REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggesstions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA, 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any oenalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
28-09-2016	Final Report	1-Oct-2014 - 30-Sep-2016

4. TITLE AND SUBTITLE
Final Report: Near Real-Time Quantification of Stochastic Model Parameters

5a. CONTRACT NUMBER

5b. GRANT NUMBER
W911NF-14-C-0151

5c. PROGRAM ELEMENT NUMBER
665502

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

6. AUTHORS
Dr. William J. Browning, Thomas E. Wood, Dr. Roger Ghanem, Dr. Paul K. Newton, Dr. Panagiotis Tsilifis, Dr. H.T. Banks, Dr. Jared Catenacci

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
Applied Mathematics, Inc.
1622 Route 12, Box 637

Gales Ferry, CT 06335 -1532

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
66142-MA-ST2.1

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT
Report developed under Topic #A13A-T009, contract W911NF-14-C-0151.
Frequentist and Bayesian approaches for solving the inverse problem are developed with the model parameters treated as stochastic parameters. The algorithms are coded and results are compared for a bacteria growth model and experiment datasets. Forward uncertainty propagation methods are developed that account for the variability and covariance structures of the estimated model parameters due to variability in the measurements and the model sensitivity.

15. SUBJECT TERMS
STTR Report, Inverse problem, Stochastic parameter estimation, non-parametric estimation, uncertainty quantification

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			William Browning
UU	UU	UU	UU		19b. TELEPHONE NUMBER 860-464-7259

Standard Form 298 (Rev 8/98)
Prescribed by ANSI Std. Z39.18

Report Title

Final Report: Near Real-Time Quantification of Stochastic Model Parameters

ABSTRACT

Report developed under Topic #A13A-T009, contract W911NF-14-C-0151.
Frequentist and Bayesian approaches for solving the inverse problem are developed with the model parameters treated as stochastic parameters. The algorithms are coded and results are compared for a bacteria growth model and experiment datasets. Forward uncertainty propagation methods are developed that account for the variability and covariance structures of the estimated model parameters due to variability in the measurements and the model sensitivity.

Enter List of papers submitted or published that acknowledge ARO support from the start of the project to the date of this printing. List the papers, including journal references, in the following categories:

(a) Papers published in peer-reviewed journals (N/A for none)

Received Paper

TOTAL:

Number of Papers published in peer-reviewed journals:

(b) Papers published in non-peer-reviewed journals (N/A for none)

Received Paper

TOTAL:

Number of Papers published in non peer-reviewed journals:

(c) Presentations

Number of Presentations: 0.00

Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Received Paper

TOTAL:

Number of Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Peer-Reviewed Conference Proceeding publications (other than abstracts):

Received Paper

TOTAL:

Number of Peer-Reviewed Conference Proceeding publications (other than abstracts):

(d) Manuscripts

Received Paper

TOTAL:

Number of Manuscripts:

Books

Received Book

TOTAL:

Received Book Chapter

TOTAL:

Patents Submitted

Patents Awarded

Awards

Graduate Students

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Post Doctorates

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Faculty Supported

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Under Graduate students supported

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Student Metrics

This section only applies to graduating undergraduates supported by this agreement in this reporting period

The number of undergraduates funded by this agreement who graduated during this period: 0.00

The number of undergraduates funded by this agreement who graduated during this period with a degree in science, mathematics, engineering, or technology fields:..... 0.00

The number of undergraduates funded by your agreement who graduated during this period and will continue to pursue a graduate or Ph.D. degree in science, mathematics, engineering, or technology fields:..... 0.00

Number of graduating undergraduates who achieved a 3.5 GPA to 4.0 (4.0 max scale):..... 0.00

Number of graduating undergraduates funded by a DoD funded Center of Excellence grant for Education, Research and Engineering:..... 0.00

The number of undergraduates funded by your agreement who graduated during this period and intend to work for the Department of Defense 0.00

The number of undergraduates funded by your agreement who graduated during this period and will receive scholarships or fellowships for further studies in science, mathematics, engineering or technology fields:..... 0.00

Names of Personnel receiving masters degrees

NAME

Total Number:

Names of personnel receiving PHDs

NAME

Total Number:

Names of other research staff

NAME	PERCENT_SUPPORTED

FTE Equivalent:
Total Number:

Sub Contractors (DD882)

1 a. University of Southern California

1 b. Department Of Contracts And Grant
3720 South Flower Street
Los Angeles CA 900890701

Sub Contractor Numbers (c): 15-01
Patent Clause Number (d-1):
Patent Date (d-2):
Work Description (e):
Sub Contract Award Date (f-1): 1/1/15 12:00AM
Sub Contract Est Completion Date(f-2): 9/30/16 12:00AM

---

1 a. University of Southern California

1 b. Department of Contracts and Grants
3720 South Flower Street
Los Angeles CA 900890701

Sub Contractor Numbers (c): 15-01
Patent Clause Number (d-1):
Patent Date (d-2):
Work Description (e):
Sub Contract Award Date (f-1): 1/1/15 12:00AM
Sub Contract Est Completion Date(f-2): 9/30/16 12:00AM

Inventions (DD882)

Scientific Progress

Technology Transfer

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*
28-09-2016

2. REPORT TYPE
FINAL REPORT

3. DATES COVERED *(From - To)*
1 October 2014 - 30 September 2016

4. TITLE AND SUBTITLE
PHASE II FINAL REPORT, Army STTR Topic No. A13A-009, Near Real-Time Quantification of Stochastic Model Parameters

5a. CONTRACT NUMBER
W911NF-14-C-0151

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dr. William J. Browning, Thomas E. Wood, Dr. Roger Ghanem, Dr. Paul K. Newton, Dr. Panagiotis Tsilifis, Dr. H.T. Banks, Dr. Jared Catenacci

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Applied Mathematics, Inc., 1622 Route 12, P.O. Box 637, Gales Ferry, CT 06335; University of Southern California, Department of Aerospace & Mechanical Engineering, Los Angeles, CA 90089; Center for Research in Scientific Computation, North Carolina State University, Raleigh, NC 27695

8. PERFORMING ORGANIZATION REPORT NUMBER
1501-0024

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Army Research Office
ARO-RTP
4300 S. Miami Blvd.
Durham, NC 27703

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution authorized to U.S. Government Agencies only; contains proprietary information.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Report developed under Topic #A13A-T009, contract W911NF-14-C-0151.
Frequentist and Bayesian approaches for solving the inverse problem are developed with the model parameters treated as stochastic parameters. The algorithms are coded and results are compared for a bacteria growth model and experiment datasets. Forward uncertainty propagation methods are developed that account for the variability and covariance structures of the estimated model parameters due to variability in the measurements and the model sensitivity.

15. SUBJECT TERMS
STTR Report, Inverse problem, Stochastic parameter estimation, non-parametric estimation, uncertainty quantification

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			William J. Browning
U	U	U	SAR	315	19b. TELEPHONE NUMBER *(Include area code)* 860-464-7259

Reset

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

SMALL BUSINESS TECHNOLOGY TRANSFER
PHASE II FINAL REPORT

Army STTR Topic No. A13A-009

Near Real-Time Quantification of Stochastic Model Parameters

Prepared under

Contract No. W911NF-14-C-0151

Prepared for

Dr. Joseph D. Myers
U.S. Army Research Office
Research Triangle Park, North Carolina

Prepared by

Applied Mathematics, Inc.
Gales Ferry, Connecticut

28 September 2016

DFARS 252.227-7018. Rights in Noncommercial Technical Data and Computer Software – Small Business Innovation Research (SBIR) Program (FEB 2014)

Contract No.:	W911NF-14-C-0151
Contractor Name:	Applied Mathematics, Inc.
Contractor Address:	1622 Route 12 P.O. Box 637 Gales Ferry, CT 06335

Expiration of SBIR Data Rights Period: 30 September 2021

The Government's rights to use, modify, reproduce, release, perform, display, or disclose these technical data are restricted by paragraph (b)(2) of the Rights in Noncommercial Technical Data and Computer Software--Small Business Innovative Research (SBIR) Program clause contained in the above identified contract. Any reproduction of technical data or portions thereof marked with this legend must also reproduce the markings. Any person, other than the Government, who has been provided access to such data must promptly notify the above named Contractor.

TABLE OF CONTENTS

1.0 INTRODUCTION 1-1

2.0 UNCERTAINTY QUANTIFICATION AND THE INVERSE PROBLEM 2-1
- 2.1 FREQUENTIST APPROACH 2-1
- 2.2 BAYESIAN APPROACH 2-5

3.0 AN APPLICATION 3-1
- 3.1 4-PARAMETER QUASI-CHEMICAL MODEL (QCM-4) 3-1
- 3.2 ALTERNATE QCM-4 MODEL FORMULATIONS 3-3
- 3.3 INVERSE PROBLEM: A FREQUENTIST SOLUTION 3-5
- 3.4 INVERSE PROBLEM: A BAYESIAN SOLUTION 3-10
- 3.5 FORWARD UNCERTAINTY PROPAGATION 3-12

4.0 UNCERTAINTY QUANTIFICATION APPLIED TO THE QCM-4 MODEL 4-1
- 4.1 SOLVING THE SYSTEM OF ODEs 4-1
- 4.2 MINIMIZING SSR 4-5
- 4.3 DATASETS 4-5
- 4.4 FREQUENTIST AND BAYESIAN SOLUTION COMPARISON 4-7
- 4.5 FORWARD UNCERTAINTY PROPAGATION 4-18

5.0 CONCLUSIONS 5-1

6.0 AREAS FOR FURTHER INVESTIGATION 6-1
- 6.1 REDUCING UNCERTAINTY IN PARAMETER ESTIMATES 6-1
- 6.2 ALTERNATE BAYESIAN METHODS FOR SOLVING THE INVERSE PROBLEM 6-14
- 6.3 AGGREGATE DATA 6-16

APPENDIX A, MODEL DATASETS A-1

APPENDIX B, SOURCE CODE B-1

APPENDIX C, BACTERIA COUNT VARIABILITY C-1

LIST OF ILLUSTRATIONS

Figure 3-1. QCM-4 Model State Variables vs. Time for $\kappa = [0.1, 5.0, 10.0, 2.0]^T$ 3-3

Figure 3-2. Example Plot of Four State vs. Time for Alternate Bacteria Kinetic Models 3-10

Figure 4-1. Example QCM-4 Growth Curves 4-2

Figure 4-2. Regions of Influence of Rate Parameters on Growth Curve 4-4

Figure 4-3. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 1, Dataset 1 4-8

Figure 4-4. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 1, Dataset 2 4-9

Figure 4-5. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 1, Dataset 3 4-10

Figure 4-6. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 1, Dataset 4 4-11

Figure 4-7. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 1 4-12

Figure 4-8. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 2 4-13

Figure 4-9. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 3 4-14

Figure 4-10. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 4 4-15

Figure 4-11. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 5 4-16

Figure 4-12. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 6 4-17

Figure 4-13. Forward Uncertainty Propagation for Small Initial Growth 4-21

Figure 4-14. Forward Uncertainty Propagation for Large Growth – Slow Death 4-22

Figure 4-15. Forward Uncertainty Propagation for Large Initial Growth – Fast Death 4-23

Figure 4-16. Forward Uncertainty Propagation for Large Fast Growth – Slow Death 4-24

Figure 4-17. Forward Uncertainty Propagation for Linear Death 4-25

Figure 4-18. Forward Uncertainty Propagation for Non-Linear Death 4-26

Figure 6-1. Example of Alternate Model Fits to Data with Death Only Kinetics 6-5

Figure 6-2. Example of Alternate Model Fits to Data with Primarily Growth Kinetics 6-6

Figure 6-3. Example of Alternate Model Fits to Data with Death and Growth Kinetics 6-8
Figure 6-4. Normalized Sensitivities of Model Parameters for Example with $\kappa = 1.00$, $n_s = 2$, and $n = 4$ 6-12
Figure 6-5. Bayesian Inverse Problem Using Gaussian Mixtures 6-15
Figure 6-6. Value of Model Parameters from MCMC Algorithm Runs 6-17
Figure 6-7. Prior and Posterior Distributions for A. ξ_1, B. ξ_2, C. ξ_3, and D. ξ_4 6-17
Figure 6-8. MCMC Algorithm Autocorrelations for A. ξ_1, B. ξ_2, C. ξ_3, and D. ξ_4 6-18
Figure 6-9. Model Output and Uncertainty for SDE Method 6-18
Figure 6-10. Simulated Growth Data with $\alpha = 3.7$ (blue dots) and $\alpha = 2.7$ (green circles) 6-19
Figure 6-11. Dataset Produced by Randomly Sampling the Two Trajectories in Figure 6-9 6-19
Figure A-1. QCM-4 Model Fits for Figure 1, Reference [b] Datasets A-3
Figure A-2. QCM-4 Model Fits for Figure 2, Reference [b] Datasets A-4
Figure A-3. QCM-4 Model Fits for Figure 3, Reference [b] Datasets A-5
Figure A-4. QCM-4 Model Fits for Figure 4, Reference [b] Datasets A-6
Figure A-5. QCM-4 Model Fits for Figure 5, Reference [b] Datasets A-7
Figure A-6. QCM-4 Model Fits for Figure 6, Reference [b] Datasets A-8
Figure A-7. QCM-4 Model Fits for Figure 7, Reference [b] Datasets A-9
Figure A-8. QCM-4 Model Fits for Figure 8, Reference [b] Datasets A-10
Figure A-9. QCM-4 Model Fits for Figure 9, Reference [b] Datasets A-11
Figure A-10. QCM-4 Model Fits for Figure 10, Reference [b] Datasets A-12
Figure A-11. Forward Uncertainty Propagation for Dataset 1 in Figure A-1 ..... A-13
Figure A-12. Forward Uncertainty Propagation for Dataset 2 in Figure A-1 ..... A-14
Figure A-13. Forward Uncertainty Propagation for Dataset 3 in Figure A-1 ..... A-15
Figure A-14. Forward Uncertainty Propagation for Dataset 4 in Figure A-1 ..... A-16
Figure A-15. Forward Uncertainty Propagation for Dataset 1 in Figure A-2 ..... A-17
Figure A-16. Forward Uncertainty Propagation for Dataset 2 in Figure A-2 ..... A-18
Figure A-17. Forward Uncertainty Propagation for Dataset 3 in Figure A-2 ..... A-19
Figure A-18. Forward Uncertainty Propagation for Dataset 4 in Figure A-2 ..... A-20
Figure A-19. Forward Uncertainty Propagation for Dataset 5 in Figure A-2 ..... A-21
Figure A-20. Forward Uncertainty Propagation for Dataset 6 in Figure A-2 ..... A-22

Figure A-21. Forward Uncertainty Propagation for Dataset 1 in Figure A-3 ..... A-23
Figure A-22. Forward Uncertainty Propagation for Dataset 2 in Figure A-3 ..... A-24
Figure A-23. Forward Uncertainty Propagation for Dataset 3 in Figure A-3 ..... A-25
Figure A-24. Forward Uncertainty Propagation for Dataset 4 in Figure A-3 ..... A-26
Figure A-25. Forward Uncertainty Propagation for Dataset 5 in Figure A-3 ..... A-27
Figure A-26. Forward Uncertainty Propagation for Dataset 1 in Figure A-4 ..... A-28
Figure A-27. Forward Uncertainty Propagation for Dataset 2 in Figure A-4 ..... A-29
Figure A-28. Forward Uncertainty Propagation for Dataset 3 in Figure A-4 ..... A-30
Figure A-29. Forward Uncertainty Propagation for Dataset 4 in Figure A-4 ..... A-31
Figure A-30. Forward Uncertainty Propagation for Dataset 1 in Figure A-5 ..... A-32
Figure A-31. Forward Uncertainty Propagation for Dataset 2 in Figure A-5 ..... A-33
Figure A-32. Forward Uncertainty Propagation for Dataset 3 in Figure A-5 ..... A-34
Figure A-33. Forward Uncertainty Propagation for Dataset 1 in Figure A-6 ..... A-35
Figure A-34. Forward Uncertainty Propagation for Dataset 2 in Figure A-6 ..... A-36
Figure A-35. Forward Uncertainty Propagation for Dataset 1 in Figure A-7 ..... A-37
Figure A-36. Forward Uncertainty Propagation for Dataset 2 in Figure A-7 ..... A-38
Figure A-37. Forward Uncertainty Propagation for Dataset 3 in Figure A-7 ..... A-39
Figure A-38. Forward Uncertainty Propagation for Dataset 1 in Figure A-8 ..... A-40
Figure A-39. Forward Uncertainty Propagation for Dataset 2 in Figure A-8 ..... A-41
Figure A-40. Forward Uncertainty Propagation for Dataset 3 in Figure A-8 ..... A-42
Figure A-41. Forward Uncertainty Propagation for Dataset 4 in Figure A-8 ..... A-43
Figure A-42. Forward Uncertainty Propagation for Dataset 1 in Figure A-9 ..... A-44
Figure A-43. Forward Uncertainty Propagation for Dataset 2 in Figure A-9 ..... A-45
Figure A-44. Forward Uncertainty Propagation for Dataset 3 in Figure A-9 ..... A-46
Figure A-45. Forward Uncertainty Propagation for Dataset 4 in Figure A-9 ..... A-47
Figure A-46. Forward Uncertainty Propagation for Dataset 5 in Figure A-9 ..... A-48
Figure A-47. Forward Uncertainty Propagation for Dataset 1 in Figure A-10 ..... A-49
Figure A-48. Forward Uncertainty Propagation for Dataset 2 in Figure A-10 ..... A-50
Figure A-49. Forward Uncertainty Propagation for Dataset 3 in Figure A-10 ..... A-51

LIST OF TABLES

Table 6-1. Frequentist Model Comparison Using Dataset 1 of Figure 1 in Reference [b] 6-2
Table 6-2. Frequentist Model Comparison Using Dataset 2 of Figure 1 in Reference [b] 6-2
Table 6-3. Bayesian Model Comparison Using Dataset 1 of Figure 1 in Reference [b] 6-3
Table 6-4. Bayesian Model Comparison Using Dataset 2 of Figure 1 in Reference [b] 6-3
Table 6-5. Alternate and QCM-4 Model Fits and Standard Errors for Dataset 5 of Figure 3, Reference [b] 6-4
Table 6-6. AIC Analysis Values for Dataset 5 of Figure 3, Reference [b] 6-4
Table 6-7. Alternate and QCM-4 Model Fits and Standard Errors for Dataset 1 of Figure 3 in Reference [b] 6-6
Table 6-8. AIC Analysis Values for Dataset 1 of Figure 3, Reference [b] 6-7
Table 6-9. Alternate and QCM-4 Model Fits and Standard Errors for Dataset 4 of Figure 1, Reference [b] 6-8
Table 6-10. AIC Analysis Values for Dataset 4 of Figure 1, Reference [b] 6-9
Table 6-11. Values of Auxiliary Parameters Used in the Example 6-11
Table 6-12. Results of Algorithm 2 for $n_s = 2$ and $n = 4$ 6-11
Table 6-13. Results of Algorithm 2 for $n_s = 2$ and $n = 5$ 6-11
Table 6-14. Results of Algorithm 2 for $n_s = 2$ and $n = 6$ 6-12
Table 6-15. Value of ELBO for Gaussian Mixtures 6-14
Table A-1. Description of Datasets A-2
Table B-1. Opensource Functions from MATLAB Central B-1
Table B-2. Frequentist UQ MATLAB Functions B-2
Table B-3. Opensource Python Packages B-158
Table B-4. Bayesian UQ Python Script Files B-158

ACKNOWLEDGEMENT

We would like to acknowledge the splendid cooperation and support we received from the Army Research Office Project Manager, Dr. Joseph D. Myers.

We would also like to thank Dr. Christopher Doona, Molecular Sciences and Engineering Team, and John Player and Steve Arcidiacono, Biological Sciences Team, Natick Soldier Research, Development & Engineering Center (NSRDEC), for helpful discussions.

The project team members are Dr. William J. Browning, Thomas E. Wood, Lisa McManus, Applied Mathematics, Inc.; Dr. Roger Ghanem, Dr. Paul K. Newton, Dr. Panagiotis Tsilifis, University of Southern California, Department of Aerospace and Mechanical Engineering; Dr. H.T. Banks, Dr. Jared Catenacci, North Carolina State University, Center for Research in Scientific Computation.

1.0 INTRODUCTION

Mathematical models of physical and biological systems with parameters distributed probabilistically requires the estimation of a probability measure over the set of admissible parameters. For use in applications, algorithms that can rapidly estimate model parameters in real-time are needed.

We quantify the uncertainty in estimated model parameters using both a Bayesian and a Frequentist approach. We then apply these methods to a class of quasi-chemical models (QCM) developed by the U.S. Army Natick Soldier Research Development and Engineering Center (NSRDEC) (References [a] and [b]). The QCM models, developed to model bacteria growth on food under various environmental conditions, are capable of capturing the effects of microbial lag, inactivation and tailing.

Bayesian and Frequentist approaches for solving the inverse problem are presented in this report. Solutions using the two approaches for the datasets from Reference [b] are compared. Uncertainty Quantification (UQ) methods for Forward Uncertainty Propagation are also applied using the datasets.

Organization of Report. The report is organized as follows. The inverse problem and its mathematical solution using both a Bayesian and a Frequentist approach are presented in Section 2. A 4-parameter Quasi-Chemical Model (QCM-4) for bacteria growth developed by the U.S. Army is described in Section 3. The QCM-4 model is used as an example application for applying the UQ methods. The results of applying the UQ methods to the QCM-4 model are presented in Section 4. Conclusions are provided in Section 5. Areas for further investigation are discussed in detail in Section 6.

The QCM-4 experiment datasets are provided in Appendix A. Code for solving the inverse problem is given in Appendix B. Variability in plate counting of bacteria colonies is discussed in Appendix C.

2.0 UNCERTAINTY QUANTIFICATION AND THE INVERSE PROBLEM

In this section we describe how UQ methods are used to solve and evaluate a type of problem that consists of a model of a dynamic process and measurements related to the process, in which there is inherent uncertainty in both the model and the measurements.

Assume there is some process in which a state $\mathbf{x} = \left[x_1, \ldots, x_{N_x}\right]^{\mathrm{T}}$ of N_x variables changes over time, the initial value of the state ($\mathbf{x}(0)$) is known, and there is a mathematical model of N_κ parameters $\boldsymbol{\kappa}$ that relates the change in the state over time to functions $\mathbf{f}$ of the state variables. Mathematically this can be represented as

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{f}\left(\mathbf{x}\left(t;\boldsymbol{\kappa}\right);\boldsymbol{\kappa}\right) . \tag{2-1}$$

If a set of measurements exist that are related to the dynamic process, then the inverse problem is to estimate the model parameters, $\hat{\boldsymbol{\kappa}}$, that best fit the measurements. UQ methods can be applied for formulating and solving the inverse problem, conducting sensitivity analysis to understand the sensitivity of the model to variability in parameter estimates, developing parameter reduction techniques for reducing uncertainty, and propagating uncertainty forward through the model.

Sections 2.1 and 2.2 discuss frequentist and Bayesian approaches for solving the inverse problem, respectively.

2.1 FREQUENTIST APPROACH

2.1.1 The Inverse Problem

Let $g(t;\boldsymbol{\kappa})$ be the observation process, which is a function of the state $\mathbf{x}$ of the modeled process at time t (Reference [c]). To fit the model to a set of n measurements, assume a statistical model of the form

$$Y_j = g\left(t_j;\boldsymbol{\kappa}_0\right) + \varepsilon_j , \quad j = 1,\ldots,n , \tag{2-2}$$

where Y_j is a measurement of the dynamic process at time t_j with error ε_j, and $\boldsymbol{\kappa}_0$ are the true parameters of the model at time t_j (i.e. $\mathrm{E}[Y_j] = g(t_j;\boldsymbol{\kappa}_0)$). The estimated parameters, $\boldsymbol{\kappa}$, can be found using ordinary least squares given by

$$\boldsymbol{\kappa} = \underset{\boldsymbol{\kappa}\in\Omega}{\arg\min} \sum_{j=1}^{n} \left(Y_j - g\left(t_j;\boldsymbol{\kappa}\right)\right)^2 , \tag{2-3}$$

where Ω is the sample space for the model parameters. If y_j is a realization of Y_j, and ε_j a realization of $\mathcal{E}_j$ for j = 1,...,n measurements, then

$$y_j = g(t_j;\boldsymbol{\kappa}_0) + \varepsilon_j, \quad j = 1,\ldots,n, \tag{2-4}$$

and the estimated model parameters for the realized measurements, $\hat{\boldsymbol{\kappa}}$, that solve the inverse problem are

$$\hat{\boldsymbol{\kappa}} = \arg\min_{\boldsymbol{\kappa}\in\Omega} \sum_{j=1}^{n} \left(y_j - g(t_j;\boldsymbol{\kappa})\right)^2 . \tag{2-5}$$

The quantity being minimized in Equation (2-5), the objective or cost function for the minimization problem, is referred to as the sum of squared residuals (SSR).

2.1.2 Sensitivity Analysis

Sensitivity analysis examines the sensitivity of the process state relative to the model parameters. Traditional sensitivity functions (TSFs) can be used to evaluate the local sensitivity of an inverse problem solution as well as estimate the covariance of the estimated model parameters. The covariance can then be used to forward propagate the uncertainty in a solution through the model.

The local sensitivity of an inverse problem relates the change in the function g to changes in the model parameters $\boldsymbol{\kappa}$. For dynamic system defined by a system of differential equations, the matrix sensitivity equations are given by (Reference [c])

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\kappa}}\right) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \boldsymbol{\kappa}} + \frac{\partial \mathbf{f}}{\partial \boldsymbol{\kappa}} . \tag{2-6}$$

For longitudinal measurements of the form $\boldsymbol{g}(t;\boldsymbol{\kappa}) = C\mathbf{x}(t;\boldsymbol{\kappa})$ where C is an observation operator, Equation (2-6) can be used to obtain

$$\frac{\partial \boldsymbol{g}(t;\boldsymbol{\kappa})}{\partial \kappa_j} = C\frac{\partial \mathbf{x}(t;\boldsymbol{\kappa})}{\partial \kappa_j} . \tag{2-7}$$

The traditional sensitivity functions, $\mathbf{s}$, are given by

$$\mathbf{s}_{\kappa_j}(t) = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\kappa_j} . \tag{2-8}$$

The $N_x \times N_\kappa$ matrix $\mathbf{s}$ is found by solving the matrix system

$$\frac{\mathrm{d}\mathbf{s}}{\mathrm{d}t} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{s}(t) + \frac{\partial \mathbf{f}}{\partial \boldsymbol{\kappa}}, \quad \mathbf{s}(0) = \mathbf{0}_{N_x \times N_\kappa} . \tag{2-9}$$

The sensitivity matrix $\mathbf{F}(\boldsymbol{\kappa})$ for n measurements is then defined by

$$\mathbf{F}_{ij}(\boldsymbol{\kappa}) = \left(\frac{\partial \boldsymbol{g}(t_i;\boldsymbol{\kappa})}{\partial \kappa_j} \right). \tag{2-10}$$

Propagating the uncertainty of a solution through the model uses the fact that the least squares estimator $\boldsymbol{\kappa}$ has the asymptotic properties (References [c] through [e]),

$$\boldsymbol{\kappa} \sim \mathcal{N}(\boldsymbol{\kappa}_0, \Sigma_0), \tag{2-11}$$

where $\boldsymbol{\kappa}_0$ are the true model parameters, and $\boldsymbol{\Sigma}_0$ is the true covariance of $\boldsymbol{\kappa}$. If σ_0^2 is the true variance in the measurements, then the covariance can be approximated by

$$\Sigma_0 \approx \sigma_0^2 \left(\mathbf{F}^{\mathrm{T}}(\boldsymbol{\kappa}_0)\mathbf{F}(\boldsymbol{\kappa}_0) \right)^{-1}. \tag{2-12}$$

The values for $\boldsymbol{\kappa}_0$ and σ_0^2 are unknown, but can be estimated by $\hat{\boldsymbol{\kappa}}$ from Equation (2-5) and

$$\hat{\sigma}_0^2 = \frac{1}{n - N_\kappa} \sum_{j=1}^{n} \left(y_j - g(t_j;\hat{\boldsymbol{\kappa}}) \right)^2 , \tag{2-13}$$

respectively. The covariance can then be estimated using

$$\Sigma_0 \approx \hat{\Sigma} = \hat{\sigma}_0^2 \left(\mathbf{F}^{\mathrm{T}}(\hat{\boldsymbol{\kappa}}_0)\, \mathbf{F}(\hat{\boldsymbol{\kappa}}_0) \right)^{-1}. \tag{2-14}$$

The uncertainty in a solution across all model parameters can be forward propagated assuming the multivariate normal distribution given in Equation (2-11). For individual model parameter sensitivity, it is often convenient to construct confidence intervals. Let $SE_j = \sqrt{\hat{\Sigma}_{jj}}$ be the standard error of the jth model parameter. Let $t_{1-\rho/2}$ be the critical value for a Student's t-distribution where

$$\mathrm{P}\left[S > t_{1-\rho/2} \right] = \rho/2 , \tag{2-15}$$

for a random variate S with n-N_κ degrees of freedom. The 100 × (1-ρ)% confidence interval for model parameter κ_j is then given by

$$\left[\hat{\kappa}_j - t_{1-\rho/2}\, SE_j , \hat{\kappa}_j + t_{1-\rho/2}\, SE_j \right] . \tag{2-16}$$

2.1.3 Model Reduction via Parameter Ranking and Model Comparison Testing

Models of complex, dynamic processes do not always have analytic solutions. Estimating the model parameters that best fit a given set of measured data is accomplished using a numerical solver. Solutions may be insensitive to one or more of the solution parameters due to inherent conditions of the process over the model parameter space, or numerical issues related to the tolerance values of the solver and the error in the measured data used to estimate the model parameters. In either case, some components of the model may need to be replaced, modified, or removed (Reference [f]) in order to reduce sensitivity.

One method for handling situations in which a model solution is insensitive to one or more model parameters is to determine which parameters are significant, set the insignificant parameters to nominal values, and solve Equation (2-5) using a reduced set of model parameters. The first step is to perform a parameter ranking based on the orthogonalization of the sensitivity matrix $\mathbf{F}(\boldsymbol{\kappa})$. This can be achieved using a thin, or economy-size, QR decomposition of the matrix $\mathbf{F}$ so that $\mathbf{FP} = \mathbf{QR}$, where $\mathbf{P}$ is a permutation matrix. The order of the permutations gives the ranking of the parameters where the rankings are chosen according to the 2-norm of the sensitivity with respect to the parameter.

The reduction process begins by first estimating only the most significant parameter from the sensitivity-based parameter ranking scheme while the remaining parameters are set to nominal values. The choice of nominal values for testing the significance of the model parameters is arbitrary. If a solution is truly insensitive to a model parameter, then any value for an insignificant parameter should not alter the choice of a significant parameter.

The second step is to estimate the two most significant parameters and use a statistically-based model comparison (i.e. hypothesis) test to determine if the data can be adequately described by the single parameter model as compared to the two parameter model. If it is determined that the second parameter significantly improves the model fit to the data, then the process of adding an additional significant parameter to the model will continue until there is no significant change to the model fit.

Since nested models are being compared, the null hypothesis being tested is that the constrained model provides an adequate fit to the data. Let $J(\boldsymbol{\kappa};\mathbf{y})$ and $J(\boldsymbol{\kappa}_H;\mathbf{y})$ denote the value of the objective function in Equation (2-5), where $\boldsymbol{\kappa}_H \subset \boldsymbol{\kappa}$ and $\dim(\boldsymbol{\kappa} - \boldsymbol{\kappa}_H) = 1$ (i.e. $\boldsymbol{\kappa}_H$ contains one less model parameter than $\boldsymbol{\kappa}$). Let T be the difference in the objective functions given by

$$T(\mathbf{y}) = J(\boldsymbol{\kappa}_H;\mathbf{y}) - J(\boldsymbol{\kappa};\mathbf{y}) \geq 0, \tag{2-17}$$

and define the test statistic for the hypothesis test, $\mathcal{U}(\mathbf{y})$, as

$$\mathcal{U}(\mathbf{y}) = \frac{n\,T(\mathbf{y})}{J(\boldsymbol{\kappa};\mathbf{y})}. \tag{2-18}$$

The value of $\mathcal{U}(\mathbf{y})$ converges to $\mathcal{U}$ in distribution (Reference [c]), where $\mathcal{U} \sim \chi^2(1)$, and $\chi^2(1)$ is a chi-square distribution with one degree of freedom. If the statistic $\mathcal{U}(\mathbf{y}) > \tau$, then the null hypothesis is rejected at a 100 × (1-β)% confidence level where

$$\mathrm{P}[\chi(1) > \tau] = \beta . \tag{2-19}$$

2.2 BAYESIAN APPROACH

2.2.1 Variational Bayesian Inference

Assume that the observations (i.e. measurements) y_j are realizations given by Equation (2-4) where ε_i are realizations of random measurement noise at time t_i. Let $\mathbf{y} = [y_1,...,y_n]^T$ be the set of all observations and $\boldsymbol{\varepsilon} = [\varepsilon_1,...,\varepsilon_n]^T$ the set of additive noises. The statistical model of the observations can then be written as

$$\mathbf{y} = \mathcal{G}(\boldsymbol{\kappa}) + \boldsymbol{\varepsilon} , \tag{2-20}$$

where $\mathcal{G}(\boldsymbol{\kappa}) = [g(t_1;\boldsymbol{\kappa}),..., g(t_n;\boldsymbol{\kappa})]^T$. The additive noise ε_i for each observation is assumed to be independent and identically distributed Gaussian random variables that are independent of the parameters $\boldsymbol{\kappa}$. That is, $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0},\sigma\mathbf{I}_n)$, and $\mathrm{p}(\boldsymbol{\kappa},\sigma) = \mathrm{p}(\boldsymbol{\kappa})\mathrm{p}(\sigma)$ where p() is a probability. Since σ is non-negative, and for the specific example that will be discussed in Section 3 $\boldsymbol{\kappa}$ is non-negative, it is convenient to parameterize the problem over the real numbers as

$$\boldsymbol{\xi} = \ln(\boldsymbol{\kappa}), \text{ and} \tag{2-21}$$

$$\omega = \tfrac{1}{2}\ln(\sigma). \tag{2-22}$$

The Bayesian paradigm consists of considering that $\boldsymbol{\xi}$ follows a prior distribution p(ω) which, after collecting the observations $\mathbf{y}$, is updated using Bayes' rule to

$$\mathrm{p}(\boldsymbol{\xi} \mid \mathbf{y}) = \frac{\mathrm{p}(\mathbf{y} \mid \boldsymbol{\xi})\mathrm{p}(\boldsymbol{\xi})}{\mathrm{p}(\mathbf{y})} = \frac{\mathrm{p}(\mathbf{y} \mid \boldsymbol{\xi})\mathrm{p}(\boldsymbol{\xi})}{\int \mathrm{p}(\mathbf{y} \mid \boldsymbol{\xi})\mathrm{p}(\boldsymbol{\xi})d\boldsymbol{\xi}} . \tag{2-23}$$

To allow for inference on the Gaussian measurement noise, Equation (2-23) can be modified as

$$\mathrm{p}(\boldsymbol{\xi},\omega \mid \mathbf{y}) = \frac{\mathrm{p}(\mathbf{y} \mid \boldsymbol{\xi},\omega)\mathrm{p}(\boldsymbol{\xi})\mathrm{p}(\omega)}{\mathrm{p}(\mathbf{y})} = \frac{\mathrm{p}(\mathbf{y} \mid \boldsymbol{\xi},\omega)\mathrm{p}(\boldsymbol{\xi})\mathrm{p}(\omega)}{\iint \mathrm{p}(\mathbf{y} \mid \boldsymbol{\xi},\omega)\mathrm{p}(\boldsymbol{\xi})\mathrm{p}(\omega)\mathrm{d}\boldsymbol{\xi}\,\mathrm{d}\omega} , \tag{2-24}$$

and denote with $\boldsymbol{\theta} \in \mathbb{R}^{N_\kappa+1}$ the augmented set of parameters $\boldsymbol{\theta} = [\boldsymbol{\xi},\omega]$.

Typically, the posterior distribution $\mathrm{p}(\boldsymbol{\theta}|\mathbf{y})$ is explored via Markov Chain Monte Carlo (MCMC) techniques (Reference [g]) which enable sampling from a Markov Chain that has $\mathrm{p}(\boldsymbol{\theta}|\mathbf{y})$ as its

invariant distribution and, therefore, the generated samples can be thought of as (asymptotically) drawn from that distribution.

In order for that assumption to be accurate, it is often required to draw thousands of samples until the generated chain approaches its invariant density. The speed of convergence of the chain depends on the nature of the true posterior and on the generating procedure. For instance, variations of the Metropolis-Hastings algorithm (References [h] through [k]) exist in the literature which provide different acceptance rates and eventually faster or slower convergence. However, in all but the simplest cases, several thousands samples are required, which means that the forward model must run equally many times, resulting in large or possibly unaffordable computational costs. In order to avoid excessive simulations and reduce the computational cost of exploring the posterior distribution, a variational approach of the problem is considered (References [l] and [m]) that is based on approximating $p(\boldsymbol{\theta}|\mathbf{y})$ by a known probability distribution chosen from a known family.

Let $\mathcal{Q}$ be a family of probability densities parameterized by $\boldsymbol{\lambda}$ $\left(\mathcal{Q} = \left\{ q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}), \boldsymbol{\lambda} \in \mathbb{R}^{N_\lambda} \right\}\right)$. The goal is to find the element of $\mathcal{Q}$ that is as close as possible, in some sense, to the true posterior $p(\boldsymbol{\theta}|\mathbf{y})$. The distance between two probability distributions can be quantified using the Kullback-Leibler (KL) divergence given by (Reference [n])

$$\mathrm{KL}\left[q(\boldsymbol{\theta}) \,\|\, p(\boldsymbol{\theta} \mid \mathbf{y})\right] = \int_{\mathbb{R}^{N_\kappa+1}} \ln\left[\frac{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})}{p(\boldsymbol{\theta} \mid \mathbf{y})}\right] q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) d\boldsymbol{\theta} \,. \tag{2-25}$$

The KL divergence does not define a metric since it does not satisfy the triangle inequality. However, it is often used as such since it is always non-negative and $\mathrm{KL}[q\|p] = 0 \Leftrightarrow q = p$ a.s. In addition, $\mathrm{KL}[q\|p] \to 0$ implies that $q \to p$ in total variation, a result that guarantees proximity between two probability measures. Intuitively, the KL divergence can be thought of as the information loss of approximating the true posterior $p(\boldsymbol{\kappa}|\mathbf{y})$ by $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$. It is clear from the above that minimizing the KL divergence can provide a way to find approximations of the true posterior; we, therefore, state the optimization problem as finding $\boldsymbol{\lambda}^*$ such that

$$q\left(\boldsymbol{\theta} \mid \boldsymbol{\lambda}^*\right) = \underset{\boldsymbol{\lambda} \in \mathbb{R}^{N_\lambda}}{\arg\min}\, \mathrm{KL}\left[\, q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) \,\|\, p(\boldsymbol{\theta} \mid \mathbf{y})\,\right]. \tag{2-26}$$

Trivially, if the true posterior $p(\boldsymbol{\theta}|\mathbf{y})$ is in $\mathcal{Q}$, one should expect the result of the optimization to be 0; otherwise, it will be a strictly positive value. In practice, it is not possible to evaluate the above expression since $p(\boldsymbol{\theta}|\mathbf{y})$ is not known. In the next section, we formulate an equivalent optimization problem that is possible to solve by deriving a lower bound of the evidence $p(\mathbf{y})$.

2.2.2 Evidence Lower Bound (ELBO)

Substituting Baye's rule (Equation (2-24)) into the KL divergence (Equation(2-25)) gives the relation

$$\ln p(\mathbf{y}) = \mathcal{F}[q] + \mathrm{KL}[q(\boldsymbol{\theta}) \| p(\boldsymbol{\theta} \mid \mathbf{y})], \tag{2-27}$$

where

$$\mathcal{F}[q] = \int \ln\left[\frac{p(\mathbf{y},\boldsymbol{\theta})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})}\right] q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) d\boldsymbol{\lambda}\,, \tag{2-28}$$

$$= \mathcal{H}[q] + \int \ln[p(\mathbf{y},\boldsymbol{\theta})] q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) d\boldsymbol{\theta}\,, \tag{2-29}$$

with

$$\mathcal{H}[q] = -\int q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) \ln[q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})] d\boldsymbol{\theta} \tag{2-30}$$

being the entropy of $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$. Observing that the left hand side of Equation (2-27) is independent of $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$, minimizing $\mathrm{KL}[q||p]$ is equivalent to maximizing the quantity $\mathcal{F}[q]$ which, due to the positivity of $\mathrm{KL}[q||p]$, always satisfies

$$\mathcal{F}[q] \le p(\mathbf{y})\,. \tag{2-31}$$

$\mathcal{F}[q]$ is referred to as the evidence lower bound (ELBO). Maximization of $\mathcal{F}[q]$ is a feasible task since it does not depend on the posterior or the evidence distributions, but only on the joint distribution $p(\mathbf{y},\boldsymbol{\theta}) = p(\mathbf{y}|\boldsymbol{\theta})\, p(\boldsymbol{\theta})$ which, in general, is known.

2.2.3 Approximating Schemes

In order to evaluate the integral in Equation (2-29) a numerical method must be used. Numerical integration or Monte Carlo will be able to provide accurate results which, when used within an optimization algorithm can be as expensive as performing MCMC. Instead approximating schemes for Equation (2-29) are developed that are valid when the approximating family of distributions $\mathcal{Q}$ is the family of Gaussian mixtures, that is,

$$\mathcal{Q}_L = \left\{ q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) \,\middle|\, q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) = \frac{1}{L}\sum_{i=1}^{L} \mathcal{N}(\boldsymbol{\theta} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \right\}. \tag{2-32}$$

The optimization problem needs to be solved with respect to the parameters $\boldsymbol{\lambda} = \{\boldsymbol{\mu}_i\}_{i=1}^{L} \cup \{\boldsymbol{\Sigma}_i\}_{i=1}^{L}$ where L, the number of Gaussian components, will be assumed to be fixed and $\boldsymbol{\Sigma}_i$ must be restricted to be positive definite. For the purpose of inferring the parameters in the example discussed in Section 3, $\boldsymbol{\Sigma}_i$ will also be assumed to be diagonal which will further simplify the calculations. The latter implies that the parameters will be *a posteriori* independent, which is not always the case, but this constraint is added here for the sake of simplicity.

First, a lower bound for $\mathcal{H}$ is derived using Jensen's inequality (Reference [o]) to obtain

$$\mathcal{H}[\mathrm{q}] \geq \mathcal{H}_0[\mathrm{q}], \tag{2-33}$$

where

$$\mathcal{H}_0[\mathrm{q}] = -\frac{1}{L}\sum_{i=1}^{L} \ln q_i \ , \tag{2-34}$$

with

$$q_i = \frac{1}{L}\sum_{j=1}^{L} \mathrm{q}\left(\boldsymbol{\mu}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j\right). \tag{2-35}$$

By replacing $\mathcal{H}$[q] with $\mathcal{H}_0$[q] in Equation (2-29), a lower bound for the ELBO is derived. Next, define $\mathcal{L}$[q] to be the second term in Equation (2-29),

$$\mathcal{L}[\mathrm{q}] = \int \ln[\mathrm{p}(\mathbf{y} \mid \boldsymbol{\theta})]\mathrm{q}(\boldsymbol{\theta} \mid \boldsymbol{\lambda})\mathrm{d}\boldsymbol{\theta}\,, \tag{2-36}$$

which, by substituting for q($\boldsymbol{\theta}$|$\boldsymbol{\lambda}$) using Equation (2-32), gives

$$\mathcal{L}[\mathrm{q}] = \frac{1}{L}\int \ln[\mathrm{p}(\mathbf{y} \mid \boldsymbol{\theta})]\mathcal{N}(\boldsymbol{\theta} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\mathrm{d}\boldsymbol{\theta}\,, \tag{2-37}$$

and each integral term can be approximated by taking a 2nd order Taylor expansion of ln[p($\mathbf{y}$,$\boldsymbol{\theta}$)] about $\boldsymbol{\theta} = \boldsymbol{\mu}_i$. This gives

$$\ln \mathrm{p}(\mathbf{y},\boldsymbol{\theta}) \approx \ln \mathrm{p}(\mathbf{y},\boldsymbol{\mu}_i) + \nabla_\theta \ln[\mathrm{p}(\mathbf{y},\boldsymbol{\mu}_i)](\boldsymbol{\theta} - \boldsymbol{\mu}_i) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_i)^{\mathrm{T}} \nabla_\theta^2 \ln[\mathrm{p}(\mathbf{y},\boldsymbol{\mu}_i)](\boldsymbol{\theta} - \boldsymbol{\mu}_i), \tag{2-38}$$

where $\nabla_{\boldsymbol{\theta}}$ is the Jacobian and $\nabla_{\boldsymbol{\theta}}^2$ is the Hessian with respect to $\boldsymbol{\theta}$. Upon substituting Equation (2-38) into Equation (2-36), take the approximation for $\mathcal{L}$[q] as

$$\mathcal{L}_2[\mathrm{q}] = \mathcal{L}_0[\mathrm{q}] + \frac{1}{2L}\sum_{i=1}^{L} \mathrm{Tr}\left[\boldsymbol{\Sigma}_i \nabla_\theta^2 \ln \mathrm{p}(\mathbf{y},\boldsymbol{\mu}_i)\right], \tag{2-39}$$

where Tr is the matrix trace function, and $\mathcal{L}_0$[q] is the term resulting from the zeroth order approximation in Equation (2-38) given by

$$\mathcal{L}_0[\mathrm{q}] = \frac{1}{L}\sum_{i=1}^{L} \ln \mathrm{p}(\mathbf{y},\boldsymbol{\mu}_i). \tag{2-40}$$

Combining Equations (2-34) and (2-39) yields an approximation of the ELBO, $\mathcal{F}_2[\mathrm{q}]$, given by

$$\mathcal{F}_2[\mathrm{q}] = \mathcal{H}_0[\mathrm{q}] + \mathcal{L}_2[\mathrm{q}], \tag{2-41}$$

that can be used in an optimization algorithm to compute a local maximum of Equation (2-29).

2.2.4 Stochastic Differential Equation

In Section 2.1, the inverse problem was formulated assuming that the model parameters are random relative to experimental measurements, but constant over time. The inverse problem can be extended to allow for time-varying model parameters that represent a random process. To formulate this let $\tilde{\boldsymbol{\kappa}} = \left[k_1(t), \ldots, k_{N_\kappa}(t)\right]^{\mathrm{T}}$ and introduce Gaussian white noise on the random parameters by writing

$$k_i(t) = k_0^i + \psi_i \frac{\mathrm{d}W_t^i}{\mathrm{d}t}, \quad i = 1, \cdots, N_\kappa, \tag{2-42}$$

where k_0^i is a random initial value for $k_i(t)$; W_t^i are independent Brownian motions with $W_0^i = 0$; and ψ_i are the white noise amplitudes. The main properties of W_t^i include that any increments $\mathrm{d}W_t^i$ and $\mathrm{d}W_s^i$ for $t \neq s$ are independent and that $\mathrm{d}W_t \sim \mathcal{N}\left(0, \sqrt{\mathrm{d}t}\right)$. The system of differential equations for the inverse problem in Equation (2-1) can then be replaced with the Itô stochastic differential equation (SDE) given by

$$\mathrm{d}\mathbf{X}_t = \tilde{\mathbf{f}}\left(\mathbf{X}_t, \tilde{\boldsymbol{\kappa}}_0\right)\mathrm{d}t + \tilde{\boldsymbol{\sigma}}\left(\mathbf{X}_t, \boldsymbol{\psi}\right)\mathrm{d}\mathbf{W}_t, \tag{2-43}$$

where $\mathbf{X}_t = \left[X_t^1, \ldots, X_t^{N_x}\right]^{\mathrm{T}}$, $X_t^i = x_i$, $\tilde{\boldsymbol{\kappa}}_0 = \left[k_0^1, \ldots, k_0^{N_\kappa}\right]^{\mathrm{T}}$, $\boldsymbol{\psi} = \left[\psi_1, \ldots, \psi_{N_\kappa}\right]^{\mathrm{T}}$, and $\mathbf{W}_t = \left[W_t^1, \ldots, W_t^{N_\kappa}\right]^{\mathrm{T}}$. The values for the drift $\tilde{\mathbf{f}}$ are similar to those for $\mathbf{f}$ in Equation (2-1), and the values for the diffusion $\tilde{\boldsymbol{\sigma}}$ are related to the derivatives of $\mathbf{f}$. In Section 3 we will show how $\tilde{\mathbf{f}}$ and $\tilde{\boldsymbol{\sigma}}$ are constructed.

Calling Equation (2-43) an Itô-SDE refers to the fact that when the equation is written in integral form

$$\mathbf{X}_t = \mathbf{X}_0 + \int_0^t \tilde{\mathbf{f}}\left(\mathbf{X}_s, \tilde{\boldsymbol{\kappa}}_0\right)\mathrm{d}s + \int_0^t \tilde{\boldsymbol{\sigma}}\left(\mathbf{X}_s, \boldsymbol{\psi}\right)\mathrm{d}\mathbf{W}_s, \tag{2-44}$$

the integral $\int_0^t \tilde{\boldsymbol{\sigma}}\left(\mathbf{X}_s, \boldsymbol{\psi}\right)\mathrm{d}\mathbf{W}_s$ is understood to be an Itô integral (Reference [p]). Equation (2-43) now satisfies the necessary growth and Lipschitz conditions that guarantee existence and uniqueness of the solution. Therefore, the smoothing problem of identifying the paths conditioned on partial observations is a well-defined problem.

2.2.5 Smoothing and Parameter Inference

Data Assimilation Problem. Since the uncertainty about $\mathbf{X}_t$ is due to the Brownian motion $\mathbf{W}_t$ and the uncertain parameters $\boldsymbol{\psi}$ and $\tilde{\boldsymbol{\kappa}}_0$, the parameter inference problem discussed in Sections 2.2.1 through 2.2.3 becomes a data assimilation problem that treats Equation (2-43) as a signal $\mathbf{X}_t$ and aims to extract information about $\mathbf{X}_t$ from observations $\mathbf{Y} = \{y_{t_i}\}_{i=1}^n$ through updating the knowledge about $\mathbf{W}_t$, $\boldsymbol{\psi}$, and $\tilde{\boldsymbol{\kappa}}_0$. The observations are obtained by

$$y_{t_i} = h\left(\mathbf{X}_{t_i}\right) + \varepsilon_i \ , \quad i = 1, \cdots, n \tag{2-45}$$

where $h(\cdot)$ is an observation operator, ε_i is additive measurement noise, and $\{t_i\}_{i=1}^n$ are the discrete time instances where observations are collected. The additive measurement noise will be independent and identically distributed Gaussian, that is $\varepsilon_i \sim \mathcal{N}\left(0, \gamma^2\right)$.

The data assimilation problem is essentially a combination of the smoothing problem that consists of discovering the possible paths of the signal $\mathbf{X}_{[0,T]}$ that were realized, given partial observations $\mathbf{Y}$ and the parameter inference problem for estimating $\{\tilde{\boldsymbol{\kappa}}_0, \boldsymbol{\psi}\}$, again given $\mathbf{Y}$. Thus, in a Bayesian formulation of the problem, the main interest is in the joint conditional distribution of $\mathbf{X}_{[0,T]}, \tilde{\boldsymbol{\kappa}}_0, \boldsymbol{\psi} | \mathbf{Y}$, or equivalently that of $\mathbf{W}_{[0,T]}, \tilde{\boldsymbol{\kappa}}_0, \boldsymbol{\psi} | \mathbf{Y}$ since the noise paths $\mathbf{W}_{[0,T]}$ suffice to completely determine $\mathbf{X}_{[0,T]}$. For convenience, the noise quantities $\Xi_t^i = \psi_i W_t$ will be considered thus merging $\boldsymbol{\psi}$ with $\mathbf{W}_t$ since the first is nothing but a noise amplitude and setting $\boldsymbol{\Xi}_t = \left[\Xi_t^1, \ldots, \Xi_t^{N_\kappa}\right]^{\mathrm{T}}$. Also, to work with a parameter space over the entire reals, let $\tilde{\boldsymbol{\xi}} = \ln\left(\tilde{\boldsymbol{\kappa}}_0\right)$. The data assimilation problem can be stated as that of estimating the posterior distribution

$$\mathrm{d}\mu^{\mathbf{Y}}\left(\tilde{\boldsymbol{\xi}}, \boldsymbol{\Xi}_{[0,T]}\right) \propto \mathcal{L}\left(\mathbf{Y} \mid \tilde{\boldsymbol{\xi}}, \boldsymbol{\Xi}_{[0,T]}\right) \mathrm{d}\mu_0\left(\tilde{\boldsymbol{\xi}}, \boldsymbol{\Xi}_{[0,T]}\right) , \tag{2-46}$$

where $\mathcal{L}\left(\mathbf{Y} \mid \tilde{\boldsymbol{\xi}}, \boldsymbol{\Xi}_{[0,T]}\right)$ is the likelihood function and $\mathrm{d}\mu_0\left(\tilde{\boldsymbol{\xi}}, \boldsymbol{\Xi}_{[0,T]}\right) = \mathrm{d}\mu_{\tilde{\xi}}\left(\tilde{\boldsymbol{\xi}}\right) \mathrm{d}\mu_{\boldsymbol{\Xi}}\left(\boldsymbol{\Xi}_{[0,T]}\right)$ is the prior distribution as follows from Bayes' rule. In Equaion (2-46), $\mu^{\mathbf{Y}}$ is the posterior joint probability measure of $\tilde{\boldsymbol{\xi}}, \boldsymbol{\Xi}_{[0,T]}$ conditioned on the data $\mathbf{Y}$, while the joint prior μ_0 can be decomposed with respect to the measures of $\mu_{\tilde{\xi}}$ and $\mu_{\boldsymbol{\Xi}}$ of $\tilde{\boldsymbol{\xi}}$ and $\boldsymbol{\Xi}_{[0,T]}$ respectively. Note that it is assumed a priori that $\tilde{\boldsymbol{\xi}}$ and $\boldsymbol{\Xi}_{[0,T]}$ are independent. The use of this measure-theoretic notation is preferred mostly due to the fact that $\boldsymbol{\Xi}_{[0,T]}$ is infinite dimensional and, in general, its distribution does not have a density function. If the data-misfit function is defined as

$$\Phi\left(\tilde{\boldsymbol{\xi}}, \boldsymbol{\Xi}_{[0,T]}; \mathbf{Y}\right) = \frac{1}{2\gamma^2}\left(\mathbf{Y} - h\left(\mathbf{X}_{[0,T]}\right)\right)^T \left(\mathbf{Y} - h\left(\mathbf{X}_{[0,T]}\right)\right), \tag{2-47}$$

then the likelihood term takes the form

$$\mathcal{L}\left(\mathbf{Y} \mid \tilde{\xi}, \Xi_{[0,T]}\right) \propto e^{\left(-\Phi\left(\tilde{\xi}, \Xi_{[0,T]}; \mathbf{Y}\right)\right)}. \tag{2-48}$$

A Metropolis-within-Gibbs Algorithm for Posterior Exploration. In order to explore the smoothing distribution of Equation (2-46), MCMC methods and variations of the Metropolis-Hastings (MH) algorithm (References [h] and [i]) are used. A Metropolis-within-Gibbs algorithm will be used that was developed in Reference [g]. The general concept in a MH algorithm is to construct a Markov Chain $\left\{\tilde{\xi}^{j}, \Xi^{j}_{[0,T]}\right\}_{j=1}^{n}$ that converges in distribution to a target distribution, here taken to be $\mu^{\mathbf{Y}}\left(\tilde{\xi}, \Xi_{[0,T]}\right)$. This is done by an interative procedure that draws samples from a proposal distribution, say $\mathrm{d}\nu\left(\tilde{\xi}', \Xi' \mid \tilde{\xi}^{j-1}, \Xi^{j}_{[0,T]}\right)$, conditional on the previous step, and accepts with acceptance probability

$$\alpha = \min\left\{1, \frac{\mathrm{d}\mu^{\mathbf{Y}}\left(\tilde{\xi}, \Xi_{[0,T]}\right)\mathrm{d}\nu\left(\tilde{\xi}^{j-1}, \Xi^{j-1}_{[0,T]} \mid \tilde{\xi}', \Xi'_{[0,T]}\right)}{\mathrm{d}\mu^{\mathbf{Y}}\left(\tilde{\xi}^{j-1}, \Xi^{j-1}_{[0,T]}\right)\mathrm{d}\nu\left(\tilde{\xi}', \Xi'_{[0,T]} \mid \tilde{\xi}^{j-1}, \Xi^{j-1}_{[0,T]}\right)}\right\}. \tag{2-49}$$

The main issue with this approach, for high dimensional distributions is that it is hard to choose an efficient proposal distribution that generates joint samples $\tilde{\xi}, \Xi_{[0,T]}$. Note that $\Xi_{[0,T]}$ is an infinite dimensional process for which the standard MH algorithm can become arbitrarily slow. This is due to the fact that in infinite dimensions, the absolute continuity condition between the measures defined in the numerator and denominator of Equation (2-49) fails to be satisfied and is no longer a valid acceptance probability (Reference [q]). In practice, the algorithm can still run after a finite discretization but becomes extremely slow under mesh refinement. One way to bypass this issue is by formulating the algorithm in function space (References [r] and [s]), which is the approach discussed here. In addition, to further improve the performance of MCMC, a MH-within-Gibbs scheme is employed in order to provide a separate proposal for updating $\tilde{\xi}$ which is assumed to be statistically independent of $\Xi_{[0,T]}$.

A MH-within-Gibbs algorithm takes into account the Gibbs sampler principle, according to which, the chain $\left\{\tilde{\xi}^{j}, \Xi^{j}_{[0,T]}\right\}_{j=1}^{n}$ constructed by sampling $\tilde{\xi}^{j} \sim \mathrm{d}\mu^{\mathbf{Y}}_{\tilde{\xi}}\left(\tilde{\xi} \mid \Xi_{[0,T]}\right)$ and $\Xi^{j}_{[0,T]} \sim \mathrm{d}\mu^{\mathbf{Y}}_{\Xi}\left(\Xi_{[0,T]} \mid \tilde{\xi}\right)$ is Markovian and converges to the joint posterior distribution of $\tilde{\xi}, \Xi_{[0,T]} \mid \mathbf{Y}$. This results in a block-updating procedure which consists of first updating $\tilde{\xi}^{j}$ and then $\Xi^{j}_{[0,T]}$ by drawing from different proposals and accepting with different probabilities. Denote with $\mathrm{d}\nu_{\tilde{\xi}}\left(\tilde{\xi}' \mid \tilde{\xi}^{j-1}, \Xi^{j-1}_{[0,T]}\right)$ and $\mathrm{d}\nu_{\Xi}\left(\Xi'_{[0,T]} \mid \tilde{\xi}^{j}, \Xi^{j-1}_{[0,T]}\right)$ the corresponding proposal distributions for generating $\tilde{\xi}^{j}$ and $\Xi^{j}_{[0,T]}$ respectively at the jth step. Then the MH-within-Gibbs pseudo-algorithm that asymptotically generates samples for the target posterior would consist of the steps displayed in Algorithm 1.

Algorithm 1: MH-within-Gibbs

Initialize: Choose $\tilde{\xi}^0$, $\Xi^0_{[0,T]}$, and n.

for j = 1 to n do

1. Generate $\tilde{\xi}' \sim \mathrm{d}\nu_{\tilde{\xi}}\left(\tilde{\xi}' \mid \tilde{\xi}^{j-1}, \Xi^{j-1}_{[0,T]}\right)$
2. Accept with probability

$$\alpha_{\tilde{\xi}} = \min\left\{1, \frac{\mathrm{d}\mu^{\mathbf{Y}}_{\tilde{\xi}}\left(\tilde{\xi}' \mid \Xi^{j-1}_{[0,T]}\right)\mathrm{d}\nu_{\tilde{\xi}}\left(\tilde{\xi}^{j-1} \mid \tilde{\xi}', \Xi^{j-1}_{[0,T]}\right)}{\mathrm{d}\mu^{\mathbf{Y}}_{\tilde{\xi}}\left(\tilde{\xi}^{j-1} \mid \Xi^{j-1}_{[0,T]}\right)\mathrm{d}\nu_{\tilde{\xi}}\left(\tilde{\xi}' \mid \tilde{\xi}^{j-1}, \Xi^{j-1}_{[0,T]}\right)}\right\} \tag{2-50}$$

3. Generate $\mathbf{X}'_{[0,T]} \sim \mathrm{d}\nu_{\Xi}\left(\Xi'_{[0,T]} \mid \tilde{\xi}^{j}, \Xi^{j-1}_{[0,T]}\right)$
4. Accept with probability

$$\alpha_{\Xi} = \min\left\{1, \frac{\mathrm{d}\mu^{\mathbf{Y}}_{\Xi}\left(\Xi'_{[0,T]} \mid \tilde{\xi}^{j}\right)\mathrm{d}\nu_{\Xi}\left(\Xi^{j-1}_{[0,T]} \mid \tilde{\xi}^{j}, \Xi'_{[0,T]}\right)}{\mathrm{d}\mu^{\mathbf{Y}}_{\Xi}\left(\Xi^{j-1}_{[0,T]} \mid \tilde{\xi}^{j}\right)\mathrm{d}\nu_{\Xi}\left(\Xi'_{[0,T]} \mid \tilde{\xi}^{j}, \Xi^{j-1}_{[0,T]}\right)}\right\} \tag{2-51}$$

end

Typically the above procedure also includes a problem-specific burn-in period and thinning of the chain in order to ensure that our samples can be considered as i.i.d. samples drawn from the target posterior. The first might be necessary due to the fact that our samples are generated until some further step $m << n$ should be discarded since we assume that our chain that started from an arbitrary point $\left\{\tilde{\xi}^0, \Xi^0_{[0,T]}\right\}$ and might have not approached the distribution yet. The second is due to the Markovian nature of the chain and the fact that, as will be seen in choosing the proposal probabilities, the samples are typically autocorrelated, and discarding samples every few steps is one way to reduce the degree of autocorrelation.

In order to implement Algorithm 1, the two proposal distributions $\mathrm{d}\nu_{\tilde{\xi}}\left(\tilde{\xi}' \mid \tilde{\xi}^{j-1}, \Xi^{j-1}_{[0,T]}\right)$ and $\mathrm{d}\nu_{\Xi}\left(\Xi'_{[0,T]} \mid \tilde{\xi}^{j}, \Xi^{j-1}_{[0,T]}\right)$ need to be specified. For $\tilde{\xi}$, choose an independence sampler proposal and generate samples from its prior $\mathrm{d}\mu_{\tilde{\xi}}\left(\tilde{\xi}\right)$, which at every step are independent of the previously accepted sample $\tilde{\xi}^{j-1}$. For $\Xi_{[0,T]}$, use a preconditioned Crank-Nicolson (pCN) proposal, first introduced in Reference [r], that updates according to

$$\Xi'_{[0,T]} = \left(1-\beta^2\right)^{1/2}\Xi^{n-1}_{[0,T]} + \beta\zeta \tag{2-52}$$

where $\zeta = \left[\zeta^1_{[0,T]}, \ldots, \zeta^{N_\kappa}_{[0,T]}\right]$ and ζ^i_t is a Brownian motion with variance $\mathrm{Var}(\mathrm{d}\zeta^i_t) = \mathrm{Var}(\psi_i \mathrm{d}W^i_t) = \psi_i^2\,\mathrm{d}t$ and $\beta \in (0,1]$ is a tuning parameter. More sophisticated pCN proposals that further speed up MCMC could be also be considered by replacing the scalar β by an operator (Reference [t]).

The acceptance probabilities corresponding to the proposal probabilities must also be derived. For $\alpha_{\tilde{\xi}}$, the acceptance probabilities are calculated as

$$\alpha_{\xi} = \min\left\{1, \frac{\mathrm{d}\mu_{\tilde{\xi}}^{\mathbf{Y}}\left(\tilde{\boldsymbol{\xi}}' \mid \boldsymbol{\Xi}_{[0,T]}^{j-1}\right)\mathrm{d}\mu_{\tilde{\xi}}\left(\tilde{\boldsymbol{\xi}}^{j-1}\right)}{\mathrm{d}\mu_{\tilde{\xi}}^{\mathbf{Y}}\left(\tilde{\boldsymbol{\xi}}^{j-1} \mid \boldsymbol{\Xi}_{[0,T]}^{j-1}\right)\mathrm{d}\mu_{\tilde{\xi}}\left(\tilde{\boldsymbol{\xi}}'\right)}\right\}, \tag{2-53}$$

$$= \min\left\{1, \frac{\mathcal{L}\left(\mathbf{Y} \mid \tilde{\boldsymbol{\xi}}', \boldsymbol{\Xi}_{[0,T]}^{j-1}\right)}{\mathcal{L}\left(\mathbf{Y} \mid \tilde{\boldsymbol{\xi}}^{j-1}, \boldsymbol{\Xi}_{[0,T]}^{j-1}\right)}\right\}, \tag{2-54}$$

$$= \min\left\{1, \exp\left(\Phi\left(\tilde{\boldsymbol{\xi}}^{j-1}, \boldsymbol{\Xi}_{[0,T]}^{j-1}; \mathbf{Y}\right) - \Phi\left(\tilde{\boldsymbol{\xi}}', \boldsymbol{\Xi}_{[0,T]}^{j-1}; \mathbf{Y}\right)\right)\right\}. \tag{2-55}$$

For α_{Ξ}, the reversibility property with respect to the prior can be verified by

$$\mathrm{d}\nu_{\Xi}\left(\boldsymbol{\Xi}'_{[0,T]} \mid \boldsymbol{\Xi}_{[0,T]}^{j-1}\right)\mathrm{d}\mu_{\Xi}\left(\boldsymbol{\Xi}_{[0,T]}^{j-1}\right) = \mathrm{d}\nu_{\Xi}\left(\boldsymbol{\Xi}_{[0,T]}^{j-1} \mid \boldsymbol{\Xi}'_{[0,T]}\right)\mathrm{d}\mu_{\Xi}\left(\boldsymbol{\Xi}'_{[0,T]}\right) \tag{2-56}$$

which gives

$$\alpha_{\Xi} = \min\left\{1, \frac{\mathcal{L}\left(\mathbf{Y} \mid \tilde{\boldsymbol{\xi}}^{j}, \boldsymbol{\Xi}'_{[0,T]}\right)}{\mathcal{L}\left(\mathbf{Y} \mid \tilde{\boldsymbol{\xi}}^{j}, \boldsymbol{\Xi}_{[0,T]}^{j-1}\right)}\right\}, \tag{2-57}$$

$$= \min\left\{1, \exp\left(\Phi\left(\tilde{\boldsymbol{\xi}}^{j}, \boldsymbol{\Xi}_{[0,T]}^{j-1}; \mathbf{Y}\right) - \Phi\left(\tilde{\boldsymbol{\xi}}^{j}, \boldsymbol{\Xi}'_{[0,T]}; \mathbf{Y}\right)\right)\right\}. \tag{2-58}$$

We note that Equation (2-56) is essentially the property that makes the pCN proposal dimension-independent since the acceptance probability depends only on the data misfit function.

3.0 AN APPLICATION

In this section, we apply the UQ methods developed in Section 2 to a bacteria growth model developed by the U.S. Army.

Bacteria Growth in Food. Typical bacterial growth in a closed environment goes through four stages (Reference [u]): lag phase, exponential growth phase, stationary phase, and a mortality (i.e. death) phase. The ideal representation of growth kinetics is the bacterial growth curve which is a plot of the number of living cells as a function of time.

In practice, the growth curve is a record of the number of countable cells obtained at a set of sampling times. In a bacterial growth experiment, samples from an inoculated food product under a set of environmental conditions are collected over time. The bacteria in the sample are recovered and enumerated. A standard enumeration method is plate counting in which a petri dish with a growth medium is cultured with the recovered, possibly serially diluted, bacteria and incubated. Viable bacteria cells in the growth medium will form colonies which can be counted as colony forming units (CFU). Reported values are often presented as concentrations or base 10 logarithms of concentrations of the enumeration counts in CFU relative to the volume of the sample (i.e. milliliters).

Reference [u] lists more than 20 mathematical models used for modeling one or more of the stages of bacterial growth in food. In this section, we apply UQ methods to a specific population dynamics model called the Quasi-Chemical Model (QCM) which was developed at the U.S. Army Natick Soldier Research Development and Engineering Center (NSRDEC) (References [a] and [b]).

3.1 4-PARAMETER QUASI-CHEMICAL MODEL (QCM-4)

Several versions of the QCM model were developed to model characteristics of bacterial growth kinetics observed during experiments. The initial version, which is called the 4-parameter QCM (QCM-4) model in this report, was used to model the growth and death of Staphylococcus Aureus (S. aureus) in intermediate moisture bread. Later, the QCM-4 model was used to model the growth kinetics of S. aureus; Listeria monocytogenes (Listeria); and Escherichia coli (E. coli) on hosts of bread, turkey meat, ham, commercial cheese products, or whey protein at different values of pH, temperature (T), water activity (Aw), and pressure (P) (Reference [b]).

The QCM-4 model consists of four ordinary differential equations (ODEs) that describe the dynamics of a four element state. Let M be the concentration of quiescent microbial cells (i.e. lag phase cells); M^* the concentration of microbial cells activated for cell division and multiplication (growth phase cells); A the concentration of an antagonistic metabolite or extracellular death factor (i.e. quorum sensing molecule); and D the concentration of dead microbial cells. The four ODEs that describe the state $\mathbf{x}(t;\boldsymbol{\kappa}) = [M(t;\boldsymbol{\kappa}), M^*(t;\boldsymbol{\kappa}), A(t;\boldsymbol{\kappa}), D(t;\boldsymbol{\kappa})]^T$ are derived from methods used in nonlinear chemical dynamics for analyzing complex chemical reactions (Reference [v]) and are given by

$$\frac{\mathrm{d}M}{\mathrm{d}t} = -k_1 M \ , \tag{3-1}$$

$$\frac{\mathrm{d}M^*}{\mathrm{d}t} = k_1 M + (k_2 - k_4) M^* - h k_3 M^* A \ , \tag{3-2}$$

$$\frac{\mathrm{d}A}{\mathrm{d}t} = k_2 M^* - h k_3 M^* A \ , \tag{3-3}$$

$$\frac{\mathrm{d}D}{\mathrm{d}t} = k_4 M^* + h k_3 M^* A \ , \tag{3-4}$$

where $\boldsymbol{\kappa} = [k_1, k_2, k_3, k_4]^{\mathrm{T}}$ are non-negative rate parameters and h is a scaling constant equal to 10^{-9}. The QCM-4 model can be expressed in the vector form of Equation (2-1) where $[x_1, x_2, x_3, x_4] = [M, M^*, A, D]$, $\mathbf{f} = [\ f_1, f_2, f_3, f_4]^{\mathrm{T}}$, and

$$f_1 = -k_1 M \ , \tag{3-5}$$

$$f_2 = k_1 M + (k_2 - k_4)\ M^* - h k_3 M^* A \ , \tag{3-6}$$

$$f_3 = k_2 M^* - h k_3 M^* A \ , \tag{3-7}$$

$$f_4 = k_4 M^* + h k_3 M^* A \ . \tag{3-8}$$

The ODEs model bacterial kinetics that would occur in an experiment (Reference [a]) in which a food product is inoculated with an initial concentration of bacterial cells I, and $M(0;\boldsymbol{\kappa}) = I$. After a period of time, the bacterial cells become activated ($M \rightarrow M^*$) and undergo a multiplication step through reproduction by cell division that can produce an antagonistic metabolite ($M^* \rightarrow 2M^*+A$). Over time, death will occur either naturally ($M^* \rightarrow D$), or through sensitization by the antagonistic metabolite ($M^*+A \rightarrow M^{**} \rightarrow D$) where M^{**} is an intermediate phase. The initial conditions for the model are $\mathbf{x}(0;\boldsymbol{\kappa}) = [I, 0, 0, 0]^{\mathrm{T}}$.

Enumeration methods such as plate counting measure the number of living cells in a sample at time t, $U(t;\boldsymbol{\kappa})$, which is given by the model as

$$U(t;\boldsymbol{\kappa}) = M(t;\boldsymbol{\kappa}) + M^*(t;\boldsymbol{\kappa}) . \tag{3-9}$$

For a given value of $\boldsymbol{\kappa}$, the ODEs in Equations (3-1) through (3-4) can be solved numerically to give the value of the state $\mathbf{x}$ over time. As an example, Figure 3-1 plots the values of the state variables as well as the value of U versus time for the case where $\boldsymbol{\kappa} = [0.1, 5.0, 10.0, 2.0]^{\mathrm{T}}$.

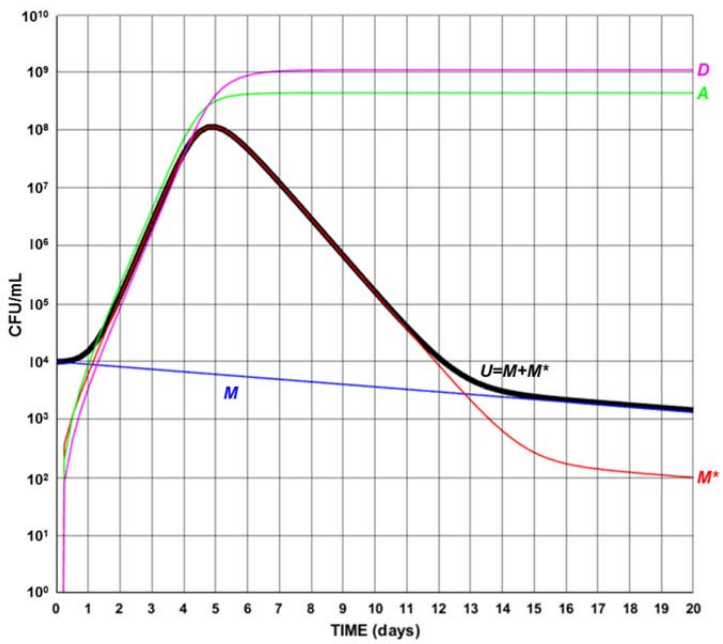


Figure 3-1. QCM-4 Model State Variables vs. Time for $\boldsymbol{\kappa}$ = [0.1, 5.0, 10.0, 2.0]^T

3.2 ALTERNATE QCM-4 MODEL FORMULATIONS

The QCM-4 model in Section 3.1 is a mathematical model defined by four ODEs in four state variables with four model parameters. We examine four alternate formulations of the model in applying UQ methods to the QCM-4 model. The alternate formulations are, in certain cases, numerically more stable and less sensitive to uncertainties in model parameters. The formulation of the QCM-4 model in Section 3.1 will be referred to as Formulation A. Formulations B, C, D, and E of the QCM-4 model are described in this section.

3.2.1 Model Formulation B

Equation (3-1) can be solved analytically for M by integration which gives

$$M(t;\boldsymbol{\kappa}) = Ie^{-k_1 t}, \tag{3-10}$$

and Equation (3-9) can be written as

$$U(t;\boldsymbol{\kappa}) = Ie^{-k_1 t} + M^*(t;\boldsymbol{\kappa}). \tag{3-11}$$

The concentration of dead bacterial cells, D, is uncoupled from Equations (3-2) and (3-3). Since D is not measured, Equation (3-4) is not necessary to evaluate U or solve for $\boldsymbol{\kappa}$. Formulation B of the QCM-4 can then be defined by Equation (2-1) for the state $\mathbf{x} = [M^*, A]^T$ with initial

condition $\mathbf{x}(0) = [0,\ 0]^T$, rate parameters $\boldsymbol{\kappa} = [k_1,\ k_2,\ k_3,\ k_4]^T$, and ODE functions $\mathbf{f} = [\ f_1,\ f_2]^T$ where

$$f_1 = Ie^{-k_1 t} + \left(k_2 - k_4\right)M^* - hk_3 M^* A\,, \tag{3-12}$$

$$f_2 = k_2 M^* - hk_3 M^* A\,. \tag{3-13}$$

3.2.2 Model Formulation C

In Equation (3-2), the term $(k_2 - k_4)M^*$ results in correlation between the rate parameters k_2 and k_4. The difference between k_2 and k_4 is defined as the net natural growth rate in Reference [a]. Define α to be $k_2 - k_4$. Formulation C of the QCM-4 model is based on Formulation B and uses the same state vector and value for f_2. Formulation C differs from Formulation B in that the rate parameters are now $\boldsymbol{\kappa} = [k_1, k_2, k_3, \alpha]^T$ and f_1 is given by

$$f_1 = Ie^{-k_1 t} + \alpha M^* - hk_3 M^* A \tag{3-14}$$

The change from Formulation B to C has no effect on numerically solving the system of two ODEs, but can affect the optimization problem. For Formulations A and B, $\boldsymbol{\kappa}$ is a vector of non-negative values. In Formulation C, the value of α can take on any real value which changes the solution space that numerical solvers optimize over.

3.2.3 Model Formulation D

As will be discussed in Section 3.5, the forward uncertainty propagation method randomly draws values for $\boldsymbol{\kappa}$ from a multivariate normal distribution given by Equation (2-11). When the covariance of the rate parameters is large relative to the mean value, negative rate parameters can be drawn. To avoid this problem, Formulation D uses a logarithm space for the rate parameters by letting $z_i = \ln(k_i)$ and setting $\boldsymbol{\kappa} = [z_1,\ z_2,\ z_3,\ z_4]^T$. The values for z_i are defined on the reals, which will map into the positive reals when converted back to k_i. The state for Formulation D is $\mathbf{x} = [M^*, A]^T$, and the differential equations are defined by $\mathbf{f} = [\ f_1, f_2]$ where

$$f_1 = Ie^{z_1 - e^{z_1} t} + \left(e^{z_2} - e^{z_4}\right)M^* - he^{z_3} M^* A\,, \tag{3-15}$$

$$f_2 = e^{z_2} M^* - he^{z_3} M^* A\,. \tag{3-16}$$

3.2.4 Model Formulation E

Formulation E is a modification to Formulation D that we will use in forward uncertainty propagation. In Formulation C, when $\alpha > 0$, the output of the QCM-4 model shows both bacteria growth and death. If $\alpha \le 0$, then bacteria death-only occurs. During forward uncertainty propagation, a mean solution for $\boldsymbol{\kappa}$ exists and whether there is bacteria growth or death is known; however, random draws on $\boldsymbol{\kappa}$ using Formulation D can generate bacteria growth or death. For

numerical reasons discussed in Section 3.5, bacteria death-only cases are handled using a different model. For bacteria growth and death cases, α is positive and Formulation E defines $\boldsymbol{\kappa} = [z_1, z_2, z_3, z_4]^T = [\ln(k_1), \ln(k_2), \ln(k_3), \ln(\alpha),]^T$. The state for Formulation E is the same as Formulation D, and the two functions that define the differential equations are given by

$$f_1 = Ie^{z_1 - e^{z_1} t} + e^{z_4} M^* - he^{z_3} M^* A, \tag{3-17}$$

$$f_2 = e^{z_2} M^* - he^{z_3} M^* A. \tag{3-18}$$

3.3 INVERSE PROBLEM: A FREQUENTIST SOLUTION

3.3.1 Inverse Problem for QCM-4 Model

The frequentist approach for solving the inverse problem is to solve Equation (2-5) using QCM-4 Formulation A, B, or C. To solve Equation (2-5), the value of $g(t;\boldsymbol{\kappa})$ must be identified. Enumeration values from bacterial growth experiments are typically reported as the base 10 logarithm of the bacterial concentration. The observation process $g(t;\boldsymbol{\kappa})$ discussed in Section 2.1.1 can then be defined as

$$g(t;\boldsymbol{\kappa}) = \log_{10}(U(t;\boldsymbol{\kappa})). \tag{3-19}$$

The value of $g(t;\boldsymbol{\kappa})$, a component in the objective function of Equation (2-5), is a nonlinear function of $\boldsymbol{\kappa}$. Given a set of measured data, a nonlinear solver is used to find an estimate of $\boldsymbol{\kappa}$ which minimizes the SSR in Equation (2-5).

3.3.2 QCM-4 Model Sensitivity Analysis

Section 2.1.2 discussed how sensitivity analysis can be applied to the solution $\hat{\boldsymbol{\kappa}}$ of an inverse problem. In this section we apply the sensitivity analysis to Formulation C of the QCM-4 model. Application of the sensitivity analysis to other formulations is similar.

Let $\mathbf{x} = [x_1, x_2]^T = [M^*, A]^T$ and $\hat{\boldsymbol{\kappa}} = \left[\hat{k}_1, \hat{k}_2, \hat{k}_3, \hat{\alpha}\right]^T$. The traditional sensitivity functions, $\mathbf{s}$, are defined by Equation (2-8), the number of state variables ($N_{\mathbf{x}}$) is two, and the number of rate parameters ($N_{\boldsymbol{\kappa}}$) is 4. The sensitivity functions $\mathbf{s}$ are given by

$$\boldsymbol{s} = \begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \end{bmatrix}. \tag{3-20}$$

With $\mathbf{f} = [\ f_1, f_2]^T$ defined by Equations (3-14) and (3-13), $\mathbf{s}$ is the solution to Equation (2-9) where

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \alpha - h\hat{k}_3 x_2 & -h\hat{k}_3 x_1 \\ \hat{k}_2 - h\hat{k}_3 x_2 & -h\hat{k}_3 x_1 \end{bmatrix}, \text{ and} \tag{3-21}$$

$$\frac{\partial \mathbf{f}}{\partial \hat{\boldsymbol{\kappa}}} = \begin{bmatrix} Ie^{-\hat{k}_1 t} - \hat{k}_1 tIe^{-\hat{k}_1 t} & 0 & -hx_1 x_2 & x_1 \\ 0 & x_1 & -hx_1 x_2 & 0 \end{bmatrix}. \tag{3-22}$$

Using Equations (3-21) and (3-22) in Equation (2-9), ***s*** is then the solution to the following system of eight ODEs

$$\frac{\mathrm{d}s_{11}}{\mathrm{d}t} = \left(\alpha - h\hat{k}_3 x_2\right)s_{11} - h\hat{k}_3 x_1 s_{21} + Ie^{-\hat{k}_1 t} - \hat{k}_1 tIe^{-\hat{k}_1 t}\ , \tag{3-23}$$

$$\frac{\mathrm{d}s_{12}}{\mathrm{d}t} = \left(\alpha - h\hat{k}_3 x_2\right)s_{12} - h\hat{k}_3 x_1 s_{22}\ , \tag{3-24}$$

$$\frac{\mathrm{d}s_{13}}{\mathrm{d}t} = \left(\alpha - h\hat{k}_3 x_2\right)s_{13} - h\hat{k}_3 x_1 s_{23} - hx_1 x_2\ , \tag{3-25}$$

$$\frac{\mathrm{d}s_{14}}{\mathrm{d}t} = \left(\alpha - h\hat{k}_3 x_2\right)s_{14} - h\hat{k}_3 x_1 s_{24} + x_1\ , \tag{3-26}$$

$$\frac{\mathrm{d}s_{21}}{\mathrm{d}t} = \left(\hat{k}_2 - h\hat{k}_3 x_2\right)s_{11} - h\hat{k}_3 x_1 s_{21}\ , \tag{3-27}$$

$$\frac{\mathrm{d}s_{22}}{\mathrm{d}t} = \left(\hat{k}_2 - h\hat{k}_3 x_2\right)s_{12} - h\hat{k}_3 x_1 s_{22} + x_1\ , \tag{3-28}$$

$$\frac{\mathrm{d}s_{23}}{\mathrm{d}t} = \left(\hat{k}_2 - h\hat{k}_3 x_2\right)s_{13} - h\hat{k}_3 x_1 s_{23} - hx_1 x_2\ , \text{ and} \tag{3-29}$$

$$\frac{\mathrm{d}s_{24}}{\mathrm{d}t} = \left(\hat{k}_2 - h\hat{k}_3 x_2\right)s_{14} - h\hat{k}_3 x_1 s_{24}\ . \tag{3-30}$$

To find the sensitivity of $g(t;\hat{\boldsymbol{\kappa}})$, the sensitivity of $U(t;\hat{\boldsymbol{\kappa}})$ as defined in Equation (3-11) must first be calculated. The sensitivity of the total microbial population relative to rate parameter κ_j is

$$\frac{\partial U(t;\hat{\boldsymbol{\kappa}})}{\partial \hat{\kappa}_j} = s_{1j} - Ite^{-\hat{k}_1 t}\delta_{1j}\ , \tag{3-31}$$

where δ_{1j} is the Kronecker delta function. The sensitivity of $g(t;\hat{\boldsymbol{\kappa}})$ can be found by differentiating Equation (3-19) and applying Equation (3-31) to get

$$\frac{\partial g(t;\hat{\boldsymbol{\kappa}})}{\partial \hat{\kappa}_j} = \left(U(t;\hat{\boldsymbol{\kappa}})\ln(10)\right)^{-1}\frac{\partial U(t;\hat{\boldsymbol{\kappa}})}{\partial \hat{\kappa}_j}\ . \tag{3-32}$$

The sensitivity matrix $\mathbf{F}(\hat{\boldsymbol{\kappa}})$ and covariance $\hat{\boldsymbol{\Sigma}}$ can then be calculated from Equation (3-32) using Equations (2-10) and (2-14).

3.3.3 Alternate Models for Use in Sensitivity Analysis

Section 2.1.3 discussed problems that can occur in a model of a dynamic process in terms of the sensitivity of the model parameters. The QCM-4 models a wide range of bacteria growth kinetics that cover lag, growth, and death phases. Issues with solution sensitivity in the QCM-4 model can occur when data only show growth or death. For instance, a bacteria death-only case can show death that is linear relative to the logarithm of the bacterial concentration. In which case only one parameter is needed.

Section 2.1.3 discussed using model reduction via parameter ranking. Another approach for sensitivity analysis is to replace the QCM-4 model with a simpler model. In this section we describe four alternate models that can be used to model simple growth kinetics. Results of applying these simple growth kinetic models and comparing them with results from the QCM-4 model are discussed in Section 4.

Let $x(t)$ be the total concentration of cells alive at time t. Assuming that the cells grow or die at a rate proportional to the total concentration at time t gives

$$\frac{\mathrm{d}x}{\mathrm{d}t} = ax \,, \tag{3-33}$$

$$x(0) = I \,, \tag{3-34}$$

where a is the growth rate, and I is the initial bacterial concentration. Equations (3-33) and (3-34) can be solved to give

$$x(t;\mathbf{q}) = Ie^{at} \,, \tag{3-35}$$

where $\mathbf{q} = [a]$. Equation (3-35) is a one parameter model of exponential growth ($a > 0$) or death ($a < 0$), and is referred to as the 1-stage exponential (1EX) model.

To add a lag phase to the 1-stage exponential model, let $\mathbf{x} = [x_1, x_2]$, and define

$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = -k_1 x \,, \tag{3-36}$$

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = k_1 x_1 + a x_2 \tag{3-37}$$

$$x_1(0) = I \,,\; x_2(0) = 0 \,, \tag{3-38}$$

where k_1 is the lag rate. The total bacterial concentration is $x(t) = x_1(t)+x_2(t)$, and can be solved explicitly from Equations (3-36) through (3-38) as

$$x(t;\mathbf{q}) = Ie^{-k_1 t} + \frac{k_1 I}{a+k_1}\left(e^{at} - e^{-k_1 t}\right), \tag{3-39}$$

where $\mathbf{q} = [a, k_1]$. Equation (3-39) is a two parameter model of exponential growth or death with a lag phase, and is referred to as the 1-stage exponential with lag (1EL) model.

The 1EX and 1EL models can only be used to model constant growth or constant death kinetics. To model kinetics that exhibit both growth and death, two stage models must be used. Let $x(t)$ be the total concentration of cells alive at time t. Assuming that over the interval $0 \le t \le \tau$, the cells grow at a rate proportional to the total concentration at time t gives

$$\frac{\mathrm{d}x}{\mathrm{d}t} = ax\,, \quad 0 \le t \le \tau \tag{3-40}$$

$$x(0) = I\,, \tag{3-41}$$

where $a > 0$ is the growth rate, and I is the initial bacterial concentration. Suppose that at time $t = \tau$ the bacterial cells enter the death phase. Assuming that during the death phase the cells die at a rate proportional to the total concentration at time t gives

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -bx\,, \quad t > \tau \tag{3-42}$$

$$x(\tau) = Ie^{a\tau}\,, \tag{3-43}$$

where $b > 0$ is the death rate, and the initial condition at $t = \tau$ is chosen so that the cell concentration is continuous. Solving Equations (3-40) through (3-43) gives the piecewise defined model

$$x(t;\mathbf{q}) = \begin{cases} Ie^{at}\,, & 0 \le t \le \tau \\ Ie^{-b(t-\tau)+a\tau}\,, & t > \tau \end{cases} \tag{3-44}$$

where $\mathbf{q} = [a, b, \tau]^{\mathrm{T}}$. Equation (3-44) is a three parameter model of exponential growth and death, and is referred to as the 2-stage exponential (2EX) model.

To add a lag phase to the 2-stage exponential model, let $\mathbf{x} = [x_1, x_2]$, and define

$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = -k_1 x \,, \tag{3-45}$$

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = \begin{cases} k_1 x_1 + a x_2 \,, & 0 \le t < \tau \\ k_1 x_1 - b x_2 \,, & t \ge \tau \end{cases} \tag{3-46}$$

$$x_1(0) = I \,,\; x_2(0) = 0 \,, \tag{3-47}$$

where k_1 is the lag rate. Imposing the requirement that the solution be continuous at $t = \tau$, the total bacteria concentration is $x(t) = x_1(t)+x_2(t)$, and can be solved explicitly from Equations (3-45) through (3-47) as the piecewise model

$$x(t;\mathbf{q}) = \begin{cases} I e^{-k_1 t} + \dfrac{k_1 I}{a + k_1}\left(e^{at} - e^{-k_1 t}\right), & 0 \le t < \tau \\ I e^{-k_1 t} + \left(x_\tau - \dfrac{k_1 I}{a + k_1} e^{-k_1 \tau} \right) e^{b(\tau - t)} + \dfrac{k_1 I}{a + k_1} e^{-k_1 t} \,, & t \ge \tau \end{cases} \tag{3-48}$$

where $\mathbf{q} = [k_1, a, b, \tau]^{\mathrm{T}}$, and x_τ is defined as

$$x_\tau = \frac{k_1 I}{a + k_1}\left(e^{a\tau} - e^{-k_1 \tau}\right). \tag{3-49}$$

Equation (3-49) is a four parameter model of exponential growth and death, and is referred to as the 2-stage exponential with lag (2EL) model.

Figure 3-2 shows plots of the four alternate models discussed in this section for the example parameter values of $a = 1.0$, $b = 0.7$, $\tau = 8.0$, $k_1 = 0.5$, and the initial conditions $I = 10^5$ CFU/mL.

As with the QCM-4 model, Equation (2-5) can be solved to estimate the model parameters for the four models described in this section. The sensitivity analysis of Section 2.1.2 can be applied to calculate the standard error in the model for a given solution. One method for comparing the models developed in this section with the QCM-4 model is to compare the standard errors which are a measure of the local sensitivity of a model.

A second method for comparing competing models is based on information theory. The Akaike Information Criteria (AIC) is a model selection criteria which allows for the comparison between models without the necessary condition that the models are nested. The AIC is based on the Kullback-Leibler information and maximum likelihood estimation as a way to balance model bias and variance (Reference [c]). The AIC is given by

$$\mathrm{AIC} = n \ln\left(\frac{J(\boldsymbol{\kappa})}{n} \right) + 2\left(N_\kappa + 1\right), \tag{3-50}$$

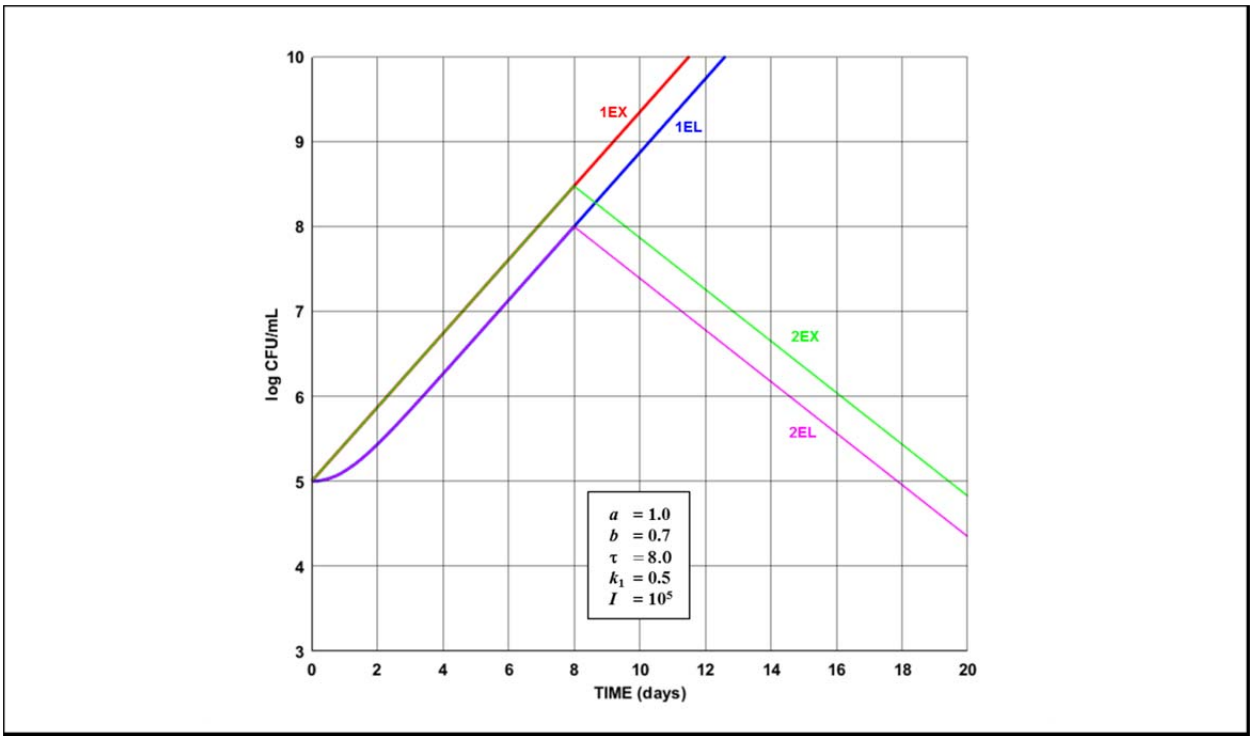


Figure 3-2. Example Plot of Four State vs. Time for Alternate Bacteria Kinetic Models

where n is the number of observations, J is the sum of squared residuals between the observations and the model output, and N_κ is the number of estimated model parameters. If the number of observations is few relative to N_κ^2, then a correction to Equation (3-50) is needed to account for finite sample sizes, and the resulting value, AIC_c, is given by

$$\mathrm{AIC}_c = \mathrm{AIC} + \frac{2(N_\kappa + 1)(N_\kappa + 2)}{n - N_\kappa - 2}. \tag{3-51}$$

For a given set of observations, among the competing models, the best approximating model is the one with the minimum value for AIC_c.

3.4 INVERSE PROBLEM: A BAYESIAN SOLUTION

The Bayesian approach for solving the inverse problem was applied using Formulation A of the QCM-4 model. As with the frequentist approach, $\boldsymbol{\kappa} = [k_1, k_2, k_3, k_4]^T$ and the value of $g(t;\boldsymbol{\kappa})$ is given by Equation (3-19).

3.4.1 Variational Bayesian Inference

As discussed in Section 2.2.1, the variational Bayesian inference method estimates the model parameters based on approximating the parameters' posterior distribution by a probability measure chosen from a parametric family, and minimizing their Kullback-Leibler divergence.

For the QCM-4 model, the parametric family we choose is a family of Gaussian mixtures with L components.

Finding the posterior distribution that minimizes the KL divergence depends on the choice of the prior $\mathrm{p}(\boldsymbol{\theta})$ (Equation (2-23)). The prior must be wide enough so that the mean of the posterior can be detected within the support of the prior with a relatively high probability. We assume that all model parameters are independent, which allow the prior to be factored as

$$\mathrm{p}(\boldsymbol{\theta}) = \left(\prod_{i=1}^{N_\kappa} \mathrm{p}(\xi_i)\right) \times \mathrm{p}(\omega). \tag{3-52}$$

The estimate of the model parameters are the means of the posterior distributions found through the minimization given in Equation (2-26), or equivalently

$$\mathrm{q}^* = \arg\min_{\mathrm{q} \in \mathcal{Q}_L} \mathrm{KL}[\mathrm{q}][\mathrm{p}], \tag{3-53}$$

for $L = 1$. The reasoning behind trying families of Gaussians on the QCM-4 model for $L > 1$ is based on the fact that $\mathcal{Q}_L \subset \mathcal{Q}_{L+1}$ for $L \geq 1$. It follows then that

$$\min_{\mathbf{q} \in \mathcal{Q}_L} \mathrm{KL}[\mathrm{q}][\mathrm{p}] \geq \min_{\mathbf{q} \in \mathcal{Q}_{L+1}} \mathrm{KL}[\mathrm{q}][\mathrm{p}], \tag{3-54}$$

which indicates that the solution found in $\mathcal{Q}_{L+1}$ could be preferable over the one found in $\mathcal{Q}_L$ since it can result in a smaller value for the KL divergence.

3.4.2 Comparing Model Solutions

Section 3.3.3 presented a method for comparing the results from different models based on the AIC. For Bayesian methods, a similar method exists called the Bayesian Information Criterion (BIC). The BIC is defined as (Reference [w])

$$\mathrm{BIC} = \ln\left(\mathrm{p}\left(\mathbf{y} \mid \boldsymbol{\kappa}^*\right)\right) - \tfrac{1}{2} N_\kappa \ln(n), \tag{3-55}$$

where $\mathrm{p}(\mathbf{y}|\boldsymbol{\kappa}^*)$ is the log-likelihood evaluated at the *maximum a posteriori* (MAP) estimate $\boldsymbol{\kappa}^*$, N_κ is the number of estimated parameters, and n is the number of observations. The value for $\boldsymbol{\kappa}^*$ will be the mean of the variational posterior $\mathrm{q}(\boldsymbol{\kappa})$ which is Gaussian, and is, therefore, the MAP estimate under the assumption that q is the true posterior. As with the AIC, the BIC is only known to be valid when $n >> N_\kappa$. For small sample sizes, the value of $\mathcal{F}[\mathrm{q}]$ from Equation (2-28) can be used as a second criterion for model comparison. This is motivated by the fact that the BIC was essentially proposed as a criterion that serves as an estimate of the evidence $\mathrm{p}(\mathbf{y})$. Recall that in the optimization problem, $\mathcal{F}[\mathrm{q}]$ is a lower bound of the evidence so it can give an additional idea about the true value of the evidence. In general, larger values of BIC tend to suggest a preferable model and the sharper the increase in its value, the stronger the evidence that the model with the largest value is closest to the truth.

3.4.3 The Stochastic Differential Equation and Data Assimilation Problem

Equation (2-43) is an Itô stochastic differential equation that can be applied to the QCM-4 model. Using Formulation A, let $\tilde{\boldsymbol{\kappa}}_0 = \left[k_0^1, k_0^2, k_0^3, k_0^4\right]^{\mathrm{T}}$, $\mathbf{x} = \left[x_1, x_2, x_3, x_4\right]^{\mathrm{T}} = \left[M, M^*, A, D\right]^{\mathrm{T}}$, and $X_t^i = x_i$. The values for $\tilde{\mathbf{f}}\left(\mathbf{X}_t, \tilde{\boldsymbol{\kappa}}_0\right)$ and $\tilde{\mathbf{o}}\left(\mathbf{X}_t, \boldsymbol{\psi}\right)$ in Equation (2-43) are then given by

$$\tilde{\mathbf{f}}\left(\mathbf{X}_t, \tilde{\boldsymbol{\kappa}}_0\right) = \begin{bmatrix} -k_0^1 X_t^1 \\ k_0^1 X_t^1 + \left(k_0^2 - k_0^4\right) X_t^2 - h k_0^3 X_t^2 X_t^3 \\ k_0^2 X_t^2 - h k_0^3 X_t^2 X_t^3 \\ k_0^4 X_t^2 + h k_0^3 X_t^2 X_t^3 \end{bmatrix}, \tag{3-56}$$

and

$$\tilde{\boldsymbol{\sigma}}\left(\mathbf{X}_t, \boldsymbol{\psi}\right) = \begin{bmatrix} -\psi_1 X_t^1 & 0 & 0 & 0 \\ \psi_1 X_t^1 & \psi_2 X_t^2 & -h\psi_3 X_t^2 X_t^3 & -\psi_4 X_t^2 \\ 0 & \psi_2 X_t^2 & -h\psi_3 X_t^2 X_t^3 & 0 \\ 0 & 0 & h\psi_3 X_t^2 X_t^3 & \psi_4 X_t^2 \end{bmatrix}. \tag{3-57}$$

Since the total number of viable bacterial cells at any time is given by Equation (3-9), let the observation operator in Equation (2-45) be defined as

$$h\left(\mathbf{X}_t\right) = X_t^1 + X_t^2 . \tag{3-58}$$

3.5 FORWARD UNCERTAINTY PROPAGATION

Most of the development discussed up to this point has focused on estimating the model parameters that best fit the QCM-4 model to a set of measured data. Both the Frequentist and Bayesian approaches give methods for specifying confidence intervals about each model parameter, but these methods assume that the parameters are independent. In this section, a method for forward uncertainty propagation is developed that accounts for the variability and covariance structure of the estimated model parameters due to variability in the observations and the model sensitivity.

Let $y_i = y(t_i)$ for $i = 1,...,n$ be a set of measured data and $\hat{\boldsymbol{\kappa}}$ be an estimate of the N_κ model parameters for a bacteria kinetics model. Each y_i is the base 10 logarithm of a bacterial count, and the bacterial kinetics model can be either one of the formulations of the QCM, or one of the alternate models described in Section 3.3.3. The method for forward uncertainty propagation consists of the following three steps:

1. Estimate the covariance in $\hat{\boldsymbol{\kappa}}$,

2. Generate a set of random model parameters, $\boldsymbol{\kappa}_j$, from a multivariate normal distribution, and

3. Generate and plot the set of random kinetics curves corresponding to the random model parameters $\boldsymbol{\kappa}_j$.

Section 2.1.2 discusses how the covariance for the estimate $\hat{\boldsymbol{\kappa}}$, $\hat{\boldsymbol{\Sigma}}$, can be calculated. The method for generating random model parameters and bacteria growth curves are discussed in the following sections.

3.5.1 Generating Random Model Parameters

To generate a set of random model parameters, we assume that the model parameters $\boldsymbol{\kappa}$ are random variates from a multivariate normal distribution with mean $\hat{\boldsymbol{\kappa}}$ and covariance $\hat{\boldsymbol{\Sigma}}$. To generate the random variates, let $\mathbf{A}$ be a decomposition of $\hat{\boldsymbol{\Sigma}}$ such at $\mathbf{A}^T\mathbf{A} = \hat{\boldsymbol{\Sigma}}$. If $\mathbf{z}$ is a vector of N_κ independent standard normal variates, then $\boldsymbol{\kappa}$ is given by (Reference [x])

$$\boldsymbol{\kappa} = \hat{\boldsymbol{\kappa}} + \mathbf{A}\,\mathbf{z}\,. \tag{3-59}$$

The matrix $\mathbf{A}$ is found using a Cholesky decomposition of $\hat{\boldsymbol{\Sigma}}$, which requires $\hat{\boldsymbol{\Sigma}}$ to be positive semi-definite. The only cases in which the covariance matrix was found to not be positive semi-definite were those that exhibited death-only (i.e. inactivation) kinetics. These cases are discussed in the next section.

A second issue that occurs in the generation of random model parameters occurs when the standard error of the model parameters (i.e. the square root of the diagonal of the covariance matrix) are large with respect to the mean value of the model parameters. For Formulations A and B of the QCM, the model parameters are the four rate parameters [k_1, k_2, k_3, k_4]. For Formulation C, the fourth parameter is $\alpha = k_2 - k_4$. The rate constants are inverse times, and must have non-negative values. As will be discussed in Section 4, almost every dataset resulted in large numbers of negative rate parameters to be generated. To prevent this problem from biasing results, Formulations D and E were developed for use in forward uncertainty propagation.

3.5.2 Generating Random Kinetics Curves

Our current implementation of the forward uncertainty propagation algorithm (see Appendix B) generates 2500 random estimates of the model parameters. With the exception of death-only cases, kinetics curves for each random estimate of the model parameters are drawn from time zero up to a maximum time using Formulation E of the QCM model.

Several of the death-only cases resulted in covariance matrices that were not positive semi-definite, and, therefore, could not be decomposed. To be consistent, any dataset that results in a mean model parameter estimate in which k_4 is larger than k_2 (i.e. death-only) undergoes a parameter reduction to the 1-stage exponential model with lag given by Equation (3-39). The two model parameters a and k_1 are found by minimizing the sum of squared residuals (SSR) between the measured and modeled data.

4.0 UNCERTAINTY QUANTIFICATION APPLIED TO THE QCM-4 MODEL

In this section we present the results of applying the UQ methods discussed in Section 3 to the QCM-4 model using data collected from bacteria growth experiments on food.

Finding the best fit of the QCM-4 model to a set of measured data consists of two steps: (1) solving the system of ODEs given a set of rate parameters and an initial bacteria count and (2) minimizing the SSR between the QCM-4 model fit and a set of measured data.

4.1 SOLVING THE SYSTEM OF ODEs

Given a set of rate parameters $\boldsymbol{\kappa} = [k_1,\ k_2,\ k_3,\ k_4]^T$ and initial conditions for the state $\mathbf{x}(0;\boldsymbol{\kappa}) = [I,\ 0,\ 0,\ 0]^T$, the value of the state at any time t, $\mathbf{x}(t;\boldsymbol{\kappa})$, is found by solving a system of two or four differential equations as discussed in Sections 3.1 and 3.2. The system of differential equations for the QCM-4 model are nonlinear, and, in general, cannot be solved analytically; numerical solvers are used.

Computer code to implement the Frequentist approach was written using MATLAB, and the ODE solver used was either `ode45` for non-stiff systems of ODEs, or `ode15s` for stiff systems of ODEs. A system of ODEs is considered stiff when the step size needed for numerical stability must be very small, and can depend on the formulation of the QCM-4 being used and/or the set of rate parameters that define the system of ODEs. MATLAB's `ode45` solver is based on an explicit Runge-Kutta (4,5) formula called the Dormand-Prince method. The `ode15s` solver is a variable order, multistep solver based on the numerical differentiation formulas (NDFs).

For the Bayesian approach, computer code was written in Python, and used the ODE solver `ode` from the scipy package with the LSODA algorithm. LSODA (Livermore solver for ordinary differential equations with automatic switching) is part of the ODEPACK library of ODE solvers developed at Lawrence Livermore National Laboratory. LSODA automatically switches between the Adams method for non-stiff systems of ODEs, and backward differentiation formulas (BDF) for stiff systems of ODEs.

As part of our analysis of the QCM-4 model, 300 solutions were calculated for different sets of rate parameters to determine the types of bacteria growth and death that are modeled by the QCM-4 model. Figure 4-1 presents 16 examples of the solutions showing no growth or death; death only; growth only; growth and stationary and growth and death. Analysis of the solutions showed that different combinations of rate parameters can produce similar or identical growth curves. As an example, consider the case in which k_1 is zero (Figure 4-1A). Equation (3-1) shows that if k_1 is zero, then the change in lag phase bacteria cells is zero which means that the number of bacteria cells remains constant for all time, regardless of the values for the rate parameters k_2, k_3, and k_4. This implies that some types of growth kinetics may be overparameterized by the QCM-4 model. As will be seen in Section 4.3, different combinations of rate parameters can produce growth curves where the values of the resulting SSRs are the same.

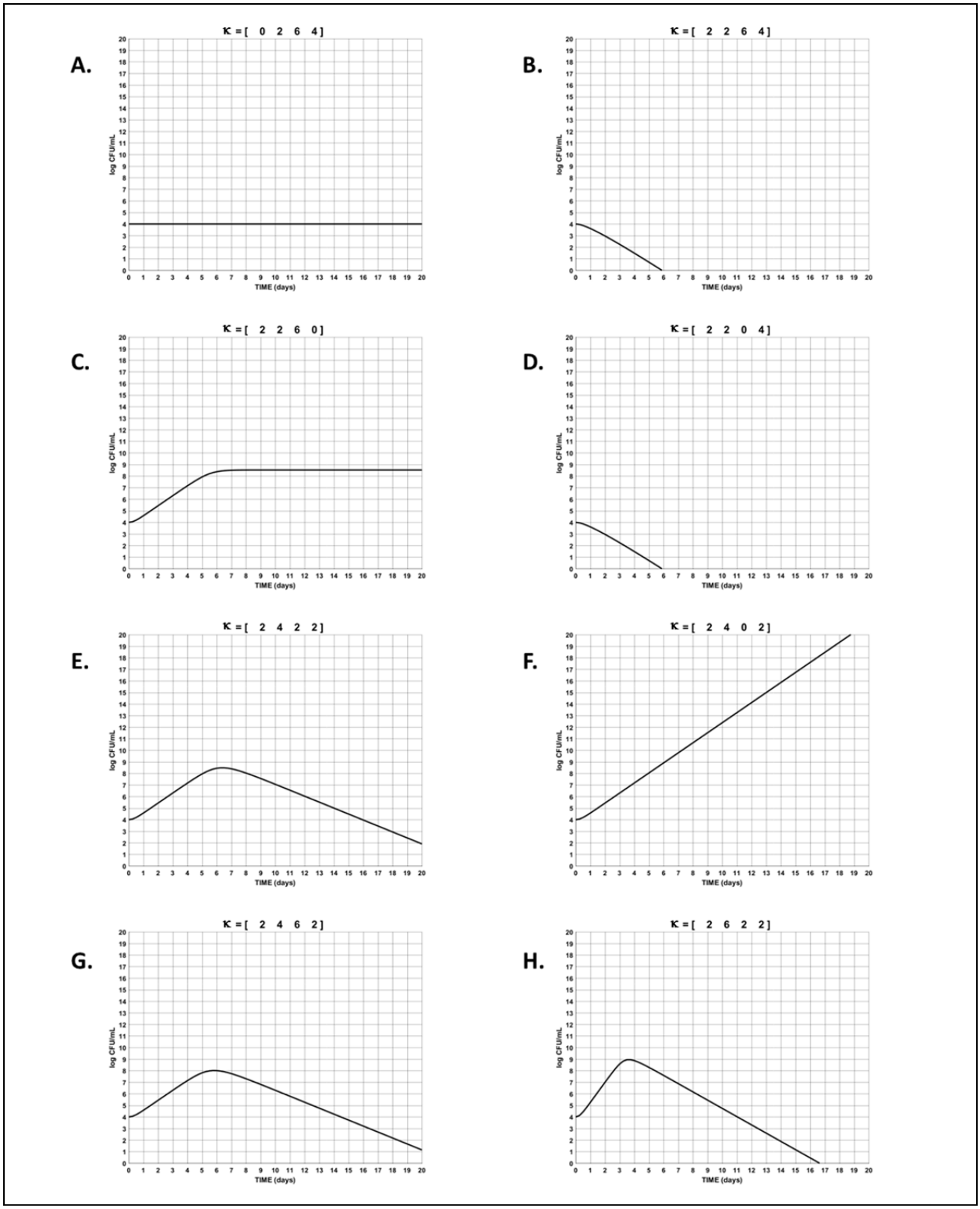


Figure 4-1. Example QCM-4 Growth Curves (Page 1 of 2)

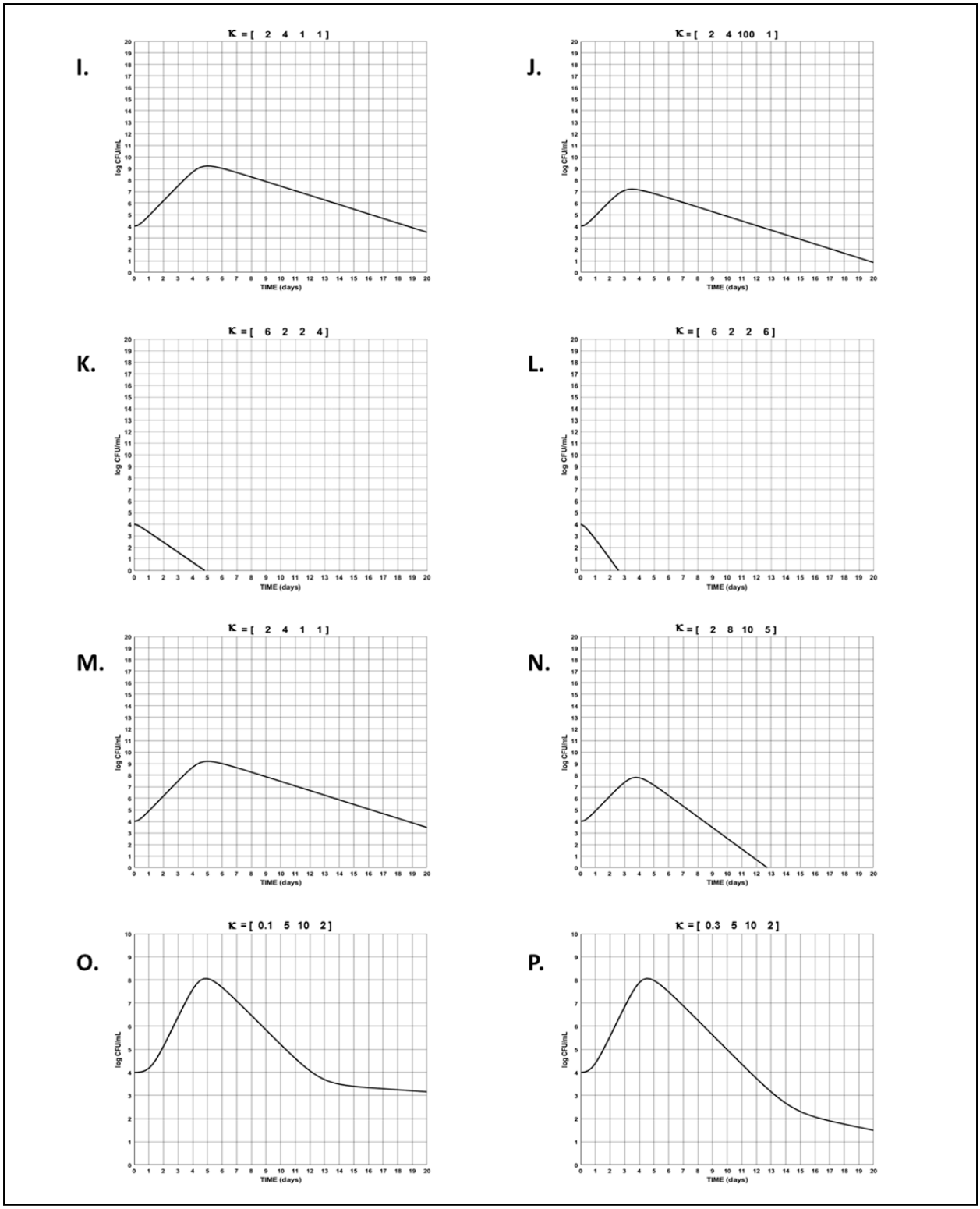


Figure 4-1. Example QCM-4 Growth Curves (Page 2 of 2)

Relationships between the rate parameters, k_1, k_2, k_3, k_4, and the solution of the QCM-4 model are:

- k_1 determines the length of the initial lag phase,
- If k_1 is zero, the count of live bacteria cells remains constant,
- Net growth of bacteria cells is a function of k_2-k_4,
- If k_3 or k_4 is zero, unlimited growth can occur,
- If k_4 is zero and k_3 is non-zero, growth followed by a stationary phase can occur,
- If k_2-k_4 is greater than zero and k_4 is non-zero, then growth and death can occur,
- If k_2-k_4 is less than or equal to zero, death only can occur, and
- The transition from growth to either stationary or death phases is a function of k_3.

These relationships provide a guide for comparing different growth curves, and for choosing initial values for the rate parameters for use in the optimization when measured data are used. Figure 4-2 shows an example of QCM-4 model solution for a case exhibiting growth and death. Different sections of the curve are annotated with the rate parameter(s) that have the most influence.

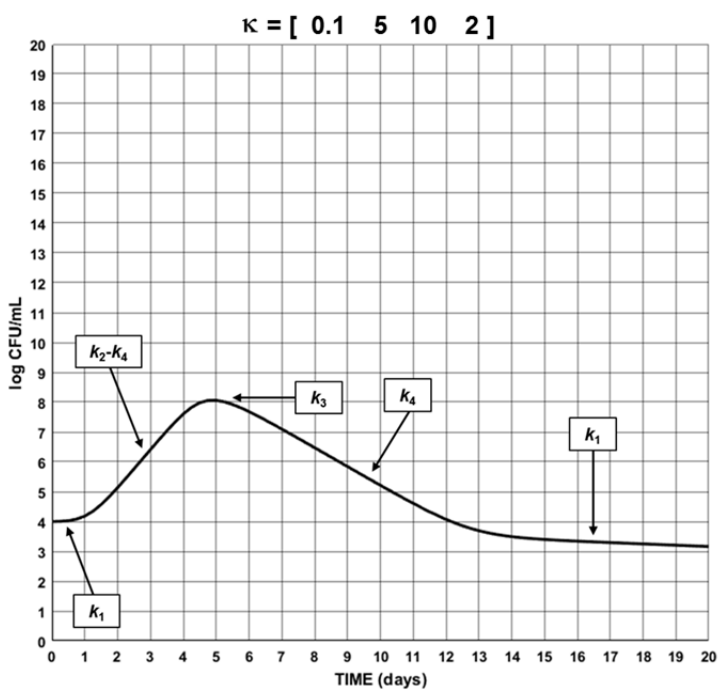


Figure 4-2. Regions of Influence of Rate Parameters on Growth Curve

4.2 MINIMIZING SSR

Given a set of bacteria count data, the inverse problem is to find the rate parameters that generate QCM-4 model output that best fits the data. Both the Frequentist approach and Bayesian variational approach solve the inverse problem by minimizing the SSR. The Frequentist approach finds the rate parameters that minimize the SSR between the measured data and the model output (Equation (2-5)). The computer code is written in MATLAB and uses the MATLAB Central function `minimize`, or the Optimization Toolbox function `fmincon`. The function `minimize` uses the Nelder-Mead minimization algorithm and allows for bounds and constraints to be specified. The function `fmincon` is run using the interior-point algorithm and also allows for specifying bounds and constraints.

The Bayesian variational approach uses a posterior probability distribution and a family of Gaussian probability distributions that are functions of the rate parameters and measured data. The solution rate parameters are those that minimize the Kullback-Leibler divergence between the estimated true posterior probability distributions and the possible families of Gaussian distributions. The computer code is written in Python and uses the `minimize` solver from the scipy package. The minimize solver is implemented using the L-BFGS-B (Limited-memory Broyden-Fletcher-Goldfarb-Shanno with box constraints) algorithm developed at Northwestern University.

The minimization algorithms used for solving the inverse problem require an initial set of rate parameters. For the Frequentist approach, the initial rate parameters were determined by examining the measured data, comparing it with the set of pre-calculated solutions discussed in Section 4.1.1 or previous solutions. If necessary, initial rate parameters are adjusted to reduce stiffness. For the Bayesian approach where the mean and variance of the logarithm of the rate parameters are to be determined, the initial parameters were assumed to be from a normal distribution. In both approaches, global optimization techniques were not used. Section 6 discusses the use of multistart methods and machine learning algorithms for global optimization.

4.3 DATASETS

Reference [b], which describes the quasi-chemical model for predicting bacteria growth in foods, contains plots of datasets collected from ten experiments. Figures 1 – 10 of Reference [b] show measured data consisting of two to six datasets of measured plate counts over time of the growth of bacteria in food-based hosts under different environmental conditions.

Specifically, the datasets present the growth of Staphylococcus Aureus (S. aureus); Listeria Monocytogenes (Listeria); or Escherichia coli (E. Coli) on hosts of bread, turkey meat, ham, commercial cheese products, or whey protein at different values of pH, temperature (T), water activity (Aw), and pressure (P). In Figure 8 of Reference [b], different concentrations of anti-microbial lactate are added to the samples. In total, there are 39 datasets of bacteria counts as a function of time for the 10 food experiments.

The datasets were digitized from Reference [b] and recorded in 39 ASCII text files for use in developing the UQ code for the QCM-4 model. Appendix A contains a description of the 39 datasets; plots of the datasets for each of the ten experiments along with the QCM-4 fits; and forward uncertainty propagation plots for each dataset and a table of the dataset time series values.

A summary description of the datasets shown in Figures 1 – 10 of Reference [b] follows:

Figure 1 contains four datasets of growth data spanning 20 days for S. aureus in bread at a pH of 5.4, T of 35°C, and four values of Aw.

Figure 2 contains six datasets of growth data spanning 65 days for S. aureus in bread at an Aw of 0.90, pH of 5.23, and six temperatures between 15°C and 40°C.

Figure 3 contains five datasets of growth data spanning 12 days for S. aureus in bread at an Aw of 0.86, T of 35°C, and five pH values between 4.97 and 5.38.

Figure 4 contains four datasets of growth data spanning 19 days for S. aureus in turkey meat at a pH of 6.35, T of 35°C, and four values of Aw between 0.84 and 0.89.

Figure 5 contains three datasets of growth data spanning 20 days for S. aureus in ham at a pH of 5.8, T of 35°C, and three values of Aw between 0.88 and 0.92.

Figure 6 contains two datasets of growth data spanning 8 days for S. aureus in commercial cheese products at a T of 35°C and two values for Aw/pH of 0.85/5.9 and 0.87/6.0.

Figure 7 contains three datasets of growth data spanning 16 days in turkey meat for S. aureus, Listeria, and E. coli at a pH of 6.4 and T of 35°C.

Figure 8 contains four datasets of growth data spanning 13 days for S. aureus in turkey meat in which anti-microbial lactate has been introduced. The growth data is for an Aw of 0.896, T of 35°C, pH of 6.4, and four concentrations of lactate between 0% and 3%.

Figure 9 contains five datasets of growth data spanning 50 minutes for E. coli in whey protein under high pressure. The growth data are at a T of 50°C and at five pressures between 207 and 345 MPa.

Figure 10 contains three datasets of growth data spanning 45 minutes for E. coli in whey protein at a P of 310 MPa, and at three temperatures between 30°C and 50°C.

4.4 FREQUENTIST AND BAYESIAN SOLUTION COMPARISON

Solutions to the inverse problem for the 10 datasets from Figure 1 and 2 of Reference [b] were calculated using the Frequentist and Bayesian approaches. Figures 4-3 through 4-12 show plots of the Frequentist and Bayesian solutions; the measured data; and a table with the mean rate parameters, the standard error (Frequentist) or standard deviation (Bayesian) of the estimated rate parameter, and the SSR of the solution relative to the measured data.

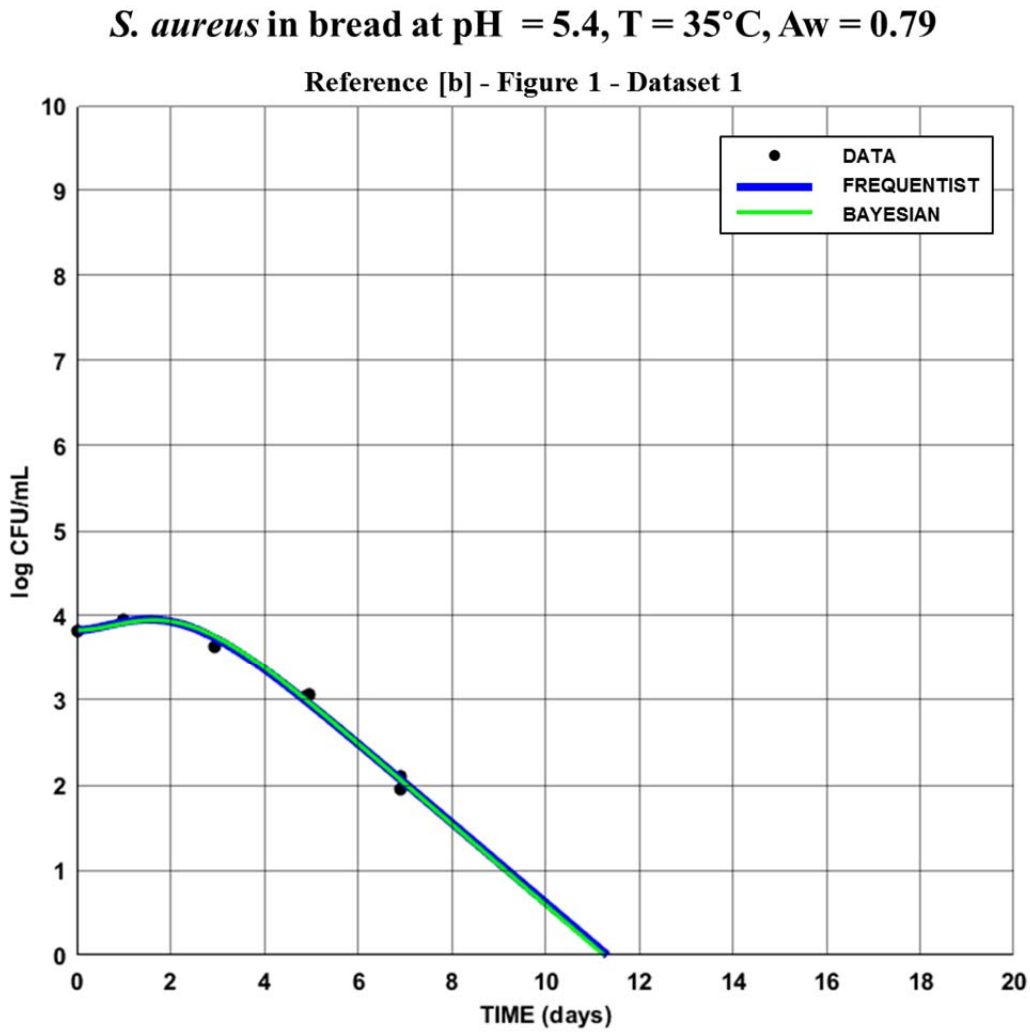
The Frequentist solutions used Formulation B or C, and estimate the mean value of the rate parameters using Equation (2-5). The standard error, a calculated quantity of the solution, is the square root of the diagonal of the covariance matrix.

The Bayesian solutions used a version of Formulation D that extends the list of parameters being solved for to include a variable related to the standard deviation of the rate parameters. The mean value of the rate parameters and standard deviation of the rate parameters are estimated using the variational method discussed in Section 3.4 for a single family of Gaussian distributions.

A comparison of the Frequentist and Bayesian solutions in Figures 4-3 through 4-12 shows that

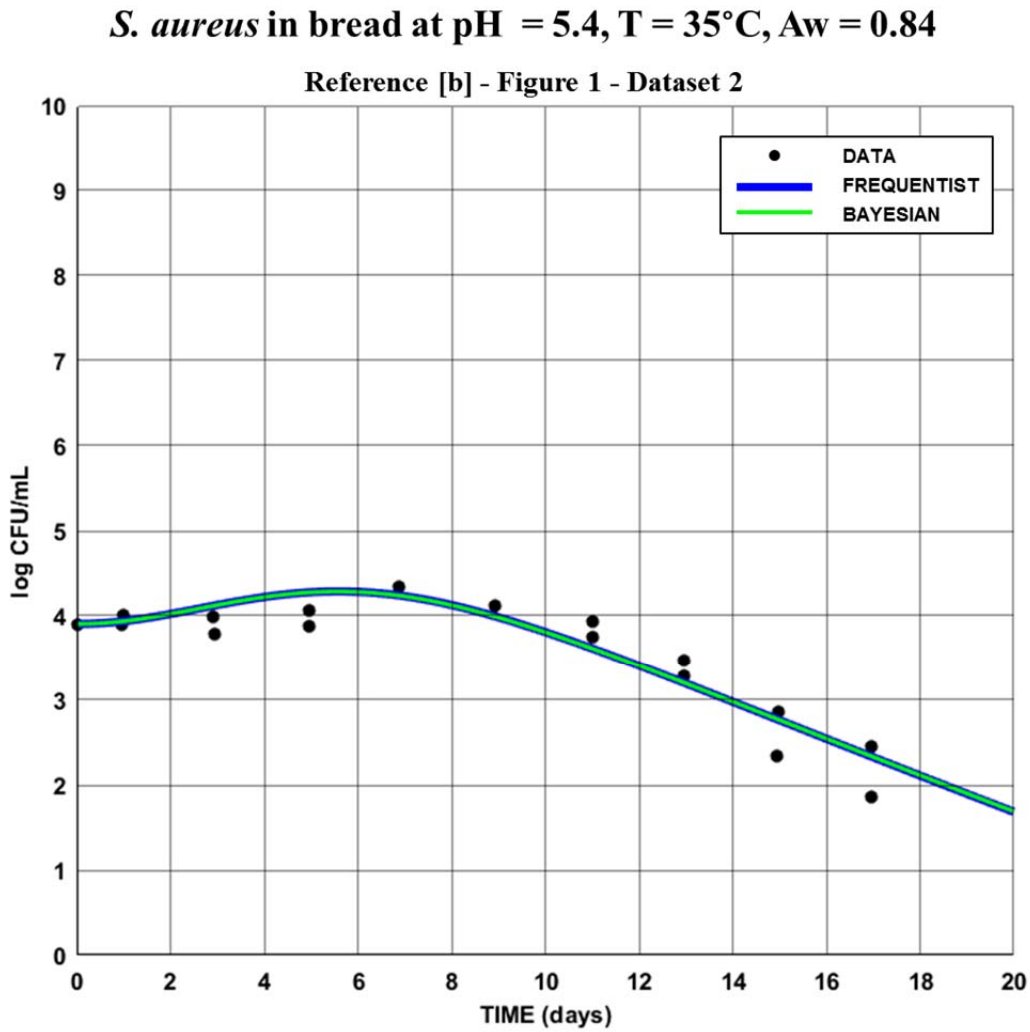
- The Frequentist solution has the smallest SSR value in 6 of the 10 datasets,
- The Bayesian solution has the smallest SSR value in 3 of the 10 datasets,
- In one dataset, the SSR of the solution is the same for both approaches to three significant digits,
- The median percentage difference relative to the mean in the SSRs over all 10 datasets is 8.3%,
- The mean solution rate parameters of the two approaches agree to less than 6% in only 2 datasets,
- The standard errors of the rate parameters for the Frequentist solutions vary from 0.05 to 6.98×10^5, and
- With the exception of the standard deviation for k_3 in Figure 4-7 ($\sigma = 7.18$), the standard deviations of the rate parameters for the Bayesian solutions vary from 1.00 to 1.89.

For all 10 datasets, the growth curves for the Frequentist and Bayesian solutions are similar, and often nearly identical even though the solution rate parameters seldom agree. For example, in Figure 4-4, the generated curves are nearly identical and the resulting SSE values are the same. The Frequentist solution rate parameters ($\boldsymbol{\kappa}_F = [0.37, 97.66, 77.06, 97.25]^T$), though, differ from the Bayesian solution rate parameters ($\boldsymbol{\kappa}_B = [0.37, 41.15, 184.31, 40.74]^T$) by more than 80% for k_2, k_3, and k_4. This shows that growth curves generated from the QCM-4 model may not be unique over the rate parameters, and the model may be overparameterized for some types of growth kinetics.



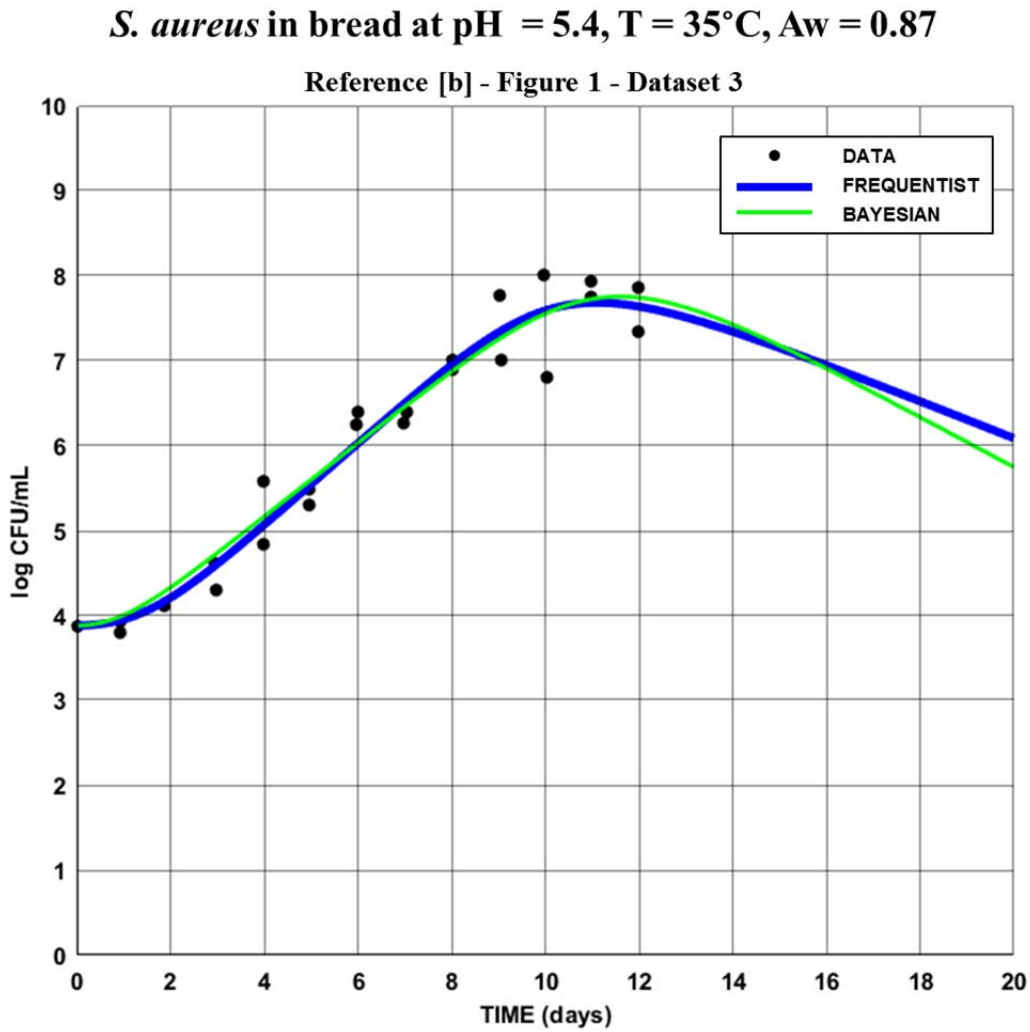
Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	1.61	13.13	307.30	5.70×10^{5}	209.50	3.90×10^{5}	306.76	5.70×10^{5}	3.88×10^{-2}
Bayesian	1.04	1.00	234.79	1.00	316.97	1.00	234.23	1.00	3.85×10^{-2}

Figure 4-3. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 1, Dataset 1



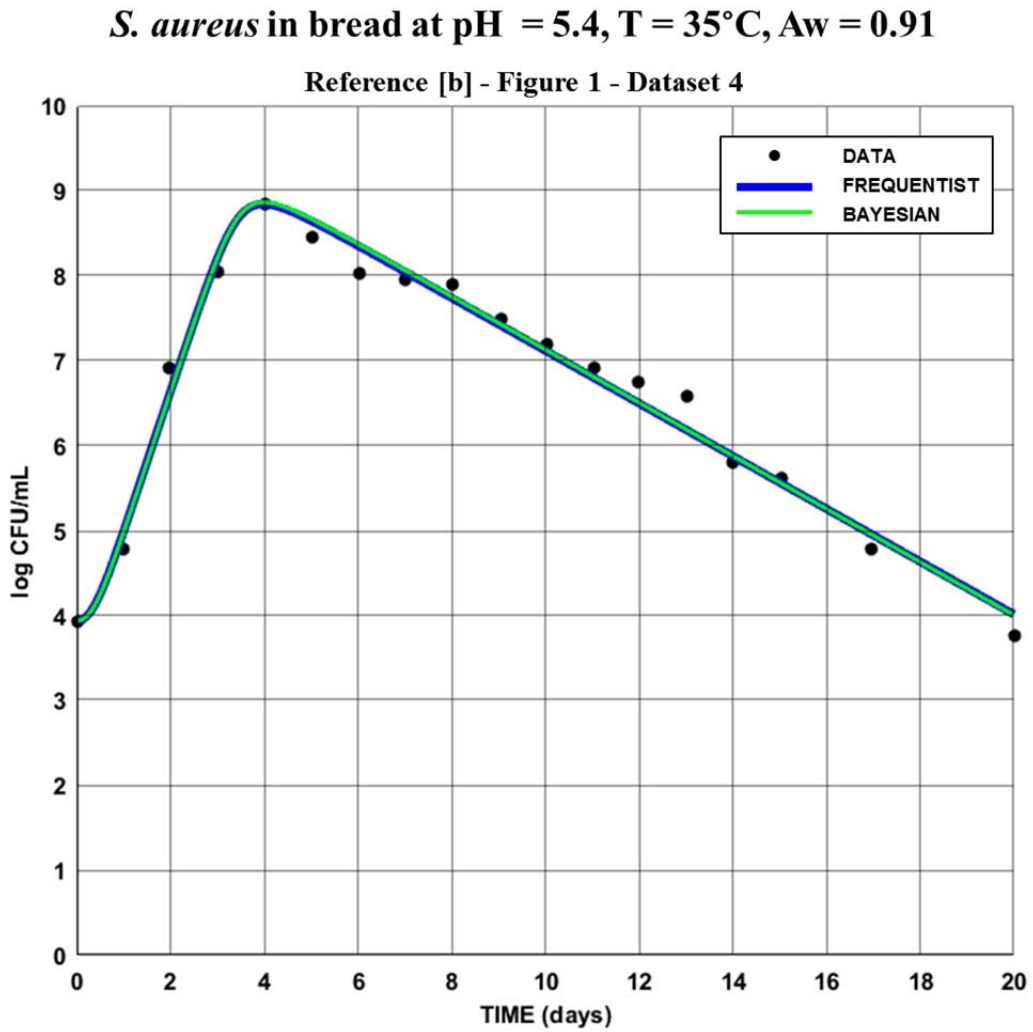
Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	0.37	0.80	97.66	1.60×10^4	77.06	1.30×10^4	97.25	1.60×10^4	1.01×10^0
Bayesian	0.37	1.25	41.15	1.00	184.31	1.25	40.74	1.00	1.01×10^0

Figure 4-4. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 1, Dataset 2



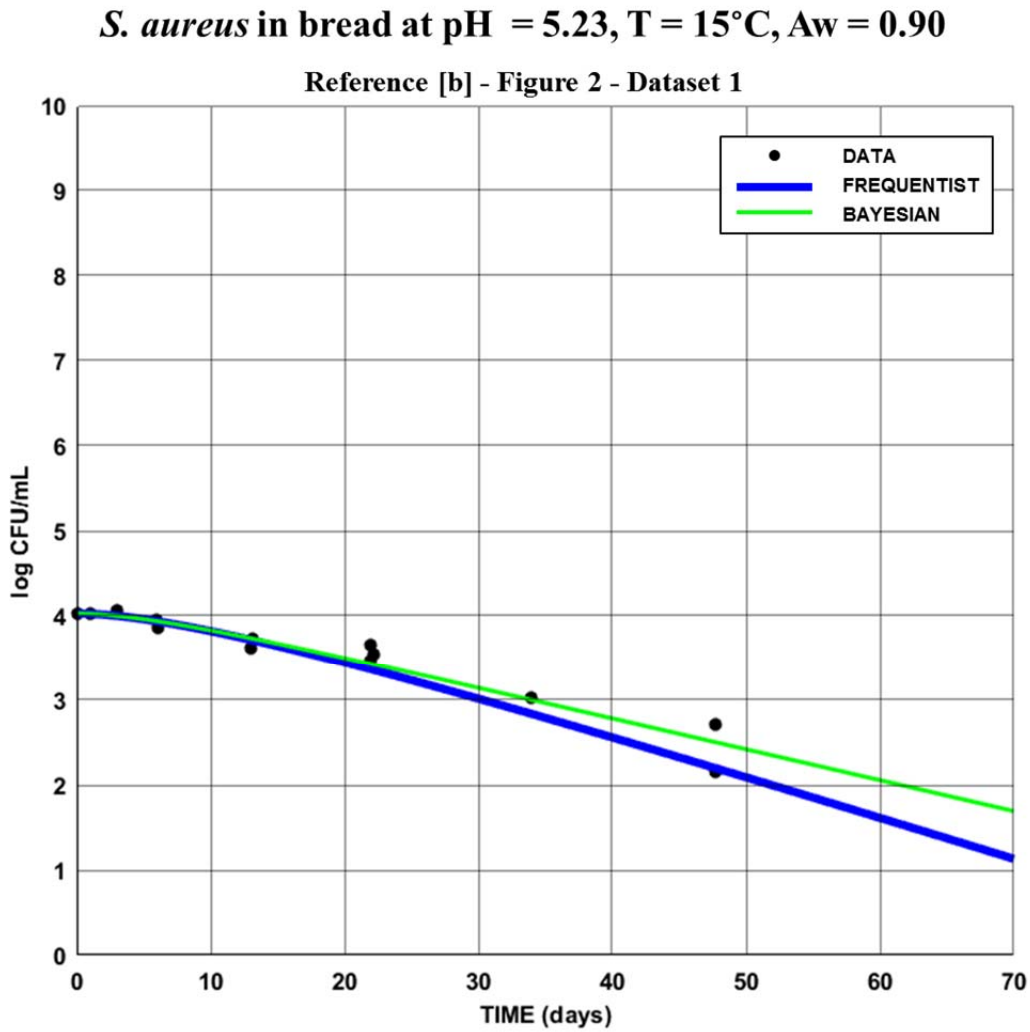
Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	0.23	0.19	1.77	3.24	9.70	28.52	0.65	3.39	2.08×10^0
Bayesian	0.53	1.73	2.44	1.03	4.24	1.89	1.44	1.04	2.20×10^0

Figure 4-5. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 1, Dataset 3



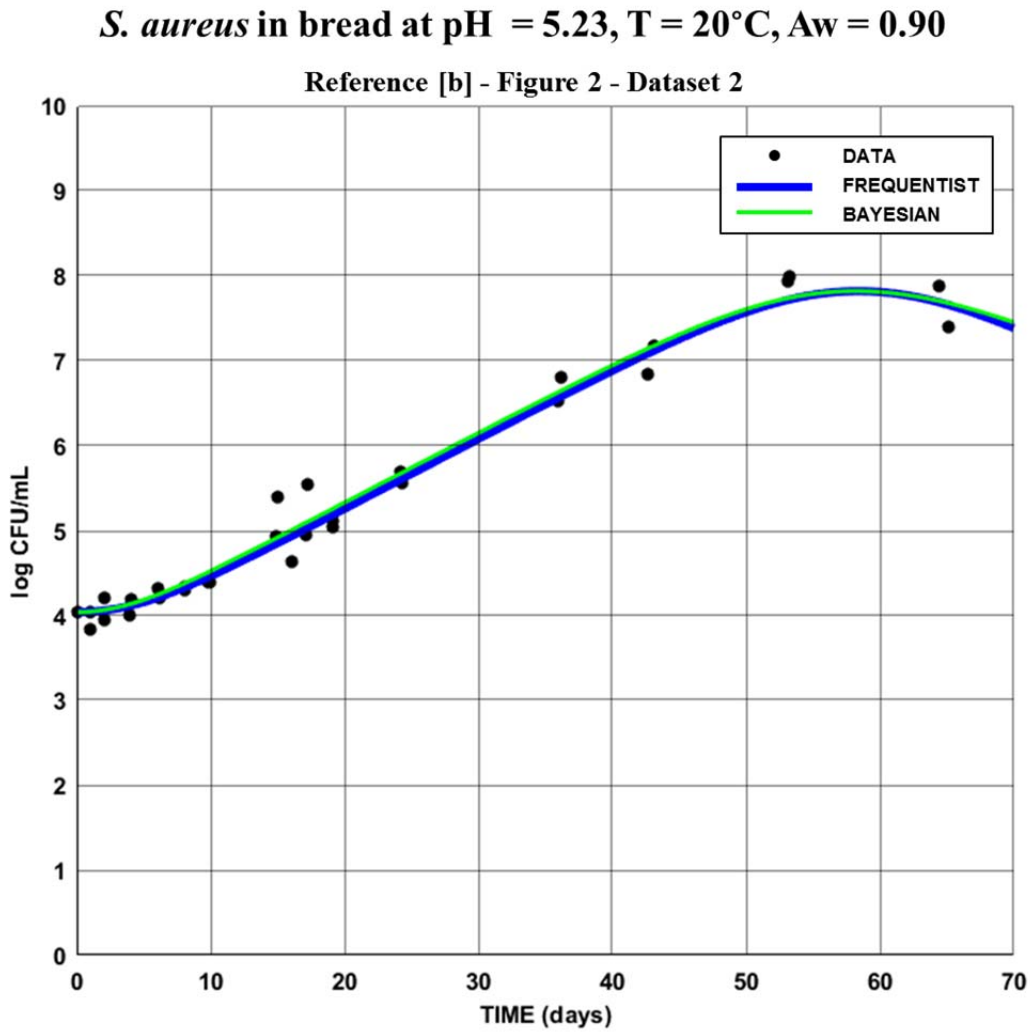
Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	1.27	1.25	4.52	0.38	3.55	0.77	0.72	0.50	6.55×10^{-1}
Bayesian	1.23	1.88	4.53	1.06	3.39	1.22	0.73	1.05	6.57×10^{-1}

Figure 4-6. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 1, Dataset 4



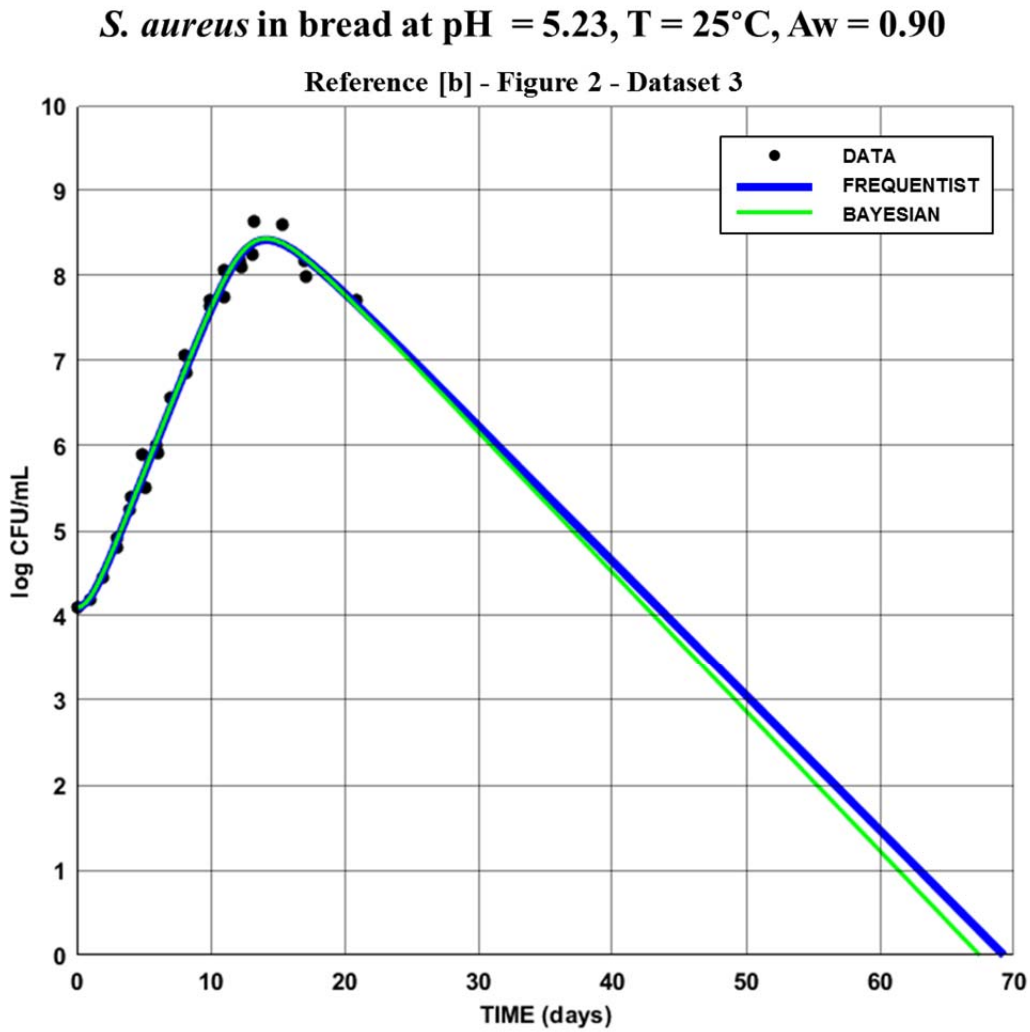
Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	0.16	26.46	37.44	6.98×10^{5}	0.66	1.24×10^{4}	37.55	6.98×10^{5}	4.61×10^{-1}
Bayesian	0.22	1.50	0.96	1.01	1.00	7.18	1.05	1.01	2.54×10^{-1}

Figure 4-7. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 1



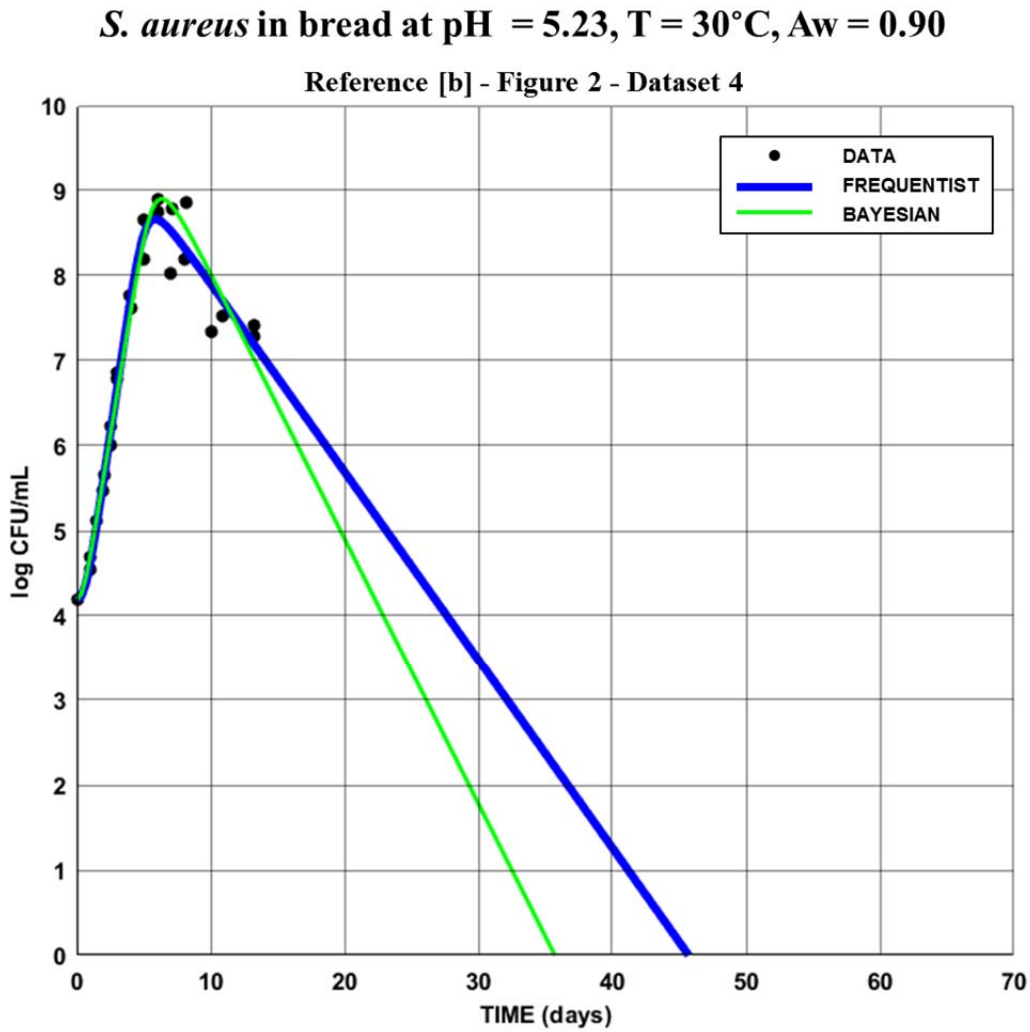
Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	0.11	0.05	5.55	2.32×10^{2}	0.05	1.96	5.36	2.32×10^{2}	1.15×10^{0}
Bayesian	0.15	1.47	0.82	1.01	0.36	1.47	0.63	1.01	1.13×10^{0}

Figure 4-8. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 2



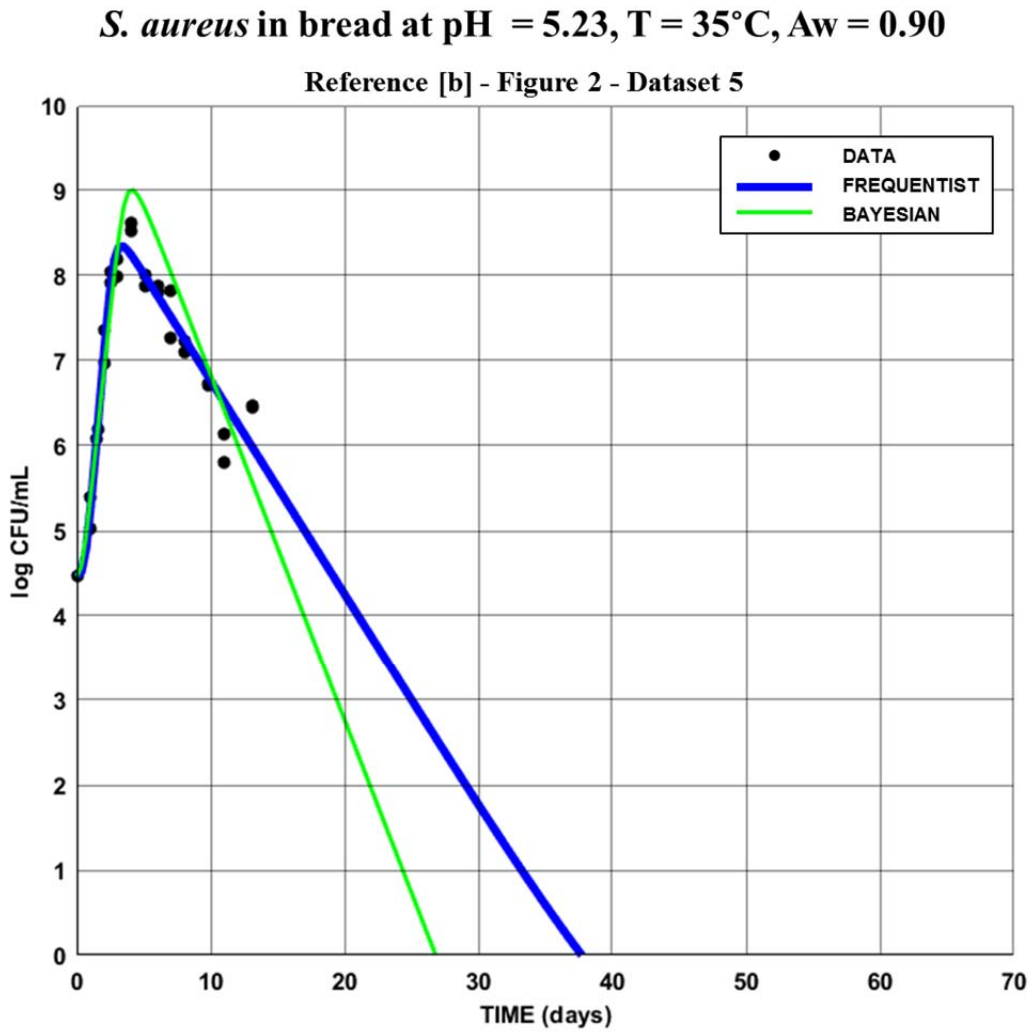
Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	0.58	0.20	1.35	0.15	1.58	0.53	0.44	0.16	4.79×10^{-1}
Bayesian	0.61	1.14	1.38	1.01	1.49	1.11	0.46	1.03	5.37×10^{-1}

Figure 4-9. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 3



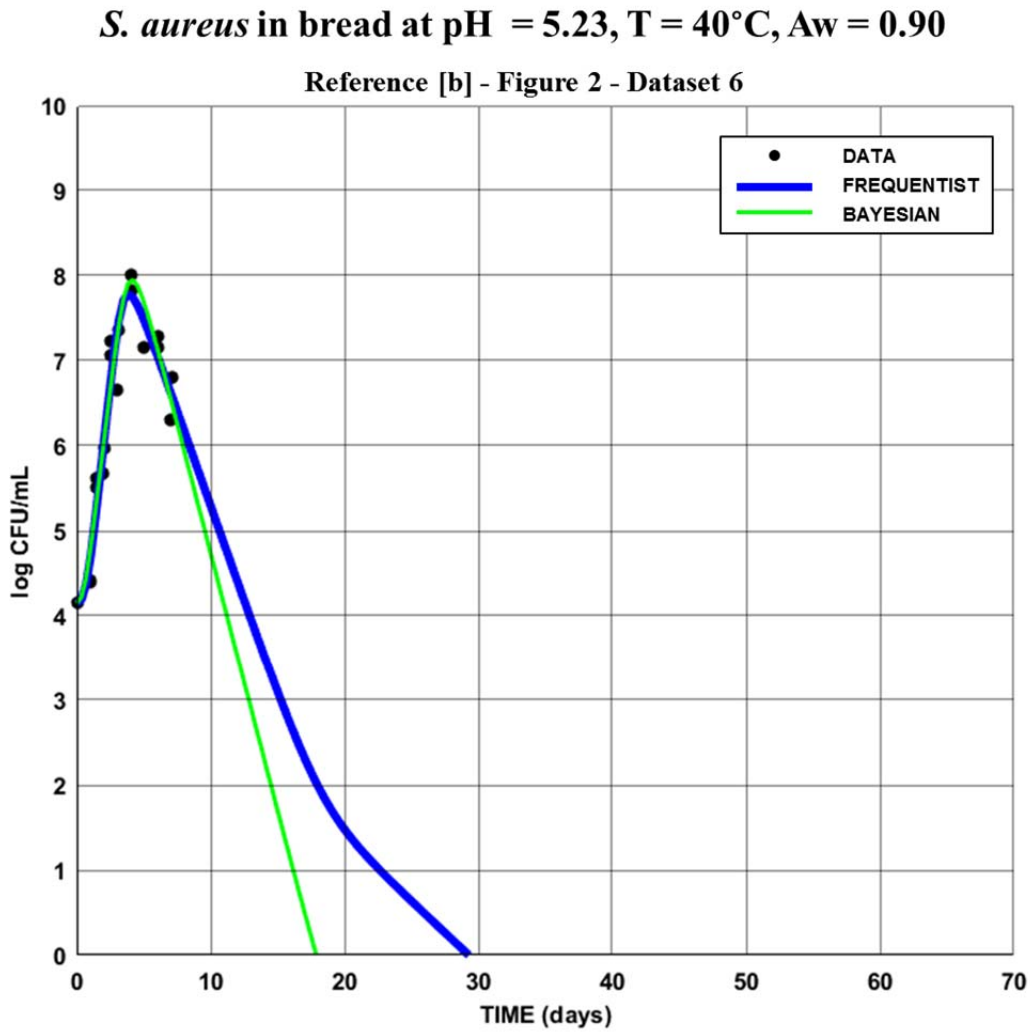
Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	0.57	0.31	2.98	0.17	3.33	0.94	0.52	0.09	1.34×10^0
Bayesian	1.06	1.41	3.02	1.05	1.44	1.32	0.79	1.16	1.93×10^0

Figure 4-10. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 4



Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	0.32	0.28	4.89	0.48	13.56	3.83	0.58	0.06	1.91×10^0
Bayesian	1.10	1.47	4.44	1.08	1.91	1.35	0.99	1.14	6.42×10^0

Figure 4-11. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 5



Approach	Solution Rate Parameters								SSR
	k_1		k_2		k_3		k_4		
	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	Mean	SE/σ	
Frequentist	0.34	0.32	4.46	0.48	29.53	17.39	1.12	0.46	1.55×10^0
Bayesian	0.68	1.72	4.86	1.05	13.57	1.49	1.85	1.14	1.73×10^0

Figure 4-12. Comparison of Frequentist and Bayesian QCM-4 Model Fits for Reference [b], Figure 2, Dataset 6

4.5 FORWARD UNCERTAINTY PROPAGATION

The goodness of model fit to data can be measured in terms of how accurate is the model fit relative to the measured data, and in how stable is the fit. For the QCM-4 model, accuracy in the model fit is given by the value of the SSR. Stability refers to the sensitivity of the model fit to uncertainty in the estimate of the model parameters. For a model such as the QCM-4 model, uncertainty can be due to (Reference [f])

- The measured data from an experiment may not be adequate (e.g. measured bacteria data may not occur at sufficient times to describe all changes in growth kinetics),
- Measured data has errors and may not be reproducible, and
- Numerical solvers give approximate solutions up to some tolerance level.

In this section we apply the method of forward uncertainty propagation discussed in Section 3.5 to analyze the effects of uncertainty in the estimated rate parameters for solutions to six of the measured datasets that are representative of the growth and death seen in the 39 datasets. The forward uncertainty propagation method first involves estimating the rate parameters that minimize the SSR to a set of measured data, and estimating the covariance in the rate parameter estimate based on the sensitivity of the QCM-4 model at the solution and the variability in the measured data about the solution. Under the assumption that the rate parameters are random variates from a multivariate normal distribution, random draws of rate parameters are used to generate a distribution of growth curves. The distribution of growth curves about the solution growth curve can then be used to show how uncertainty in the rate parameters affects the estimated bacteria growth curve. In the analysis of the uncertainty, growth rate curves are generated from the initial time out to the maximum time for each experiment. This allows for examining how uncertainty affects the estimated growth curves over the range of measured data for purposes of mean growth estimates or interpolation, and how uncertainty can affect predictions of future growth kinetics through extrapolation.

Formulation B was initially used for the forward uncertainty propagation, but two problems occurred with using this formulation. First, in Formulation B, the rate parameters are $\boldsymbol{\kappa} = [k_1, k_2, k_3, k_4]^T$ where each of the k_i is nonnegative. Referring to the Frequentist solutions in Figures 4-3 through 4-12, it can be seen that the standard error of a rate parameter can be as much as six orders of magnitude larger than its mean. With large standard errors, the chance of randomly drawing a negative value is close to 50%. The assumption that the model parameters are normally distributed under Formulation B is not valid. To avoid this problem, the model parameters were chosen to be the natural logarithm of the rate parameters which maps nonnegative real numbers into all real numbers (Formulation D).

A second problem occurred in situations in which the measured data only showed death-only kinetics. The QCM-4 solutions correctly reflected death-only kinetics in that $\alpha = k_2 - k_4 \leq 0$, but the resulting covariance matrix was often not positive semi-definite which is required for uncertainty propagation. This issue was addressed by resolving any QCM-4 solution with death-only kinetics using the two-parameter 1-stage exponential model with lag, and using the resulting two parameter covariance matrix for uncertainty propagation. For QCM-4 solutions exhibiting

growth, Formulation E was used to estimate the mean solution and covariance matrix for uncertainty propagation.

Figures 4-13 through 4-18 show the forward uncertainty propagation results for six datasets that are representative of the growth kinetics seen in all 39 datasets listed in Appendix A. Each figure shows a plot of the measured data; a plot of the growth curve resulting from the mean rate parameters (i.e. thick dark line); and plots of the randomly generated growth curves (i.e. thin light lines). A table with the mean rate parameters converted to $\boldsymbol{\kappa} = [k_1, k_2, k_3, \alpha]^T$, the associated standard errors of the rate parameters, and the SSR between the solution growth curve and the measured data are shown on each figure.

The six representative datasets are the following:

Small Initial Growth (Reference [b], Figure 1, Dataset 1). Figure 4-13 shows the forward uncertainty propagation results. This dataset consists of 8 measurements with kinetics that show a small amount of initial growth followed by death. Note that the randomly generated curves are not symmetric about the mean solution, and cover a large spread of values.

The largest number of randomly generated curves have curvature similar to the mean curve and reach a logarithmic count of zero between 11 and 14 days. These randomly generated curves are primarily due to the rate parameter with the smallest standard error which is the net growth rate, α. The net growth rate controls how fast the growth increases, and how fast death occurs. The remaining growth curves which are approximately linear are primarily due to the distribution of the lag rate, k_1. Equation (3-10) shows that the concentration of lag cells decreases exponentially at a constant rate. On a plot of the logarithm of the count as a function of time, the number of lag cells decreases linearly. The lack of symmetry of the randomly generated curves about the mean curve is due to the covariance between the parameters and the fact that death kinetics with slopes significantly greater than that of the mean curve in Figure 4-13 require values for k_2 and k_4 that are larger than can be randomly generated based on the means and standard errors shown in Figure 4-13.

Large Growth - Slow Death (Reference [b], Figure 2, Dataset 2). Figure 4-14 shows the forward uncertainty propagation results. This dataset consists of 30 measurements with kinetics that show a large amount of initial growth. The majority of the randomly generated curves for this dataset align tightly about the mean curve due to the small standard error for α. The remaining randomly generated curves that show near linear growth are due to small values for α and k_1.

Large Initial Growth - Fast Death (Reference [b], Figure 2, Dataset 5). Figure 4-15 shows the forward uncertainty propagation results. This dataset consists of 27 measurements with kinetics that show a large amount of initial growth occurring quickly, followed by a quick death phase. Over the period where measured data are available, the randomly generated curves tightly follow the mean curve with little spread. Past the measured data, the spread in the randomly generated curves begins to increase slowly. There are a small number of randomly generated curves that change slope between times of about 24 and 34 days. This change in slope is called a lag tail. It can be shown (Reference [a]) that the growth rate for the antagonistic metabolite (i.e.

dA/dt) is always positive, and increases asymptotically. If dA/dt increases quickly enough, then the transition of lag phase cells to exponential growth only occurs for a short period of time. After this time, the concentration of A is sufficiently large that lag phase cells that become activated die almost immediately, and the growth curve decreases at the lag rate.

Large Fast Growth - Slow Death (Reference [b], Figure 4, Dataset 2). Figure 4-16 shows the forward uncertainty propagation results. This dataset consists of 27 measurements with kinetics that show a large amount of initial growth that occurs quickly, followed by a slow death phase. For this dataset, the randomly generated curves tightly align symmetrically about the mean curve.

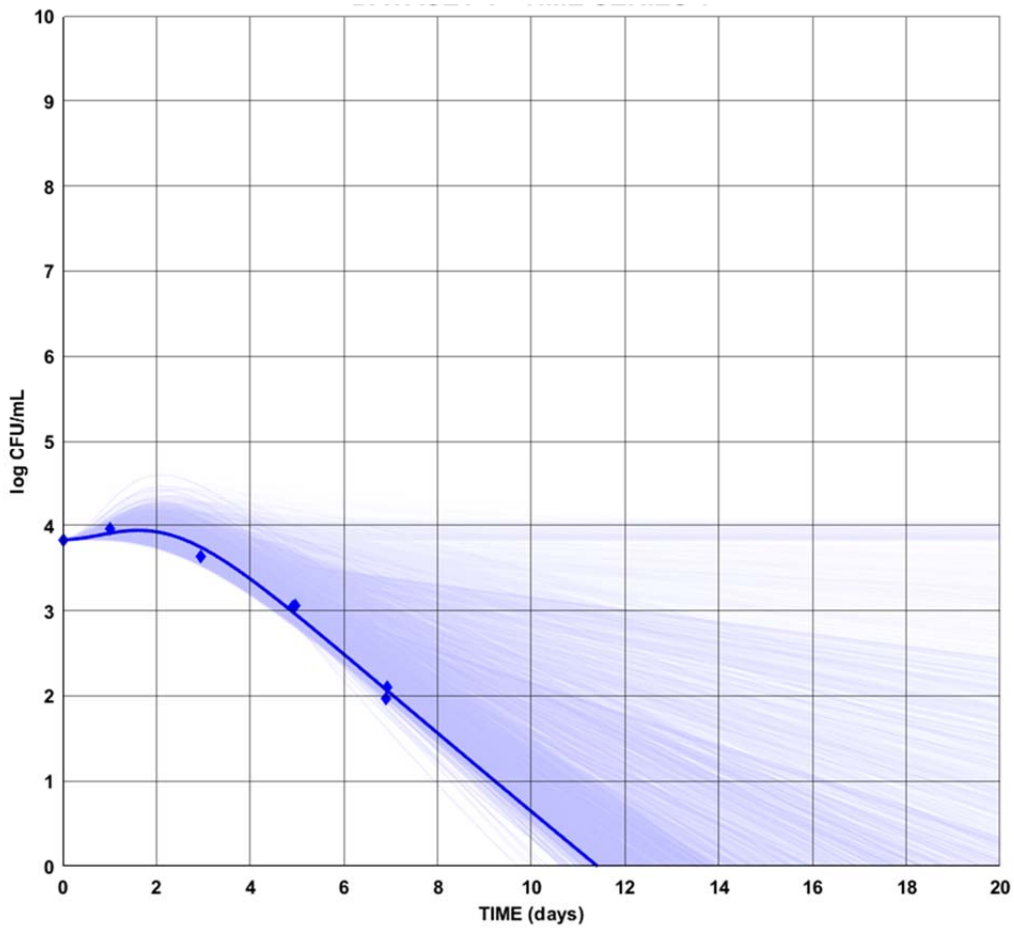
Linear Death (Reference [b], Figure 4, Dataset 4). Figure 4-17 shows the forward uncertainty propagation results. This dataset consists of 25 measurements with kinetics that show death-only kinetics that are linear. Since this is a death-only dataset, the mean solution and uncertainty propagation use the reduced parameter 1-stage exponential model with lag given by Equation (3-39). For this dataset, the mean and randomly generated curves are linear. Figure 4-17 shows that the mean value for the parameter a is largely negative. Equation (3-39) shows that as the value of a becomes large, the terms on the right approach zero, and the model becomes an exponential decay in the single parameter k_1. As a single parameter model, the randomly generated curves are also linear, and show little spread about the mean curve.

Non-linear Death (Reference [b], Figure 2, Dataset 1). Figure 4-18 shows the forward uncertainty propagation results. This dataset consists of 14 measurements that show death-only kinetics. Since this is a death-only kinetics dataset, the mean solution and uncertainty propagation use the reduced parameter 1-stage exponential model with lag. Unlike dataset 4 with linear death, this dataset has two parameter death kinetics; the randomly generated curves show the kinetics that occur between the extreme values for both parameters.

SMALL INITIAL GROWTH

S. aureus in bread at pH = 5.4, T = 35°C, Aw = 0.79

Reference [b] - Figure 1 - Dataset 1



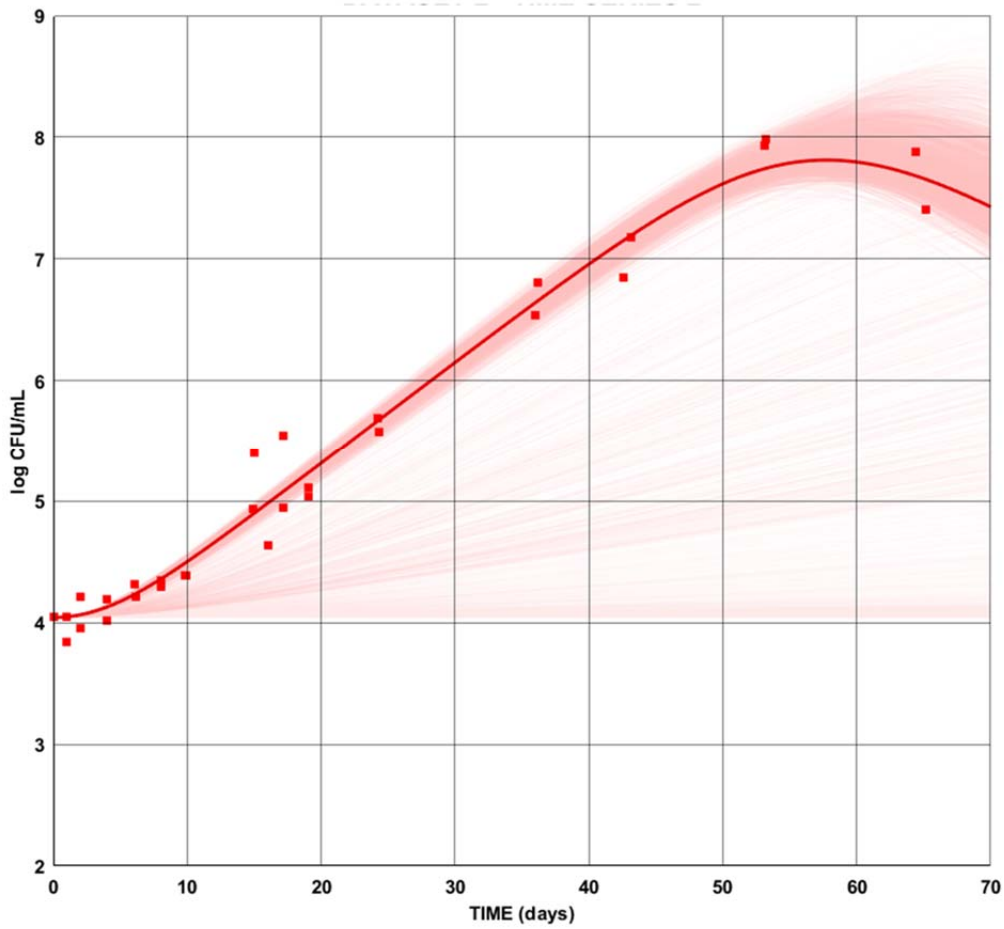
Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.98	4.98	17.66	13.51	4618.32	11.43	0.56	2.22	4.16×10^{-2}

Figure 4-13. Forward Uncertainty Propagation for Small Initial Growth

LARGE GROWTH – SLOW DEATH

S. aureus in bread at pH = 5.23, T = 20°C, Aw = 0.90

Reference [b] - Figure 2 - Dataset 2



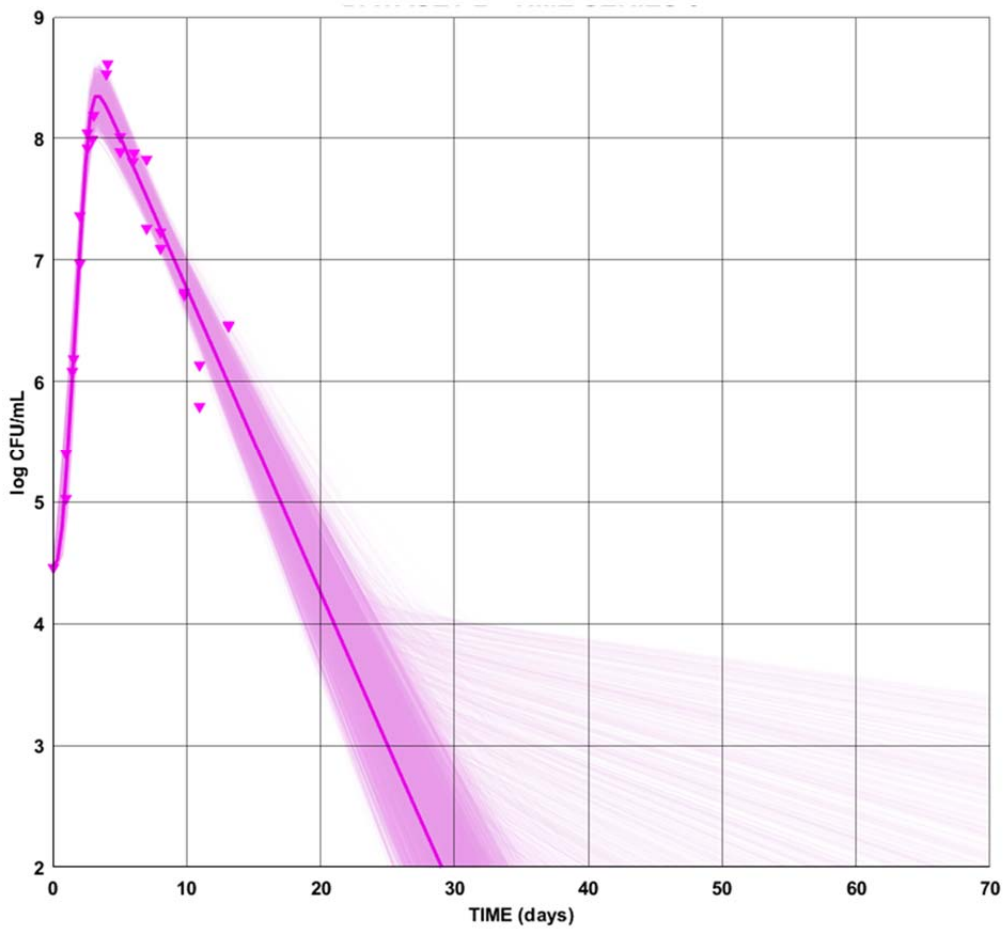
Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.12	0.12	0.71	5.16	0.44	5.98	0.19	0.03	1.12×10^{0}

Figure 4-14. Forward Uncertainty Propagation for Large Growth – Slow Death

LARGE INITIAL GROWTH – FAST DEATH

S. aureus in bread at pH = 5.23, T = 35°C, Aw = 0.90

Reference [b] - Figure 2 - Dataset 5



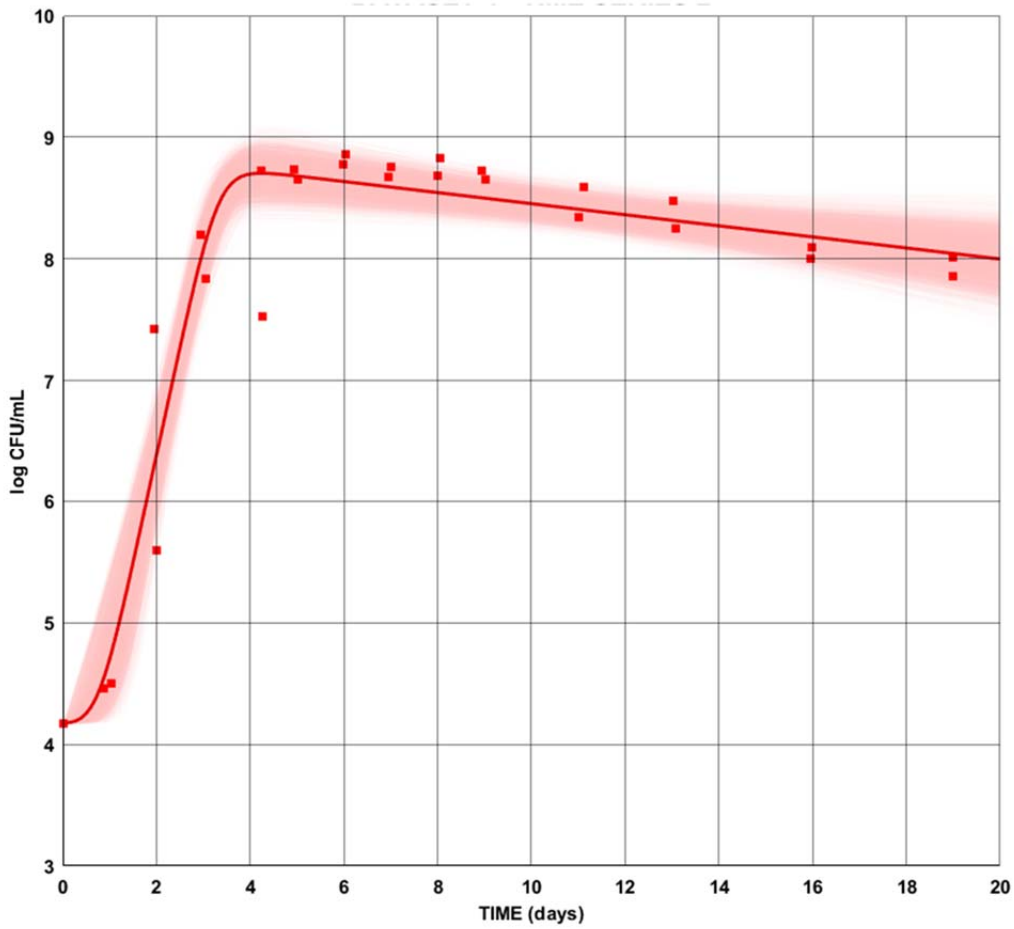
Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.32	1.34	4.89	0.10	13.55	0.28	4.31	0.12	1.91×10^{0}

Figure 4-15. Forward Uncertainty Propagation for Large Initial Growth – Fast Death

LARGE FAST GROWTH – SLOW DEATH

S. aureus in turkey at pH = 6.35, T = 35°C, Aw = 0.87

Reference [b] - Figure 4 - Dataset 2



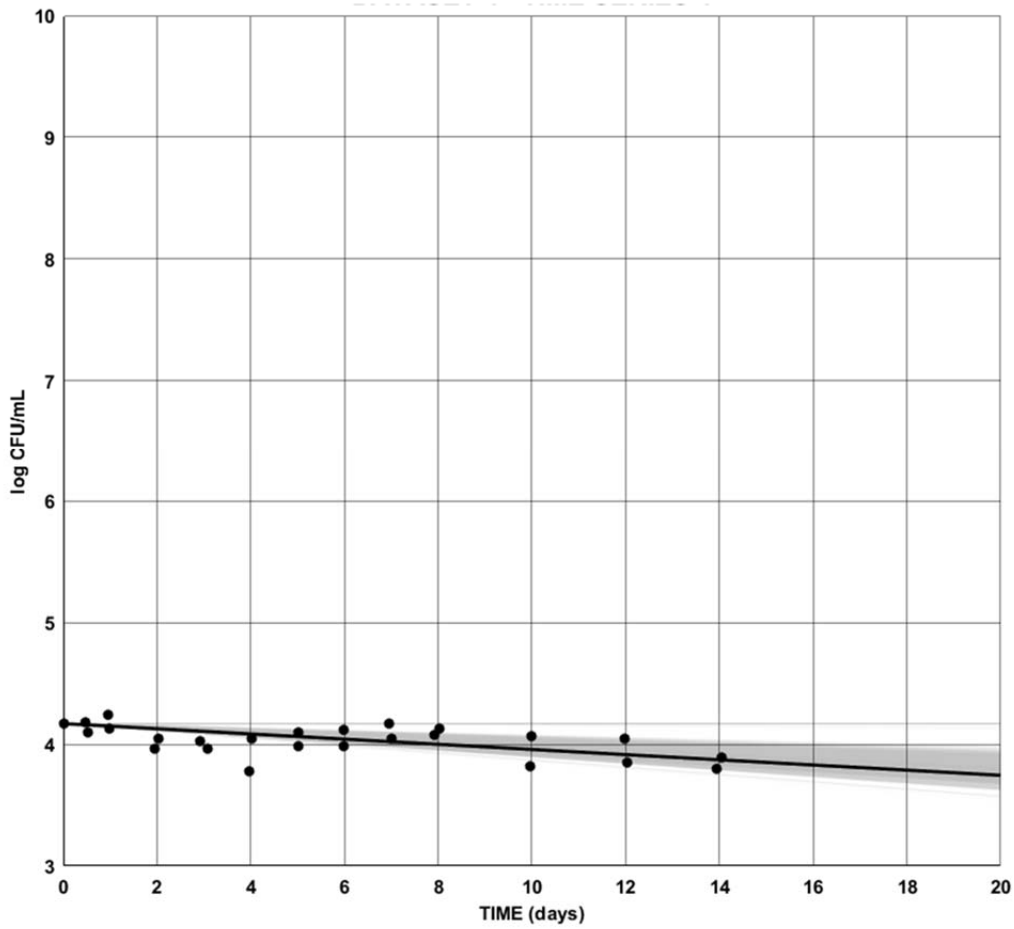
Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.17	2.14	4.25	0.16	7.44	0.40	4.15	0.16	3.94×10^{0}

Figure 4-16. Forward Uncertainty Propagation for Large Fast Growth – Slow Death

LINEAR DEATH

S. aureus in turkey at pH = 6.35, T = 35°C, Aw = 0.84

Reference [b] - Figure 4 - Dataset 4



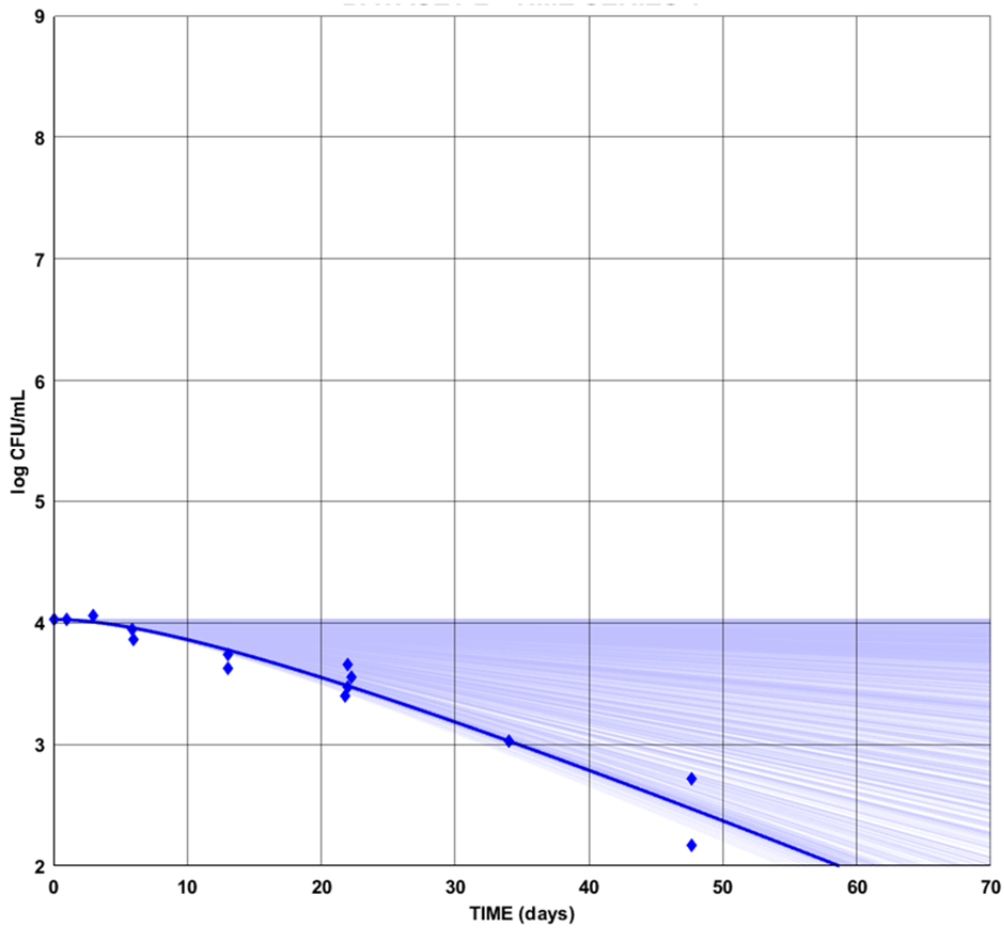
Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-998.00	1909.37	0.05	0.26	2.94×10^{-1}

Figure 4-17. Forward Uncertainty Propagation for Linear Death

NON-LINEAR DEATH

S. aureus in bread at pH = 5.23, T = 15°C, Aw = 0.90

Reference [b] - Figure 2 - Dataset 1



Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-0.11	11.00	0.12	11.28	2.36×10^{-1}

Figure 4-18. Forward Uncertainty Propagation for Non-Linear Death

5.0 CONCLUSIONS

Conclusions are:

- Treating the model parameters as stochastic quantities in solving the QCM inverse problem provides better predictive capabilities than treating the parameters as deterministic quantities.

- The Frequentist and Bayesian approaches provide the same mean solution for the inverse problem using the QCM datasets.

- The Frequentist and Bayesian results for Standard Error and Standard Deviation are different for the inverse problem using the QCM datasets.

  In the frequentist case, estimates are based on the available experimental data; while in the Bayesian setting, they depend also on the choice of the prior. In theory, the two approaches would provide the same results if the Bayesian method was using a completely uninformative prior.

- The QCM datasets do not support an accurate estimate of all the stochastic model parameters.

  In some of the QCM datasets (e.g. Figures 4-13 and 4-18) the dynamics are simple and the QCM model is overparameterized which leads to large uncertainty in estimated parameters and instability in the inverse problem. In these cases, surrogate models can be used to reduce parameter uncertainty with little increase in model error.

  The model output is similar for significantly different rate parameters; i.e. for some of the datasets (e.g. Figures 4-3, 4-4, and 4-8) there are numerous local minima. This means that the inverse problem is under-determined which can be due to an insufficient number of observations or to high correlation in model parameters, e.g. k_2 and k_4.

6.0 AREAS FOR FURTHER INVESTIGATION

In this section we discuss areas for further investigation.

6.1 REDUCING UNCERTAINTY IN PARAMETER ESTIMATES

6.1.1 Parameter Ranking

As discussed in Section 4, certain types of bacteria growth may not support a four parameter model such as the QCM-4 model in all situations; for example, if measured data are only collected over a portion of the bacterial growth cycle such as activation only or inactivation only. One approach to test for this type of overparameterization is through parameter ranking and model comparison testing. This method ranks the model parameters by sensitivity relative to some model solution, then resolves the problem by varying a subset of the model parameters while the remaining parameters are held constant. In this way the significant model parameters can be identified.

The parameter ranking and model comparison method can be applied using either a Frequentist approach or a Bayesian approach. However, the approach is a local assessment of parameter significance. The methodology relies on a set of nominal values to fix the model parameters not being tested at each stage of the model comparison. For the Frequentist approach the nominal values are taken from the initial model parameter vector; for the Bayesian approach the nominal values are taken from the prior probabilities. If a different choice is made for the nominal values, then the list of significant parameters can change.

Frequentist Approach. Section 2.1.3 presented a method for determining which model parameters are significant in a given model solution. We illustrate this by applying this method to Datasets 1 and 2 from Figure 1 of Reference [b] using Formulation C of the QCM-4 model where $\boldsymbol{\kappa} = [k_1\ k_2\ k_3\ \alpha]^T$. For Dataset 1, the ranking of the model parameters in decreasing sensitivity using the QR decomposition is $[\alpha, k_1, k_3, k_2]^T$. Table 6-1 shows the parameters used for the hypothesis tests and the resulting value for $\mathcal{U}(\mathbf{y})$ using Equation (2-18). The critical value for the chi-squared test at a 90% confidence is 2.71. The first test compares $\boldsymbol{\kappa}_H = \alpha$ and $\boldsymbol{\kappa} = [\alpha k_1]$. Since $\mathcal{U}(\mathbf{y})$ is greater than the critical value of 2.71, the null hypothesis is rejected. That is, the data does support at a significantly statistical level the model with the estimation of both α and k_1. The second hypothesis test is to see if the data will support the estimation of the third most significant parameter, k_3, by comparing the values $\boldsymbol{\kappa}_H = [\alpha\ k_1]$ and $\boldsymbol{\kappa} = [\alpha\ k_1\ k_3]$. Since the value of the test statistic, 7.93×10^{-2}, does not exceed the critical value, the null hypothesis cannot be rejected. We conclude that the model can sufficiently describe the dataset using only the parameters α and k_1. The resulting two parameter model reduces the standard errors for α and k_1 by more than 95%, with an increase in the SSR of 1.3%.

Table 6-1. Frequentist Model Comparison Using Dataset 1 of Figure 1 in Reference [b]

$\boldsymbol{\kappa}_H$	$\boldsymbol{\kappa}$	$\mathcal{U}(\mathbf{y})$
α	α , k_1	1.01×10^{2}
α , k_1	α , k_1 , k_3	7.93×10^{-2}

For Dataset 2 of Figure 1 in Reference [b], the ranking of the model parameters in decreasing sensitivity is also $[\alpha, k_1, k_3, k_2]^T$. Table 6-2 shows the parameters used for the hypothesis tests and the resulting value for $\mathcal{U}(\mathbf{y})$. The first hypothesis test with $\boldsymbol{\kappa}_H = \alpha$ and $\boldsymbol{\kappa} = [\alpha\ k_1]$ rejects the null hypothesis at a 90% confidence level, while the second hypothesis test with $\boldsymbol{\kappa}_H = [\alpha\ k_1]$ and $\boldsymbol{\kappa} = [\alpha\ k_1\ k_3]$ does not reject the null hypothesis. We conclude that the model can sufficiently describe this dataset using only the parameters α and k_1. The resulting two parameter model reduces the standard errors for α and k_1 by more than 65% with an increase in the SSR of 0.8%.

Table 6-2. Frequentist Model Comparison Using Dataset 2 of Figure 1 in Reference [b]

$\boldsymbol{\kappa}_H$	$\boldsymbol{\kappa}$	$\mathcal{U}(\mathbf{y})$
α	α , k_1	2.80×10^{0}
α , k_1	α , k_1 , k_3	1.40×10^{-1}

Bayesian Approach. The Bayesian approach is similar to the Frequentist approach in that it starts with a parameter ranking using the QR decomposition, but uses the Bayesian Information Criterion (Equation (3-55)), and the value of the evidence lower bound (Equation (2-29)) for the model comparison. Both values are calculated since the BIC is only known to be valid when the number of observations is much greater than the number of parameters estimated. The motivation of using the ELBO, $\mathcal{F}[q]$, is that the BIC is an estimate of the evidence $p(\mathbf{y})$, and $\mathcal{F}[q]$ is a lower bound of the evidence (Equation (2-31)). In general, the highest value for the BIC suggests a preferable model, and the larger the increase in its value, the stronger the evidence that the model with the largest BIC value is closest to the truth.

The Bayesian approach, with Formulation D where $\boldsymbol{\kappa} = [z_1\ z_2\ z_3\ z_4]^T$ and $z_i = \ln(k_i)$, is applied to the same two datasets used in the Frequentist approach example discussed above. For Dataset 1 of Figure 1 in Reference [b], the ranking of the model parameters in decreasing sensitivity is $[z_1, z_4, z_2, z_3]^T$. Table 6-3 shows the results. In Table 6-3, the values for both the BIC and $\mathcal{F}[q]$ decrease as the number of model parameters increase. Since the largest change in the values occurs between the one and two parameter models, the dataset supports accurate estimation of only the single parameter $z_1 = \ln(k_1)$. This differs from the results of the Frequentist approach which concludes that two parameters are significant.

Table 6-3. Bayesian Model Comparison Using Dataset 1 of Figure 1 in Reference [b]

κ_H	BIC	$\mathcal{F}$[q]
z_1	1.02	-3.54
z_1 , z_4	0.19	-27.43
z_1 , z_4 , z_2	–0.85	-36.70
z_1 , z_4 , z_2 , z_3	–2.04	-46.00

For Dataset 2 of Figure 1 in Reference [b], the ranking of the model parameters in decreasing sensitivity is $[z_2,z_1,z_4, z_3]^T$. Table 6-4 shows the results. In Table 6-4, the two parameter model has the largest values for both BIC and $\mathcal{F}$[q], which implies that the dataset supports accurate estimation of the two parameters $z_2 = \ln(k_2)$ and $z_1 = \ln(k_1)$. This result is similar to the result of the Frequentist approach for this dataset.

Table 6-4. Bayesian Model Comparison Using Dataset 2 of Figure 1 in Reference [b]

κ_H	BIC	$\mathcal{F}$[q]
z_2	-25.16	-29.33
z_2 , z_1	-4.38	-16.65
z_2 , z_1 , z_4	–5.79	-26.66
z_2 , z_1 , z_4 , z_3	–7.21	-30.01

6.1.2 Alternate Models

Section 3.3.3 discussed four alternate models for modeling bacteria growth and decay. These models are often less sensitive to uncertainty in the model parameters than the QCM-4 model, but at the expense of less accuracy. The AIC discussed in Section 3.3.3 provides a method for determining when a change in the model is beneficial to reducing uncertainty in the model output. In this section we present examples of applying the alternate growth models to datasets that exhibit different types of bacteria growth and death.

Death Only Kinetics. Death only kinetics occurs in one third of the datasets shown in Appendix A. In death only kinetics, on average, there is no change in concavity to the data over time; a one or two parameter model is likely to be accurate as the QCM-4 model. For forward uncertainty propagation, the use of the 1-stage alternate models for death only kinetics was required in order to produce a covariance matrix that was positive semi-definite.

Figure 6-1 shows the data from Dataset 5 of Figure 3 in Reference [b] as well as the fits to the data for the 1-Stage Exponential (1EQ); 1-Stage Exponential with Lag (1EL); and QCM-4 models. Figure 6-1 shows good agreement between the 1EL and QCM-4 models.

Table 6-5 shows the estimated model parameters and standard errors for the output of the three models shown in Figure 6-1. The lag rate (k_1 = 1.15) and net growth rate (k_2-k_4 = -1.15) of the QCM-4 model agree with the corresponding values for the 1EL model. The standard error in the

lag rate is smaller for the QCM-4 model than the 1EL model, but the standard error in the net growth rate for the 1EL model (i.e. a) is significantly smaller than the standard errors for k_2, k_3, and k_4 of the QCM-4 model.

Table 6-5. Alternate and QCM-4 Model Fits and Standard Errors for Dataset 5 of Figure 3, Reference [b]

1-Stage Exponential Model		
q	**Estimate**	**Standard Error**
a	-0.73	0.05
1-Stage Exponential Model With Lag		
q	**Estimate**	**Standard Error**
k_1	1.14	144.66
a	-1.14	144.64
Quasi-Chemical Model		
κ	**Estimate**	**Standard Error**
k_1	1.15	10.37
k_2	1.88	9.18×10^6
k_3	11.16	5.45×10^7
k_4	3.03	9.18×10^6

Table 6-6 shows the AIC analysis results for the model output in Figure 6-1 and includes the number of measured data values in the dataset (n), the number of parameters estimated by each model (N_κ), the value of the cost function ($J(\boldsymbol{\kappa})$), and the value of the AIC corrected for finite sample sizes (AIC_c). For the QCM-4 and the alternate models, the cost function is the SSR between the model output and the observations. Recalling from Section 3.3.3, the best approximating model is the one with the minimum value for AIC_c. In Table 6-6, the minimum value for AIC_c occurs for the 1EL model at a cost of a 6% increase in the value of the cost function.

Table 6-6. AIC Analysis Values for Dataset 5 of Figure 3, Reference [b]

Model	n	N_κ	$J(\kappa)$	AIC_c
1-Stage Exponential	11	1	0.92	-21.85
1-Stage Exponential with Lag	11	2	0.36	**-28.19**
Quasi-Chemical	11	4	0.34	-16.12

Primarily Growth Kinetics. In 36% of the datasets shown in Appendix A, the data do not show a significant death phase relative to the growth phase over the time period the data were collected. Since there is growth and some death, there is, on average, a concavity change in the data and 2-stage alternate models should be considered.

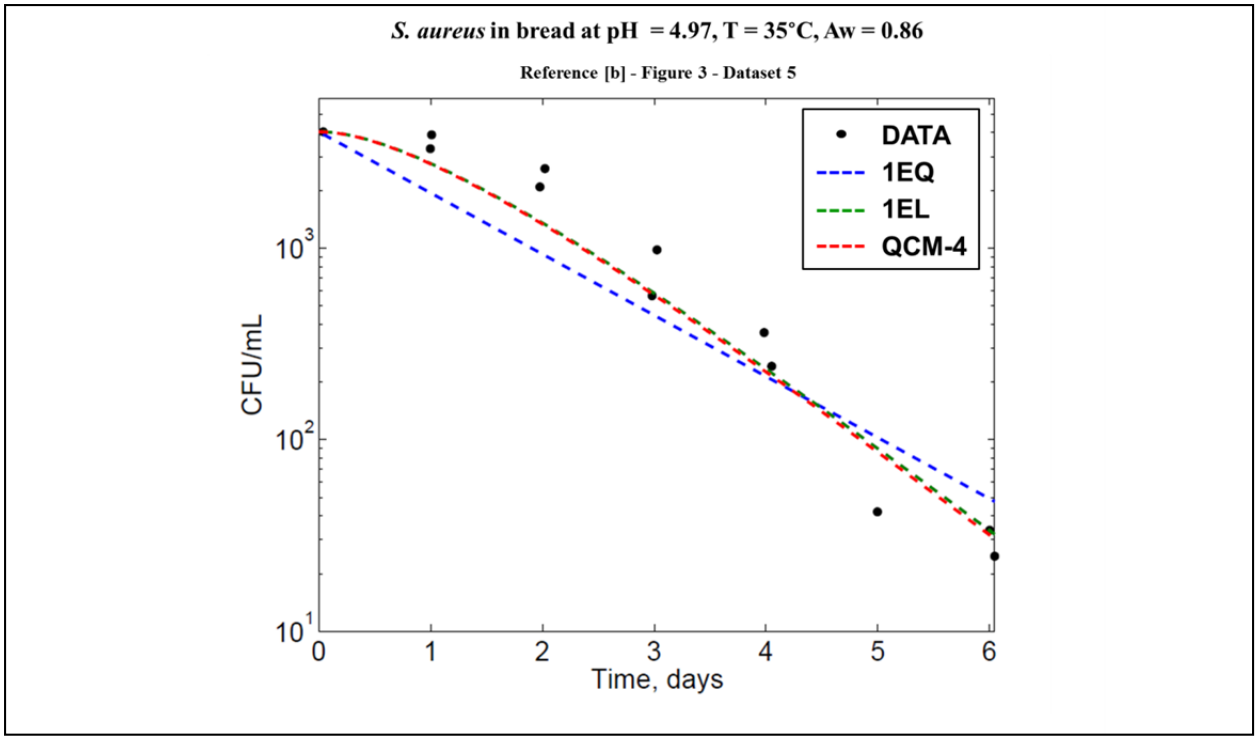


Figure 6-1. Example of Alternate Model Fits to Data with Death Only Kinetics

Figure 6-2 shows the data from Dataset 1 of Figure 3 in Reference [b] as well as the fits to the data for the 2-Stage Exponential (2EQ); 2-Stage Exponential with Lag (2EL); and QCM-4 models. During the growth phase, the 2EL model agrees well with the QCM-4 model, but during the death phase, the 2EQ model shows better agreement with the QCM-4 model than the 2EL model.

Table 6-7 shows the estimated model parameters and standard errors for the output of the three models shown in Figure 6-2. From Figure 6-2, the transition from growth to death in the QCM-4 model occurs at 8 days, which is halfway between the estimates for the 2EQ model (8.62 days) and the 2EL model (7.33 days). The lag rates for the QCM-4 and 2EL models agree to within 15%, and the initial net growth rate for the 2EL model (a = 1.10) agrees with the net growth rate for the QCM-4 model (k_2-k_4 = 1.22) to within 10%. With the exception of the QCM-4 lag rate, the standard errors of the estimated parameters for the 2EQ and 2EL models are significantly smaller than the standard errors for the QCM-4 model.

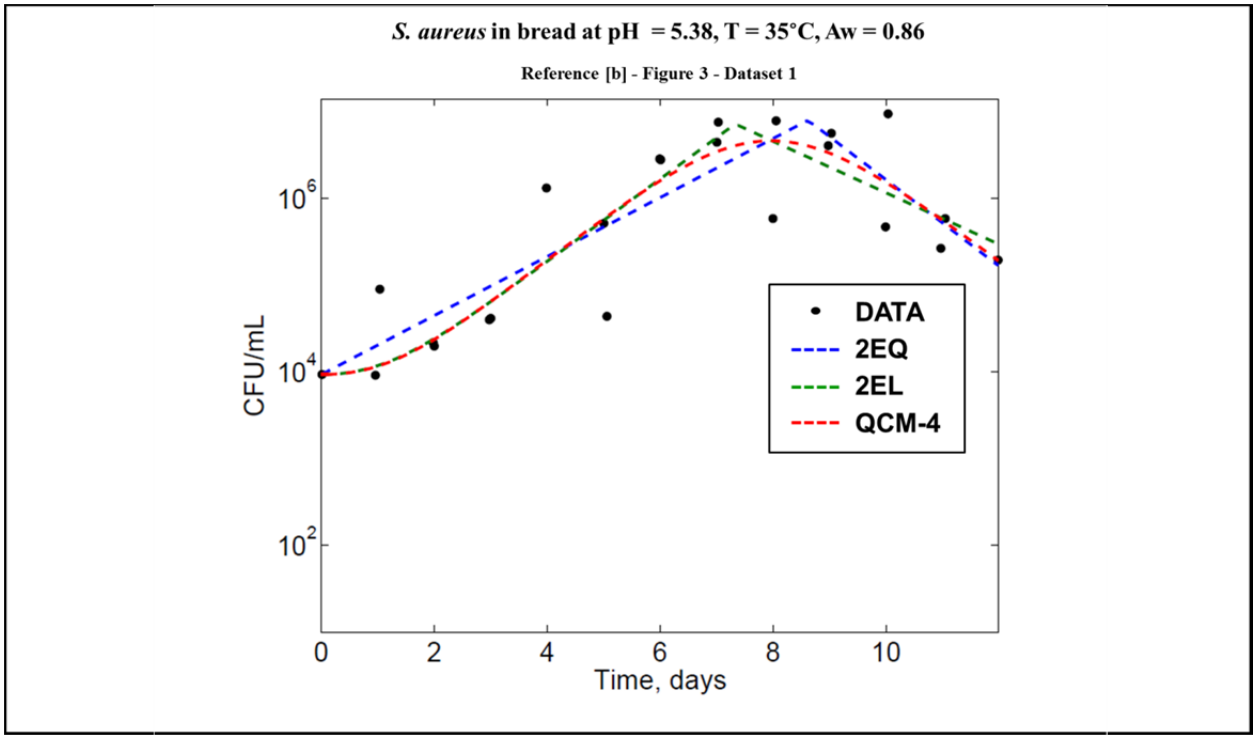


Figure 6-2. Example of Alternate Model Fits to Data with Primarily Growth Kinetics

Table 6-7. Alternate and QCM-4 Model Fits and Standard Errors for Dataset 1 of Figure 3 in Reference [b]

2-Stage Exponential Model		
q	**Estimate**	**Standard Error**
a	0.78	0.06
b	1.15	0.44
τ	8.62	2.78
2-Stage Exponential Model With Lag		
q	**Estimate**	**Standard Error**
k_1	0.36	0.54
a	1.10	0.25
b	0.68	0.31
τ	7.33	0.62
Quasi-Chemical Model		
κ	**Estimate**	**Standard Error**
k_1	0.31	0.50
k_2	42.23	1.37×10^3
k_3	3.39	1.13×10^2
k_4	41.08	1.38×10^3

Table 6-8 shows the AIC analysis results for the model output in Figure 6-2. In Table 6-8, the minimum value for AIC_c occurs for the 2EX model with a 6% increase in the value of the cost function.

Table 6-8. AIC Analysis Values for Dataset 1 of Figure 3, Reference [b]

Model	n	N_κ	$J(\kappa)$	AIC_c
2-Stage Exponential	22	3	5.40	**-20.53**
2-Stage Exponential with Lag	22	4	5.25	-17.77
Quasi-Chemical	22	4	5.10	-18.43

Growth and Death Kinetics. In 18% of the datasets in Appendix A, the data show significant growth and death phases. In many of these datasets there is a rapid transition from the lag phase to the growth phase which makes accurate estimation of the model parameters that shape the initial lag and growth phases (k_1 and α) difficult. To address this issue, the 1-stage exponential model with lag, which is a subset of Formulation C of the QCM-4 model, will be fit to the data before the growth of the bacteria slows and eventually enters the death phase. This gives estimates for k_1 and a in the 1EL model. The parameter a in the 1EL model coincides with α in Formulation C of the QCM-4 model. The two parameters k_1 and α are then fixed in Formulation C, and the QCM-4 model is solved for only k_2 and k_3 over the entire dataset. This process of fitting the data is referred to as the hybrid quasi-chemical model.

The question remains as to how to choose which data points will be included in the fitting for the 1EL model. One possibility is to view the data, and estimate when the data transitions from the growth to the death phase. A second method, which can be automated, is to start with a nominal number of the measured data points $n_0 > 2$. The 1EL model is fit to these data points and the standard errors are computed. Then an additional data point is included and the exponential model is fit to the $n_0 + 1$ data points and the standard errors are again computed. If the norm of the standard errors is larger when using the first $n_0 + 1$ data points compared to the first n_0 data points, then the estimates found using the first n_0 data points will be used. If the standard errors have decreased, an additional data point is considered so that now there are $n_0 + 2$ data points. This process is repeated until the standard errors stop decreasing.

Figure 6-3 shows the data from Dataset 4 of Figure 1 in Reference [b] as well as the fits to the data for the 2EQ, 2EL, QCM-4, and hybrid QCM-4 models. All alternate models show good agreement with the QCM-4 model in the figure.

Table 6-9 shows the estimated model parameters and standard errors for the output of the four models shown in Figure 6-3. The QCM-4 model estimated value for the lag rate k_1 is large indicating that the lag phase is short, and the associated standard error is several orders of magnitude larger than the standard error for any other estimated parameter from any of the models. The 1EL model estimates a moderate value for the lag rate, which is then used in the hybrid QCM-4 model. For k_1, k_2, and k_3, the standard errors for the hybrid model are less than the standard errors for the QCM-4 model.

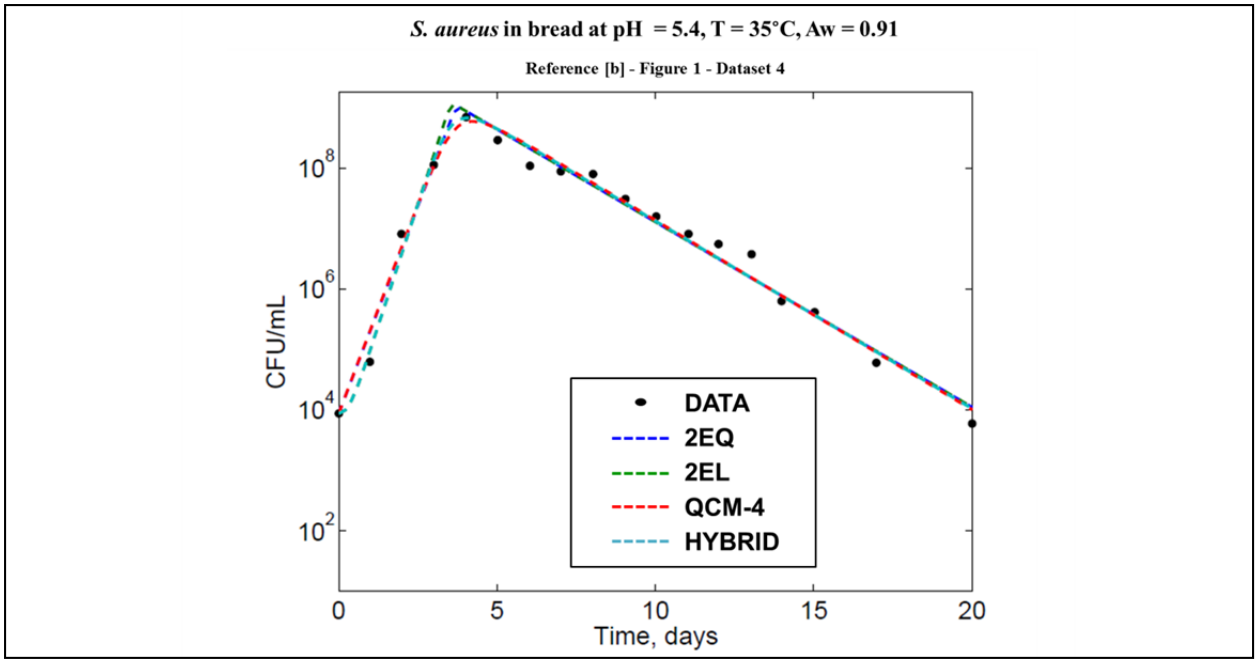


Figure 6-3. Example of Alternate Model Fits to Data with Death and Growth Kinetics

Table 6-9. Alternate and QCM-4 Model Fits and Standard Errors for Dataset 4 of Figure 1, Reference [b]

2-Stage Exponential Model		
q	**Estimate**	**Standard Error**
a	3.16	0.15
b	0.71	0.03
τ	3.71	0.25
2-Stage Exponential Model With Lag		
q	**Estimate**	**Standard Error**
k_1	1.43	1.51
a	3.70	0.38
b	0.71	0.03
τ	3.54	0.16
Quasi-Chemical Model		
κ	**Estimate**	**Standard Error**
k_1	1.71×10^3	6.93×10^5
k_2	3.94	0.36
k_3	3.31	0.79
k_4	0.74	0.04
Hybrid Quasi-Chemical Model		
κ	**Estimate**	**Standard Error**
k_1	1.43	1.98
k_2	4.43	0.03
k_3	3.55	0.71
α	3.70	0.50

Table 6-10 shows the AIC analysis results for the model output in Figure 6-3. In Table 6-10, the minimum value for AIC_c occurs for the 2EX model at a cost of a 4% increase in the value of the cost function. For the hybrid model, the value of $J(\boldsymbol{\kappa})$ is 0.66. If the hybrid model is considered a four parameter model, then the value of AIC_c for the hybrid model is -41.13, which would make it the best approximating model for this dataset with a reduction in the cost function of more than 18%.

Table 6-10. AIC Analysis Values for Dataset 4 of Figure 1, Reference [b]

Model	n	$N_{\boldsymbol{\kappa}}$	$J(\boldsymbol{\kappa})$	AIC_c
2-Stage Exponential	17	3	0.84	**-39.72**
2-Stage Exponential with Lag	17	4	0.71	-38.62
Quasi-Chemical	17	4	0.81	-36.26

6.1.3 Optimal Sampling Strategies

In this section we investigate optimal sampling strategies for data collection. Suppose the total data samples to be collected is n, we want to find the optimal sampling times t_j, $j = 1, ..., n$. We assume that the optimal sampling times will be those times which produce the most accurate parameter estimates with the highest degree of confidence. Standard optimal design criteria involve the use of the Fisher information matrix (FIM) which is defined as

$$\mathcal{F}(\mathbf{t};\boldsymbol{\kappa}) = \mathbf{F}^{\mathrm{T}}(\mathbf{t};\boldsymbol{\kappa})\mathbf{F}(\mathbf{t};\boldsymbol{\kappa}), \tag{6-1}$$

where $\mathbf{F}(\mathbf{t};\boldsymbol{\kappa})$ is the $n \times p$ sensitivity matrix given by Equation (2-10), and $\mathbf{t} = \{t_j\}_{j=1}^{n}$. There are many different design criteria one can use to attempt to maximize the information content in a dataset (References [c], [y], [z], and [aa]). Using the D-optimal design criteria, the goal is to determine the optimal sampling times $\mathbf{t}$ such that

$$\min_{\mathbf{t}\in \mathrm{T}} \det\left(\mathcal{F}(\mathbf{t};\boldsymbol{\kappa})^{-1}\right), \tag{6-2}$$

where $\mathrm{T} \subset [0,T]$ is the admissible set of sampling times. For an E-optimal design criteria, we wish to maximize the minimum eigenvalue of the FIM, that is, we solve

$$\max_{\mathbf{t}\in \mathrm{T}} \{\lambda_{\min}(\mathbf{t};\boldsymbol{\kappa})\}, \tag{6-3}$$

where $\lambda_{\min}$ is the minimum eigenvalue of $\mathcal{F}(\mathbf{t};\boldsymbol{\kappa})$. Each of these design criteria is local by nature since the results depends explicitly upon the value of the parameter vector $\boldsymbol{\kappa}$. However, before the experiment is performed we do not have an estimate for $\boldsymbol{\kappa}$. For this reason, we wish to develop an adaptive procedure for determining the optimal sampling times. The general procedure can be described in the following way. Start the experiment using a predetermined sampling strategy (e.g. linearly spaced sampling). After a prescribed number of data points n_s

have been collected, run the inverse problem to estimate the parameters $\boldsymbol{\kappa}$. Then use this estimate $\hat{\mathbf{t}}$ in the optimal design problem to determine the optimal sampling strategy for the remaining number of data points to be collected. Algorithm 2 shows how this process can be iterated each time a new data point becomes available.

Algorithm 2: Adaptive Optimal Sampling

Initialize: Set $n_s < n$; define initial sampling times t_j, $j = 1,...,n$.

Estimate $\hat{\boldsymbol{\kappa}}$ given the data (t_j,y_j), $j = 1,...,n_s$.

for $m = n_s+1$ to n do

1. Find the optimal sampling times t_k, $k = m,...,n$ using the desired design criteria given $\hat{\boldsymbol{\kappa}}$.
2. Collect a new data point at the time $t = t_m$.
3. Update the estimate given the data points (t_j,y_j), $j = 1,...,m$.

end

The following restrictions can easily be incorporated into Algorithm 2: allotting a minimum time between samples dt and/or allotting a maximum time between samples ds. In practice, the minimum time between samples is dictated by the restrictions of the data collection process. Specifying a maximum time between samples will be necessary in order to not wait too long in between collecting data, and potentially missing important dynamics altogether.

Example. To examine the feasibility of Algorithm 2, we test the algorithm on a variety of different scenarios. The data for the example scenarios was generated by

$$y_j = U(t_j;\boldsymbol{\kappa}_0) + \varepsilon_j, \quad j = 1,...,n, \tag{6-4}$$

where ε_j is a realization of a normally distributed random variable with mean zero and variance σ_0^2. The estimate $\hat{\boldsymbol{\kappa}}$ is found by solving

$$\hat{\boldsymbol{\kappa}} = \arg\min_{\boldsymbol{\kappa}\in\Omega} \sum_{j=1}^{n} \left(\log_{10}(y_j) - \log_{10}(U(t_j;\boldsymbol{\kappa}))\right)^2. \tag{6-5}$$

We compare the estimated value of the unknown parameters and the associated standard errors obtained using Algorithm 2 with the values computed using an equally spaced sampling strategy. Since the estimates and standard errors will naturally depend upon the realizations of the random variable ε_j, we will perform each problem N_O times and report the average results.

We first consider the case where only the parameter k_3 is unknown, that is, $\boldsymbol{\kappa} = k_3$. The remaining parameters are taken to be known fixed values. The initial starting point for estimation of $\boldsymbol{\kappa}$ was taken to be $0.8\boldsymbol{\kappa}_0$ and the lower and upper bounds were set as 10^{-3} and 10^3, respectively. For the true value of the unknown parameter $\boldsymbol{\kappa} = k_3$ we consider $k_3 = 1.0$ and $k_3 = 10.0$. For the case of

$k_3 = 1.0$, the maximum sampling interval was set to be $T = 14$, and for the case of $k_3 = 10.0$ the maximum sampling interval was set to $T = 10$. The reason for shorting the interval in the second case is to ensure that the data remained non-negative after we add noise. For the following results the number of sampling points used before beginning the adaptive procedure was set to be $n_s = 2$. We consider the case where the total number of samples is $n = 4$, 5 and 6, and the results are given in Tables 6-12 through 6-14. All of the auxiliary parameters used in the implementation are given in Table 6-11.

Table 6-11. Values of Auxiliary Parameters Used in the Example

Parameter	k_1	k_2	α	I	σ_0^2	dt	ds	N_O
Value	1.0	4.0	2.0	10^4	2×10^{10}	0.01	5.0	10^3

The reason for considering so few total number of samples is that we are only estimating one parameter, and if we take n large enough then a linear sampling scheme will perform with the same efficiency as using an optimal sampling scheme. When only using $n = 4$ total samples, we see that the standard error decreases in each case. With $n = 5$ and 6, the standard error decreases for each case except when using the E-optimal criteria with the true value of $k_3 = 10.0$. In general, as n increases from 4 to 6, the difference in the standard error found using the optimal and linear sampling schemes decreases.

Table 6-12. Results of Algorithm 2 for $n_s = 2$ and $n = 4$

	D-optimal		**E-optimal**		κ_0
	Optimal	**Linear**	**Optimal**	**Linear**	
k_3	1.015	1.050	1.020	1.024	1.00
SE	3.494×10^{-2}	8.942×10^{-2}	5.600×10^{-2}	7.771×10^{-2}	
k_3	10.029	10.031	10.035	10.050	10.00
SE	0.249	0.335	0.259	0.327	

Table 6-13. Results of Algorithm 2 for $n_s = 2$ and $n = 5$

	D-optimal		**E-optimal**		κ_0
	Optimal	**Linear**	**Optimal**	**Linear**	
k_3	1.019	1.026	1.021	1.021	1.00
SE	5.239×10^{-2}	6.677×10^{-2}	6.010×10^{-2}	6.813×10^{-2}	
k_3	10.039	10.026	10.017	10.013	10.00
SE	0.610	0.648	0.962	0.635	

Table 6-14. Results of Algorithm 2 for $n_s = 2$ and $n = 6$

	D-optimal		E-optimal		κ_0
	Optimal	Linear	Optimal	Linear	
k_3	1.020	1.023	1.014	1.019	1.00
SE	5.646×10^{-2}	6.199×10^{-2}	5.555×10^{-2}	6.212×10^{-2}	
k_3	10.047	10.033	10.029	10.041	10.00
SE	1.038	1.144	1.232	1.135	

In Figure 6-4 we show the normalized sensitivities of all of the model parameters, the estimated parameter k_3 and the fixed parameters k_1, k_2 and α. From here we get a further indication of why we do not obtain a larger difference in the standard errors computed using the adaptive optimal sampling strategy compared to a lineally spaced sampling scheme. The unknown parameter k_3 has a near zero sensitivity until approximately $t = 5$. At approximately $t = 10$, the sensitivity has reached a steady state at the maximum. Thus, any sampling strategy which uses the same number of points over the interval [10, 14] should be expected to yield similar standard errors.

This example suggest possible limitations of this adaptive sampling strategy. A number of initial data points must be collected to first use in the inverse problem; however, if a data set does not have sensitivity to all of the unknown model parameters in the initial sampling interval, then no useful information can be obtained about those parameters. This may result in poor initial estimates for these parameters. In this case, these poor estimates may in turn lead to suboptimal sampling strategies since the next sampling time is chosen according to an inaccurate estimate.

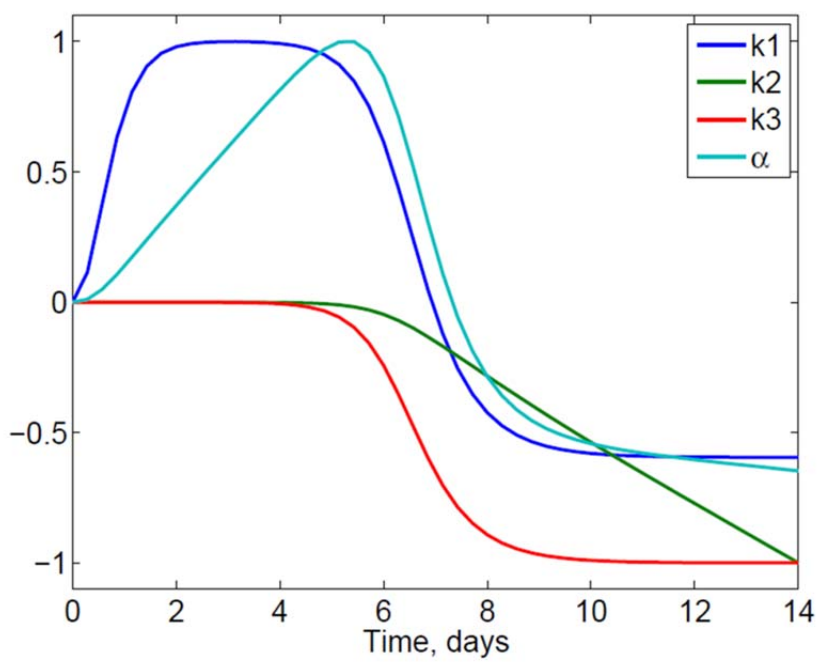


Figure 6-4. Normalized Sensitivities of Model Parameters for Example with $\kappa = 1.00$, $n_s = 2$, and $n = 4$

6.1.4 Global Optimization

Optimization methods such as Nelder-Mead and interior-point start with an initial estimate of the model parameters being solved for and then move through the parameter space in the direction of a set of parameters that either minimize or maximize a cost function. Since these solvers are optimizing within a neighborhood of the initial estimate of the parameters, they are called local solvers and only find local minima or maxima.

For the QCM-4 model, the optimization problem was to minimize the SSR between the observations and the model output for the Frequentist approach, or minimize the Kullback-Leibler divergence between the estimated true posterior probability distributions and the possible families of Gaussian distributions for the Bayesian approach. In analyzing the QCM-4 model, it was found that the kinetics of the observations resulted in a parameter space with many local minima, and the choice of the initial parameter estimates greatly affected the model solution.

Future work on models like the QCM-4 model, particularly for automated use, should examine techniques for global optimization. Three techniques to consider are the Multistart Method, Simulated Annealing, and Machine Learning.

Multistart Method – The basic idea behind the Multistart Method (Reference [f]) is to divide up the parameter space, choose an initial parameter estimate within each subspace, and locally optimize within each subspace. The solution to the subspace with the minimum cost function value is taken as the optimal solution. This approach requires setting lower and upper bounds on each parameter so that the parameter space is finite, and developing a method for dividing up the parameter space so that it is unlikely any subspace has multiple minima.

Simulated Annealing – Whereas the Multistart Method is a dense deterministic search over the parameter space, Simulated Annealing (Reference [f]) involves a random walk through the parameter space. As with a local solver, Simulated Annealing moves through the parameter space towards a local minimum, but at random times, will move away from the minimum using a statistical distribution from thermal physics. This process, which simulates the annealing process of metals, gives the algorithm the ability to move out of the neighborhood of a local minimum in the hope of finding a global minimum. This approach requires setting an annealing schedule which determines the randomness of the search and the likelihood that a global minimum is found.

Machine Learning – Machine learning is a branch of computer science that automates analytical model building through the development of algorithms that iteratively learn from data (Reference [bb]). In a machine learning approach, measured and simulated datasets covering a broad range of bacterial growth kinetics are obtained or produced, and fit using the QCM-4 model by an analyst. The data, initial parameter estimates, solution parameters, parameter bounds, algorithm parameters (i.e. ODE solver tolerances), and model outputs are collected and stored in what is referred to as a training set. When a new dataset is to be fit, machine learning algorithms compare the new measured data with the data in the training sets to retrieve the initial parameter estimates that are likely to find a global minimum.

6.2 ALTERNATE BAYESIAN METHODS FOR SOLVING THE INVERSE PROBLEM

6.2.1 Inversion Using a Mixture of Gaussians

The Bayesian results in Section 4.3 were based on solving the inverse problem through the minimization in Equation (3-53) for the case of $L = 1$. That is, each parameter being solved for is assumed to be from a Gaussian distribution and the mean and variance of that distribution is estimated for each parameter. The uncertainty in the parameter estimates could be reduced by assuming that each parameter is from a mixture of $L > 1$ Gaussian distributions.

Figure 6-5 shows results of using a single Gaussian ($L = 1$) and mixture of Gaussians ($L = 2,3,4$) for Dataset 4 of Figure 1 in Reference [b]. For each value of L, Figure 6-5 shows the marginal and joint density functions of the approximating posterior distributions for the parameters $[\xi_1, \xi_2, \xi_3, \xi_4]^T$. As the number of Gaussian components increases, the structure of the probability distributions increases. Table 6-15 shows the value of the ELBO for the four different Gaussian mixtures. The largest ELBO value occurs when L is equal to 4 which indicates that a mixture of four Gaussians is the preferable model for this dataset.

Table 6-15. Value of ELBO for Gaussian Mixtures

Number of Gaussians in Mixture (L)	ELBO ($\mathcal{F}[q]$)
1	-11.90
2	-12.11
3	-11.19
4	-9.87

An area for further investigation using this method is the stability of the solutions with regard to the number of parameters being estimated relative to the number of observations. The Bayesian variational method estimates two values (i.e. mean and variance) for each model parameter. The number of model parameters, N_κ, for the Bayesian inversion of the QCM-4 model is five. The total number of parameters being estimated is then $10L$. For a mixture of four Gaussians, 40 parameters are being estimated from only 18 data points in Dataset 4 of Figure 1 in Reference [b].

6.2.2 Stochastic Differential Equation

The SDE approach to the inverse problem for the QCM-4 model can solved using Algorithm 1 in Section 2.2.5. The initial values for the model parameters, $\hat{\xi}$, are estimated by solving the deterministic QCM-4 model using ordinary nonlinear least squares and assuming they are normally distributed. Algorithm 1 is then run 3000 times to improve the statistical estimates of the parameters.

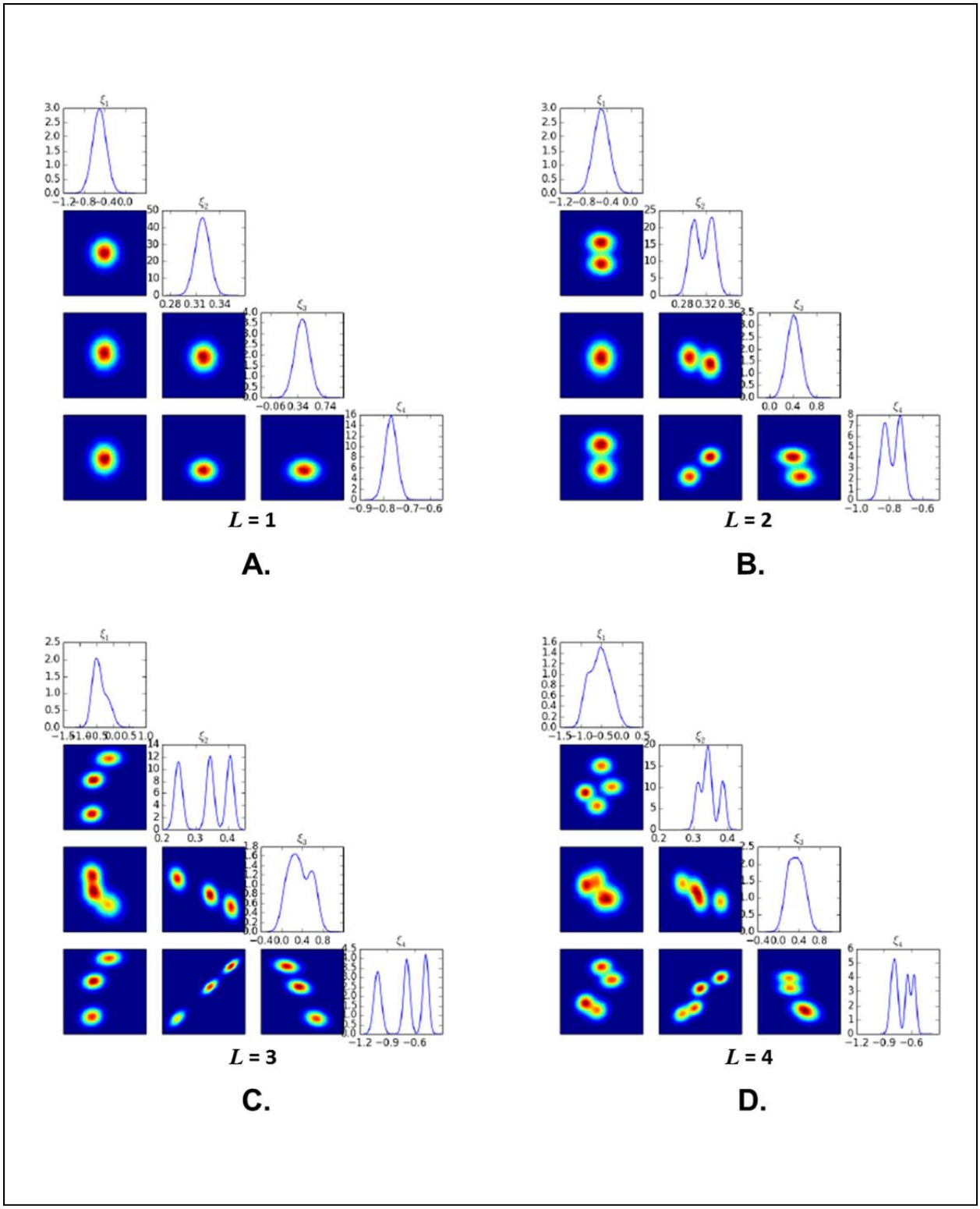


Figure 6-5. Bayesian Inverse Problem Using Gaussian Mixtures

Figure 6-6 shows the evolution of the model parameters for Dataset 4 of Figure 1 in Reference [b]. The mean values of the ξ_i stabilize after about 500 samples are drawn which indicates that no more that the first 500 need to be discarded (i.e. burn-in period). Figure 6-7 shows the prior and posterior probability distribution functions for the model parameters. Although there is little difference in the means, the variance of ξ_2, ξ_3, and ξ_4 has decreased which reduces the uncertainty in the parameter estimates. Figure 6-8 shows the autocorrelations of the model parameters which show a satisfactory level of decorrelation in the constructed chain. Figure 6-9 shows the mean of the model output along with the 95% confidence bands based on the MCMC samples. With the exception of data points near times 0 or 20 days, the model output fits the data. In particular, the confidence level near data points usually decreases since the possible realized paths should be near these points. In the regions between data points, the uncertainty increases since no information is available about these time periods through the data.

6.3 AGGREGATE DATA

Potential issues can occur with mathematical modeling and estimation in connection with certain types of experiments such as bacteria kinetics experiments. Consider an experiment in which independent samples of inoculated food are prepared and incubated. At each sample time, a sample is extracted and the bacteria are recovered and plate counted. If the sample is discarded data is called *aggregate data* which is similar to the type of data one would obtain in ecological catch and release experiments (Chapter 5 of Reference [c]). Repeated measurements of the same sample over a period of time would result in *individual data*. Aggregate data is often treated as individual data, and modeling and estimation techniques derived for the purpose of data assimilation with individual data are applied. When this is done, the effects of variability across samples is ignored, which can lead to over confidence in estimation results and overly conservative predictions. Techniques designed to handle aggregate data have been developed (Reference [c]), but require sufficient replicates at each sampling time to be collected which can greatly increase the cost of an experiment. Examples of the pitfalls which can occur when aggregate data is treated as individual data, and a more detailed discussion on aggregate data approaches applied in a pharmacokinetic setting can be found in Reference [cc]. We briefly discuss potential problems that may arise when utilizing an individual data-based mathematical modeling framework to describe an aggregate data experiment by considering the following hypothetical scenario.

Suppose we have an experiment involving two different strains of a microorganism with different growth rates (α). In Figure 6-10 we present two simulations of the model, in one strain $\alpha = 2.7$ (green circles) and in the other strain $\alpha = 3.7$ (blue dots). In this depiction we have created data which matches our hypothetical scenario.

Assume that there are microorganisms from several different strains, and that any sample is randomly selected from this data. In Figure 6-11 we depict how a data set collected in this manner might appear. To construct this data set at each time point we randomly select a data point from one of the two trajectories shown in Figure 6-10. If we look at the data set given in

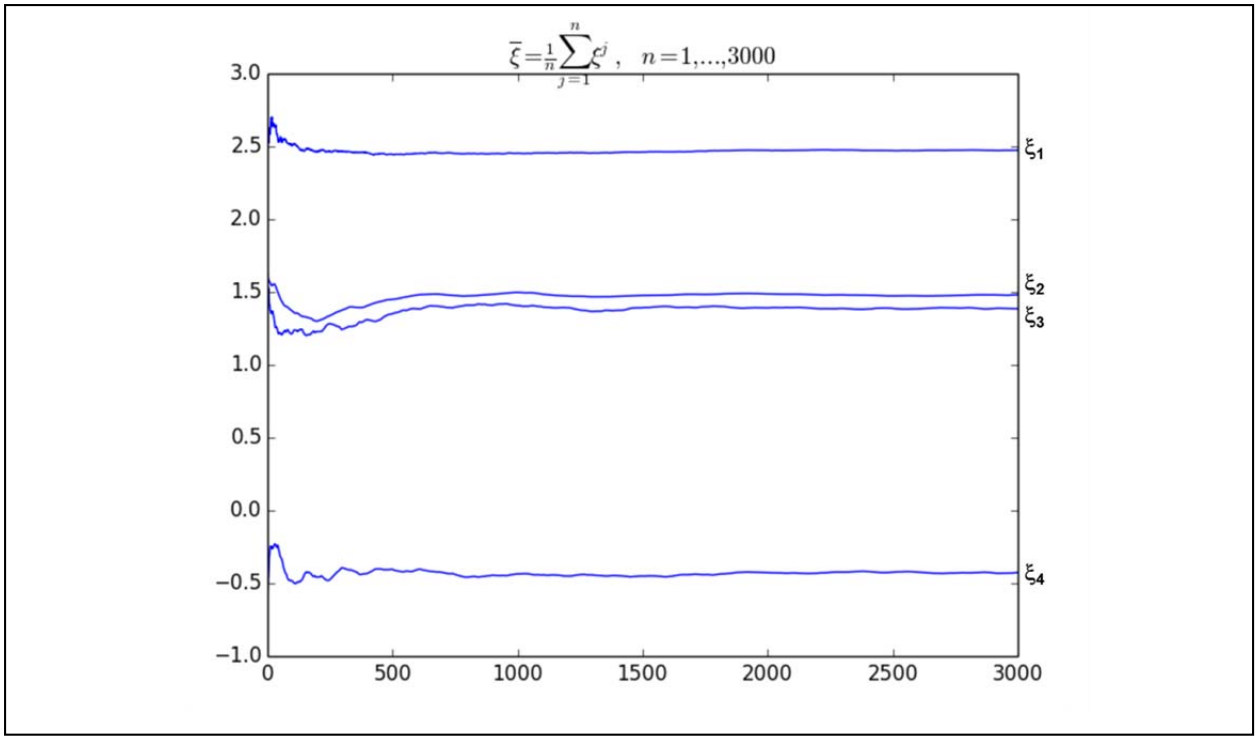


Figure 6-6. Value of Model Parameters from MCMC Algorithm Runs

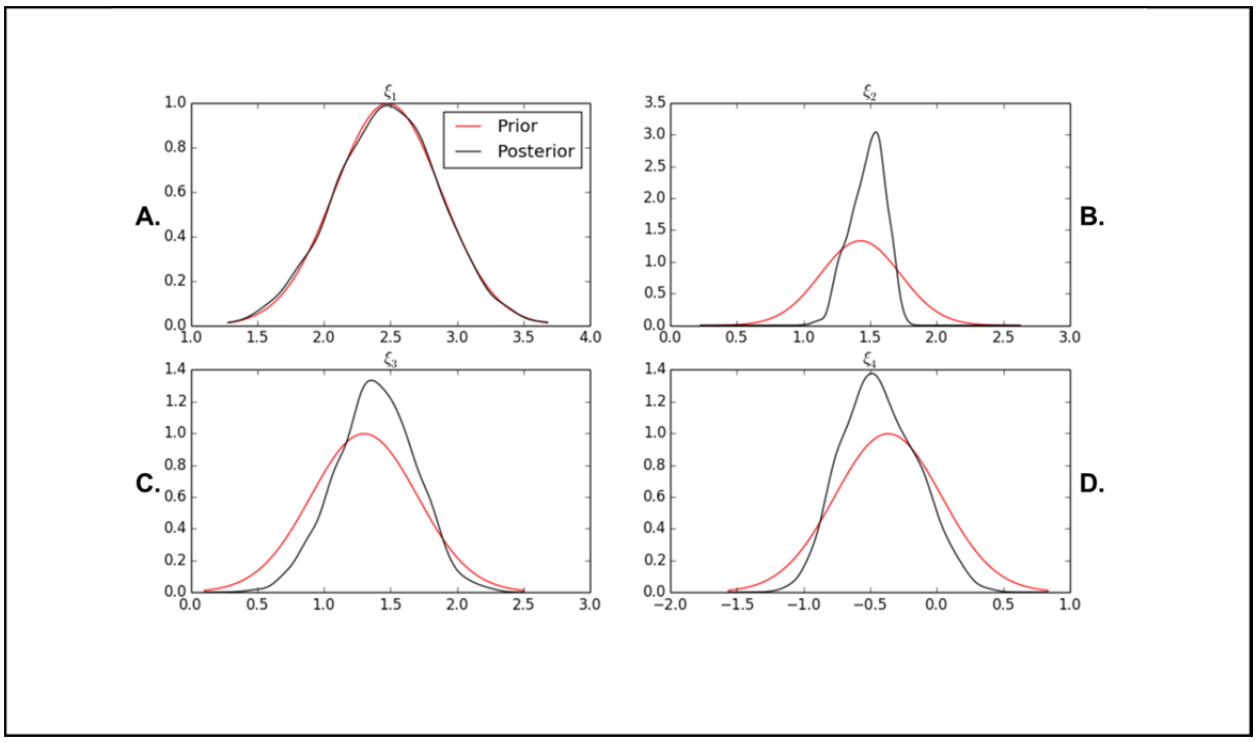


Figure 6-7. Prior and Posterior Distributions for A. ξ_1, B. ξ_2, C. ξ_3, and D. ξ_4

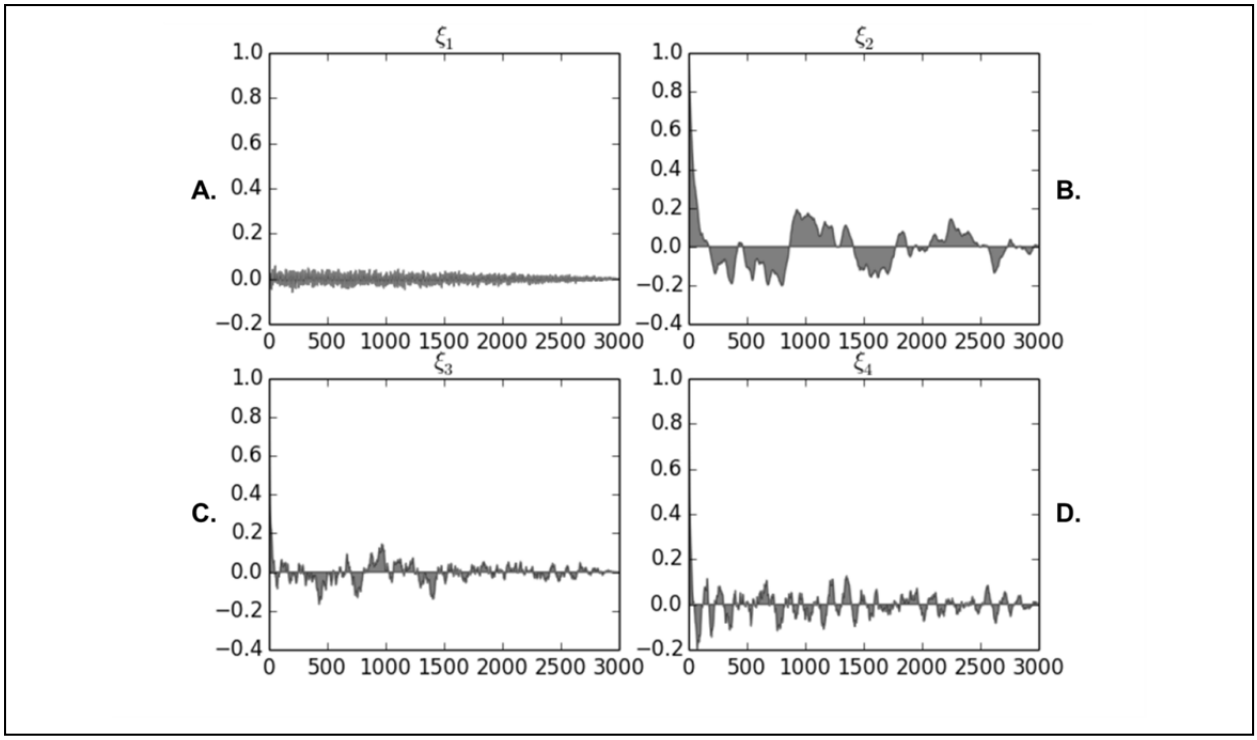


Figure 6-8. MCMC Algorithm Autocorrelations for A. ξ_1, B. ξ_2, C. ξ_3, and D. ξ_4

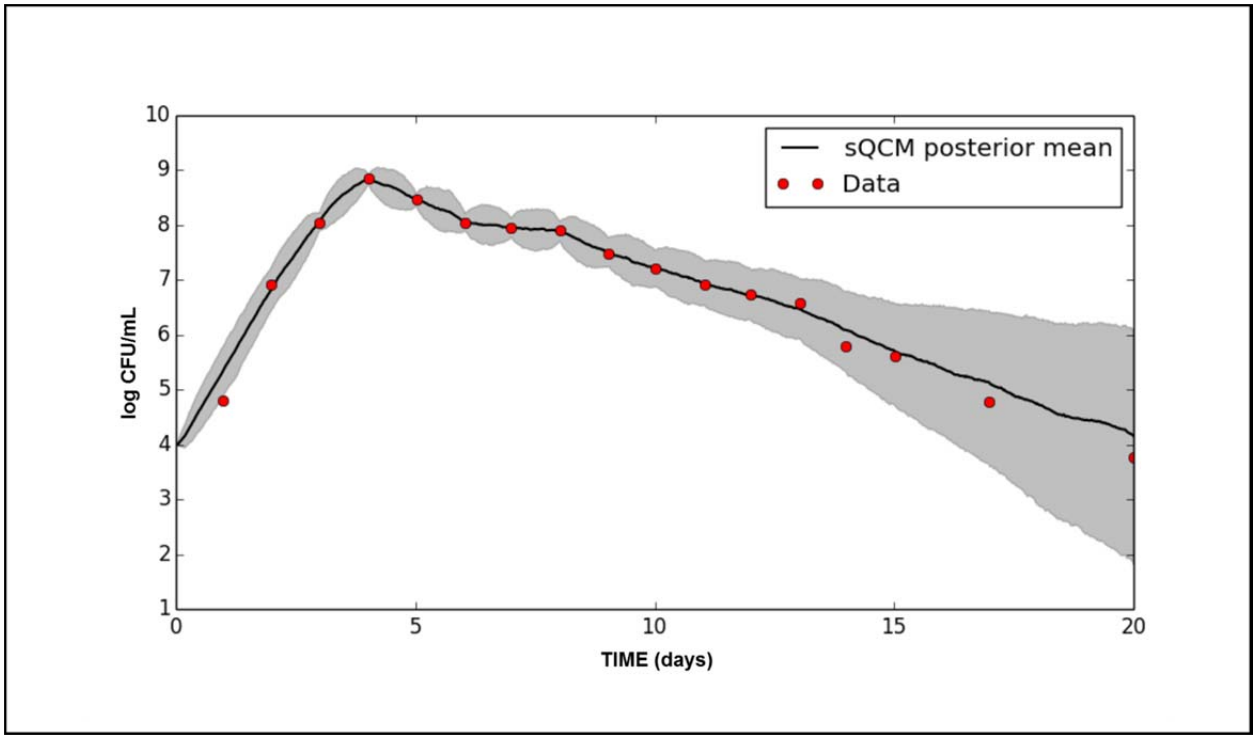


Figure 6-9. Model Output and Uncertainty **for SDE Method**

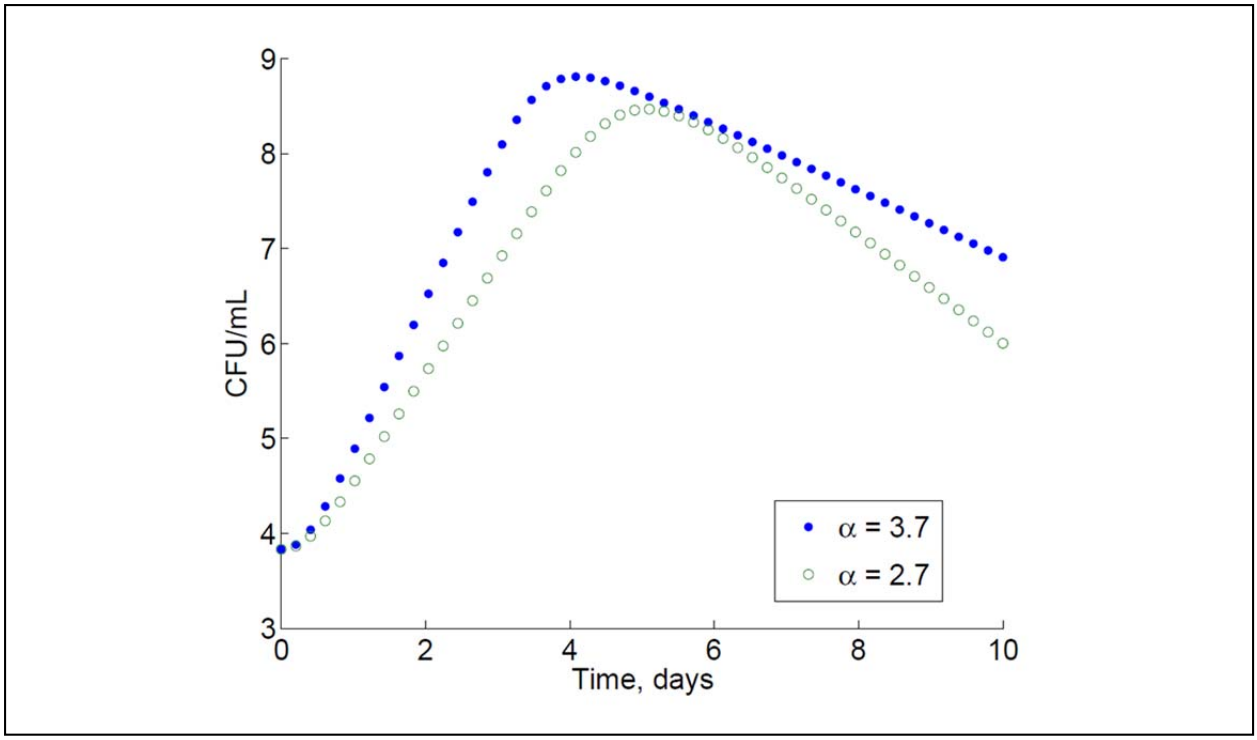


Figure 6-10. Simulated Growth Data with α = 3.7 (blue dots) and α = 2.7 (green circles)

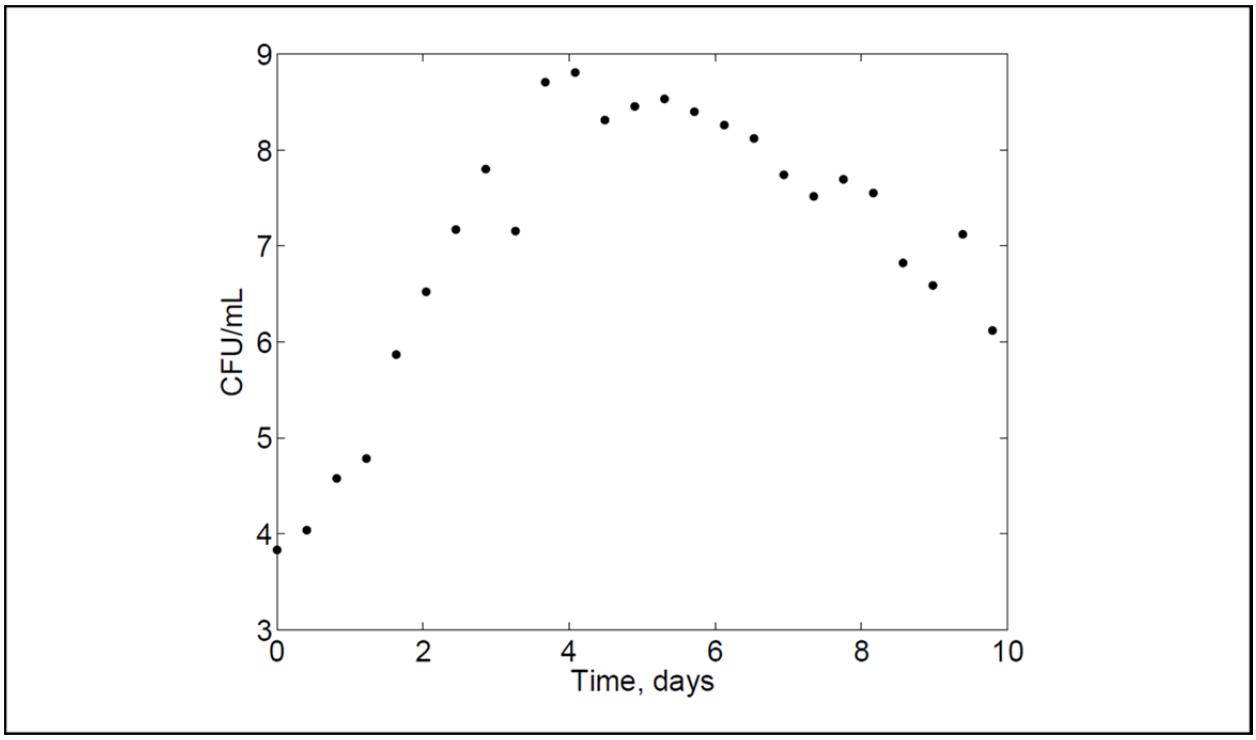


Figure 6-11. Dataset Produced by Randomly Sampling the Two Trajectories in Figure 6-9

Figure 6-10, it appears to resemble a noisy data set representing a population with constant growth rate. Therefore, if we use this data set to estimate α, we will estimate a value for α somewhere roughly midway between $\alpha = 2.7$ and 3.7 (the two values used to produce the data set).

Ideally, what we would like to do is be able to use the data shown in Figure 6-10 to infer that the samples contain microorganisms that grow at two distinctly different rates. In this way, we can consider alpha as an individual based parameter which has an associated distribution. If we impose a distribution on the model parameter α to obtain a model of the form

$$v(t;G) = \int_{\Omega} g(t;\alpha)\, \mathrm{d}G(\alpha), \tag{6-6}$$

we will *not* be able to determine that two growth rates produced the data regardless of the assumptions we place on the distribution $G(\alpha)$. Even if we assume G is a discrete distribution such that the $\mathrm{Prob}\{G(\alpha) = 2.7\} = \mathrm{Prob}\{G(\alpha) = 3.7\} = 0.5$, the resulting model $\nu(t;G)$ will only exhibit the *average* behavior. In order to gain more insight into the underlying distribution $G(\alpha)$ we would need to acquire replicate data at each sampling time where the number of replicates counted at each sampling time is sufficient to identify the distribution G (Reference [cc]).

REFERENCES

a. Ross, E.W., et al., "The mathematical properties of the quasi-chemical model for microorganism growth-death kinetics in foods," *International Journal of Food Microbiology*, Volume 99, pps. 157-171, 2005.

b. Doona, C.J., F.E. Feeherry, and E.W. Ross, "A quasi-chemical model for the growth and death of microorganisms in foods by non-thermal and high-pressure processing," *International Journal of Food Microbiology*, Volume 100, pps. 21-32, 2005.

c. Banks, H.T., S. Hu, and W.C. Thompson, *Modeling and Inverse Problems in the Presence of Uncertainty*, CRC Press, Boca Raton, 2014.

d. Website, www.stat.ncsu.edu/people/davidian/courses/st762_fall2009 (as of 22 August 2016).

e. Davidian, M., and D. Giltinan, "Nonlinear models for repeated measurement data: An overview and update," *Journal of Agricultural, Biological, and Environmental Statistics*, Vol. 8, pps. 387-419, 2003.

f. Goulet, D., "Modeling, Simulating, and Parameter Fitting of Biochemical Kinetic Experiments," *SIAM Review*, Vol. 58, No. 2, pps. 331-353, 2016.

g. Robert, C., and G. Casella, *Monte Carlo Statistical Methods*, Springer Science & Business Media, New York, 2004.

h. Metropolis, N., et al., "Equation of State Calculations by Fast Computing Machines," *Journal of Chemical Physics*, Volume 21, pps. 1087-1091, 1953.

i. Hastings, W., "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika*, Volume 57, Number 1, pps. 97-109, 1970.

j. Haario, H., E. Saksman, and J. Tamminen, "An Adaptive Metropolis Algorithm," *Bernoulli*, Volume 7, Number 2, pps. 223-242, 2001.

k. Roberts, G.O., and J.S. Rosenthal, "Optimal scaling of discrete approximations to Langevin diffusions," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, Volume 60, Number 1, 1998.

l. Gershman, S.J., M.D. Hoffman, and D.M. Blei, "Nonparametric Variational Inference," *Proceedings of the 29th International Conference on Machine Learning*, 2012.

m. Tsilifis, P., Bilionis, I., Katsounaros, I. and Zabaras, N., 2016. Computationally efficient variational approximations for Bayesian inverse problems. *Journal of Verification, Validation and Uncertainty Quantification*, 1(3), p.031004.

n. Kullback, S., and R.W. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, Volume 22, Number 1, pps. 79-86, 1951.

o. Huber, M.F., et al., "On entropy approximation for Gaussian mixture random vectors," IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, 2008.

p. Øksendal, B., Stochastic Differential Equations, Springer, Berlin Heidelberg, 2003.

q. Tierney, L., “A Note on Metropolis-Hastings Kernels for General State Spaces,” *The Annals of Applied Probability*, Volume 8, Number 1, pps. 1-9, 1998.

r. Beskos, A., et al., “MCMC methods for diffusion bridges,” *Stochastics and Dynamics*, Volume 8, Number 3, pps. 319-350, 2008.

s. Cotter, S.L., et al., “MCMC Methods for Functions: Modifying Old Algorithms to Make Them Faster,” *Statistical Science*, Volume 28, Number 3, pps. 424-446, 2013.

t. Law, K.J.H., “Proposals which speed up function-space MCMC,” *Journal of Computational and Applied Mathematics*, Volume 262, pps. 127-138, 2014.

u. Peleg, M., and M.G. Corradini, “Microbial Growth Curves: What the Models Tell Us and What They Cannot,” *Critical Reviews in Food Science and Nutrition*, Vol. 51, Issue 10, pps. 917-945, 2011.

v. Sagues, F. and I.R. Epstein, “Nonlinear chemical dynamics,” *Dalton Transactions*, Issue 7, pps. 1201-1217, 2003.

w. Bishop, C., *Pattern Recognition and Machine Learning*, Springer-Verlag, New York, 2006.

x. Press, W.H., et al., *Numerical Recipes: The Art Of Scientific Programming*, Cambridge University Press, New York, 2007.

y. Atkinson, A.C., A.N. Donev, and R.D. Tobias, *Optimum Experimental Designs, with SAS, volume 34*, Oxford University Press, Oxford, 2007.

z. Fedorov, V.V., and P. Hackl, *Model-oriented Design of Experiments, Volume 125*, Springer Science & Business Media, New York, 2012.

aa. Kiefer, J.C., et al., *Jack Carl Kiefer Collected Papers: Design of Experiments*, Springer, New York, 1985.

bb. Website, www.sas.com/it_it/insights/analytics/machine-learning.html (as of 25 August 2016).

cc. Banks, H.T., et al., Parameter estimation using unidentified individual data in individual based models, Center for Research in Scientific Computation Technical Report CRSC-TR16-04, NC State University, May 2016.

dd. Website, www.mathworks.com/matlabcentral (as of 30 August 2016).

ee. Website, nickhigham.wordpress.com/2013/02/13/the-nearest-correlation-matrix/comment-page-1/ (as of 30 August 2016).

ff. Website, www.math.nus.edu.sg/~matsundf/ (as of 30 August 2016).

gg. Website, www.scipy.org (as of 30 August 2016).

hh. Website, github.com (as of 30 August 2016).

ii. AATCC Test Method 100-2004, Antibacterial Finishes on Textile Materials: Assessment of, AATCC Technical Manual/2007, pps. 145-147.

jj. Peeler, J.T., et al., “Replicate Counting Errors by Analysts and Bacterial Colony Counters,” *Journal of Food Protection*, Vol. 45, No. 3, pps. 238-240, February 1982.

APPENDIX A

MODEL DATASETS

Reference [b] describes the quasi-chemical model of bacterial growth in foods. Reference [b] contains plots of datasets collected from 10 experiments conducted at NSRDEC (Figures 1 through 10 of Reference [b]). Each of the 10 figures of Reference [b] showing measured data consists of two to six datasets of measured plate counts over time that show the growth of various bacteria in food-based hosts under different environmental conditions.

There are a total of 39 datasets in Figures 1 through 10 of Reference [b], which are summarized in Table A-1 in terms of the food product; the organism used to inoculate the food; and the experimental conditions. Figures A-1 through A-10 plot the measured data and QCM-4 model fit to the data in a format similar to that shown in Figures 1 through 10 of Reference [b]. The data measurements were digitized from Reference [b], and the model fits were calculated using Formulation B of the QCM-4 model.

Figures A-11 through A-49 show the forward uncertainty propagation algorithm applied to the 39 datasets. Each figure shows a plot of the measured data; a plot of the growth curve resulting from the mean rate parameters (i.e. thick dark line); plot of the randomly generated growth curves (i.e. thin light lines); a table with the mean rate parameters and standard errors, and a table of the measured data in terms of the measurement time and base 10 logarithm of the bacterial concentration.

Table A-1. Description of Datasets

Figure from Reference [b]	Dataset	Experiment
1	1	*S. aureus* in bread at pH = 5.4, T = 35°C, Aw = 0.79
1	2	*S. aureus* in bread at pH = 5.4, T = 35°C, Aw = 0.84
1	3	*S. aureus* in bread at pH = 5.4, T = 35°C, Aw = 0.87
1	4	*S. aureus* in bread at pH = 5.4, T = 35°C, Aw = 0.91
2	1	*S. aureus* in bread at pH = 5.23, T = 15°C, Aw = 0.90
2	2	*S. aureus* in bread at pH = 5.23, T = 20°C, Aw = 0.90
2	3	*S. aureus* in bread at pH = 5.23, T = 25°C, Aw = 0.90
2	4	*S. aureus* in bread at pH = 5.23, T = 30°C, Aw = 0.90
2	5	*S. aureus* in bread at pH = 5.23, T = 35°C, Aw = 0.90
2	6	*S. aureus* in bread at pH = 5.23, T = 40°C, Aw = 0.90
3	1	*S. aureus* in bread at pH = 5.38, T = 35°C, Aw = 0.86
3	2	*S. aureus* in bread at pH = 5.36, T = 35°C, Aw = 0.86
3	3	*S. aureus* in bread at pH = 5.31, T = 35°C, Aw = 0.86
3	4	*S. aureus* in bread at pH = 5.19, T = 35°C, Aw = 0.86
3	5	*S. aureus* in bread at pH = 4.97, T = 35°C, Aw = 0.86
4	1	*S. aureus* in turkey at pH = 6.35, T = 35°C, Aw = 0.89
4	2	*S. aureus* in turkey at pH = 6.35, T = 35°C, Aw = 0.87
4	3	*S. aureus* in turkey at pH = 6.35, T = 35°C, Aw = 0.86
4	4	*S. aureus* in turkey at pH = 6.35, T = 35°C, Aw = 0.84
5	1	*S. aureus* in ham at pH = 5.8, T = 35°C, Aw = 0.92
5	2	*S. aureus* in ham at pH = 5.8, T = 35°C, Aw = 0.90
5	3	*S. aureus* in ham at pH = 5.8, T = 35°C, Aw = 0.88
6	1	*S. aureus* in cheese at pH = 5.9, T = 35°C, Aw = 0.87
6	2	*S. aureus* in cheese at pH = 6.0, T = 35°C, Aw = 0.85
7	1	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.90
7	2	*Listeria* in turkey at pH = 6.4, T = 35°C, Aw = 0.94
7	3	*E. coli* in turkey at pH = 6.4, T = 35°C, Aw = 0.96
8	1	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.896, 0% anti-microbial lactate
8	2	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.896, 1% anti-microbial lactate
8	3	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.896, 2% anti-microbial lactate
8	4	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.896, 3% anti-microbial lactate
9	1	*E. coli* in whey protein at T = 35°C, P = 207 MPa
9	2	*E. coli* in whey protein at T = 35°C, P = 242 MPa
9	3	*E. coli* in whey protein at T = 35°C, P = 276 MPa
9	4	*E. coli* in whey protein at T = 35°C, P = 311 MPa
9	5	*E. coli* in whey protein at T = 35°C, P = 345 MPa
10	1	*E. coli* in whey protein at T = 30°C, P = 310 MPa
10	2	*E. coli* in whey protein at T = 40°C, P = 310 MPa
10	3	*E. coli* in whey protein at T = 50°C, P = 310 MPa

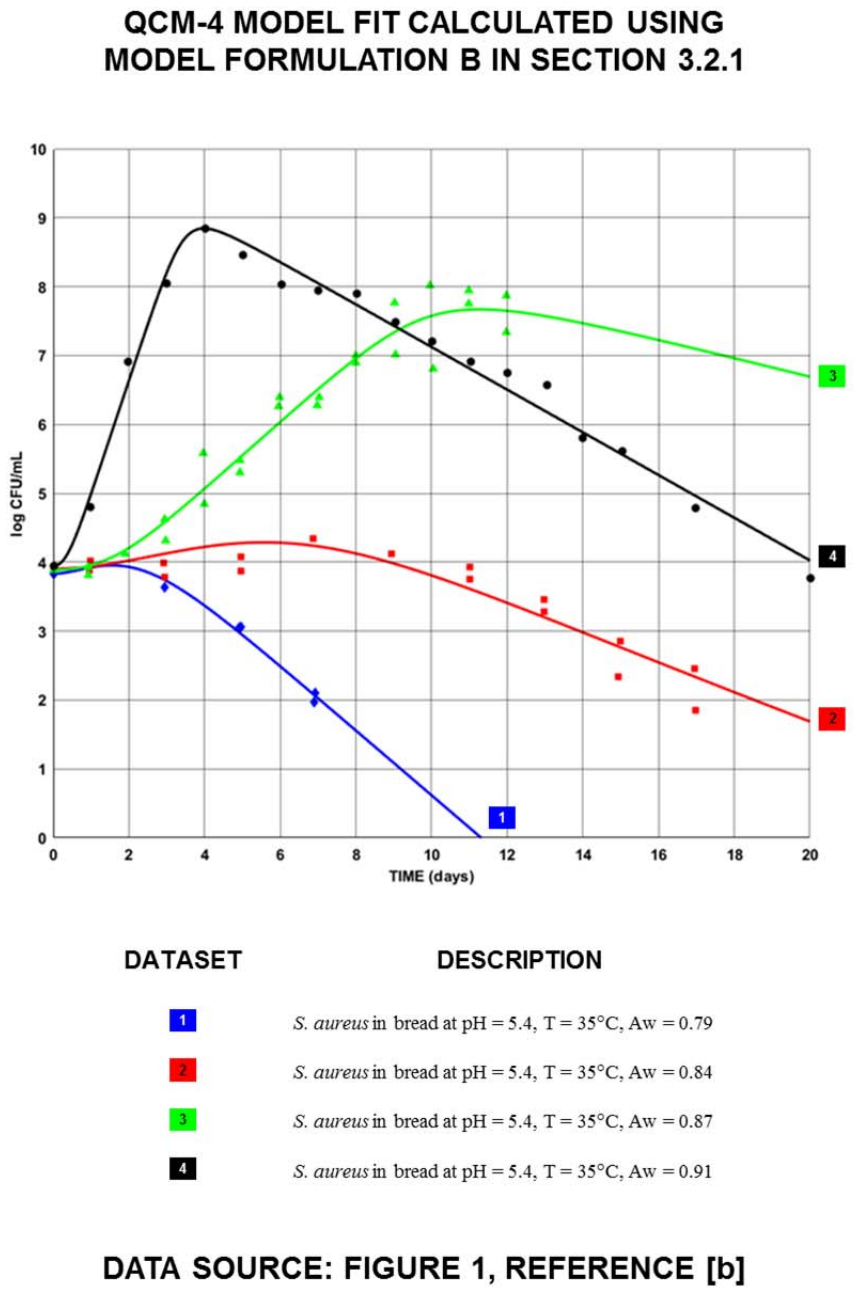
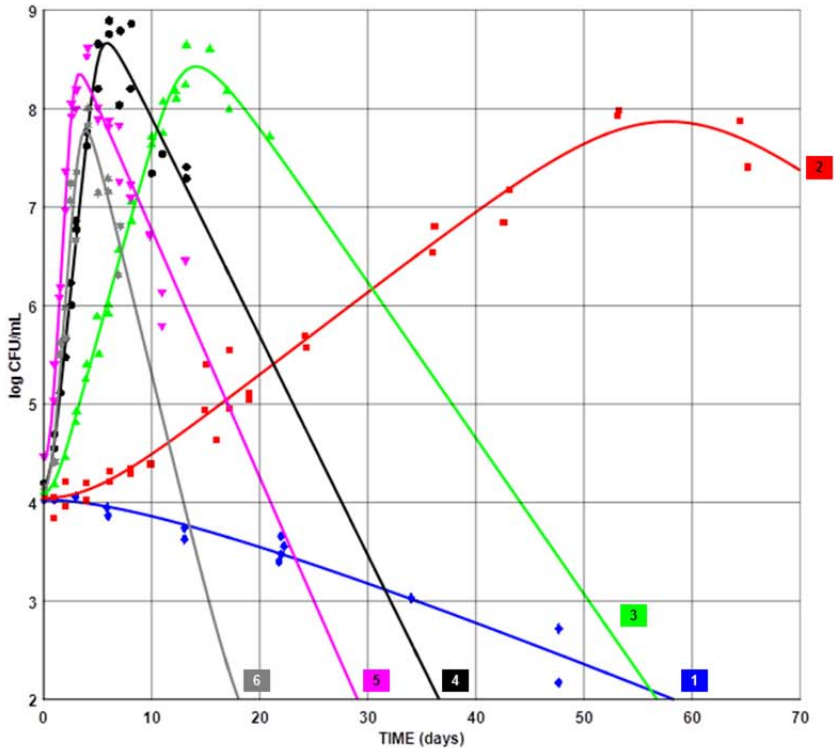


Figure A-1. QCM-4 Model Fits for Figure 1, Reference [b] Datasets

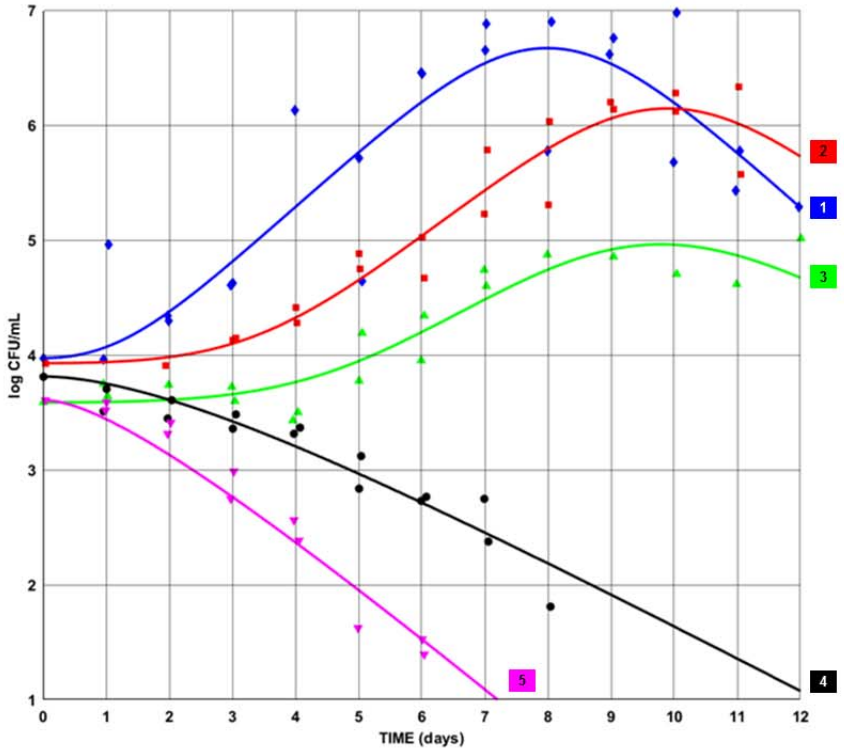


DATASET	DESCRIPTION
1	*S. aureus* in bread at pH = 5.23, T = 15°C, Aw = 0.90
2	*S. aureus* in bread at pH = 5.23, T = 20°C, Aw = 0.90
3	*S. aureus* in bread at pH = 5.23, T = 25°C, Aw = 0.90
4	*S. aureus* in bread at pH = 5.23, T = 30°C, Aw = 0.90
5	*S. aureus* in bread at pH = 5.23, T = 35°C, Aw = 0.90
6	*S. aureus* in bread at pH = 5.23, T = 40°C, Aw = 0.90

DATA SOURCE: FIGURE 2, REFERENCE [b]

Figure A-2. QCM-4 Model Fits for Figure 2, Reference [b] Datasets

QCM-4 MODEL FIT CALCULATED USING MODEL FORMULATION B IN SECTION 3.2.1

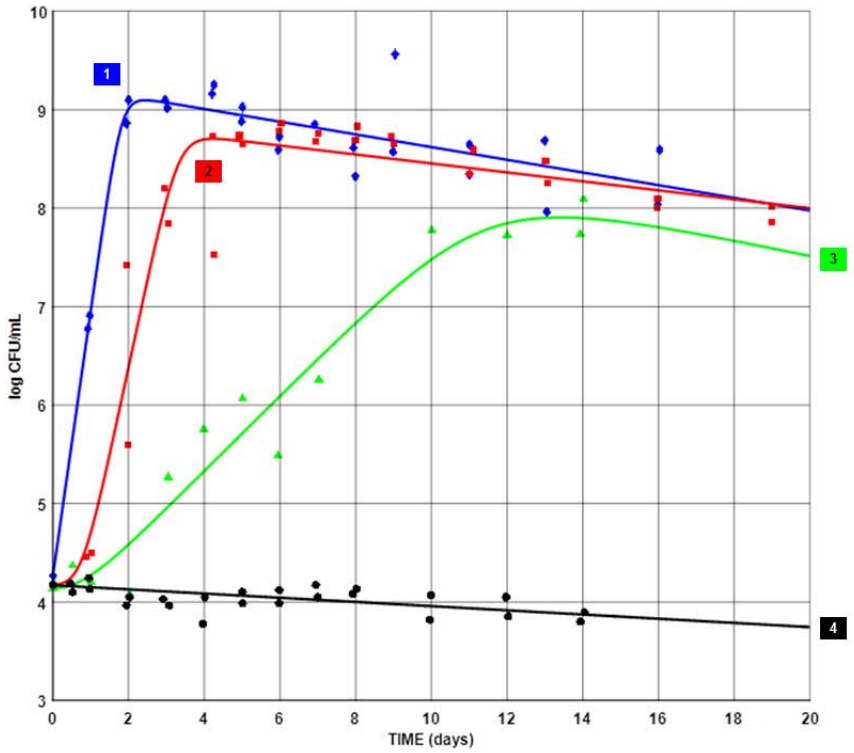


DATASET	DESCRIPTION
1	*S. aureus* in bread at pH = 5.38, T = 35°C, Aw = 0.86
2	*S. aureus* in bread at pH = 5.36, T = 35°C, Aw = 0.86
3	*S. aureus* in bread at pH = 5.31, T = 35°C, Aw = 0.86
4	*S. aureus* in bread at pH = 5.19, T = 35°C, Aw = 0.86
5	*S. aureus* in bread at pH = 4.97, T = 35°C, Aw = 0.86

DATA SOURCE: FIGURE 3, REFERENCE [b]

Figure A-3. QCM-4 Model Fits for Figure 3, Reference [b] Datasets

QCM-4 MODEL FIT CALCULATED USING MODEL FORMULATION B IN SECTION 3.2.1



DATASET	DESCRIPTION
1	*S. aureus* in turkey at pH = 6.35, T = 35°C, Aw = 0.89
2	*S. aureus* in turkey at pH = 6.35, T = 35°C, Aw = 0.87
3	*S. aureus* in turkey at pH = 6.35, T = 35°C, Aw = 0.86
4	*S. aureus* in turkey at pH = 6.35, T = 35°C, Aw = 0.84

DATA SOURCE: FIGURE 4, REFERENCE [b]

Figure A-4. QCM-4 Model Fits for Figure 4, Reference [b] Datasets

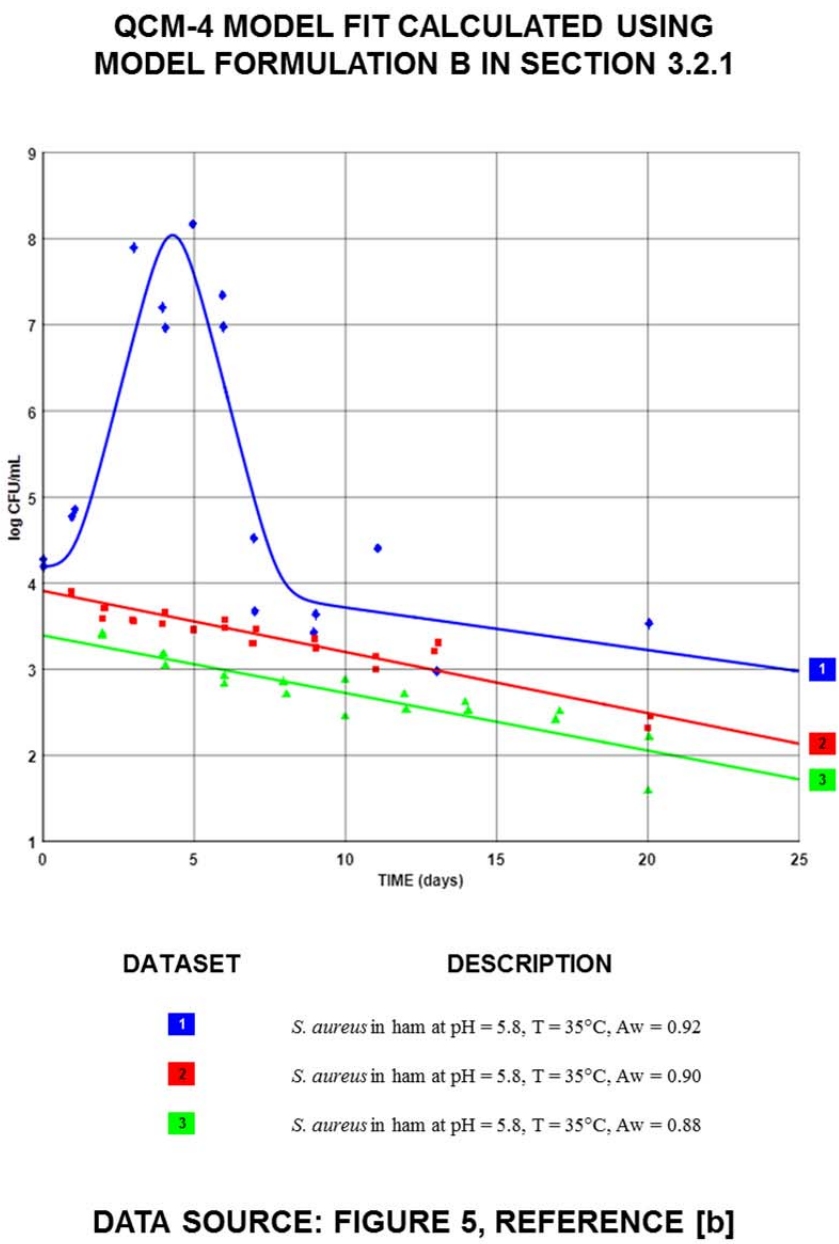


Figure A-5. QCM-4 Model Fits for Figure 5, Reference [b] Datasets

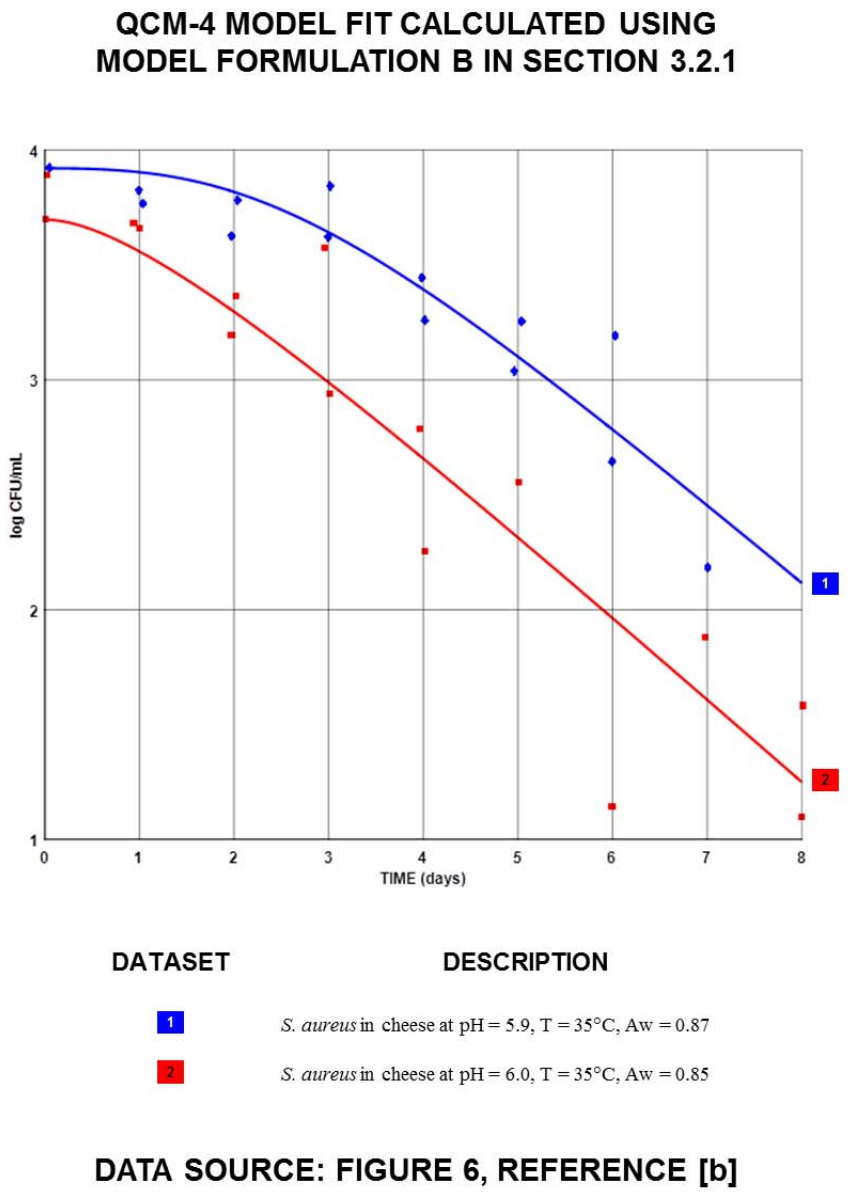
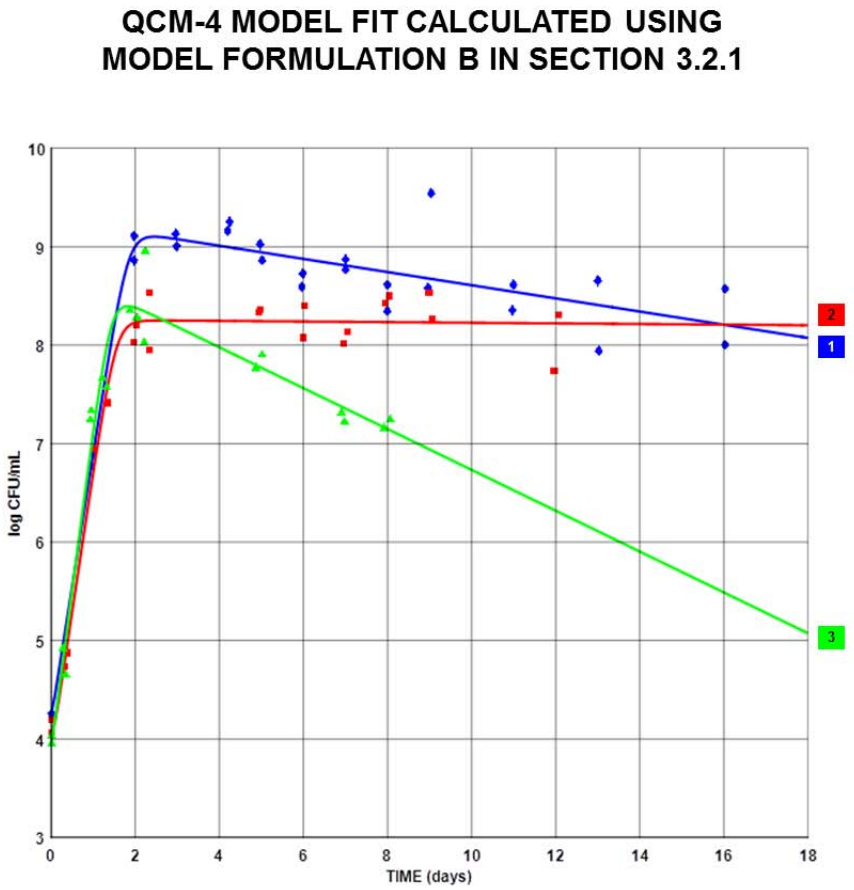


Figure A-6. QCM-4 Model Fits for Figure 6, Reference [b] Datasets

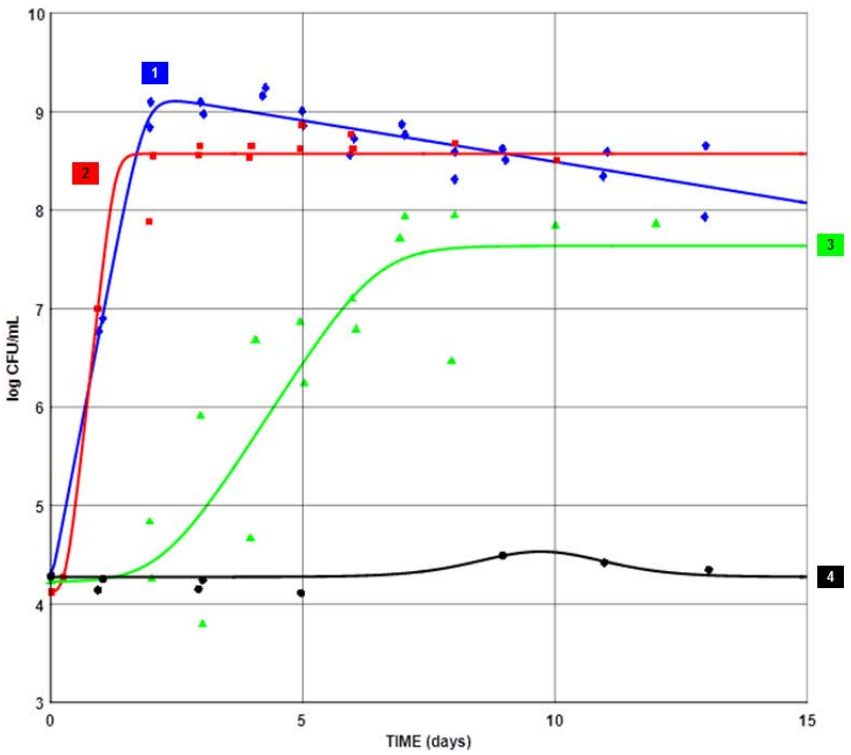


DATASET	DESCRIPTION
1	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.90
2	*Listeria* in turkey at pH = 6.4, T = 35°C, Aw = 0.94
3	*E. coli* in turkey at pH = 6.4, T = 35°C, Aw = 0.96

DATA SOURCE: FIGURE 7, REFERENCE [b]

Figure A-7. QCM-4 Model Fits for Figure 7, Reference [b] Datasets

QCM-4 MODEL FIT CALCULATED USING MODEL FORMULATION B IN SECTION 3.2.1

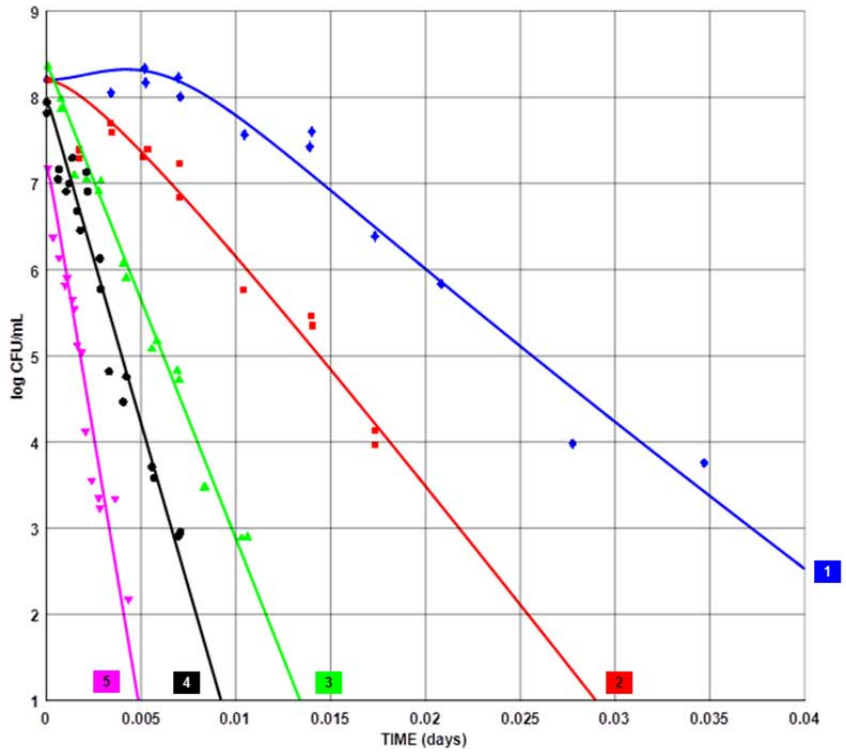


DATASET	DESCRIPTION
1	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.896, 0% anti-microbial lactate
2	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.896, 1% anti-microbial lactate
3	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.896, 2% anti-microbial lactate
4	*S. aureus* in turkey at pH = 6.4, T = 35°C, Aw = 0.896, 3% anti-microbial lactate

DATA SOURCE: FIGURE 8, REFERENCE [b]

Figure A-8. QCM-4 Model Fits for Figure 8, Reference [b] Datasets

QCM-4 MODEL FIT CALCULATED USING MODEL FORMULATION B IN SECTION 3.2.1

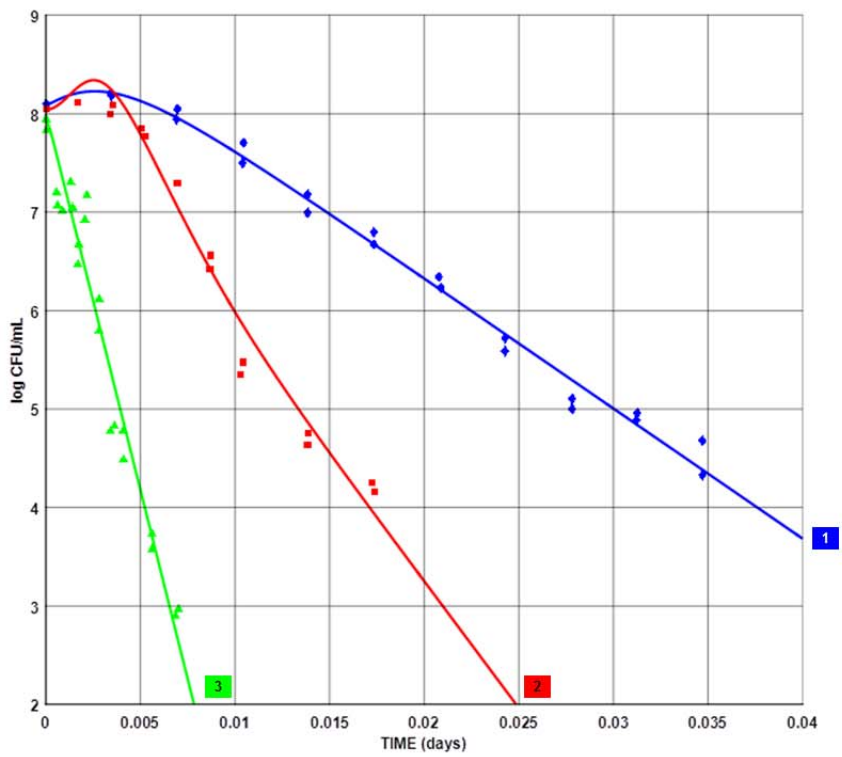


DATASET	DESCRIPTION
1	*E. coli* in whey protein at T = 35°C, P = 207 MPa
2	*E. coli* in whey protein at T = 35°C, P = 242 MPa
3	*E. coli* in whey protein at T = 35°C, P = 276 MPa
4	*E. coli* in whey protein at T = 35°C, P = 311 MPa
5	*E. coli* in whey protein at T = 35°C, P = 345 MPa

DATA SOURCE: FIGURE 9, REFERENCE [b]

Figure A-9. QCM-4 Model Fits for Figure 9, Reference [b] Datasets

QCM-4 MODEL FIT CALCULATED USING MODEL FORMULATION B IN SECTION 3.2.1

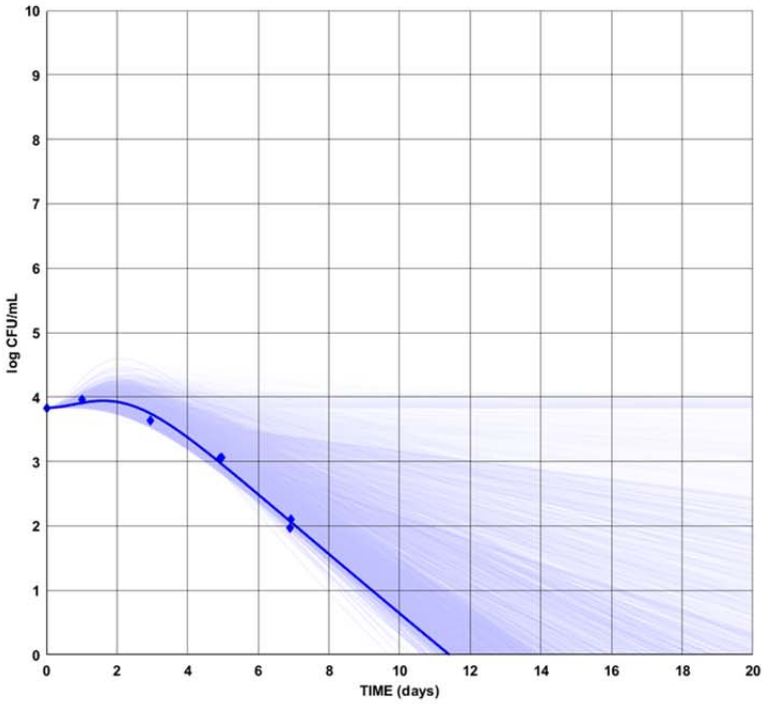


DATASET	DESCRIPTION
1	*E. coli* in whey protein at T = 30°C, P = 310 MPa
2	*E. coli* in whey protein at T = 40°C, P = 310 MPa
3	*E. coli* in whey protein at T = 50°C, P = 310 MPa

DATA SOURCE: FIGURE 10, REFERENCE [b]

Figure A-10. QCM-4 Model Fits for Figure 10, Reference [b] Datasets

S. aureus in bread at pH = 5.4, T = 35°C, Aw = 0.79



Measurements[1]

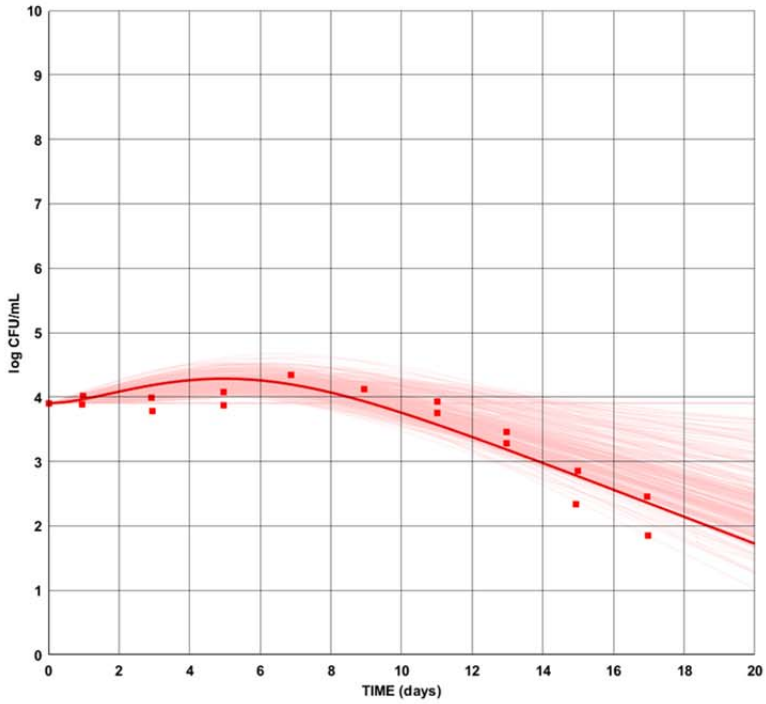
Time (days)	log CFU/mL
0.00	3.83
1.00	3.95
1.00	3.95
2.94	3.63
4.90	3.04
4.95	3.06
6.90	1.96
6.92	2.10

[1]Dataset 1 of Figure 1 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.98	4.98	17.66	13.51	4618.32	11.43	0.56	2.22	4.16×10^{-2}

Figure A-11. Forward Uncertainty Propagation for Dataset 1 in Figure A-1

S. aureus in bread at pH = 5.4, T = 35°C, Aw = 0.84



Measurements[(1)]

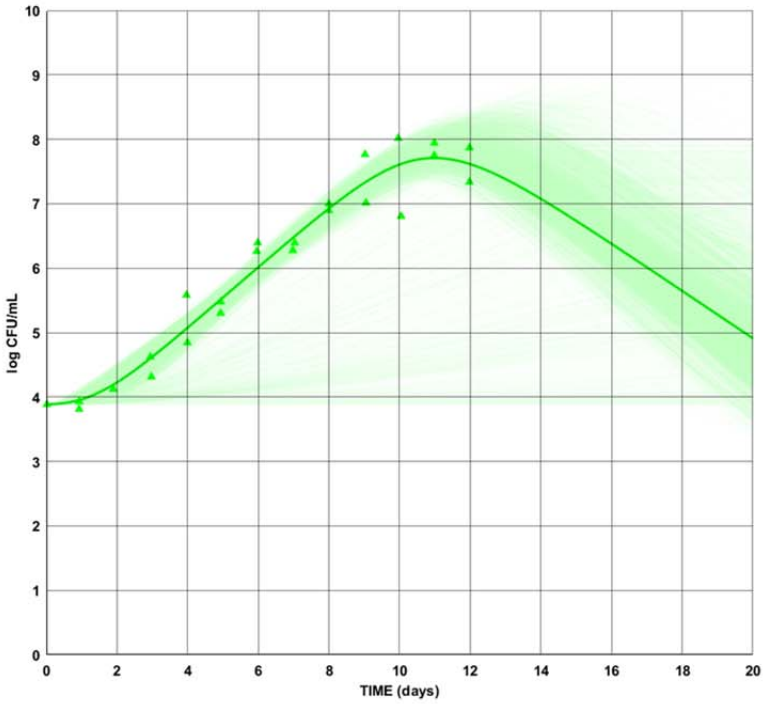
Time (days)	log CFU/mL
0.00	3.91
0.96	3.89
0.98	4.01
2.90	4.00
2.95	3.78
4.97	4.07
4.97	3.88
6.87	4.35
8.93	4.12
11.00	3.94
11.00	3.75
12.96	3.47
12.96	3.28
14.93	2.34
14.98	2.86
16.95	2.46
16.97	1.85

[(1)]Dataset 2 of Figure 1 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
1.02	5.48	65.13	113.60	94.53	115.28	0.37	2.03	1.20×10^{0}

Figure A-12. Forward Uncertainty Propagation for Dataset 2 in Figure A-1

S. aureus in bread at pH = 5.4, T = 35°C, Aw = 0.87



Measurements[1]

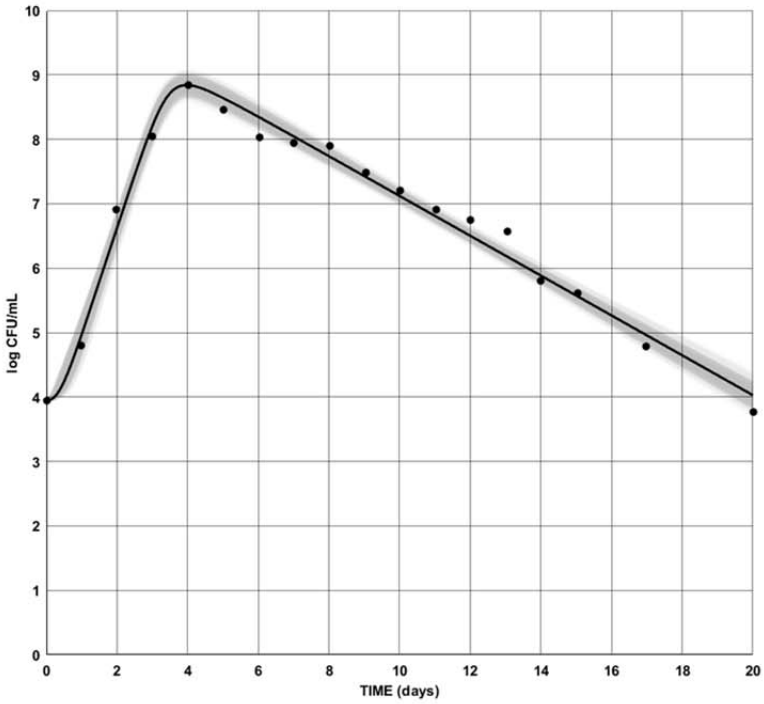
Time (days)	log CFU/mL
0.00	3.89
0.94	3.81
0.94	3.93
1.88	4.12
2.95	4.62
2.98	4.31
3.97	5.59
4.00	4.84
4.94	5.30
4.94	5.48
5.96	6.26
5.99	6.39
6.98	6.27
7.03	6.39
8.00	6.89
8.01	7.00
9.02	7.76
9.05	7.02
9.97	8.02
10.04	6.80
10.99	7.74
10.99	7.94
11.99	7.87
11.99	7.33

[1]Dataset 3 of Figure 1 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.26	0.91	3.64	5.49	3.59	6.65	1.09	0.12	2.09×10^{0}

Figure A-13. Forward Uncertainty Propagation for Dataset 3 in Figure A-1

S. aureus in bread at pH = 5.4, T = 35°C, Aw = 0.91



Measurements[1]

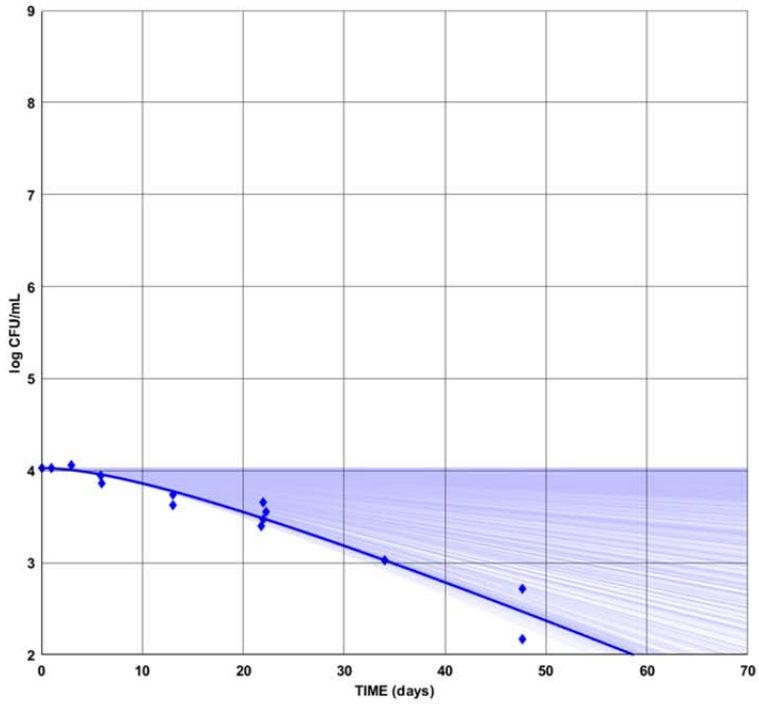
Time (days)	log CFU/mL
0.00	3.94
0.98	4.79
1.98	6.91
3.00	8.05
4.02	8.84
5.02	8.46
6.04	8.04
7.01	7.95
8.03	7.90
9.05	7.49
10.02	7.20
11.04	6.91
11.99	6.74
13.04	6.58
13.99	5.80
15.03	5.62
16.97	4.78
20.00	3.77

[1]Dataset 4 of Figure 1 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
1.14	0.97	4.55	0.06	3.61	0.22	3.83	0.07	6.54×10^{-1}

Figure A-14. Forward Uncertainty Propagation for Dataset 4 in Figure A-1

S. aureus in bread at pH = 5.23, T = 15°C, Aw = 0.90



Measurements[1]

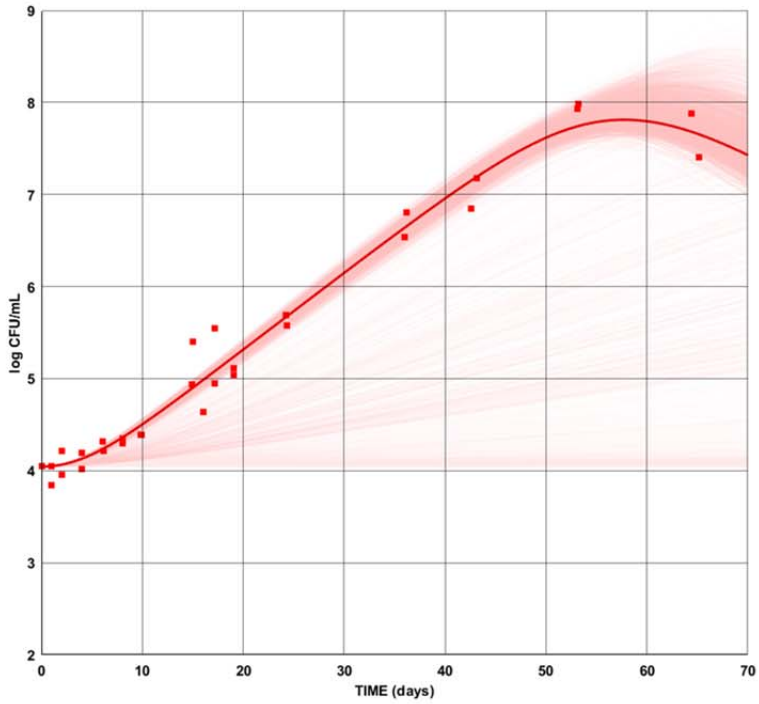
Time (days)	log CFU/mL
0.00	4.03
0.97	4.03
2.97	4.06
5.91	3.95
6.01	3.86
12.99	3.63
13.06	3.74
21.82	3.39
21.95	3.47
21.95	3.66
22.21	3.55
34.00	3.02
47.68	2.72
47.68	2.17

[1]Dataset 1 of Figure 2 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-0.11	11.00	0.12	11.28	2.36×10^{-1}

Figure A-15. Forward Uncertainty Propagation for Dataset 1 in Figure A-2

S. aureus in bread at pH = 5.23, T = 20°C, Aw = 0.90



Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.12	0.12	0.71	5.16	0.44	5.98	0.19	0.03	1.12×10^{0}

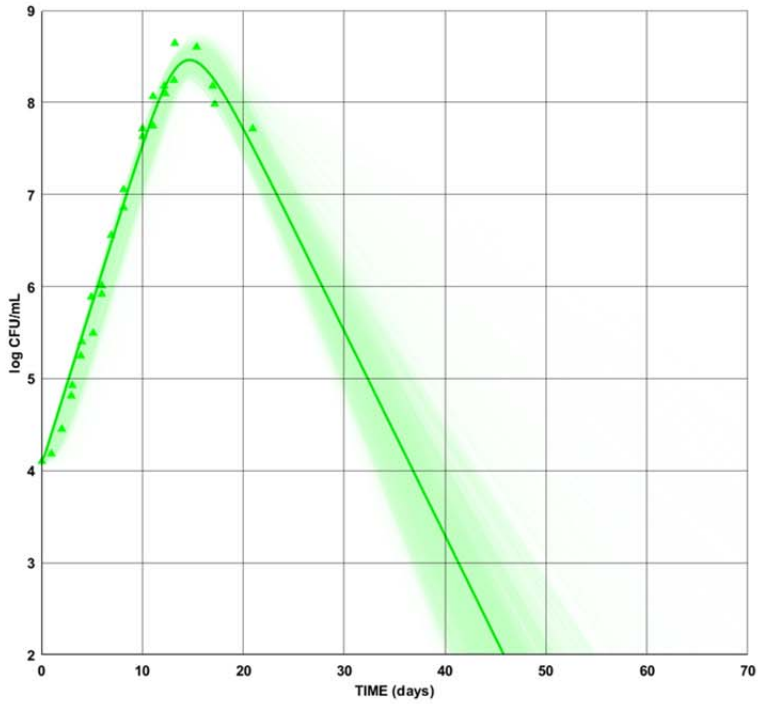
Measurements[1]

Time (days)	log CFU/mL
0.00	4.04
0.99	4.05
1.01	3.84
2.00	4.22
2.01	3.96
3.98	4.02
4.02	4.19
6.09	4.32
6.12	4.22
8.00	4.35
8.01	4.30
9.87	4.39
9.90	4.39
14.93	4.94
15.00	5.40
16.01	4.64
17.17	4.95
17.18	5.55
19.06	5.12
19.07	5.04
24.21	5.69
24.35	5.57
35.97	6.54
36.18	6.81
42.59	6.85
43.09	7.18
53.15	7.93
53.20	7.99
64.41	7.88
65.17	7.40

[1]Dataset 2 of Figure 2 in Reference [b]

Figure A-16. Forward Uncertainty Propagation for Dataset 2 in Figure A-2

S. aureus in bread at pH = 5.23, T = 25°C, Aw = 0.90



Measurements[1]

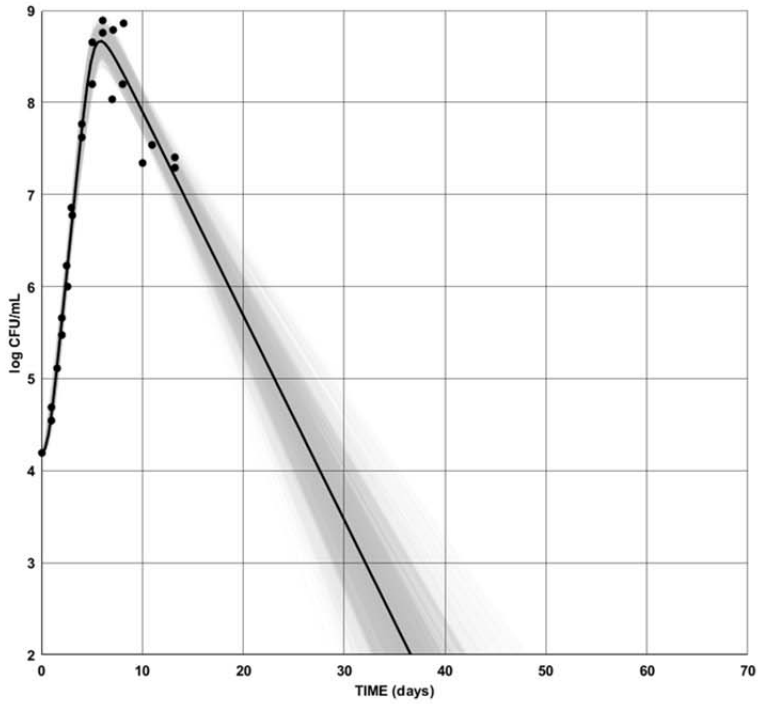
Time (days)	log CFU/mL
0.00	4.10
0.96	4.19
1.98	4.45
2.98	4.81
3.03	4.93
3.93	5.25
4.00	5.40
4.94	5.89
5.08	5.50
5.95	6.02
6.02	5.92
6.96	6.56
8.10	7.06
8.14	6.85
9.98	7.63
9.98	7.71
11.02	7.75
11.02	8.07
12.17	8.18
12.24	8.10
13.12	8.25
13.20	8.64
15.35	8.61
16.95	8.18
17.15	7.98
20.90	7.71

[1]Dataset 3 of Figure 2 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
7.25	2.24	1.76	0.36	0.77	0.66	0.81	0.04	8.65×10^{-1}

Figure A-17. Forward Uncertainty Propagation for Dataset 3 in Figure A-2

S. aureus in bread at pH = 5.23, T = 30°C, Aw = 0.90



Measurements[1]

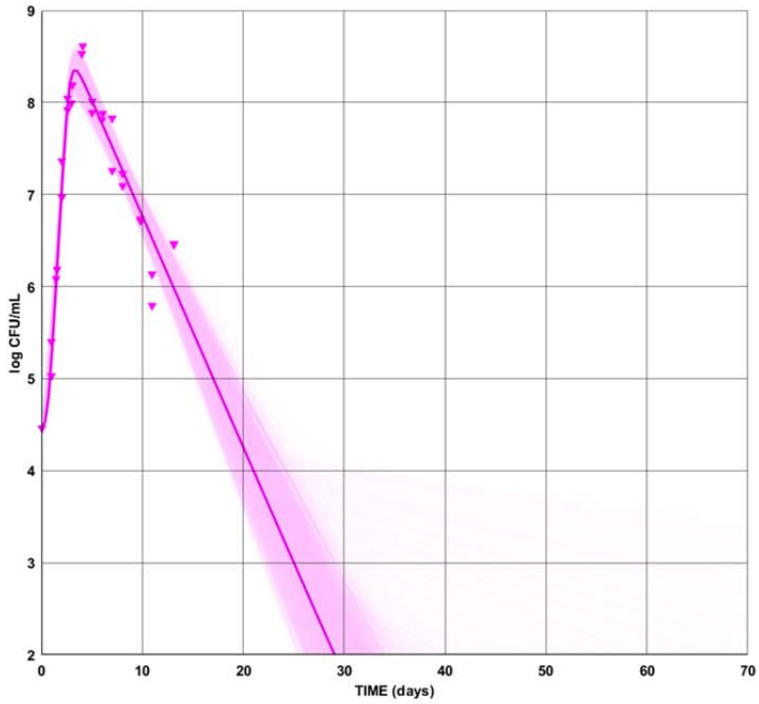
Time (days)	log CFU/mL
0.00	4.19
0.99	4.55
1.02	4.69
1.51	5.11
1.51	5.12
1.98	5.47
2.04	5.66
2.52	6.23
2.54	6.00
3.00	6.86
3.05	6.78
3.99	7.77
4.03	7.62
5.03	8.20
5.03	8.65
6.03	8.76
6.03	8.89
7.02	8.03
7.06	8.79
8.03	8.20
8.17	8.86
10.02	7.34
10.93	7.54
13.18	7.29
13.18	7.41

[1]Dataset 4 of Figure 2 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.57	0.60	2.98	0.06	3.33	0.28	2.46	0.07	1.34×10^{0}

Figure A-18. Forward Uncertainty Propagation for Dataset 4 in Figure A-2

S. aureus in bread at pH = 5.23, T = 35°C, Aw = 0.90



Measurements[1]

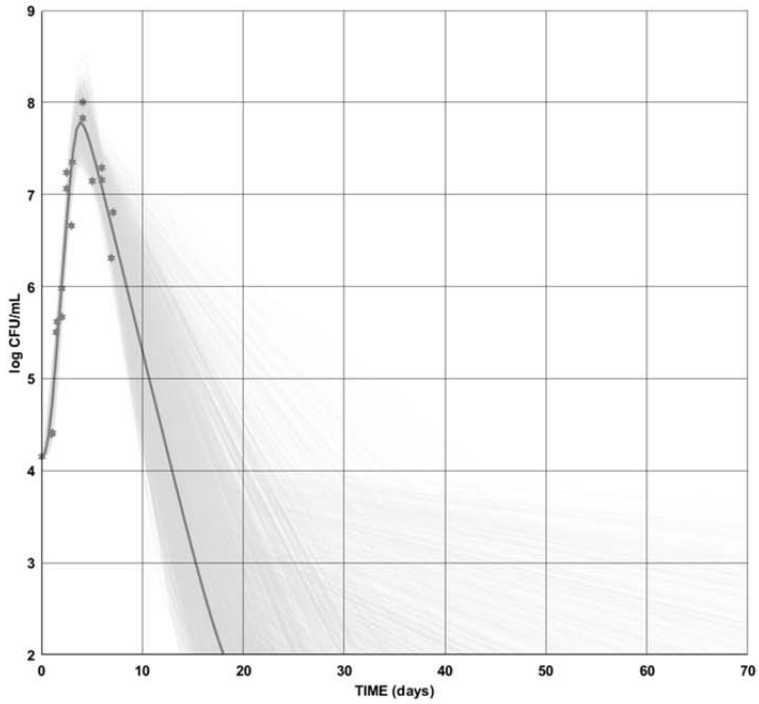
Time (days)	log CFU/mL
0.00	4.46
0.99	5.40
1.03	5.03
1.50	6.08
1.54	6.19
2.02	6.97
2.03	7.36
2.55	8.04
2.56	7.92
2.98	7.99
3.04	8.19
4.04	8.53
4.09	8.62
5.07	7.89
5.07	8.01
6.01	7.81
6.04	7.88
7.03	7.26
7.03	7.83
8.04	7.23
8.09	7.10
9.86	6.73
9.86	6.72
10.95	5.80
10.95	6.14
13.12	6.46
13.15	6.47

[1]Dataset 5 of Figure 2 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.32	1.34	4.89	0.10	13.55	0.28	4.31	0.12	1.91×10^{0}

Figure A-19. Forward Uncertainty Propagation for Dataset 5 in Figure A-2

S. aureus in bread at pH = 5.23, T = 40°C, Aw = 0.90



Measurements[1]

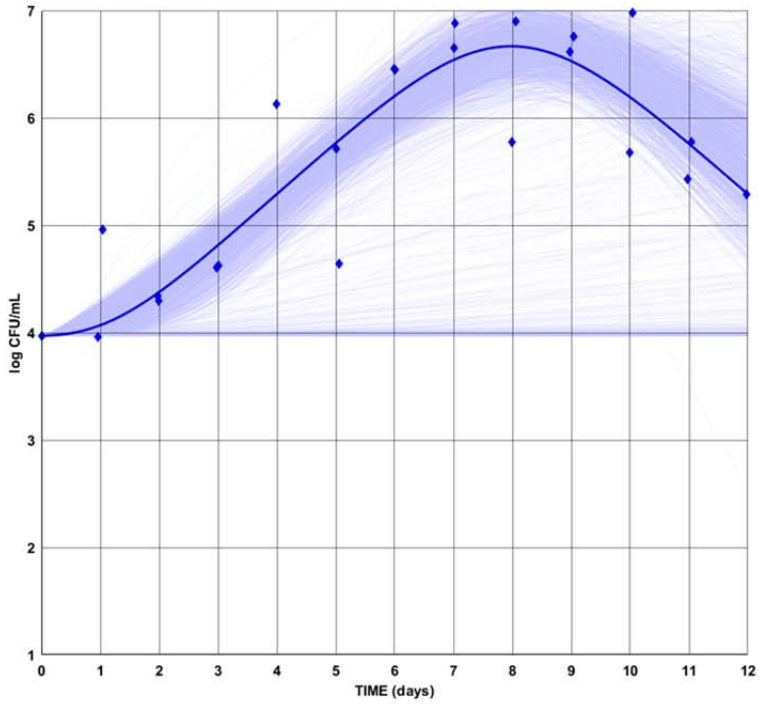
Time (days)	log CFU/mL
0.00	4.15
0.97	4.40
1.04	4.41
1.46	5.51
1.52	5.62
1.98	5.67
2.03	5.98
2.51	7.24
2.53	7.06
2.98	6.67
3.08	7.35
4.05	7.83
4.05	8.00
4.99	7.15
6.02	7.16
6.02	7.29
6.96	6.31
7.08	6.81

[1]Dataset 6 of Figure 2 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.34	1.30	4.46	0.11	29.52	0.60	3.34	0.16	1.55×10^{0}

Figure A-20. Forward Uncertainty Propagation for Dataset 6 in Figure A-2

S. aureus in bread at pH = 5.38, T = 35°C, Aw = 0.86



Measurements[1]

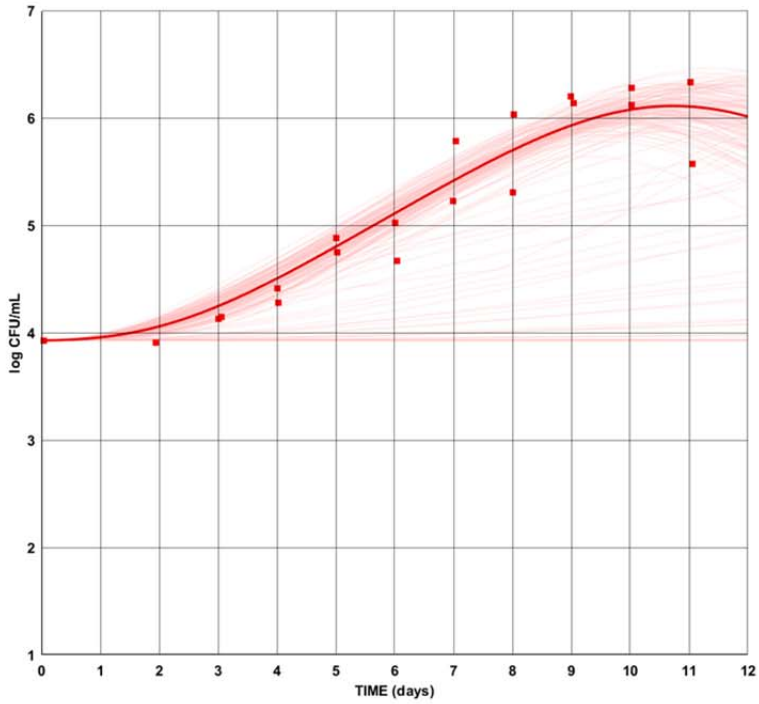
Time (days)	log CFU/mL
0.00	3.97
0.96	3.96
1.04	4.96
1.98	4.34
2.00	4.30
2.97	4.60
3.00	4.62
3.99	6.12
5.00	5.72
5.06	4.64
5.99	6.46
6.01	6.45
7.01	6.65
7.03	6.88
8.00	5.77
8.05	6.90
8.98	6.61
9.03	6.76
9.99	5.68
10.03	6.98
10.97	5.43
11.05	5.77
11.98	5.29

[1]Dataset 1 of Figure 3 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.32	2.16	45.24	33.30	3.16	34.27	1.14	0.33	5.09×10^{0}

Figure A-21. Forward Uncertainty Propagation for Dataset 1 in Figure A-3

S. aureus in bread at pH = 5.36, T = 35°C, Aw = 0.86



Measurements[1]

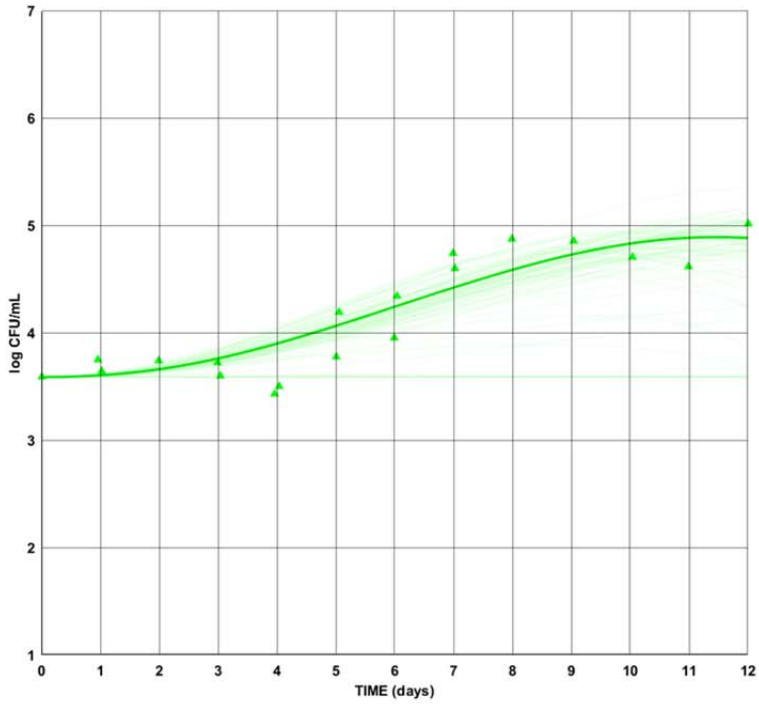
Time (days)	log CFU/mL
0.00	3.93
1.94	3.90
3.00	4.13
3.05	4.15
4.00	4.41
4.02	4.28
5.02	4.88
5.03	4.74
6.01	5.02
6.04	4.67
7.00	5.22
7.04	5.78
8.01	5.30
8.03	6.03
8.99	6.20
9.03	6.14
10.02	6.28
10.02	6.12
11.03	6.33
11.06	5.57

[1]Dataset 2 of Figure 3 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.14	0.57	33.01	237.98	7.15	239.81	0.77	0.13	1.25×10^{0}

Figure A-22. Forward Uncertainty Propagation for Dataset 2 in Figure A-3

S. aureus in bread at pH = 5.31, T = 35°C, Aw = 0.86



Measurements(1)

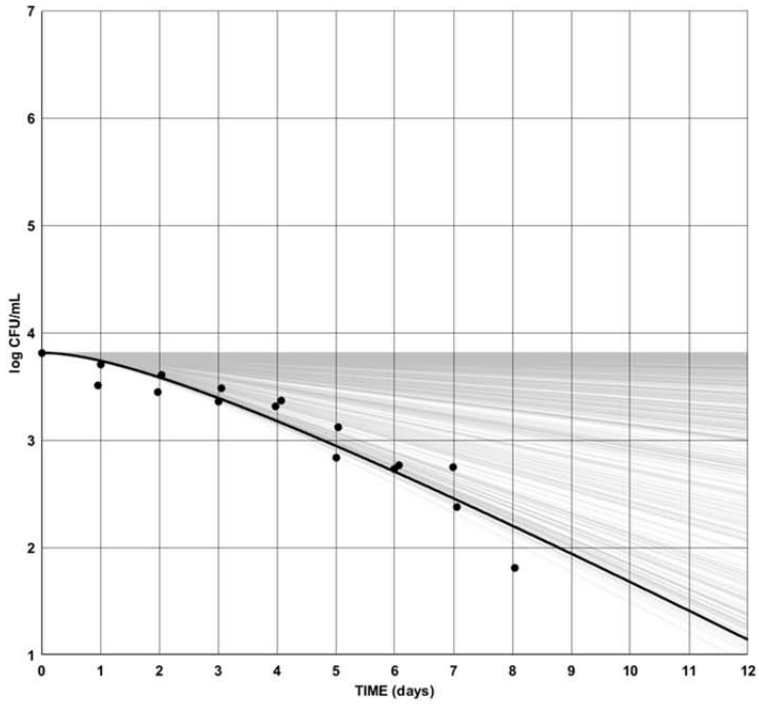
Time (days)	log CFU/mL
0.00	3.59
0.96	3.75
1.02	3.64
1.99	3.74
2.99	3.72
3.05	3.59
3.96	3.43
4.04	3.50
5.02	3.77
5.05	4.19
5.99	3.95
6.04	4.34
7.00	4.74
7.03	4.59
8.00	4.87
9.05	4.86
10.04	4.71
10.99	4.62
12.00	5.01

(1)Dataset 3 of Figure 3 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.14	0.31	38.43	428.57	46.08	431.74	0.51	0.22	9.62×10^{-1}

Figure A-23. Forward Uncertainty Propagation for Dataset 3 in Figure A-3

S. aureus in bread at pH = 5.19, T = 35°C, Aw = 0.86



Measurements[1]

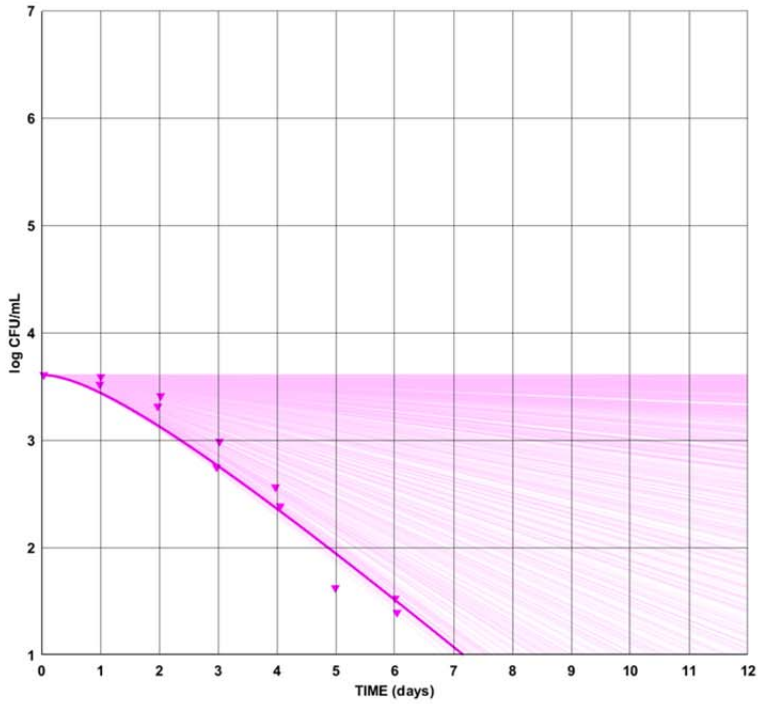
Time (days)	log CFU/mL
0.00	3.81
0.96	3.74
1.01	3.71
1.98	3.54
2.04	3.60
3.01	3.36
3.05	3.08
3.98	3.32
4.07	3.24
5.00	2.83
5.04	3.02
5.99	2.73
6.06	2.76
6.99	2.46
7.06	2.38
8.04	2.21

[1]Dataset 4 of Figure 3 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-0.69	22.82	0.70	23.16	4.33×10^{-1}

Figure A-24. Forward Uncertainty Propagation for Dataset 4 in Figure A-3

S. aureus in bread at pH = 4.97, T = 35°C, Aw = 0.86



Measurements[1]

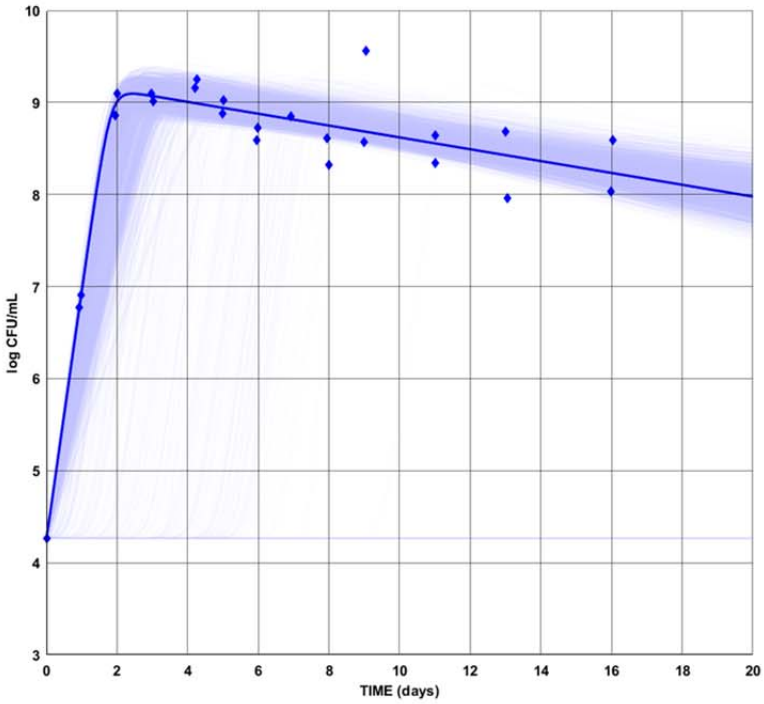
Time (days)	log CFU/mL
0.00	3.61
1.00	3.52
1.01	3.59
1.98	3.32
2.02	3.41
2.98	2.75
3.03	2.99
3.98	2.56
4.05	2.38
5.00	1.62
6.00	1.53
6.05	1.39

[1]Dataset 5 of Figure 3 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-1.14	15.36	1.15	15.66	3.60×10^{-1}

Figure A-25. Forward Uncertainty Propagation for Dataset 5 in Figure A-3

S. aureus in turkey at pH = 6.35, T = 35°C, Aw = 0.89



Measurements[1]

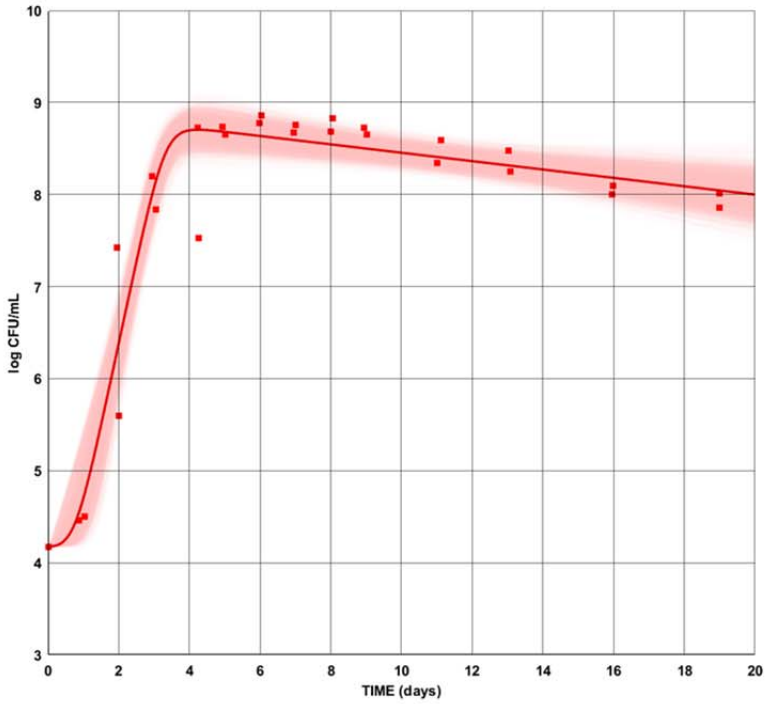
Time (days)	log CFU/mL
0.00	4.27
0.93	6.78
0.97	6.91
1.96	8.86
2.00	9.10
2.97	9.10
3.03	9.02
4.22	9.15
4.26	9.26
5.00	8.88
5.02	9.03
5.94	8.59
5.98	8.73
6.93	8.85
7.95	8.61
7.99	8.32
9.00	8.57
9.04	9.56
11.01	8.34
11.01	8.64
13.00	8.68
13.04	7.96
15.98	8.03
16.04	8.59

[1]Dataset 1 of Figure 4 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
266.19	238.94	6.36	0.33	4.54	0.49	6.21	0.34	1.73×10^{0}

Figure A-26. Forward Uncertainty Propagation for Dataset 1 in Figure A-4

S. aureus in turkey at pH = 6.35, T = 35°C, Aw = 0.87



Measurements(1)

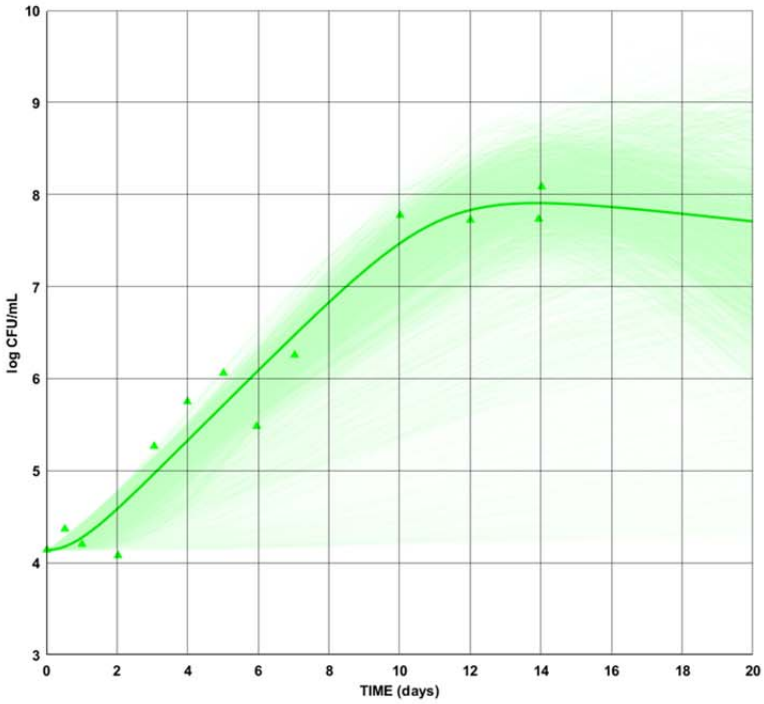
Time (days)	log CFU/mL
0.00	4.18
0.88	4.46
1.03	4.50
1.95	7.42
2.01	5.60
2.93	8.20
3.06	7.84
4.23	8.72
4.25	7.53
4.95	8.73
5.01	8.65
5.97	8.78
6.03	8.86
6.96	8.67
7.00	8.76
8.00	8.69
8.05	8.83
8.95	8.72
9.03	8.65
11.02	8.34
11.10	8.59
13.01	8.48
13.07	8.25
15.97	8.00
15.99	8.10
18.98	7.86
18.98	8.01

(1)Dataset 2 of Figure 4 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.17	2.14	4.25	0.16	7.44	0.40	4.15	0.16	3.94×10^{0}

Figure A-27. Forward Uncertainty Propagation for Dataset 2 in Figure A-4

S. aureus in turkey at pH = 6.35, T = 35°C, Aw = 0.86



Measurements[1]

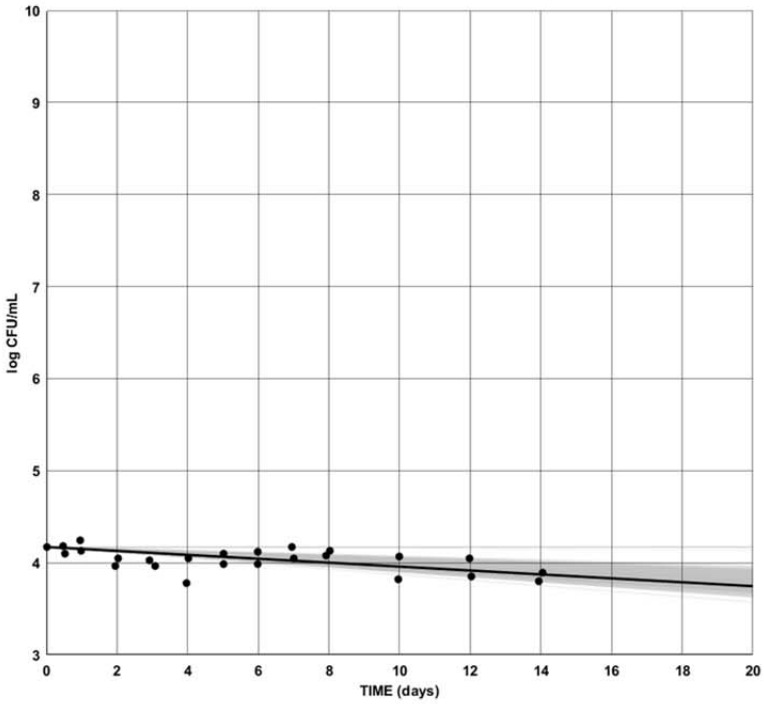
Time (days)	log CFU/mL
0.00	4.14
0.53	4.37
1.00	4.20
2.03	4.08
3.05	5.26
3.99	5.76
5.02	6.06
5.96	5.48
7.03	6.26
10.02	7.77
12.00	7.72
13.93	7.73
14.01	8.09

[1]Dataset 3 of Figure 4 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.75	1.90	0.98	1.79	8.07	4.15	0.88	0.26	1.25×10^{0}

Figure A-28. Forward Uncertainty Propagation for Dataset 3 in Figure A-4

S. aureus in turkey at pH = 6.35, T = 35°C, Aw = 0.84



Measurements[1]

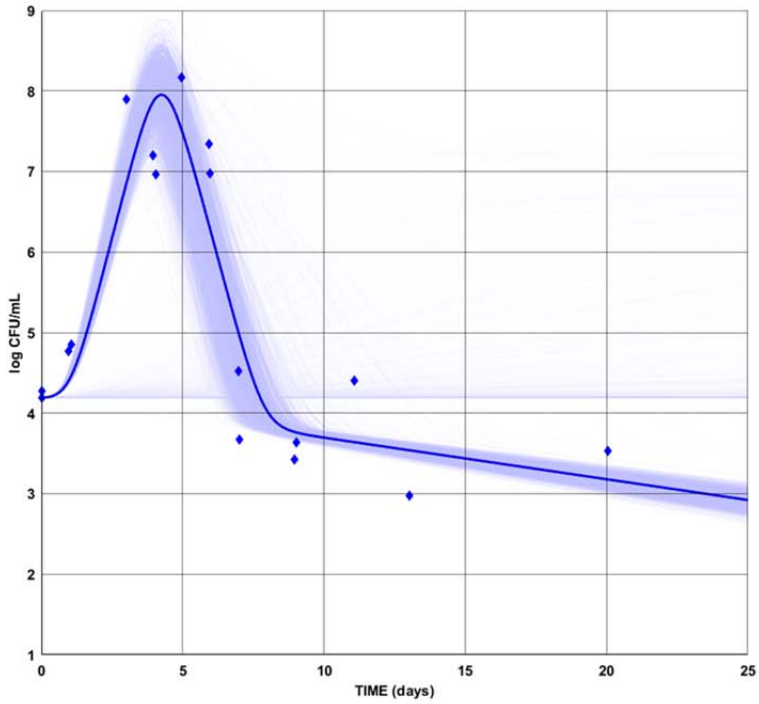
Time (days)	log CFU/mL
0.00	4.17
0.46	4.18
0.52	4.10
0.95	4.24
0.99	4.13
1.94	3.96
2.04	4.04
2.92	4.03
3.08	3.96
3.97	3.78
4.03	4.05
5.01	3.99
5.01	4.10
5.97	3.99
5.99	4.12
6.94	4.17
7.00	4.04
7.92	4.08
8.02	4.13
9.97	3.82
9.99	4.07
11.98	4.04
12.02	3.85
13.94	3.80
14.05	3.90

[1]Dataset 4 of Figure 4 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-998.00	1909.37	0.05	0.26	2.94×10^{-1}

Figure A-29. Forward Uncertainty Propagation for Dataset 4 in Figure A-4

S. aureus in ham at pH = 5.8, T = 35°C, Aw = 0.92



Measurements[1]

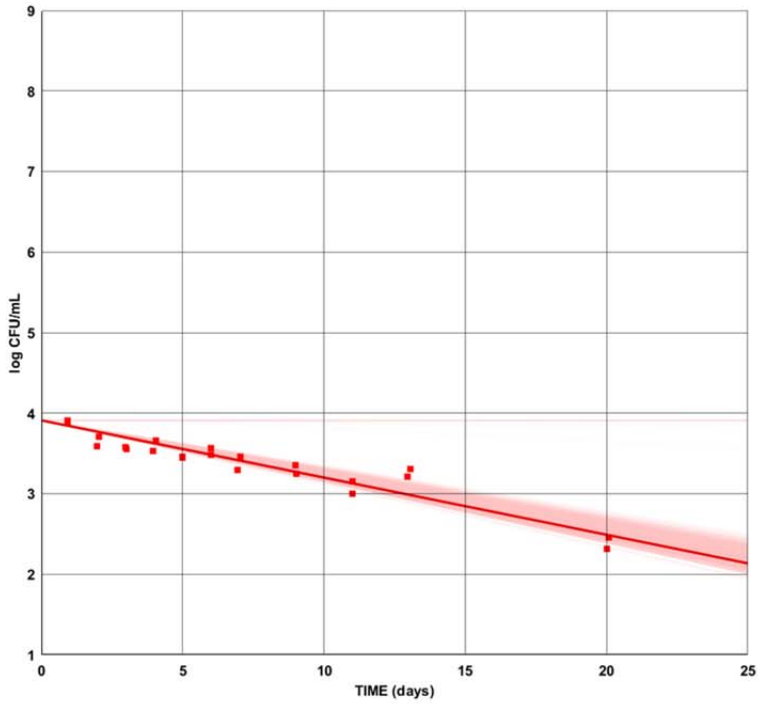
Time (days)	log CFU/mL
0.00	4.19
0.00	4.28
0.97	4.77
1.05	4.85
3.00	7.90
3.95	7.20
4.06	6.96
4.97	8.17
5.93	7.34
5.98	6.97
6.99	4.52
7.02	3.67
8.96	3.43
9.01	3.64
11.06	4.41
13.01	2.97
20.03	3.54

[1]Dataset 1 of Figure 5 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.12	0.06	51.36	9.87	1.10	9.79	3.15	0.09	7.71×10^{0}

Figure A-30. Forward Uncertainty Propagation for Dataset 1 in Figure A-5

S. aureus in ham at pH = 5.8, T = 35°C, Aw = 0.90



Measurements[1]

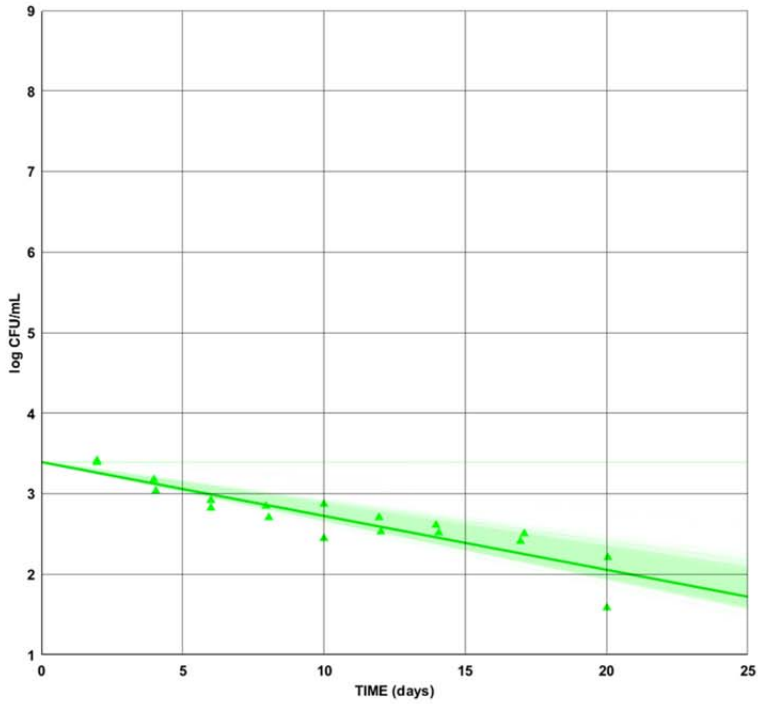
Time (days)	log CFU/mL
0.92	3.91
0.93	3.89
1.96	3.59
2.04	3.71
2.97	3.58
3.00	3.56
3.96	3.53
4.04	3.66
5.00	3.44
5.00	3.46
6.01	3.49
6.01	3.57
6.94	3.30
7.05	3.47
9.00	3.35
9.02	3.24
10.99	3.16
10.99	3.00
12.97	3.21
13.05	3.31
20.00	2.31
20.08	2.46

[1]Dataset 2 of Figure 5 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-999.00	701.59	0.16	0.08	3.38×10^{-1}

Figure A-31. Forward Uncertainty Propagation for Dataset 2 in Figure A-5

S. aureus in ham at pH = 5.8, T = 35°C, Aw = 0.88



Measurements[1]

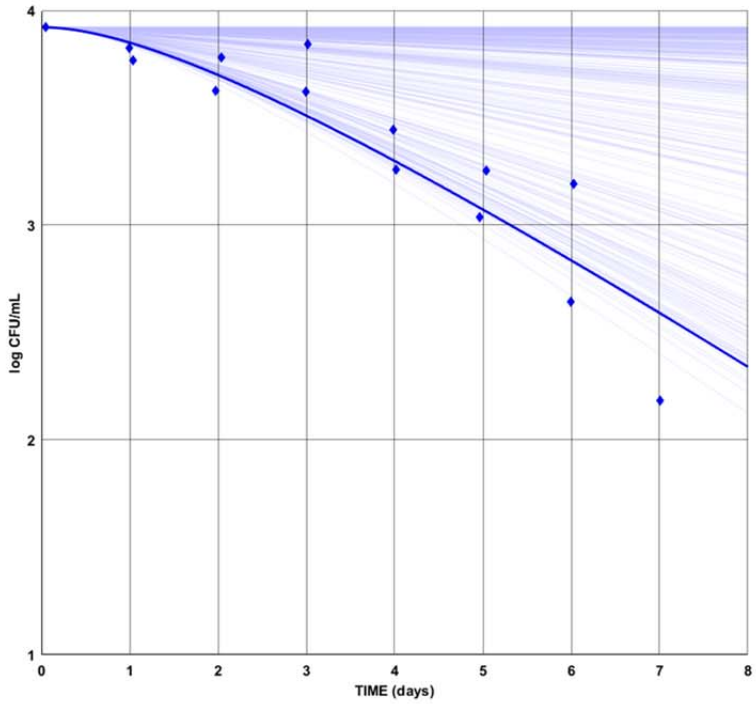
Time (days)	log CFU/mL
1.96	3.39
1.97	3.41
3.98	3.18
4.03	3.03
6.01	2.92
6.01	2.82
7.95	2.85
8.06	2.70
10.00	2.45
10.00	2.87
11.95	2.70
12.03	2.52
13.97	2.61
14.05	2.51
16.96	2.41
17.09	2.50
20.02	1.58
20.04	2.21

[1]Dataset 3 of Figure 5 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-998.95	1520.82	0.15	0.13	5.69×10^{-1}

Figure A-32. Forward Uncertainty Propagation for Dataset 3 in Figure A-5

S. aureus in cheese at pH = 5.9, T = 35°C, Aw = 0.87



Measurements[1]

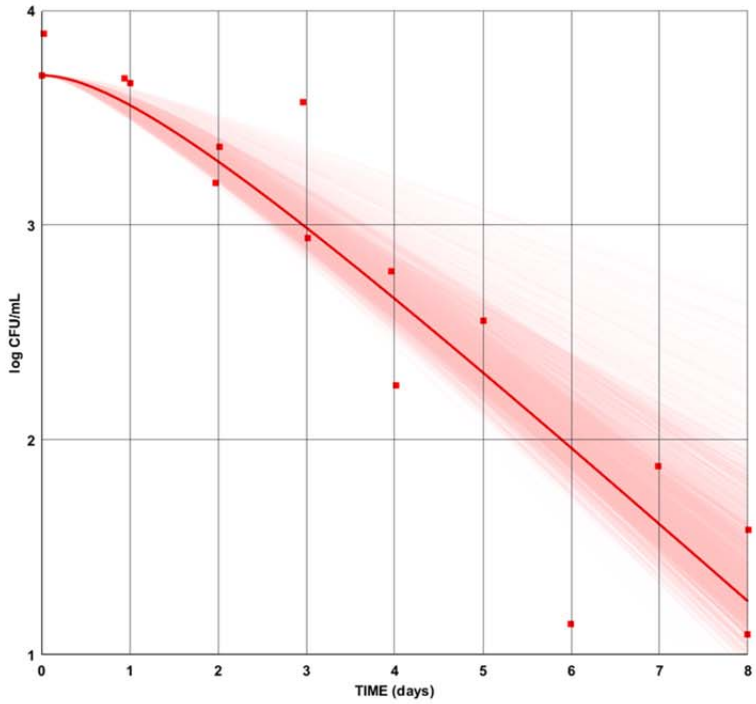
Time (days)	log CFU/mL
0.05	3.92
0.99	3.82
1.04	3.77
1.97	3.63
2.04	3.78
2.99	3.62
3.02	3.84
3.99	3.44
4.01	3.26
4.96	3.04
5.04	3.25
5.99	2.64
6.03	3.19
7.00	2.18

[1]Dataset 1 of Figure 6 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-0.68	41.04	0.69	41.56	5.41×10^{-1}

Figure A-33. Forward Uncertainty Propagation for Dataset 1 in Figure A-6

S. aureus in cheese at pH = 6.0, T = 35°C, Aw = 0.85



Measurements[1]

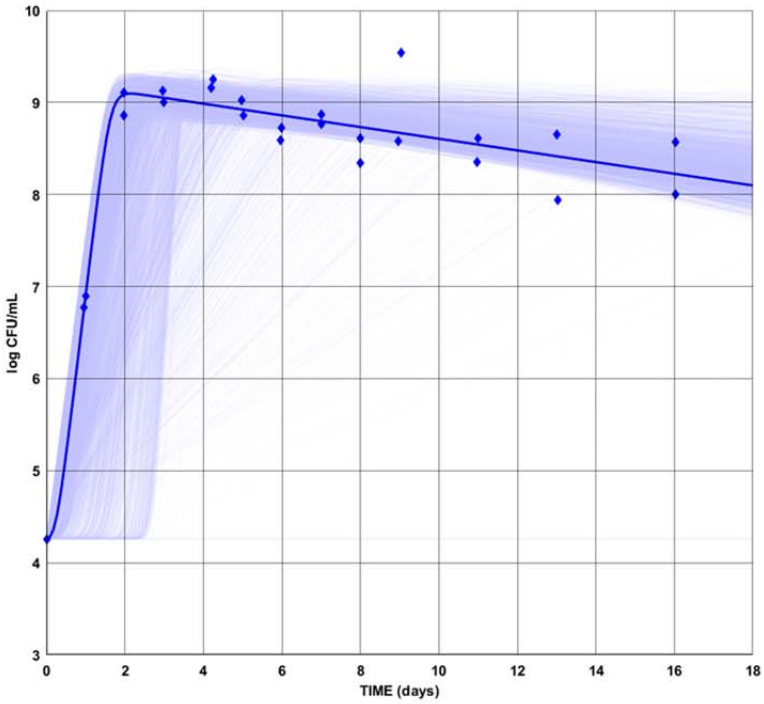
Time (days)	log CFU/mL
0.00	3.70
0.03	3.89
0.94	3.68
1.01	3.66
1.97	3.20
2.01	3.37
2.96	3.57
3.01	2.94
3.97	2.79
4.02	2.25
5.00	2.56
6.00	1.15
6.98	1.88
7.99	1.10
8.00	1.58

[1]Dataset 2 of Figure 6 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-1.27	0.85	0.84	0.36	1.51×10^{0}

Figure A-34. Forward Uncertainty Propagation for Dataset 2 in Figure A-6

S. aureus in turkey at pH = 6.4, T = 35°C, Aw = 0.90



Measurements[1]

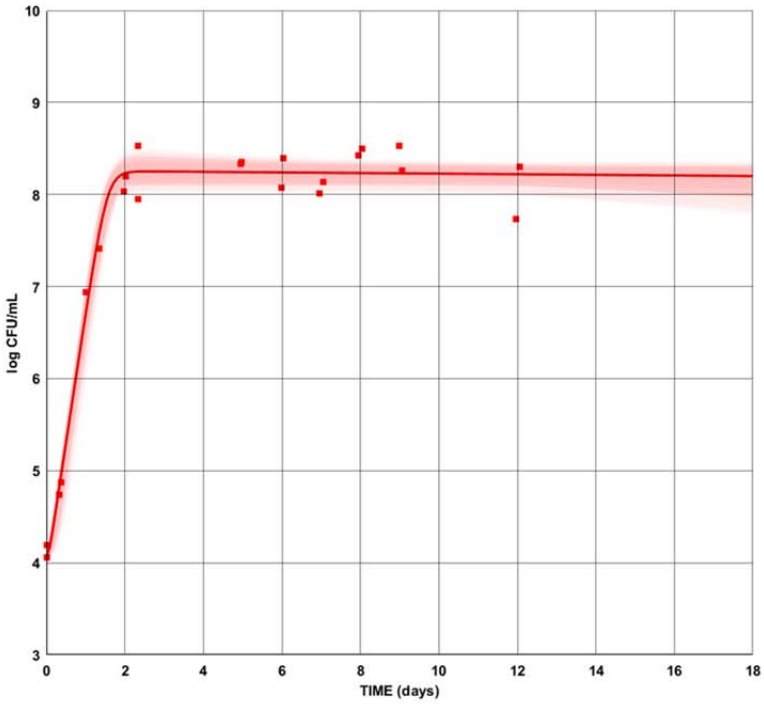
Time (days)	log CFU/mL
0.00	4.26
0.95	6.78
1.01	6.90
1.96	9.11
1.98	8.86
2.96	9.12
3.00	9.00
4.20	9.16
4.24	9.25
4.97	9.02
5.03	8.86
5.96	8.59
5.98	8.73
7.00	8.87
7.00	8.76
7.99	8.61
7.99	8.34
8.97	8.58
9.05	9.54
10.96	8.35
11.00	8.61
12.99	8.65
13.03	7.94
16.02	8.00
16.02	8.57

[1]Dataset 1 of Figure 7 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
1.28	66.75	8.24	1.04	6.01	1.09	8.09	1.06	1.72×10^{0}

Figure A-35. Forward Uncertainty Propagation for Dataset 1 in Figure A-7

Listeria in turkey at pH = 6.4, T = 35°C, Aw = 0.94



Measurements[1]

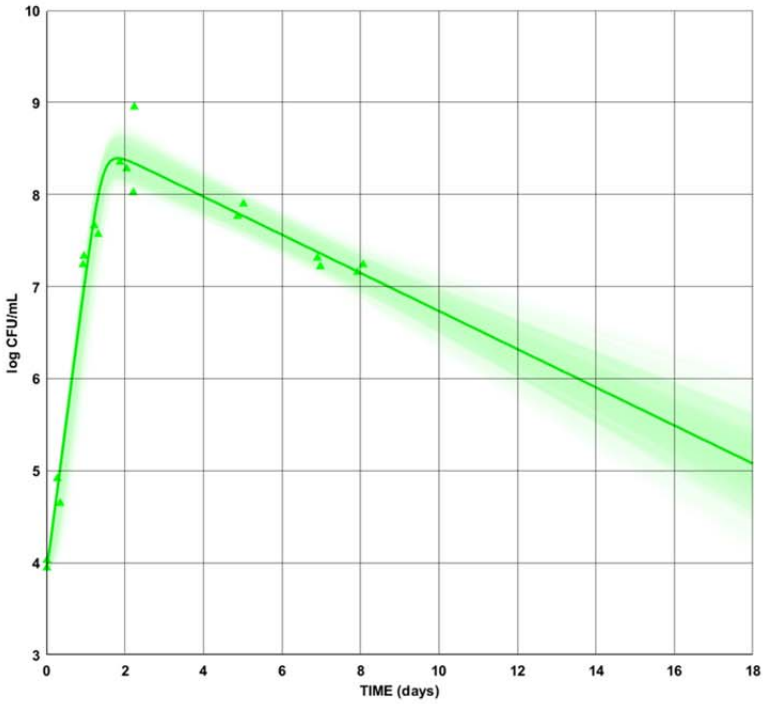
Time (days)	log CFU/mL
0.00	4.06
0.02	4.20
0.33	4.74
0.36	4.88
1.00	6.94
1.34	7.41
1.97	8.04
2.01	8.20
2.33	7.95
2.33	8.53
4.96	8.33
4.98	8.35
5.99	8.07
6.03	8.40
6.95	8.02
7.04	8.14
7.94	8.43
8.04	8.50
8.98	8.53
9.05	8.26
11.96	7.73
12.06	8.30

[1]Dataset 2 of Figure 7 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
14.18	1.27	6.49	0.06	36.12	0.26	6.49	0.06	8.87×10^{-1}

Figure A-36. Forward Uncertainty Propagation for Dataset 2 in Figure A-7

E. coli in turkey at pH = 6.4, T = 35°C, Aw = 0.96



Measurements[1]

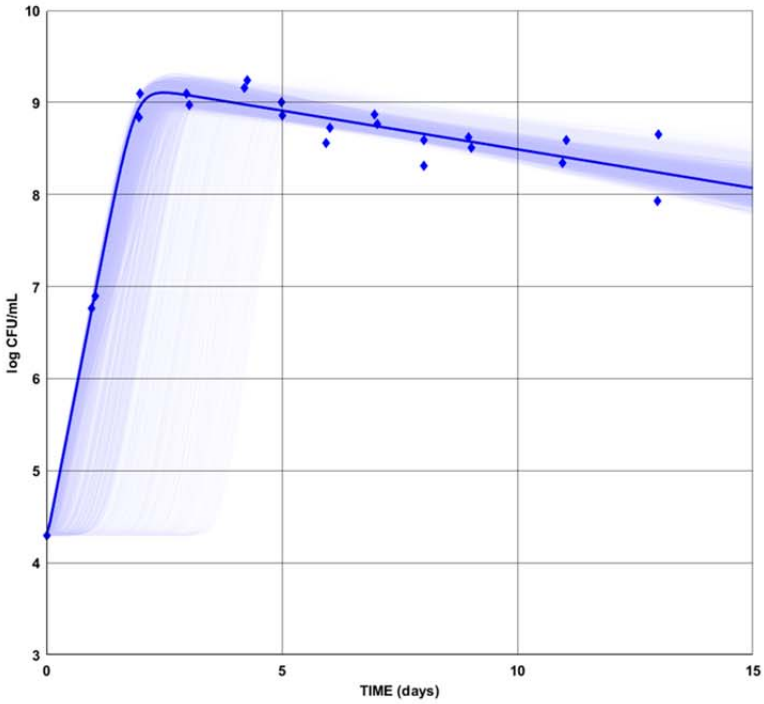
Time (days)	log CFU/mL
0.00	3.94
0.01	4.03
0.28	4.91
0.35	4.65
0.93	7.24
0.97	7.33
1.23	7.67
1.33	7.57
1.88	8.36
2.04	8.28
2.21	8.02
2.25	8.95
4.87	7.77
5.02	7.90
6.90	7.31
6.99	7.22
7.93	7.15
8.06	7.24

[1]Dataset 3 of Figure 7 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
56.61	1.93	7.96	0.04	24.73	0.33	7.49	0.04	1.15×10^{0}

Figure A-37. Forward Uncertainty Propagation for Dataset 3 in Figure A-7

S. aureus in turkey at pH = 6.4, T = 35°C, Aw = 0.896
0% anti-microbial lactate



Measurements[1]

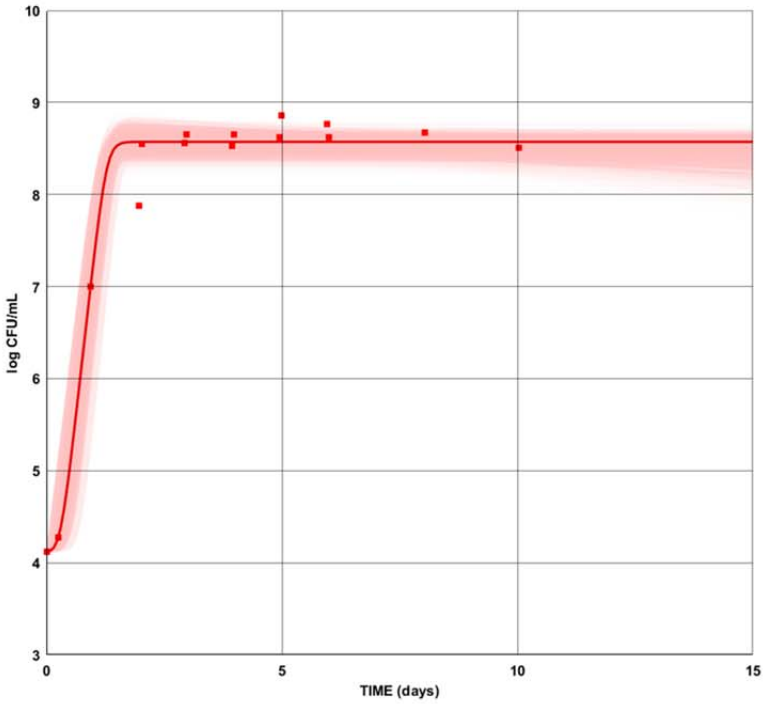
Time (days)	log CFU/mL
0.00	4.30
0.96	6.76
1.04	6.90
1.96	8.83
1.99	9.10
2.98	9.10
3.03	8.97
4.20	9.16
4.27	9.24
4.99	9.00
5.01	8.86
5.94	8.56
6.03	8.72
6.97	8.87
7.03	8.77
8.01	8.59
8.01	8.31
8.96	8.62
9.03	8.51
10.96	8.35
11.04	8.60
12.98	7.93
13.00	8.65

[1]Dataset 1 of Figure 8 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
38.80	18.39	6.20	0.15	4.17	0.28	6.01	0.15	6.91×10^{-1}

Figure A-38. Forward Uncertainty Propagation for Dataset 1 in Figure A-8

S. aureus in turkey at pH = 6.4, T = 35°C, Aw = 0.896
1% anti-microbial lactate



Measurements[1]

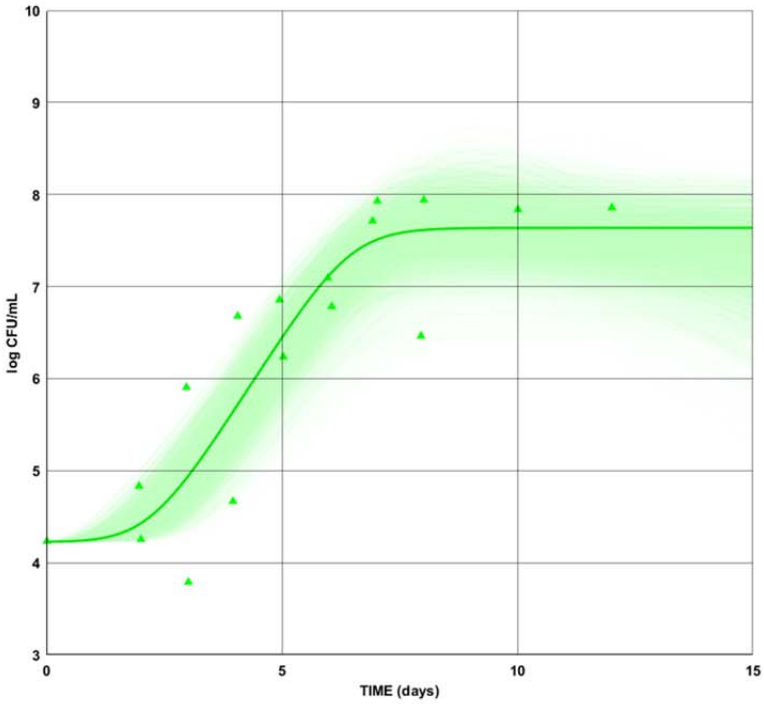
Time (days)	log CFU/mL
0.00	4.12
0.26	4.28
0.94	7.00
1.96	7.88
2.04	8.55
2.93	8.56
2.96	8.65
3.94	8.53
3.99	8.65
4.95	8.62
4.99	8.86
5.95	8.77
6.00	8.62
8.04	8.68
10.02	8.51

[1]Dataset 2 of Figure 8 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.45	2.71	10.51	0.05	28.10	0.28	10.51	0.05	6.30×10^{-1}

Figure A-39. Forward Uncertainty Propagation for Dataset 2 in Figure A-8

S. aureus in turkey at pH = 6.4, T = 35°C, Aw = 0.896
2% anti-microbial lactate



Measurements[1]

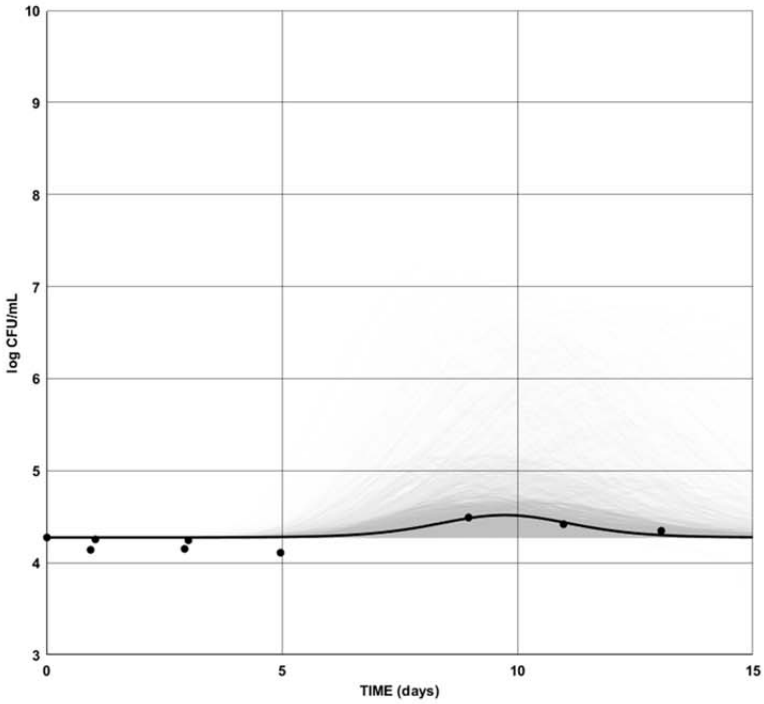
Time (days)	log CFU/mL
0.00	4.23
1.97	4.83
2.01	4.26
2.97	5.91
3.02	3.79
3.96	4.67
4.06	6.68
4.95	6.85
5.03	6.24
5.98	7.09
6.05	6.79
6.93	7.72
7.03	7.93
7.96	6.46
8.02	7.94
10.02	7.83
12.00	7.85

[1]Dataset 3 of Figure 8 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.03	1.18	1.86	0.26	42.92	1.38	1.86	0.12	6.54×10^{0}

Figure A-40. Forward Uncertainty Propagation for Dataset 3 in Figure A-8

S. aureus in turkey at pH = 6.4, T = 35°C, Aw = 0.896
3% anti-microbial lactate



Measurements(1)

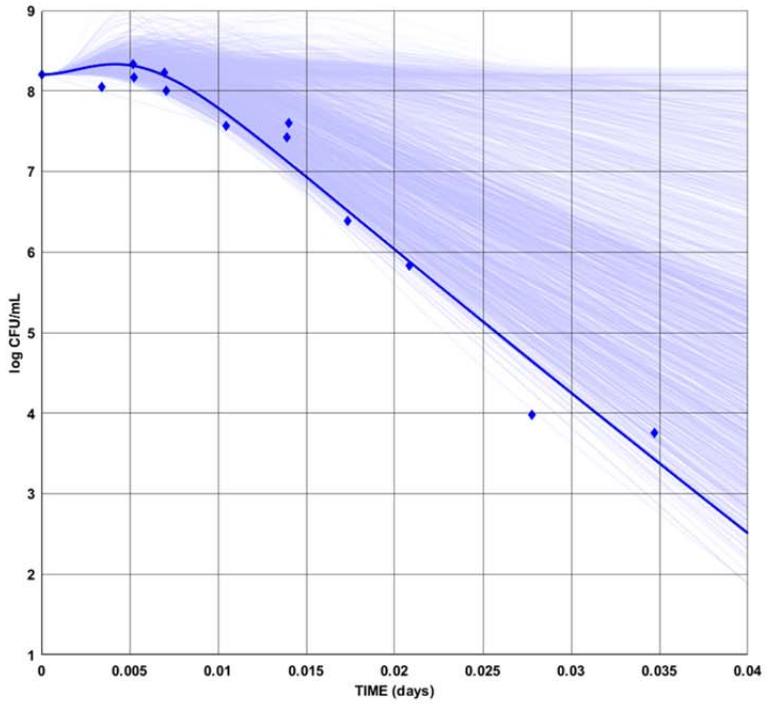
Time (days)	log CFU/mL
0.00	4.27
0.93	4.15
1.04	4.26
2.94	4.16
3.02	4.25
4.97	4.11
8.97	4.50
10.98	4.42
13.06	4.35

(1)Dataset 4 of Figure 8 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
0.00	0.04	8081.97	8.34	5.88	11.83	1.16	0.30	6.41×10^{-2}

Figure A-41. Forward Uncertainty Propagation for Dataset 4 in Figure A-8

E. coli in whey protein at T = 35°C, P = 207 MPa



Measurements[1]

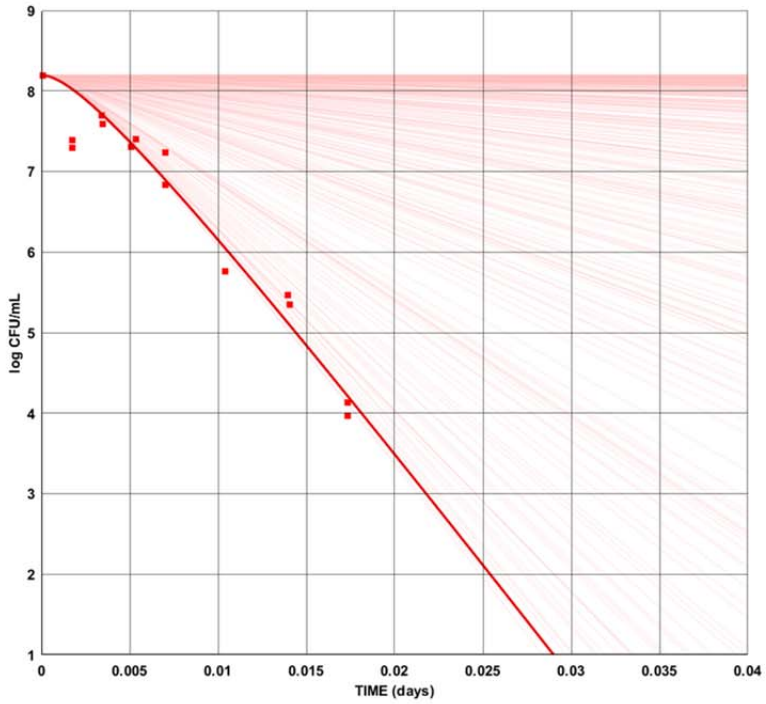
Time (days)	log CFU/mL
0.00000	8.20
0.00341	8.05
0.00519	8.33
0.00523	8.17
0.00697	8.23
0.00705	8.00
0.01046	7.57
0.01390	7.42
0.01398	7.60
0.01735	6.38
0.02083	5.83
0.02776	3.98
0.03470	3.75

[1]Dataset 1 of Figure 9 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
383.00	0.70	2643.76	12.74	222.20	15.38	250.33	1.56	1.05×10^{0}

Figure A-42. Forward Uncertainty Propagation for Dataset 1 in Figure A-9

E. coli in whey protein at T = 35°C, P = 242 MPa



Measurements[1]

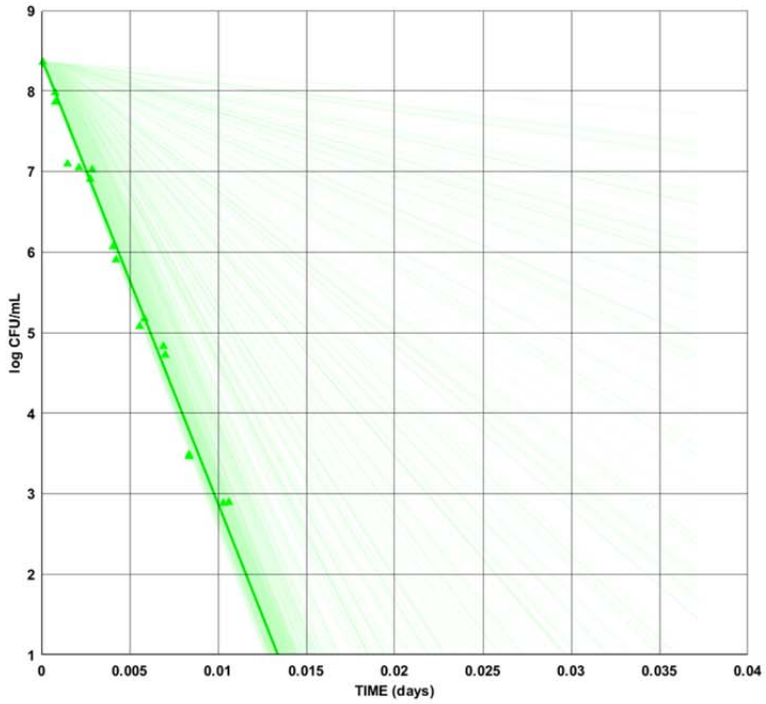
Time (days)	log CFU/mL
0.00000	8.20
0.00172	7.39
0.00173	7.30
0.00339	7.70
0.00347	7.59
0.00509	7.31
0.00533	7.40
0.00700	7.24
0.00700	6.84
0.01041	5.76
0.01394	5.46
0.01405	5.35
0.01734	4.13
0.01735	3.96

[1]Dataset 2 of Figure 9 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-675.03	44.14	676.38	44.43	1.41×10^{0}

Figure A-43. Forward Uncertainty Propagation for Dataset 2 in Figure A-9

E. coli in whey protein at T = 35°C, P = 276 MPa



Measurements[1]

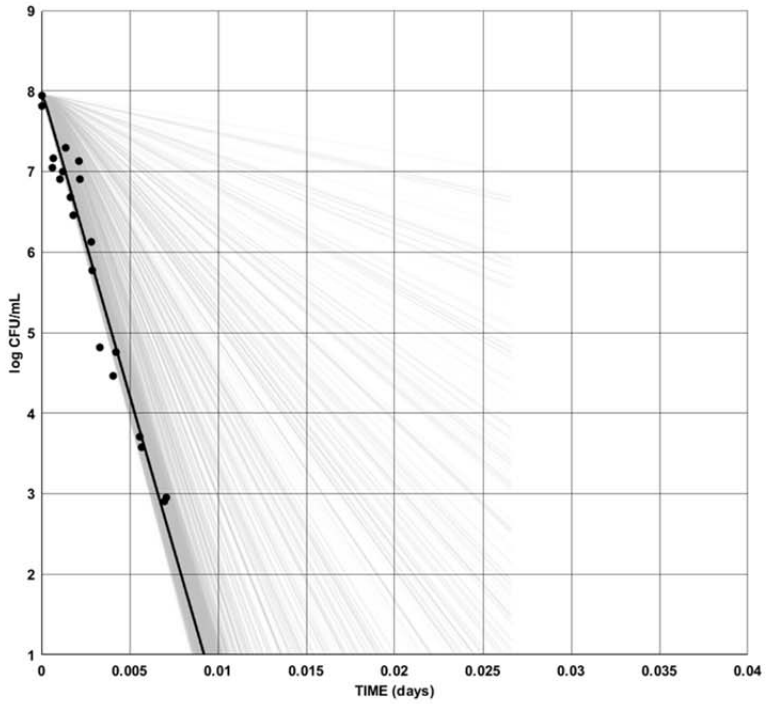
Time (days)	log CFU/mL
0.00000	8.36
0.00079	7.98
0.00079	7.86
0.00150	7.09
0.00210	7.05
0.00277	6.91
0.00289	7.02
0.00404	6.07
0.00419	5.90
0.00558	5.08
0.00582	5.17
0.00692	4.83
0.00700	4.72
0.00835	3.46
0.00836	3.49
0.01029	2.88
0.01060	2.89

[1]Dataset 3 of Figure 9 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-1275.50	0.04	9999.99	1.82	8.96×10^{-1}

Figure A-44. Forward Uncertainty Propagation for Dataset 3 in Figure A-9

E. coli in whey protein at T = 35°C, P = 311 MPa



Measurements[1]

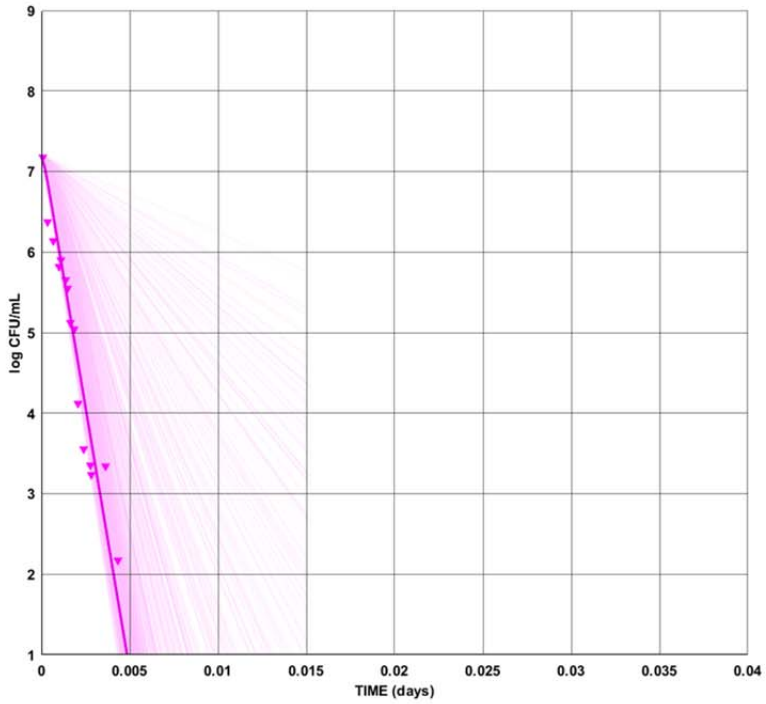
Time (days)	log CFU/mL
0.00000	7.95
0.00004	7.82
0.00059	7.05
0.00067	7.17
0.00103	6.91
0.00123	7.00
0.00135	7.30
0.00162	6.68
0.00182	6.46
0.00214	7.13
0.00218	6.91
0.00281	6.13
0.00289	5.78
0.00329	4.81
0.00408	4.46
0.00420	4.75
0.00555	3.70
0.00567	3.57
0.00697	2.90
0.00705	2.95

[1]Dataset 4 of Figure 9 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-1756.47	0.06	9999.99	1.67	2.43×10^{0}

Figure A-45. Forward Uncertainty Propagation for Dataset 4 in Figure A-9

E. coli in whey protein at T = 35°C, P = 345 MPa



Measurements[1]

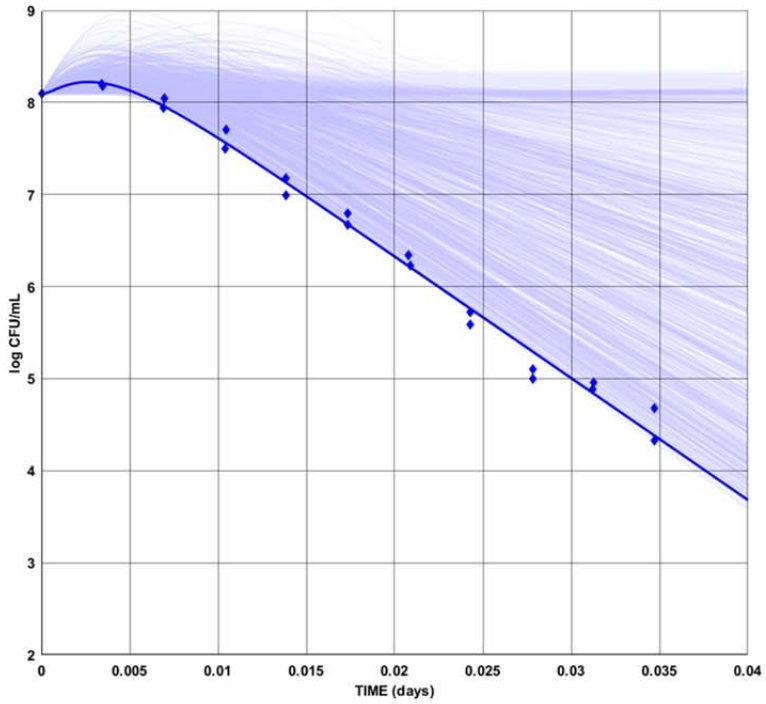
Time (days)	log CFU/mL
0.00000	7.18
0.00038	6.38
0.00065	6.14
0.00097	5.82
0.00109	5.90
0.00136	5.66
0.00148	5.55
0.00164	5.12
0.00188	5.04
0.00208	4.12
0.00239	3.55
0.00275	3.35
0.00283	3.23
0.00362	3.34
0.00433	2.17

[1]Dataset 5 of Figure 9 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-3008.36	0.08	9999.99	1.24	2.24×10^{0}

Figure A-46. Forward Uncertainty Propagation for Dataset 5 in Figure A-9

E. coli in whey protein at T = 30°C, P = 310 MPa



Measurements[1]

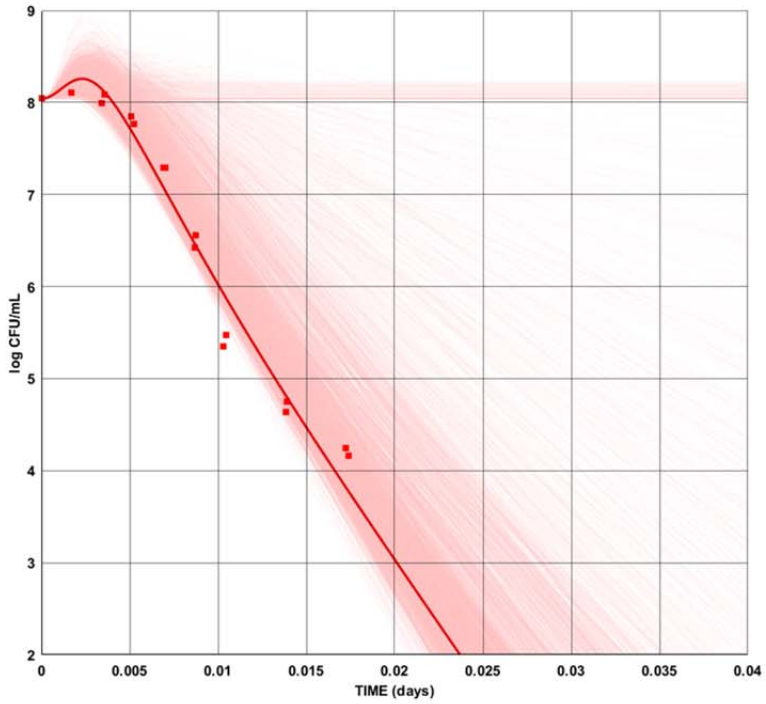
Time (days)	log CFU/mL
0.00000	8.09
0.00342	8.20
0.00345	8.18
0.00693	7.94
0.00697	8.04
0.01038	7.50
0.01045	7.71
0.01384	7.17
0.01387	6.99
0.01732	6.80
0.01735	6.67
0.02080	6.34
0.02087	6.23
0.02425	5.59
0.02429	5.72
0.02780	5.11
0.02784	5.00
0.03122	4.88
0.03129	4.96
0.03471	4.32
0.03472	4.68

[1]Dataset 1 of Figure 10 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
2849.58	28.17	694.39	0.47	1357.06	5.18	265.88	2.44	3.49×10^{-1}

Figure A-47. Forward Uncertainty Propagation for Dataset 1 in Figure A-10

E. coli in whey protein at T = 40°C, P = 310 MPa



Measurements[1]

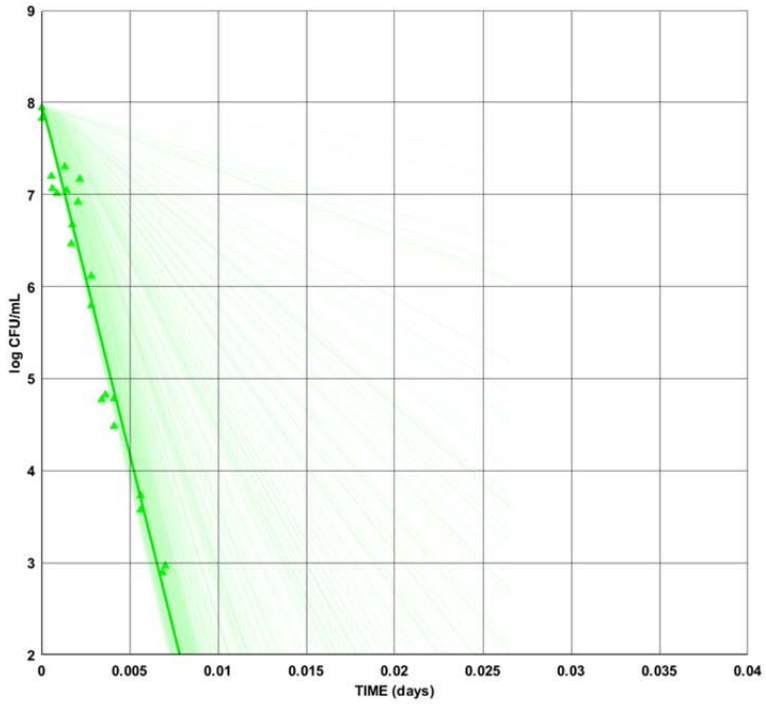
Time (days)	log CFU/mL
0.00000	8.04
0.00171	8.11
0.00342	7.99
0.00358	8.09
0.00509	7.85
0.00526	7.77
0.00693	7.29
0.00700	7.30
0.00867	6.42
0.00874	6.56
0.01031	5.35
0.01045	5.48
0.01386	4.64
0.01390	4.75
0.01724	4.24
0.01738	4.16

[1]Dataset 2 of Figure 10 in Reference [b]

Solution Rate Parameters								SSR
k_1		k_2		k_3		α		
Mean	SE	Mean	SE	Mean	SE	Mean	SE	
636.41	0.21	3063.31	2.47	1424.31	3.83	753.44	0.88	1.05×10^{0}

Figure A-48. Forward Uncertainty Propagation for Dataset 2 in Figure A-10

E. coli in whey protein at T = 50°C, P = 310 MPa



Measurements[1]

Time (days)	log CFU/mL
0.00000	7.94
0.00003	7.83
0.00057	7.19
0.00060	7.06
0.00091	7.01
0.00134	7.30
0.00141	7.04
0.00171	6.47
0.00174	6.67
0.00208	6.91
0.00218	7.17
0.00282	5.79
0.00282	6.11
0.00342	4.77
0.00365	4.82
0.00412	4.48
0.00413	4.78
0.00560	3.57
0.00563	3.73
0.00684	2.90
0.00701	2.97

[1]Dataset 3 of Figure 10 in Reference [b]

Solution Rate Parameters				SSR
a		k_1		
Mean	SE	Mean	SE	
-9999.00	1.63	1770.14	0.06	2.49×10^{0}

Figure A-49. Forward Uncertainty Propagation for Dataset 3 in Figure A-10

APPENDIX B

SOURCE CODE

This appendix contains the computer source code that implements the Frequentist and Bayesian approaches to applying UQ methods to the QCM-4 model. The Frequentist code is written in MATLAB and discussed in Section B.1. The Bayesian code is written in Python and discussed in Section B.2.

B.1 FREQUENTIST SOURCE CODE

The MATLAB code that implements the Frequentist methods was developed under MATLAB R2016a. The code does not use any MATLAB toolboxes, but does use the opensource functions minimize() obtained from the MATLAB Central repository (Reference [dd]), nearcorr() obtained from Reference [ee], and CorrelationMatrix() from Reference [ff]. These three opensource functions are listed in Table B-1. The code fits any of the 39 datasets discussed in Appendix A using the QCM-4 model; estimates the model parameter sensitivities; runs the parameter reduction algorithm; and runs the forward uncertainty propagation. The code is run by activating the main function in the file uq2_run_fqp.m using the instructions given in the file header. Data for the datasets is given in Figures A-11 through A-49, and should be entered as ASCII tab-separated data using the format discussed in the header of uq2_run_fqp.m.

Table B-1. Opensource Functions from MATLAB Central

Function	Description
minimize()	Constrained nonlinear solver
nearcorr()	Finds nearest correlation matrix to given matrix using full or partial eigendecomposition
CorrelationMatrix()	Finds nearest correlation matrix to given matrix using a quadratically convergent Newton method

The MATLAB source code for the Frequentist methods consists of 22 m-files that support 22 functions with one function per file. A description of each function is provided in Table B-2. A listing of the code for the 22 m-files is given on pages B-3 through B-157.

Table B-2. Frequentist UQ MATLAB Functions

Function	Description
uq2_run_fqp()	Main driver program for Frequentist UQ code
qcm_ini_prm()	Initializes the global parameters and structures used in the program
qm4_fit_vqm()	Finds the optimal QCM-4 model parameters by minimizing the SSR for formulations A-E
qm4_nlv_int()	Common interface function for the nonlinear solver using any formulation
qm4_slv_ode()	Solves the QCM-4 system of ODEs using formulation A
qm4_clc_dif()	Calculates the ODE differentials used in qm4_slv_ode()
Bq4_slv_ode()	Solves the QCM-4 system of ODEs using formulations B or C
Bq4_clc_dif()	Calculates the ODE differentials used in Bm4_slv_ode()
Dq4_slv_ode()	Solves the QCM-4 system of ODEs using formulations D or E
Dq4_clc_dif()	Calculates the ODE differentials used in Dm4_slv_ode()
qm4_evl_nam()	Evaluates the alternate (reduced parameter) models to the QCM-4
qm4_slv_sBC()	Solves for QCM-4 sensitivity to model parameters for all formulations B-E
qm4_clc_sdB()	Calculates the formulation B ODE differentials for qm4_slv_sBC()
qm4_clc_sdC()	Calculates the formulation C ODE differentials for qm4_slv_sBC()
qm4_clc_sdD()	Calculates the formulation D ODE differentials for qm4_slv_sBC()
qm4_clc_sdE()	Calculates the formulation E ODE differentials for qm4_slv_sBC()
qm4_cse_sBC()	Calculates standard error for QCM-4 sensitivity using data times
qm4_cse_aBC()	Calculates standard error for QCM-4 sensitivity using uniform times
qm4_qmr_fBC()	Finds model reduction of QCM-4 model using formulations B-E
qm4_cso_fBC()	Calculates sensitivity function, sensitivity matrix, covariance matrix, and standard error of QCM-4 model
qm4_nam_cse()	Calculates sensitivity function, sensitivity matrix, covariance matrix, and standard error of alternate models
qm4_dor_cnv()	Calculates parameter reduction for death-only cases using 1EL model

```
%%%
```

```
                                                % LOCAL DECLARATIONS           %
dl2_sen_fun  = double( 0.0 )          ;         % Sensitivity function (s)     %
dl2_cnt_mat  = double( 0.0 )          ;         % Count matrix                 %
dl2_sen_cnt  = double( 0.0 )          ;         % Count matrix for sensitivity %
dl2_rcv_ini  = double( 0.0 )          ;         % Initial rate constants matrix%
dl2_rcv_mat  = double( 0.0 )          ;         % Final rate constants matrix  %
dl2_cnt_sen  = double( 0.0 )          ;         % Sensitivity of counts        %
dl2_sen_mat  = double( 0.0 )          ;         % Sensitivity matrix (F)       %
dl2_cov_dat  = double( 0.0 )          ;         % Covariance matrix over data  %
dl2_cov_tim  = double( 0.0 )          ;         % Covariance matrix over time  %
dl2_cov      = double( 0.0 )          ;         % Working covariance matrix    %
dl2_cor      = double( 0.0 )          ;         % Working correlation matrix   %
dl2_var      = double( 0.0 )          ;         % Variance matrix              %
dl2_tm1      = double( 0.0 )          ;         % Temporary matrices           %
dl2_tm2      = double( 0.0 )          ;

dl1_sc1_pos  = double( 0.0 )          ;         % Screen 1 position vector     %
dl1_sc2_pos  = double( 0.0 )          ;         % Screen 2 position vector     %
dl1_sc3_pos  = double( 0.0 )          ;         % Screen 3 position vector     %
dl1_sc4_pos  = double( 0.0 )          ;         % Screen 4 position vector     %
dl1_sc5_pos  = double( 0.0 )          ;         % Screen 5 position vector     %
dl1_xlm_vec  = double( 0.0 )          ;         % X-limit vector               %
dl1_ylm_vec  = double( 0.0 )          ;         % Y-limit vector               %
dl1_xtk_vec  = double( 0.0 )          ;         % X-tick vector                %
dl1_ytk_vec  = double( 0.0 )          ;         % Y-tick vector                %
dl1_tim_vec  = double( 0.0 )          ;         % Time vector (days)           %
dl1_cnt_vec  = double( 0.0 )          ;         % Count vector (CFU/mL)        %
dl1_lgc_vec  = double( 0.0 )          ;         % Log count vector (log CFU/mL)%
dl1_ini_rcv  = double( 0.0 )          ;         % Rate constant values         %
dl1_def_rcv  = double( 0.0 )          ;         % Default rate constant values %
dl1_opt_rcv  = double( 0.0 )          ;         % Optimal rate constant values %
dl1_evl_tim  = double( 0.0 )          ;         % Evaluation time (days)       %
dl1_ret_tim  = double( 0.0 )          ;         % Returned time (days)         %
dl1_ret_cnt  = double( 0.0 )          ;         % Returned count (CFU/mL)      %
dl1_min_ssr  = double( 0.0 )          ;         % Minimum SSR vector           %
dl1_std_err  = double( 0.0 )          ;         % Standard error vector        %
dl1_std_er2  = double( 0.0 )          ;         % ALT: Standard error vector   %
dl1_sol_vec  = double( 0.0 )          ;         % Solution vector for 1EL      %
dl1_rnd_ssr  = double( 0.0 )          ;         % Random solutions SSR vector  %
dl1_sig      = double( 0.0 )          ;         % Standard deviation vector    %
dl1_tol      = double( 0.0 )          ;         % Tolerance vector             %
dl1_clr      = double( 0.0 )          ;         % Color vector                 %
dl1_cl2      = double( 0.0 )          ;         % Color vector for propagation %
dl1_xcr      = double( 0.0 )          ;         % X-coordinates                %
dl1_ycr      = double( 0.0 )          ;         % Y-coordinates                %
dl1_tim      = double( 0.0 )          ;         % Time for evaluation          %
dl1_tm1      = double( 0.0 )          ;         % Temporary vectors            %
dl1_tm2      = double( 0.0 )          ;

dl0_min_tim  = double( 0.0 )          ;         % Minimum time (days)          %
dl0_max_tim  = double( 0.0 )          ;         % Maximum time (days)          %
dl0_min_cnt  = double( 0.0 )          ;         % Minimum count (log CFU/mL)   %
dl0_max_cnt  = double( 0.0 )          ;         % Maximum count (log CFU/mL)   %
dl0_int_tim  = double( 0.0 )          ;         % Time interval (days)         %
dl0_tim_inc  = double( 0.0 )          ;         % Time increment (days)        %
dl0_min_xpl  = double( 0.0 )          ;         % Minimum x-coord for plot     %
dl0_max_xpl  = double( 0.0 )          ;         % Maximum x-coord for plot     %
dl0_min_ypl  = double( 0.0 )          ;         % Minimum y-coord for plot     %
dl0_max_ypl  = double( 0.0 )          ;         % Maximum y-coord for plot     %
dl0_sng_mny  = double( 0.0 )          ;         % Sens of g minimum y-coord    %
dl0_sng_mxy  = double( 0.0 )          ;         % Sens of g maximum y-coord    %
dl0_sng_mnx  = double( 0.0 )          ;         % Sens of g minimum x-coord    %
dl0_sng_mxx  = double( 0.0 )          ;         % Sens of g maximum x-coord    %
dl0_max_sen  = double( 0.0 )          ;         % Max sens of f for plots      %
dl0_min_ssr  = double( 0.0 )          ;         % Minimum SSR value            %
dl0_pct_vrv  = double( 0.0 )          ;         % Percent of valid rnd variates%
dl0_pct_nrm  = double( 0.0 )          ;         % Likelihood (%) normal        %
dl0_tst_sym  = double( 0.0 )          ;         % Symmetry test                %
```

```
   dl0_1EL_rat  = double( 0.0 )         ;       % 1EL ratio (alpha/k1)         %
   dl0_tm1      = double( 0.0 )         ;       % Temporary variables          %
   I            = double( 0.0 )         ;       % Index vector                 %

   nl1_bad_rcv  = uint32( 0 )           ;       % Num of bad rate constants    %

   nl0_num_rrc  = uint32( 0 )           ;       % Num of random rate constants %
   nl0_num_tcg  = uint32( 0 )           ;       % Num of total constants gen   %
   nl0_num_rvp  = uint32( 0 )           ;       % Num of rnd variates to plot  %
   nl0_mod_num  = uint32( 0 )           ;       % Temporary model number       %
   nl0_src      = uint32( 0 )           ;       % Status return code           %

   jl1_tm1      = int32( 0 )            ;       % Temporary vector             %

   jl0_dbg_flg  = int32( 0 )            ;       % Debug flag                   %
   jl0_plt_flg  = int32( 0 )            ;       % Plot flag                    %
   jl0_nam_num  = int32( 0 )            ;       % Additional model number      %
   jl0_dfi      = int32( 0 )            ;       % Debug file ID number         %
   jl0_flg      = int32( 0 )            ;       % Test flag                    %
   m            = int32( 0 )            ;       % Indexes/counters             %
   n            = int32( 0 )            ;
   p            = int32( 0 )            ;
   q            = int32( 0 )            ;
   r            = int32( 0 )            ;

   cl2_fil_nam  = char( '' )            ;       % Array of filenames           %

   cl1_inp_fnm  = char( '' )            ;       % Input filename               %
   cl1_tit_str  = char( '' )            ;       % Title string                 %
   cl1_buf      = char( '' )            ;       % String buffer                %

   Sl1_pms      = []                    ;       % Propagated model solutions   %
   Sl1_mrs      = []                    ;       % Reduced model struct vector  %
   Sl1_mrd      = []                    ;       % Death model struct vector    %

   Sl0_ops      = []                    ;       % Optimization structure       %
   Sl0_ocs      = []                    ;       % Output control structure     %
   Sl0_cps      = []                    ;       % Covariance propagation struct%
   Sl0_nam      = []                    ;       % Alternate model structure    %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DECLARATIONS        %
   dc0_qcm_mvc  = double( 1.0e-12 )     ;       % Minimum valid count          %
   dc0_qcm_mvl  = double(   -12.0 )     ;       % Minimum valid log count      %

   dc0_rat_llm  = double( -1.0005 )     ;       % 1EL ratio lower limit        %
   dc0_rat_ulm  = double( -0.9995 )     ;       % 1EL ratio upper limit        %

   dc0_cnf_lim  = double(  0.95   )     ;       % Confidence limit             %

   nc0_ini_rns  = 1234567               ;       % Initial random number seed   %

   jc0_rcv_len  = 4                     ;       % Rate constant vector length  %
   jc0_sen_len  = 2                     ;       % Sensitivity vec/mat length   %
   jc0_num_psv  = 2500                  ;       % Num of propagated sol vectors%
   jc0_num_svu  = 2500                  ;       % Num of sol vectors to update %
   jc0_num_dpp  = 200                   ;       % Num of data points for plots %

   jc0_cov_dat  = 1                     ;       % Covariance matrix over data  %
   jc0_cov_tim  = 2                     ;       % Covariance matrix over time  %

   jc0_drc_opt  = 1                     ;       % Default RCV optimal          %
   jc0_drc_ini  = 2                     ;       % Default RCV initial          %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SET RUN PARAMETERS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Run number/dataset           %
   nc0_run_num  = 1                     ;       % Figure  1 (4 datasets)       %
%   nc0_run_num = 2                     ;       % Figure  2 (6 datasets)       %
```

```
%   nc0_run_num = 3                     ;       % Figure  3 (5 datasets)       %
%   nc0_run_num = 4                     ;       % Figure  4 (4 datasets)       %
%   nc0_run_num = 5                     ;       % Figure  5 (3 datasets)       %
%   nc0_run_num = 6                     ;       % Figure  6 (2 datasets)       %
%   nc0_run_num = 7                     ;       % Figure  7 (3 datasets)       %
%   nc0_run_num = 8                     ;       % Figure  8 (4 datasets)       %
%   nc0_run_num = 9                     ;       % Figure  9 (5 datasets)       %
%   nc0_run_num = 10                    ;       % Figure 10 (3 datasets)       %

   nc0_fil_num  = 1                     ;       % File (dataset) number        %

                                                % Model formulation            %
%   nc0_mod_num = Sl0_qgd.J_QCMB4P      ;       % QCM-4 formulation B          %
%   nc0_mod_num = Sl0_qgd.J_QCMC4P      ;       % QCM-4 formulation C          %
%   nc0_mod_num = Sl0_qgd.J_QCMD4P      ;       % QCM-4 formulation D          %
   nc0_mod_num  = Sl0_qgd.J_QCME4P      ;       % QCM-4 formulation E          %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Solution model               %
%   nc0_sol_mod = Sl0_qgd.J_QCMB4P      ;       % QCM-4 formulation B          %
%   nc0_sol_mod = Sl0_qgd.J_QCMD4P      ;       % QCM-4 formulation D          %
   nc0_sol_mod  = nc0_mod_num           ;

   jc0_cov_flg = jc0_cov_dat            ;       % Covariance matrix over data  %
%   jc0_cov_flg = jc0_cov_tim           ;       % Covariance matrix over time  %

%   jc0_drc_flg = jc0_drc_opt           ;       % Default RCV flag (optimal)   %
   jc0_drc_flg  = jc0_drc_ini           ;       % Default RCV flag (initial)   %

%   jc0_mrd_flg = jc0_cov_dat           ;       % Model reduct data flag (data)%
   jc0_mrd_flg  = jc0_cov_tim           ;       % Model reduct data flag (time)%

%   jc0_ovt_flg = false                 ;       % Override time flag (false)   %
   jc0_ovt_flg  = true                  ;       % Override time flag (time)    %

   cc1_dbg_fil  = 'dbg_run_fep.txt'     ;       % Debug file name              %
   cc1_dsl_str  = 'DATASET '            ;       % Dataset lead string          %
   cc1_tsl_str  = ' - TIME SERIES '     ;       % Time series lead string      %

   cc2_clr      = { ' BLUE    '         , ...   % Color tags                   %
                    ' RED     '         , ...
                    ' GREEN   '         , ...
                    ' BLACK   '         , ...
                    ' MAGENTA '         , ...
                    ' GRAY    '         }       ;
                                                % Dataset 1 filenames          %
   cc2_fil_nm1  = { 'F1_aw_79.txt'      , ...           % 8 measurements       %
                    'F1_aw_84.txt'      , ...           % 17 measurements      %
                    'F1_aw_87.txt'      , ...           % 24 measurements      %
                    'F1_aw_91.txt'      }       ;       % 18 measurements      %
                                                % Dataset 2 filenames          %
   cc2_fil_nm2  = { 'F2-15.txt'         , ...           % 14 measurements      %
                    'F2-20.txt'         , ...           % 30 measurements      %
                    'F2-25.txt'         , ...           % 26 measurements      %
                    'F2-30.txt'         , ...           % 25 measurements      %
                    'F2-35.txt'         , ...           % 27 measurements      %
                    'F2-40.txt'         }       ;       % 18 measurements      %
                                                % Dataset 3 filenames          %
   cc2_fil_nm3  = { 'F3-538.txt'        , ...           % 23 measurements      %
                    'F3-536.txt'        , ...           % 20 measurements      %
                    'F3-531.txt'        , ...           % 19 measurements      %
                    'F3-519.txt'        , ...           % 16 measurements      %
                    'F3-497.txt'        }       ;       % 12 measurements      %
                                                % Dataset 4 filenames          %
   cc2_fil_nm4  = { 'F4-89.txt'         , ...           % 24 measurements      %
                    'F4-87.txt'         , ...           % 27 measurements      %
                    'F4-86.txt'         , ...           % 13 measurements      %
                    'F4-84.txt'         }       ;       % 25 measurements      %
```

```
                                                % Dataset 5 filenames          %
   cc2_fil_nm5  = { 'F5-92.txt'         , ...           % 17 measurements      %
                    'F5-90.txt'         , ...           % 22 measurements      %
                    'F5-88.txt'         }       ;       % 18 measurements      %
                                                % Dataset 6 filenames          %
   cc2_fil_nm6  = { 'F6-87.txt'         , ...           % 14 measurements      %
                    'F6-85.txt'         }       ;       % 15 measurements      %
                                                % Dataset 7 filenames          %
   cc2_fil_nm7  = { 'F7-SA.txt'         , ...           % 25 measurements      %
                    'F7-LS.txt'         , ...           % 22 measurements      %
                    'F7-EC.txt'         }       ;       % 18 measurements      %
                                                % Dataset 8 filenames          %
   cc2_fil_nm8  = { 'F8-0.txt'          , ...           % 23 measurements      %
                    'F8-1.txt'          , ...           % 15 measurements      %
                    'F8-2.txt'          , ...           % 17 measurements      %
                    'F8-3.txt'          }       ;       % 9 measurements       %
                                                % Dataset 9 filenames          %
   cc2_fil_nm9  = { 'F9-207.txt'        , ...           % 13 measurements      %
                    'F9-242.txt'        , ...           % 14 measurements      %
                    'F9-276.txt'        , ...           % 17 measurements      %
                    'F9-311.txt'        , ...           % 20 measurements      %
                    'F9-345.txt'        }       ;       % 15 measurements      %
                                                % Dataset 10 filenames         %
   cc2_fil_nm0  = { 'F10-30.txt'        , ...           % 21 measurements      %
                    'F10-40.txt'        , ...           % 16 measurements      %
                    'F10-50.txt'        }       ;       % 21 measurements      %

                                                % Dataset 3 TS 4 special       %
   cc2_fil_34A  = { 'F3-519.txt'        , ...           % 16 measurements      %
                    'F3-519_A1.txt'     , ...           % 16 measurements      %
                    'F3-519_A2.txt'     , ...           % 16 measurements      %
                    'F3-519_A3.txt'     , ...           %  9 measurements      %
                    'F3-519_A4.txt'     }       ;       %  9 measurements      %


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
                                                % Set output control structure %
   Sl0_ocs.j0dbgf = Sl0_qgd.J_DBGLV3    ;       % Debug flag                   %
   Sl0_ocs.j0dfid = 1                   ;       % Debug fid (1=screen; []=file)%
%   Sl0_ocs.j0dfid = []                 ;       % Debug fid (1=screen; []=file)%
   Sl0_ocs.c1dfnm = cc1_dbg_fil         ;       % Debug filename               %
   Sl0_ocs.j0pltf = true                ;       % Plot flag                    %

   jl0_plt_flg    = Sl0_ocs.j0pltf      ;       % Set local plot flag          %
   jl0_dbg_flg    = Sl0_ocs.j0dbgf      ;       % Set local debug flag         %

   if ( isempty( Sl0_ocs.j0dfid ) )             % Open debug file and set fid  %
      Sl0_ocs.j0dfid = fopen( Sl0_ocs.c1dfnm , 'w+' )   ;
      jl0_dfi        = Sl0_ocs.j0dfid                   ;

      if ( jl0_dfi == -1 )
         error( 'uq2_run_fqp: Cannot open debug file' ) ;
      else
         fprintf( jl0_dfi , '%s\n' , datetime('now') )  ;
      end
   else
      jl0_dfi = 1                       ;       % Set local debug fid          %
   end

   if ( Sl0_ocs.j0dbgf > Sl0_qgd.J_DBGLV0 )     % Print function name         %
      fprintf( jl0_dfi , '*** uq2_run_fqp ***\n' )      ;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Initialize data              %
   rng( nc0_ini_rns )                   ;       % Initialize random numbers    %
   dl1_opt_rcv = []                     ;       % Init optimal rate constants  %
```

```
if ( nc0_run_num == 1 )                      % Initialize run constants     %
   cl2_fil_nam    = cc2_fil_nm1                                   ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =    2.0000 ;               % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =   20.0000 ;
   dl0_min_cnt    =    0.0000 ; dl0_max_cnt    =   10.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.2e+01 , -1.2e+01 , -1.2e+01 , -1.2e+01 ] ;
      Sl0_ops.d1maxk = [  1.2e+01 ,  1.2e+01 ,  1.2e+01 ,  1.2e+01 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.2e+01 , -1.2e+01 , -1.2e+01 , -1.2e+01 ] ;
      Sl0_ops.d1maxk = [  1.2e+01 ,  1.2e+01 ,  1.2e+01 ,  1.2e+01 ] ;
   else                                              % Use alpha = k2 - k4  %
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+03 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   end

   if      ( nc0_fil_num == 1 )                 % Set time series specific     %
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -5.0000 ; dl0_sng_mxy    =    5.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   20.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 31.000 , 100.00 , 30.000 ] ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -5.0000 ; dl0_sng_mxy    =    5.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   20.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 21.000 , 30.000 , 20.000 ] ;
   elseif ( nc0_fil_num == 3 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =   -5.0000 ; dl0_sng_mxy    =    5.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   20.0000 ;
      dl1_ini_rcv    =  [ 0.5000 , 5.0000 , 20.000 , 4.0000 ] ;
   elseif ( nc0_fil_num == 4 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -10.0000 ; dl0_sng_mxy    =   10.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   20.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 5.0000 , 10.000 , 4.0000 ] ;
   end
elseif ( nc0_run_num == 2 )
   cl2_fil_nam    = cc2_fil_nm2                                   ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =   10.0000 ;               % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =   70.0000 ;
   dl0_min_cnt    =    2.0000 ; dl0_max_cnt    =    9.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -2.5e+01 , -2.5e+01 , -2.5e+01 , -2.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.3e+01 ,  1.3e+01 ,  1.3e+01 ,  1.3e+01 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.2e+01 , -1.2e+00 , -1.2e+00 , -1.2e+00 ] ;
      Sl0_ops.d1maxk = [  1.3e+01 ,  1.3e+01 ,  1.3e+01 ,  1.3e+01 ] ;
   else                                              % Use alpha = k2 - k4  %
```

```
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+03 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   end

   if     ( nc0_fil_num == 1 )              % Set time series specific     %
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000        ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   70.0000        ;
      dl1_ini_rcv    =  [ 0.5000 , 1.0000 , 20.000 , 2.0000 ]        ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000        ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   70.0000        ;
      dl1_ini_rcv    =  [ 0.5000 , 2.0000 , 10.000 , 1.0000 ]        ;

      if ( nc0_sol_mod == Sl0_qgd.J_QCME4P )
         Sl0_ops.d1mink = [ -2.0e+01 , -1.2e+00 , -1.2e+00 , -1.2e+00 ] ;
      end
   elseif ( nc0_fil_num == 3 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000        ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   70.0000        ;
      dl1_ini_rcv    =  [ 1.0000 , 3.0000 , 20.000 , 1.0000 ]        ;
   elseif ( nc0_fil_num == 4 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000        ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   70.0000        ;
      dl1_ini_rcv    =  [ 1.0000 , 4.0000 , 25.000 , 1.0000 ]        ;
   elseif ( nc0_fil_num == 5 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000        ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   70.0000        ;
      dl1_ini_rcv    =  [ 1.0000 , 5.0000 , 30.000 , 1.0000 ]        ;
   elseif ( nc0_fil_num == 6 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000        ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   70.0000        ;
      dl1_ini_rcv    =  [ 1.0000 , 6.0000 , 35.000 , 1.0000 ]        ;
   end
elseif ( nc0_run_num == 3 )
   cl2_fil_nam    = cc2_fil_nm3                              ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =    1.0000 ;              % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =   12.0000 ;
   dl0_min_cnt    =    1.0000 ; dl0_max_cnt    =    7.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.3e+01 , -1.3e+01 , -1.3e+01 , -1.3e+01 ] ;
      Sl0_ops.d1maxk = [  1.3e+01 ,  1.3e+01 ,  1.3e+01 ,  1.3e+01 ] ;
      dl0_sng_mny    =  -50.0000 ; dl0_sng_mxy    =   50.0000 ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.3e+01 , -1.3e+01 , -1.3e+01 , -1.3e+01 ] ;
      Sl0_ops.d1mink = [ -2.0e+00 , -2.0e+00 , -2.0e+00 , -2.0e+00 ] ;
      Sl0_ops.d1maxk = [  1.3e+01 ,  1.3e+01 ,  1.3e+01 ,  1.3e+01 ] ;
      dl0_sng_mny    =  -50.0000 ; dl0_sng_mxy    =   50.0000 ;
   else                                              % Use alpha = k2 - k4  %
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+03 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   end

   if     ( nc0_fil_num == 1 )              % Set time series specific     %
```

```
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   12.0000 ;
      dl1_ini_rcv    =  [ 0.5000 , 50.000 , 5.0000 , 40.000 ] ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   12.0000 ;
      dl1_ini_rcv    =  [ 0.2000 , 25.000 ,  8.000 , 23.000 ] ;
   elseif ( nc0_fil_num == 3 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   12.0000 ;
      dl1_ini_rcv    =  [ 0.1000 , 48.000 ,  7.000 , 47.000 ] ;
   elseif ( nc0_fil_num == 4 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   12.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 1.0000 , 5.0000 , 2.0000 ] ;
   elseif ( nc0_fil_num == 5 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   12.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 1.0000 , 10.000 , 3.0000 ] ;
   end
elseif ( nc0_run_num == 4 )
   cl2_fil_nam    = cc2_fil_nm4                              ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =    2.0000 ;              % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =   20.0000 ;
   dl0_min_cnt    =    3.0000 ; dl0_max_cnt    =   10.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -2.5e+01 , -2.5e+01 , -2.5e+01 , -2.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -2.5e+01 , -2.5e+01 , -2.5e+01 , -2.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   else                                              % Use alpha = k2 - k4  %
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+03 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   end

   if     ( nc0_fil_num == 1 )              % Set time series specific     %
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   20.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 7.0000 , 10.000 , 2.0000 ] ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   20.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 4.0000 , 10.000 , 1.0000 ] ;
   elseif ( nc0_fil_num == 3 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   20.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 6.0000 , 15.000 , 2.0000 ] ;
   elseif ( nc0_fil_num == 4 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
```

```
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   20.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 2.0000 , 5.0000 , 3.0000 ] ;
   end
elseif ( nc0_run_num == 5 )
   cl2_fil_nam    = cc2_fil_nm5                              ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =    5.0000 ;              % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =   25.0000 ;
   dl0_min_cnt    =    1.0000 ; dl0_max_cnt    =    9.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.2e+01 , -1.2e+01 , -1.2e+01 , -1.2e+01 ] ;
   Sl0_ops.d1mink = [ -2.5e+01 , -2.5e+01 , -2.5e+01 , -2.5e+01 ]    ;
      Sl0_ops.d1maxk = [  1.2e+01 ,  1.2e+01 ,  1.2e+01 ,  1.2e+01 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.2e+01 , -1.2e+01 , -1.2e+01 , -1.2e+01 ] ;
   Sl0_ops.d1mink = [ -2.5e+01 , -2.5e+01 , -2.5e+01 , -2.5e+01 ]    ;
      Sl0_ops.d1maxk = [  1.2e+01 ,  1.2e+01 ,  1.2e+01 ,  1.2e+01 ] ;
   else                                              % Use alpha = k2 - k4  %
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+03 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   end

   if     ( nc0_fil_num == 1 )              % Set time series specific     %
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   25.0000 ;
      dl1_ini_rcv    =  [ 0.5000 , 3.0000 , 10.000 , 1.0000 ] ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   25.0000 ;
      dl1_ini_rcv    =  [ 0.5000 , 1.0000 , 10.000 , 3.0000 ] ;
   elseif ( nc0_fil_num == 3 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   25.0000 ;
      dl1_ini_rcv    =  [ 0.5000 , 1.0000 , 10.000 , 3.0000 ] ;
   end
elseif ( nc0_run_num == 6 )
   cl2_fil_nam    = cc2_fil_nm6                              ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =    1.0000 ;              % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =    8.0000 ;
   dl0_min_cnt    =    1.0000 ; dl0_max_cnt    =    4.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.2e+01 , -1.2e+01 , -1.2e+01 , -1.2e+01 ] ;
      Sl0_ops.d1maxk = [  1.2e+01 ,  1.2e+01 ,  1.2e+01 ,  1.2e+01 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.2e+01 , -1.2e+01 , -1.2e+01 , -1.2e+01 ] ;
      Sl0_ops.d1maxk = [  1.2e+01 ,  1.2e+01 ,  1.2e+01 ,  1.2e+01 ] ;
   else                                              % Use alpha = k2 - k4  %
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+03 ] ;
```

```
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   end

   if     ( nc0_fil_num == 1 )              % Set time series specific     %
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -5.0000 ; dl0_sng_mxy    =    5.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    8.0000 ;
      dl1_ini_rcv    =  [ 0.5000 , 1.0000 , 10.000 , 3.0000 ] ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -5.0000 ; dl0_sng_mxy    =    5.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    8.0000 ;
      dl1_ini_rcv    =  [ 0.5000 , 1.0000 , 10.000 , 3.0000 ] ;
   end
elseif ( nc0_run_num == 7 )
   cl2_fil_nam    = cc2_fil_nm7                                ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =    2.0000 ;              % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =   18.0000 ;
   dl0_min_cnt    =    3.0000 ; dl0_max_cnt    =   10.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -2.5e+01 , -2.5e+01 , -2.5e+01 , -2.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -2.5e+01 , -2.5e+01 , -2.5e+01 , -2.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   else                                              % Use alpha = k2 - k4  %
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+03 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   end

   if     ( nc0_fil_num == 1 )              % Set time series specific     %
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -10.0000 ; dl0_sng_mxy    =   10.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   18.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 5.0000 , 5.0000 , 1.0000 ] ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -10.0000 ; dl0_sng_mxy    =   10.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   18.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 5.0000 , 5.0000 , 3.0000 ] ;
   elseif ( nc0_fil_num == 3 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -10.0000 ; dl0_sng_mxy    =   10.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   18.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 5.0000 , 10.000 , 4.0000 ] ;
   end
elseif ( nc0_run_num == 8 )
   cl2_fil_nam    = cc2_fil_nm8                                ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =    5.0000 ;              % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =   15.0000 ;
   dl0_min_cnt    =    3.0000 ; dl0_max_cnt    =   10.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
```

```
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -2.5e+01 , -2.5e+01 , -2.5e+01 , -2.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -2.5e+01 , -2.5e+01 , -2.5e+01 , -2.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   else                                              % Use alpha = k2 - k4  %
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+03 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   end

   if     ( nc0_fil_num == 1 )              % Set time series specific     %
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   15.0000 ;
      dl1_ini_rcv    =  [ 10.000 , 5.0000 , 4.0000 , 1.0000 ] ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   15.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 2.0000 , 10.000 , 0.1000 ] ;
   elseif ( nc0_fil_num == 3 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   15.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 2.0000 , 20.000 , 0.0000 ] ;
   elseif ( nc0_fil_num == 4 )
      dl0_max_sen    =   1.0e+08 ;
      dl0_sng_mny    =  -15.0000 ; dl0_sng_mxy    =   15.0000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =   15.0000 ;
      dl1_ini_rcv    =  [ 1.0000 , 1.0000 , 10.000 , 0.1000 ] ;
   end
elseif ( nc0_run_num == 9 )
   cl2_fil_nam    = cc2_fil_nm9                              ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =    0.0050 ;              % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =    0.0400 ;
   dl0_min_cnt    =    1.0000 ; dl0_max_cnt    =    9.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+04 ,  1.0e+04 ,  1.0e+04 ,  1.0e+04 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.5e+01 , -1.5e+01 , -1.5e+01 , -1.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.5e+01 , -1.5e+01 , -1.5e+01 , -1.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   else                                              % Use alpha = k2 - k4  %
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+04 ] ;
      Sl0_ops.d1maxk = [  1.0e+04 ,  1.0e+04 ,  1.0e+04 ,  1.0e+04 ] ;
   end

   if     ( nc0_fil_num == 1 )              % Set time series specific     %
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -0.1000 ; dl0_sng_mxy    =    0.1000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    0.0400 ;
      dl1_ini_rcv    =  [ 500.0000 , 850.000 ,  50.00 , 840.000 ] ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -0.1000 ; dl0_sng_mxy    =    0.1000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    0.0400 ;
      dl1_ini_rcv    =  [ 10.000 , 20.000 , 100.00 , 30.000 ] ;
```

```
   elseif ( nc0_fil_num == 3 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -0.1000 ; dl0_sng_mxy    =    0.1000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    0.0400 ;
      dl1_ini_rcv    =  [ 20.000 , 20.000 , 100.00 , 40.000 ] ;
   elseif ( nc0_fil_num == 4 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -0.1000 ; dl0_sng_mxy    =    0.1000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    0.0400 ;
      dl1_ini_rcv    =  [ 20.000 , 20.000 , 100.00 , 50.000 ] ;
   elseif ( nc0_fil_num == 5 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -0.1000 ; dl0_sng_mxy    =    0.1000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    0.0400 ;
      dl1_ini_rcv    =  [ 20.000 , 20.000 , 100.00 , 60.000 ] ;
   end
elseif ( nc0_run_num == 10 )
   cl2_fil_nam    = cc2_fil_nm0                              ;
   cl1_tit_str    = [ cc1_dsl_str , num2str( nc0_run_num ) , ...
                      cc1_tsl_str , num2str( nc0_fil_num ) ] ;

   dl0_tim_inc    =    0.0050 ;              % Plot limits (time/count)     %
   dl0_min_tim    =    0.0000 ; dl0_max_tim    =    0.0400 ;
   dl0_min_cnt    =    2.0000 ; dl0_max_cnt    =    9.0000 ;
   Sl0_ops.d0atol =   1.0e-12 ; Sl0_ops.d0rtol =   1.0e-04 ;

   if ( ( nc0_sol_mod == Sl0_qgd.J_QCM4PR ) || ...   % 4 k-values           %
        ( nc0_sol_mod == Sl0_qgd.J_QCMB4P ) )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+04 ,  1.0e+04 ,  1.0e+04 ,  1.0e+04 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCMD4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.5e+01 , -1.5e+01 , -1.5e+01 , -1.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   elseif ( nc0_sol_mod == Sl0_qgd.J_QCME4P )        % Log of rate constants%
      Sl0_ops.d1mink = [ -1.5e+01 , -1.5e+01 , -1.5e+01 , -1.5e+01 ] ;
      Sl0_ops.d1maxk = [  1.5e+01 ,  1.5e+01 ,  1.5e+01 ,  1.5e+01 ] ;
   else                                              % Use alpha = k2 - k4  %
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 , -1.0e+04 ] ;
      Sl0_ops.d1maxk = [  1.0e+04 ,  1.0e+04 ,  1.0e+04 ,  1.0e+04 ] ;
   end

   if     ( nc0_fil_num == 1 )              % Set time series specific     %
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -0.1000 ; dl0_sng_mxy    =    0.1000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    0.0400 ;
      dl1_ini_rcv    =  [ 400.00 , 900.00 , 800.00 , 700.00 ] ;
   elseif ( nc0_fil_num == 2 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -0.1000 ; dl0_sng_mxy    =    0.1000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    0.0400 ;
      dl1_ini_rcv    =  [ 400.00 , 900.00 , 800.00 , 700.00 ] ;

      if ( nc0_sol_mod == Sl0_qgd.J_QCME4P )
          Sl0_ops.d1mink = [ -3.0e+00 , -3.0e+00 , -3.0e+00 , -3.0e+00 ]     ;
      end
   elseif ( nc0_fil_num == 3 )
      dl0_max_sen    =   1.0e+05 ;
      dl0_sng_mny    =   -0.1000 ; dl0_sng_mxy    =    0.1000 ;
      dl0_sng_mnx    =    0.0000 ; dl0_sng_mxx    =    0.0400 ;
      dl1_ini_rcv    =  [ 800.00 , 1.0000 , 100.00 , 800.00 ] ;
   end
end

if ( ( nc0_sol_mod == Sl0_qgd.J_QCMD4P ) && ...      % Log of rate constants%
     ( ~isempty( dl1_ini_rcv )          ) )
   dl1_ini_rcv = max( dl1_ini_rcv , 1.0e-20 )                          ;
   dl1_ini_rcv = log( dl1_ini_rcv )                                    ;
end
```

```
if ( ( nc0_sol_mod == Sl0_qgd.J_QCME4P ) && ...      % Log of rate constants%
     ( ~isempty( dl1_ini_rcv )         ) )
   dl1_ini_rcv = max( dl1_ini_rcv , 1.0e-20 )                        ;
   dl1_ini_rcv = log( dl1_ini_rcv )                                  ;
end
                                             % Set to 1EL for death only    %
if ( nc0_run_num < 9 )
   Sl0_nam.d1mink = [ -1.0e+06 ,  1.0e-05 ,      [] ,      [] ]      ;
   Sl0_nam.d1maxk = [ -1.0e-05 ,  1.0e+06 ,      [] ,      [] ]      ;
   Sl0_nam.d1ircv = [  0.0000  ,   1.0000 ,     NaN ,     NaN ]      ;
else
   Sl0_nam.d1mink = [ -1.0e+07 ,  1.0e-05 ,      [] ,      [] ]      ;
   Sl0_nam.d1maxk = [ -1.0e-05 ,  1.0e+07 ,      [] ,      [] ]      ;
   Sl0_nam.d1ircv = [  0.0000  ,   1.0000 ,     NaN ,     NaN ]      ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Finish initializations       %
if ( ( ~isempty( nc0_fil_num )            ) && ...  % Check for error      %
     ( nc0_fil_num > length( cl2_fil_nam ) ) )
   no0_src = Sl0_qgd.E_INCARG        ;
   return                            ;
end

dl0_min_xpl = dl0_min_tim                    ; % X-coordinate plot limits   %
dl0_max_xpl = dl0_max_tim                    ;
dl0_min_ypl = dl0_min_cnt                    ; % Y-coordinate plot limits   %
dl0_max_ypl = dl0_max_cnt                    ;
dl0_int_tim = dl0_max_tim / jc0_num_dpp      ; % Generate points to plot    %

dl2_rcv_ini = zeros( 1 , jc0_rcv_len )       ; % Initialize output buffers  %
dl2_rcv_mat = zeros( 1 , jc0_rcv_len )       ;
dl1_min_ssr = zeros( 1 , 1 )                 ;

dl1_evl_tim = [           0 : ...            % Evaluation times (days)      %
                dl0_int_tim : ...
                dl0_max_tim ]'               ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Initialize graphics          %
dl1_scr_siz  = get( 0 , 'ScreenSize' )       ;       % Get screensize       %

dl1_sc1_pos = [                              ...     % Screen position      %
                   40                        ...
                   70                        ...
                   dl1_scr_siz(3) / 2.00     ...
                   dl1_scr_siz(4) / 1.30     ]       ;
dl1_sc2_pos = [                              ...
                   140                       ...
                   90                        ...
                   dl1_scr_siz(3) / 2.00     ...
                   dl1_scr_siz(4) / 1.30     ]       ;
dl1_sc3_pos = [                              ...
                   240                       ...
                   110                       ...
                   dl1_scr_siz(3) / 2.00     ...
                   dl1_scr_siz(4) / 1.30     ]       ;
dl1_sc4_pos = [                              ...
                   340                       ...
                   130                       ...
                   dl1_scr_siz(3) / 2.00     ...
                   dl1_scr_siz(4) / 1.30     ]       ;
dl1_sc5_pos = [                              ...
                   440                       ...
                   150                       ...
                   dl1_scr_siz(3) / 2.00     ...
                   dl1_scr_siz(4) / 1.30     ]       ;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%   The integer variables m and q are used as indexes throughout this func- %
% tion.  Once set, they should not be altered or used as loop indexes.         %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Loop through dataset files   %
m           = nc0_fil_num           ;        % Set file number index        %
cl1_inp_fnm = cl2_fil_nam{m}        ;        % Get filename                 %

 [                     ...                   % Get data from text file      %
     dl1_xcr         , ...
     dl1_ycr           ...
 ] =                   ...
textread               ...
 (                     ...
     cl1_inp_fnm     , ...
     '%f %f'           ...
 )                                  ;

dl1_xcr = dl1_xcr'                  ;        % Transpose data               %
dl1_ycr = dl1_ycr'                  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Initialize QCM parameters    %
dl1_tim_vec    = dl1_xcr            ;        % Get data for QCM             %
dl1_cnt_vec    = 10.^dl1_ycr        ;        % Convert to counts from log   %
dl1_ini_cnt    = [ dl1_cnt_vec(1) , ...      % Set initial counts (x(0))    %
                   0              , ...
                   0              , ...
                   0              ] ;
dl1_tim_vec(1) = 0.0                ;        % First time must be zero      %

[ dl1_tim_vec , I ] = sort( dl1_tim_vec )    ;
dl1_cnt_vec         = dl1_cnt_vec( I )       ;

for ( n = 2:length(dl1_tim_vec) )            % Prevent same times           %
   if ( dl1_tim_vec(n) <= dl1_tim_vec(n-1) )
      dl1_tim_vec(n) = dl1_tim_vec(n-1) + 1.0e-6     ;
   end
end                                          % End for ( n )                %

                                             % Set line color               %
if ( m == 1 ) , dl1_clr = [ 0.00 0.00 0.90 ]     ; end       ;
if ( m == 2 ) , dl1_clr = [ 0.90 0.00 0.00 ]     ; end       ;
if ( m == 3 ) , dl1_clr = [ 0.00 0.90 0.00 ]     ; end       ;
if ( m == 4 ) , dl1_clr = [ 0.00 0.00 0.00 ]     ; end       ;
if ( m == 5 ) , dl1_clr = [ 0.90 0.00 0.90 ]     ; end       ;
if ( m == 6 ) , dl1_clr = [ 0.50 0.50 0.50 ]     ; end       ;
                                             % Set distribution line color  %
if ( m == 1 ) , dl1_cl2 = [ 0.75 0.75 1.00 0.1 ] ; end       ;
if ( m == 2 ) , dl1_cl2 = [ 1.00 0.75 0.75 0.1 ] ; end       ;
if ( m == 3 ) , dl1_cl2 = [ 0.75 1.00 0.75 0.1 ] ; end       ;
if ( m == 4 ) , dl1_cl2 = [ 0.75 0.75 0.75 0.1 ] ; end       ;
if ( m == 5 ) , dl1_cl2 = [ 0.90 0.60 0.90 0.2 ] ; end       ;
if ( m == 6 ) , dl1_cl2 = [ 0.85 0.85 0.85 0.1 ] ; end       ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%    Formulation E is run as Formulation D  to get a solution. If  alpha is %
% negative, then Formulation  E is permanently changed to D.  Otherwise the %
% rate parameters are set to [ln(k1) ln(k2) ln(k3) ln(k2-k4)].                 %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Run QCM algorithm             %
 [                     ...                   % Fit QCM-4 to data             %
     dl1_ret_tim     , ...                   % Output vector of times (day) %
```

```
     dl1_ret_cnt     , ...                   % Output count vector (CFU/mL) %
     dl2_cnt_mat     , ...                   % Count matrix from ODE        %
     dl1_opt_rcv     , ...                   % Final rate constants (k-vals)%
     dl0_min_ssr     , ...                   % Minimum SSR (CFU/mL)         %
     nl0_src           ...                   % Status return code           %
 ] =                   ...
qm4_fit_vqm            ...
 (                     ...
     dl1_tim_vec     , ...                   % Input time vector (day)      %
     dl1_cnt_vec     , ...                   % Input count vector (CFU/mL)  %
     dl1_ini_rcv     , ...                   % Initial rate constants (k)   %
     dl1_ini_cnt     , ...                   % Initial count vector (x0)    %
     []              , ...                   % Rate cnst solution index vec %
     dl1_evl_tim     , ...                   % Evaluation times             %
     nc0_sol_mod     , ...                   % Model number                 %
     Sl0_ops         , ...                   % Optimization parameters      %
     Sl0_ocs           ...                   % Debug flag                   %
 )                                   ;

if ( nc0_mod_num  == Sl0_qgd.J_QCME4P )      % Modify for formulation E     %
   if ( dl1_opt_rcv(4) >  dl1_opt_rcv(2) )           % Death only           %
      nc0_mod_num = Sl0_qgd.J_QCMD4P         ;       % Set to formulation D %
   else                                              % Get ln(alpha)        %
      dl0_tm1        = exp( dl1_opt_rcv(2) ) - ...
                       exp( dl1_opt_rcv(4) ) ;
      dl1_opt_rcv(4) = log( dl0_tm1 )         ;
   end
end

dl2_rcv_ini(1,:) = dl1_ini_rcv       ;       % Store initial rate constants %
dl2_rcv_mat(1,:) = dl1_opt_rcv       ;       % Store final rate constants   %
dl1_min_ssr(1)   = dl0_min_ssr       ;       % Store minimum SSRs           %

                                             % Get log10 values             %
dl1_lgc_vec = log10( dl1_ret_cnt )   ;       % Count vector                 %

                                             % Set max index based on min   %
q = find( dl1_lgc_vec < dc0_qcm_mvl , 1 , 'first' )  ;

if ( isempty( q ) )
   q = length( dl1_ret_tim )         ;
elseif ( q > 1 )
   q = q - 1                         ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%    In the reduced QCM formulations (B-E), the sensitivity is only depend- %
% ent on the counts for  M* and A  which are the only values  solved for in %
% the  system of ODEs.  The count matrix dl2_cnt_sen holds count values for %
% M, M*, A, and D where D is set to 10e-15. The values needed for sensitiv- %
% ity calculations are then dl2_cnt_mat(:,2:3).                              %
%                                                                              %
%    The call to qm4_slv_sBC is only needed for the sensitivity plots.       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Calculate sensitivities      %
dl2_sen_cnt = zeros( 2 , q )                 ;
dl2_sen_cnt = dl2_cnt_mat(1:q,2:3)'          ;

 [                     ...                   % Calc QCM solution sensitivity%
     dl2_sen_fun     , ...                   % Sensitivity function (s)     %
     ~               , ...                   % Count sensitivity matrix     %
     dl2_sen_mat     , ...                   % Sensitivity matrix (F)       %
     nl0_src           ...                   % Status return code           %
 ] =                   ...
qm4_slv_sBC            ...
 (                     ...
```

```
     dl1_opt_rcv     , ...                    % Solution rate constants      %
     dl1_ini_cnt(1)  , ...                    % Initial count of M           %
     dl1_ret_tim(1:q), ...                    % Time vector (day)            %
     dl2_sen_cnt     , ...                    % Solution counts (M* and A)   %
     nc0_mod_num     , ...                    % Model formulation number     %
     jl0_dbg_flg       ...                    % Debug flag                   %
 )                            ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % Calculate SE and covariance  %
 [                     ...                    % Calc std error over data     %
     dl1_std_err     , ...                    % Standard error               %
     dl2_cov_dat     , ...                    % Covariance matrix over data  %
     nl0_src           ...                    % Status return code           %
 ] =                   ...
qm4_cse_sBC            ...
 (                     ...
     dl1_opt_rcv     , ...                    % Solution rate constants      %
     dl1_tim_vec     , ...                    % Vector of count times (day)  %
     dl1_cnt_vec     , ...                    % Vector of measured counts    %
     nc0_mod_num     , ...                    % Model formulation number     %
     Sl0_ocs           ...                    % Output control structure     %
 )                            ;

 [                     ...                    % Calc std error over time     %
     dl1_std_er2     , ...                    % Standard error               %
     dl2_cov_tim     , ...                    % Covariance matrix over time  %
     nl0_src           ...                    % Status return code           %
 ] =                   ...
qm4_cse_aBC            ...
 (                     ...
     dl1_opt_rcv     , ...                    % Solution rate constants      %
     dl1_tim_vec     , ...                    % Vector of count times (day)  %
     dl1_cnt_vec     , ...                    % Vector of measured counts    %
     dl1_ret_tim(1:q), ...                    % Evaluation times (day)       %
     nc0_mod_num     , ...                    % Model formulation number     %
     Sl0_ocs           ...                    % Output control structure     %
 )                            ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if ( jc0_drc_flg == jc0_drc_opt )             % Default RCV flag (optimal)   %
   dl1_def_rcv = dl1_opt_rcv         ;
else                                          % Default RCV flag (initial)   %
   dl1_def_rcv = dl1_ini_rcv         ;
end

if ( jc0_mrd_flg == jc0_cov_dat )             % Model reduction by data      %
    [                  ...
     Sl1_mrs         , ...                    % Model reduction struct vector%
     nl0_src           ...                    % Status return code           %
    ] =                ...
   qm4_qmr_fBC         ...
    (                  ...
     dl1_def_rcv     , ...                    % Default rate constants       %
     dl1_tim_vec     , ...                    % Vector of count times (day)  %
     dl1_cnt_vec     , ...                    % Vector of measured counts    %
     []              , ...                    % Evaluation times (optional)  %
     nc0_mod_num     , ...                    % Model formulation number     %
     []              , ...                    % Reduction for death          %
     Sl0_ops         , ...                    % Optimization parameters      %
     Sl0_ocs           ...                    % Output control structure     %
    )                            ;
else                                          % Model reduction by time      %
    [                  ...
     Sl1_mrs         , ...                    % Model reduction struct vector%
     nl0_src           ...                    % Status return code           %
    ] =                ...
```

```
   qm4_qmr_fBC         ...
    (                  ...
     dl1_def_rcv     , ...                    % Default rate constants       %
     dl1_tim_vec     , ...                    % Vector of count times (day)  %
     dl1_cnt_vec     , ...                    % Vector of measured counts    %
     dl1_ret_tim(1:q), ...                    % Evaluation times (optional)  %
     nc0_mod_num     , ...                    % Model formulation number     %
     []              , ...                    % Reduction for death          %
     Sl0_ops         , ...                    % Optimization parameters      %
     Sl0_ocs           ...                    % Output control structure     %
    )                       ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % Generate random solutions    %
if ( jc0_cov_flg == jc0_cov_dat )
   Sl0_cps.d2covm = dl2_cov_dat               ; % Covariance matrix          %
   Sl0_cps.d1time = dl1_tim_vec'              ; % Time vector                %
   Sl0_cps.d1rcnt = Sl0_cps.d1time            ; % Rate count vector (dummy)  %
   Sl0_cps.d1lgrc = Sl0_cps.d1time            ; % Log10 of rate count (dummy)%
   Sl0_cps.d1orcv = dl1_opt_rcv               ; % Optimal rate constant vec  %
   Sl0_cps.d1rrcv = zeros( 1 , jc0_rcv_len )  ; % Random rate constant vec   %
   Sl0_cps.d1icnt = dl1_ini_cnt               ; % Initial count vector       %
   Sl0_cps.j0nump = length( dl1_tim_vec )     ; % Number of points           %
else
   Sl0_cps.d2covm = dl2_cov_tim               ; % Covariance matrix          %
   Sl0_cps.d1time = dl1_ret_tim(1:q)          ; % Time vector                %
   Sl0_cps.d1rcnt = Sl0_cps.d1time            ; % Rate count vector (dummy)  %
   Sl0_cps.d1lgrc = Sl0_cps.d1time            ; % Log10 of rate count (dummy)%
   Sl0_cps.d1orcv = dl1_opt_rcv               ; % Optimal rate constant vec  %
   Sl0_cps.d1rrcv = zeros( 1 , jc0_rcv_len )  ; % Random rate constant vec   %
   Sl0_cps.d1icnt = dl1_ini_cnt               ; % Initial count vector       %
   Sl0_cps.j0nump = q                         ; % Number of points           %
end
                                              % Expand random limits (B or C)%
if ( ( nc0_mod_num == Sl0_qgd.J_QCMB4P ) || ...
     ( nc0_mod_num == Sl0_qgd.J_QCMC4P ) )
   Sl0_cps.d1mink = Sl0_ops.d1mink            ;
   Sl0_cps.d1maxk = Sl0_ops.d1maxk .* 1000.0  ;
end

if ( jc0_ovt_flg )                            ; % Override evaluation time   %
   Sl0_cps.d1time = dl1_ret_tim(1:q)          ; % Time vector                %
   Sl0_cps.d1rcnt = Sl0_cps.d1time            ; % Rate count vector (dummy)  %
   Sl0_cps.d1lgrc = Sl0_cps.d1time            ; % Log10 of rate count (dummy)%
   Sl0_cps.j0nump = q                         ; % Number of points           %
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%    In 13 of the 39 bacteria count time series, the sensitivity matrix is  %
% rank deficient (i.e. singular). The Gram matrix formed from the sensitiv- %
% ity matrix is inverted to get the covariance matrix.  For the rank defi-  %
% cient cases, a pseudoinverse is used to estimate the covariance which may %
% result in a covariance matrix that is not symmetric and/or positive semi- %
% definite (PSD).                                                             %
%                                                                             %
%    The functions nearcorr and CorrelationMatrix can be used to try and    %
% find the nearest PSD covariance matrix.  Both functions work on a corre-  %
% lation matrix which can be calculated from a covariance matrix.  For some %
% time series, especially those with rank 1, neither function will find a   %
% true PSD correlation matrix. The covariance matrix must be PSD to use the %
% oct_mvnrnd function to generate a multi-variate normal random vector.     %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % Make covariance symmetric    %
dl2_cov        = Sl0_cps.d2covm                      ;
Sl0_cps.d2covm = ( dl2_cov + dl2_cov' ) / 2.0        ;
```

```
dl0_tst_sym    = 1                                     ;

                                                % Test for positive semi-def   %
[ ~ , dl2_tm1 ] = eig( Sl0_cps.d2covm )                 ;

if ( min( diag( dl2_tm1 ) ) < 0.0 )             % Not PSD                      %
   dl2_cov = Sl0_cps.d2covm                     ;       % Covariance matrix    %
   dl1_sig = sqrt( diag( dl2_cov ) )'           ;       % Std dev vector       %
   dl2_var = dl1_sig' * dl1_sig                 ;       % Variance matrix      %
   dl2_cor = dl2_cov ./ dl2_var                 ;       % Correlation matrix   %

   dl1_tol = [ eps/1000.0 , eps/1000.0 ]        ;       % Tolerance vector     %

    dl2_cor =          ...                               % Returned corr matrix %
   nearcorr            ...                               % Nearest PSD          %
    (                  ...
     dl2_cor         , ...                               % Correlation matrix   %
     dl1_tol         , ...                               % Tolerance vector     %
     1               , ...                               % Eig decomp flag      %
     2500              ...                               % Iterations           %
    )                                           ;

   dl2_tm2 = dl2_cor .* dl2_var                 ;       % Covariance matrix    %

   [ ~ , dl2_tm1 ] = eig( dl2_tm2 )             ;       % Test for PSD         %

   if ( min( diag( dl2_tm1 ) ) < 0.0 )          % Try Qi/Sun method            %
      dl2_cor = dl2_cov ./ dl2_var              ;       % Correlation matrix   %
      n       = length( dl2_cor )               ;       % Size of corr matrix  %

       [                          ...
             dl2_cor            , ...                   % Correlation matrix   %
             ~                    ...
       ] =                        ...
      CorrelationMatrix           ...                   % Nearest PSD          %
       (                          ...
             dl2_cor            , ...                   % Correlation matrix   %
             ones(n,1)          , ...                   % b value              %
             eps * 10           , ...                   % Tau                  %
             eps                  ...                   % Tol                  %
       )                                        ;

      dl2_tm2 = dl2_cor .* dl2_var              ;       % Covariance matrix    %

      [ ~ , dl2_tm1 ] = eig( dl2_tm2 )          ;       % Test for PSD         %

      if ( min( diag( dl2_tm1 ) ) < 0.0 )
         dl2_cov = Sl0_cps.d2covm               ;       % Reset cov matrix     %
      else
         dl2_cov = dl2_tm2                      ;       % Update cov matrix    %
      end
   else
      dl2_cov = dl2_tm2                         ;       % Update cov matrix    %
   end

   Sl0_cps.d2covm = dl2_cov                     ;       % Stored cov matrix    %
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Calculate FUP                 %
if ( ( min( diag( dl2_tm1 ) ) < 0.0             ) || ...% Not PSD               %
     ( Sl0_cps.d1orcv(2) < Sl0_cps.d1orcv(4) ) )     % Death only             %

   Sl1_pms     = []                             ;
   nl0_num_rrc = 0                              ;

   if ( nc0_mod_num == Sl0_qgd.J_QCMD4P )
      dl1_def_rcv    = exp( dl1_def_rcv )       ;
```

```
   Sl0_ops.d1mink = exp( Sl0_ops.d1mink ) ;
   Sl0_ops.d1maxk = exp( Sl0_ops.d1maxk ) ;

   if ( nc0_run_num < 9 )
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+03 ,  1.0e+03 ,  1.0e+03 ,  1.0e+03 ] ;
   else
      Sl0_ops.d1mink = [  1.0e-05 ,      0.0 ,      0.0 ,      0.0 ] ;
      Sl0_ops.d1maxk = [  1.0e+04 ,  1.0e+04 ,  1.0e+04 ,  1.0e+04 ] ;
   end

   Sl0_cps.d1orcv = exp( Sl0_cps.d1orcv ) ;
   nl0_mod_num    = Sl0_qgd.J_QCMB4P      ;
else
   nl0_mod_num    = nc0_mod_num           ;
end

                                          % Special case for death only  %
if ( Sl0_cps.d1orcv(2) < Sl0_cps.d1orcv(4) )
   if ( jc0_mrd_flg == jc0_cov_dat )      % Model reduction by data      %
       [                        ...
          Sl1_mrd         , ...           % Model reduction struct vector%
          nl0_src           ...           % Status return code           %
       ] =                  ...
      qm4_qmr_fBC           ...
       (                    ...
          dl1_def_rcv     , ...           % Default rate constants       %
          dl1_tim_vec     , ...           % Vector of count times (day)  %
          dl1_cnt_vec     , ...           % Vector of measured counts    %
          []              , ...           % Evaluation times (optional)  %
          nl0_mod_num     , ...           % Model formulation number     %
          true            , ...           % Reduction for death          %
          Sl0_ops         , ...           % Optimization parameters      %
          Sl0_ocs           ...           % Output control structure     %
       )                          ;
   else                                   % Model reduction by time      %
       [                        ...
          Sl1_mrd         , ...           % Model reduction struct vector%
          nl0_src           ...           % Status return code           %
       ] =                  ...
      qm4_qmr_fBC           ...
       (                    ...
          dl1_def_rcv     , ...           % Default rate constants       %
          dl1_tim_vec     , ...           % Vector of count times (day)  %
          dl1_cnt_vec     , ...           % Vector of measured counts    %
          dl1_ret_tim(1:q), ...           % Evaluation times (optional)  %
          nl0_mod_num     , ...           % Model formulation number     %
          true            , ...           % Reduction for death          %
          Sl0_ops         , ...           % Optimization parameters      %
          Sl0_ocs           ...           % Output control structure     %
       )                          ;
   end

   dl1_sol_vec = [ Sl1_mrd(2).d1srcv(2) -   ...   % a parameter          %
                   Sl1_mrd(2).d1srcv(4)   , ...
                   Sl1_mrd(2).d1srcv(1)   , ...   % k1 parameter         %
                   NaN                    , ...
                   NaN                    ]       ;

   dl0_1EL_rat = dl1_sol_vec(1) / dl1_sol_vec(2)  ;

   if ( ( dl0_1EL_rat < dc0_rat_llm ) || ...      % Ratio <> -1          %
        ( dl0_1EL_rat > dc0_rat_ulm ) )
      jl0_nam_num    = Sl0_qgd.J_NAM1EL           ;
      Sl0_nam.d1rtim = Sl0_cps.d1time             ; % Return time        %
      Sl0_nam.d1icnt = Sl0_cps.d1icnt             ; % Init count vector  %

       [                            ...   % Evaluate alt model           %
```

```
      Sl0_nam.d1rcnt          , ...   % Model count vector (CFU/mL)  %
      nl0_src                   ...   % Status return code           %
   ] =                                    ...
   qm4_evl_nam                  ...
   (                            ...
      jl0_nam_num             , ...   % Model number                 %
      dl1_sol_vec             , ...   % Solution vector              %
      Sl0_nam.d1rtim          , ...   % Time vector (day)            %
      Sl0_nam.d1icnt(1)       , ...   % Initial count (CFU/mL)       %
      Sl0_ocs.j0dbgf            ...   % Debug flag                   %
   )                                          ;

   Sl0_nam.d1lcnt = log10( Sl0_nam.d1rcnt )    ; % Log of count vector%
   Sl0_nam.d2lcnt = Sl0_nam.d1lcnt             ; % Log of count matrix%
   Sl0_nam.d2cmat = Sl0_nam.d1rcnt             ; % Count matrix       %
   Sl0_nam.d1time = Sl0_cps.d1time             ; % Time vector        %
   Sl0_nam.d1orcv = dl1_sol_vec(1:2)           ; % Rate constant vals %
   Sl0_nam.d1rrcv = zeros( 1 , 2 )             ; % Random RCV         %
   Sl0_nam.d0mnsr = Sl1_mrd(2).d0mssr          ; % Minimum SSR        %
else                                     % Ratio == -1 -> adjust        %
   jl0_nam_num    = Sl0_qgd.J_NAM1EL           ;
   Sl0_ops.d1mink = Sl0_nam.d1mink             ;
   Sl0_ops.d1maxk = Sl0_nam.d1maxk             ;
   Sl0_nam.d1icnt = dl1_ini_cnt                ; % Init count vector  %

   [                    ...           % Refit to 1EL model           %
      Sl0_nam.d1rtim  , ...           % Output vector of times (day) %
      Sl0_nam.d1rcnt  , ...           % Output count vector (CFU/mL) %
      Sl0_nam.d2cmat  , ...           % Count matrix from ODE        %
      Sl0_nam.d1orcv  , ...           % Final rate constants (k-vals)%
      Sl0_nam.d0mnsr  , ...           % Minimum SSR (CFU/mL)         %
      nl0_src           ...           % Status return code           %
   ] =                  ...
   qm4_fit_vqm          ...
   (                    ...
      dl1_tim_vec     , ...           % Input time vector (day)      %
      dl1_cnt_vec     , ...           % Input count vector (CFU/mL)  %
      Sl0_nam.d1ircv  , ...           % Initial rate constants (k)   %
      Sl0_nam.d1icnt  , ...           % Initial count vector (x0)    %
      []              , ...           % Rate cnst solution index vec %
      dl1_ret_tim(1:q), ...           % Evaluation times             %
      jl0_nam_num     , ...           % Model number                 %
      Sl0_ops         , ...           % Optimization parameters      %
      Sl0_ocs           ...           % Debug flag                   %
   )                                  ;

   Sl0_nam.d1lcnt = log10( Sl0_nam.d1rcnt )    ; % Log of count vector%

   dl0_1EL_rat    = Sl0_nam.d1orcv(1) / ...    % Recalculate ratio    %
                    Sl0_nam.d1orcv(2)          ;

   if ( ( dl0_1EL_rat > dc0_rat_llm ) && ...   % Still equal to -1    %
        ( dl0_1EL_rat < dc0_rat_ulm ) )        %   -> increase k1     %
      Sl0_nam.d1orcv(2) = Sl0_nam.d1orcv(2) * ... % Increase k1       %
                          1.01                 ;

      dl0_1EL_rat       = Sl0_nam.d1orcv(1) / ... % Recalculate ratio %
                          Sl0_nam.d1orcv(2)    ;

      dl1_sol_vec = [ Sl0_nam.d1orcv(1) , ...     % Update soln vector%
                      Sl0_nam.d1orcv(2) , ...
                      NaN               , ...
                      NaN               ]      ;

       [                                  ...   % Evaluate alt model   %
               Sl0_nam.d1rcnt          , ...   % Model count vector   %
               nl0_src                   ...   % Status return code   %
       ] =                                ...
```

```
      qm4_evl_nam                          ...
       (                                   ...
               jl0_nam_num              , ...   % Model number         %
               dl1_sol_vec              , ...   % Solution vector      %
               Sl0_nam.d1rtim           , ...   % Time vector (day)    %
               Sl0_nam.d1icnt(1)        , ...   % Initial cnt (CFU/mL) %
               Sl0_ocs.j0dbgf             ...   % Debug flag           %
       )                                        ;

      Sl0_nam.d1lcnt = log10( Sl0_nam.d1rcnt ) ;
   end

   Sl0_nam.d2lcnt = Sl0_nam.d1lcnt     ;       % Log of count matrix  %
   Sl0_nam.d2cmat = Sl0_nam.d1rcnt     ;       % Count matrix         %
   Sl0_nam.d1time = Sl0_nam.d1rtim     ;       % Time vector          %
   Sl0_nam.d1rrcv = zeros( 1 , 2 )     ;       % Random RCV           %
end

if ( jc0_cov_flg == jc0_cov_dat )      % Covariance by data           %
    [                                ...
       Sl0_nam                    , ...   % Output NAM structure         %
       nl0_src                      ...   % Status return code           %
    ] =                             ...
   qm4_nam_cse                      ...
    (                               ...
       jl0_nam_num                , ...   % Model flag                   %
       Sl0_nam                    , ...   % Input NAM structure          %
       dl1_tim_vec                , ...   % Time vector (day)            %
       Sl0_cps.d1icnt(1)          , ...   % Initial count (CFU/mL)       %
       []                         , ...   % Debug flag                   %
       Sl0_ocs                      ...   % Output control structure     %
    )                             ;
else                                       % Covariance by time           %
    [                                ...
       Sl0_nam                    , ...   % Output NAM structure         %
       nl0_src                      ...   % Status return code           %
    ] =                             ...
   qm4_nam_cse                      ...
    (                               ...
       jl0_nam_num                , ...   % Model flag                   %
       Sl0_nam                    , ...   % Input NAM structure          %
       dl1_ret_tim(1:q)'          , ...   % Time vector (day)            %
       Sl0_cps.d1icnt(1)          , ...   % Initial count (CFU/mL)       %
       []                         , ...   % Debug flag                   %
       Sl0_ocs                      ...   % Output control structure     %
    )                             ;
end
                                           % Test for positive semi-def   %
[ ~ , dl2_tm1 ] = eig( Sl0_nam.d2covm )        ;

if ( min( diag( dl2_tm1 ) ) < 0.0 )    % Not positive semi-definite   %
   Sl1_pms     = []                                    ;
   nl0_num_rrc = 0                                     ;
   nl1_bad_rcv = zeros( 1 , 4 )                        ;
elseif ( nc0_mod_num == Sl0_qgd.J_QCMD4P )     % Formulation D        %
   if ( jc0_cov_flg == jc0_cov_dat )   % Covariance by data           %
      dl1_tim = dl1_tim_vec            ;
   else                                % Covariance by time           %
      dl1_tim = dl1_ret_tim(1:q)'      ;
   end

    [                                ...   % Death-only reduction conv    %
       Sl0_nam                    , ...   % Output NAM structure         %
       nl0_src                      ...   % Status return code           %
    ] =                             ...
   qm4_dor_cnv                      ...
    (                               ...
       Sl0_nam                    , ...   % Input NAM structure          %
```

```
        dl1_tim                  , ...   % Time vector (day)            %
        Sl0_cps.d1icnt(1)        , ...   % Initial count (CFU/mL)       %
        []                       , ...   % Debug flag                   %
        Sl0_ocs                    ...   % Output control structure     %
     )                           ;

   Sl1_pms     = repmat( Sl0_nam , 1 , jc0_num_psv )   ;
   nl0_num_rrc = 0                                     ;
   nl0_num_tcg = 0                                     ;
   nl1_bad_rcv = zeros( 1 , 2 )                        ;

                                       % No bounds on Formulation D   %
   for ( n = 1:jc0_num_psv )           % Generate random solutions    %
      Sl1_pms(n).d1rrcv = oct_mvnrnd                 ...
                          (                          ...
                              Sl1_pms(n).d1orcv' , ...
                              Sl1_pms(n).d2covm    ...
                          )                           ;

                                       % Set infinity limits         %
      Sl1_pms(n).d1rrcv(1) = min( Sl1_pms(n).d1rrcv(1) , 100 ) ;
      Sl1_pms(n).d1rrcv(1) = max( Sl1_pms(n).d1rrcv(1) , -20 ) ;
      Sl1_pms(n).d1rrcv(2) = min( Sl1_pms(n).d1rrcv(2) , 100 ) ;
      Sl1_pms(n).d1rrcv(2) = max( Sl1_pms(n).d1rrcv(2) , -20 ) ;
                                       % Check if d-k1 <> 0           %
      dl0_1EL_rat = Sl1_pms(n).d1rrcv(1) / ...   % Calculate ratio    %
                    Sl1_pms(n).d1rrcv(2)       ;
      dl0_1EL_rat = -dl0_1EL_rat                 ; % d=exp(z1) > 0      %

      if ( ( dl0_1EL_rat > dc0_rat_llm ) && ...% Test if equal to -1  %
           ( dl0_1EL_rat < dc0_rat_ulm ) )     %   -> increase k1     %
         Sl1_pms(n).d1rrcv(2) = Sl1_pms(n).d1rrcv(2) * ...
                                 1.001                 ;
      end

      nl0_num_tcg = nl0_num_tcg + 1                    ;

      if ( mod( nl0_num_tcg , jc0_num_svu/10 ) == 0 )
         fprintf( jl0_dfi , 'Generated variates: %d\n' , nl0_num_tcg );
      end
                                       % Convert back to a and k1     %
      dl1_sol_vec = [ -exp(Sl1_pms(n).d1rrcv(1)), ...  % a parameter  %
                      exp(Sl1_pms(n).d1rrcv(2)), ...  % k1 parameter %
                      NaN                       , ...
                      NaN                        ]      ;

       [                               ...      % Evaluate alt model   %
               Sl1_pms(n).d1rcnt    , ...      % Model count vector   %
               nl0_src               ...      % Status return code   %
       ] =                             ...
      qm4_evl_nam                      ...
       (                               ...
               jl0_nam_num          , ...      % Model number         %
               dl1_sol_vec          , ...      % Solution vector      %
               Sl1_pms(n).d1time    , ...      % Time vector (day)    %
               Sl1_pms(n).d1icnt(1) , ...      % Initial count        %
               0                       ...      % Debug flag           %
       )                                                 ;

      Sl1_pms(n).d1lgrc = log10( Sl1_pms(n).d1rcnt )   ;
      nl0_num_rrc       = nl0_num_rrc + 1              ;
   end                                 % End for ( n )               %
else
   Sl1_pms     = repmat( Sl0_nam , 1 , jc0_num_psv )   ;
   nl0_num_rrc = 0                                     ;
   nl0_num_tcg = 0                                     ;
   nl1_bad_rcv = zeros( 1 , 2 )                        ;
```

```
for ( n = 1:jc0_num_psv )           % Generate random solutions    %
   jl0_flg = 0              ;       % Test flag                     %
   r       = 0              ;       % Test counter                  %

   while( ~jl0_flg )                % Within min/max bounds        %
      Sl1_pms(n).d1rrcv = oct_mvnrnd             ...
                          (                      ...
                           Sl1_pms(n).d1orcv' , ...
                           Sl1_pms(n).d2covm    ...
                          )                          ;
                                    % Bounds test                  %
      dl1_tm1 = Sl1_pms(n).d1rrcv                    ;
      dl1_tm2 = max( dl1_tm1 , Sl0_nam.d1mink )      ;
      dl1_tm2 = min( dl1_tm2 , Sl0_nam.d1maxk )      ;
      jl0_flg = isequal( dl1_tm1 , dl1_tm2 )         ;
                                    % Check if a+k1 <> 0           %
      if ( ( Sl0_qgd.J_NAM1EL        == jl0_nam_num ) && ...
           ( dl1_tm1(1) + dl1_tm1(2) == 0.0         ) )
         jl0_flg = 0                                 ;
      end

      r           = r + 1                            ;
      nl0_num_tcg = nl0_num_tcg + 1                  ;

      if ( mod( nl0_num_tcg , jc0_num_svu ) == 0 )
         fprintf(                                ...
            jl0_dfi                            , ...
            'Generated variates: %d\n'         , ...
            nl0_num_tcg                        )       ;
      end

      if ( ~jl0_flg )
         jl1_tm1     = Sl1_pms(n).d1rrcv < Sl1_pms(n).d1mink;
         nl1_bad_rcv = nl1_bad_rcv + double( jl1_tm1 )      ;

         jl1_tm1     = Sl1_pms(n).d1rrcv > Sl1_pms(n).d1maxk;
         nl1_bad_rcv = nl1_bad_rcv + double( jl1_tm1 )      ;
      end
   end                              % End while( jl0_flg )         %

   if ( isempty( Sl1_pms(n).d1rrcv ) )      % No random variates   %
      Sl1_pms(n).d1lgrc = []                         ;
   else                             % Solve w/ rnd variate %
      if ( jl0_nam_num == Sl0_qgd.J_NAM1EL )
         dl1_sol_vec = [ Sl1_pms(n).d1rrcv(1), ...  % a parameter  %
                         Sl1_pms(n).d1rrcv(2), ...  % k1 parameter %
                         NaN                  , ...
                         NaN                  ]      ;
     else
         dl1_sol_vec = [ Sl1_pms(n).d1rrcv(1), ...  % k1 parameter %
                         NaN                  , ...
                         NaN                  , ...
                         NaN                  ]      ;
     end

       [                          ...      % Evaluate alt model   %
            Sl1_pms(n).d1rcnt    , ...      % Model count vector   %
            nl0_src                ...      % Status return code   %
       ] =                         ...
      qm4_evl_nam                  ...
       (                           ...
            jl0_nam_num          , ...      % Model number         %
            dl1_sol_vec          , ...      % Solution vector      %
            Sl1_pms(n).d1time    , ...      % Time vector (day)    %
            Sl1_pms(n).d1icnt(1) , ...      % Initial count        %
            0                      ...      % Debug flag           %
       )                                           ;
```

```
                Sl1_pms(n).d1lgrc = log10( Sl1_pms(n).d1rcnt );
                nl0_num_rrc       = nl0_num_rrc + 1           ;
             end
          end                                   % End for ( n )                %
       end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif ( nc0_mod_num == Sl0_qgd.J_QCMD4P )   % Formulation D generation     %
    Sl1_pms     = repmat( Sl0_cps , 1 , jc0_num_psv ) ;
    nl0_num_rrc = 0                                   ;
    nl0_num_tcg = 0                                   ;
    nl1_bad_rcv = zeros( 1 , 4 )                      ;

                                                      % No bounds on Formulation D   %
    for ( n = 1:jc0_num_psv )                         % Generate random solutions    %
       Sl1_pms(n).d1rrcv = oct_mvnrnd                       ...
                           (                                ...
                             Sl1_pms(n).d1orcv'       , ...
                             Sl1_pms(n).d2covm          ...
                           )                          ;
       Sl1_pms(n).d1rrcv = min                            ... % Infinity limit %
                           (                                ...
                             Sl1_pms(n).d1rrcv        , ...
                             100                        ...
                           )                          ;
       Sl1_pms(n).d1rrcv = max                            ... % Infinity limit %
                           (                                ...
                             Sl1_pms(n).d1rrcv        , ...
                             -20                        ...
                           )                          ;

       nl0_num_tcg = nl0_num_tcg + 1                  ;

       if ( mod( nl0_num_tcg , jc0_num_svu/10 ) == 0 )
          fprintf( jl0_dfi , 'Generated variates: %d\n' , nl0_num_tcg ) ;
       end

        [                                 ...
              Sl1_pms(n).d1rcnt       , ...   % Output count vector (CFU/mL) %
              ~                       , ...   % Count matrix from ODE        %
              nl0_src                   ...   % Status return code           %
        ] =                             ...
       Dq4_slv_ode                      ...   % Solve ODE formulation B      %
        (                               ...
              Sl1_pms(n).d1rrcv'      , ...   % Rate constant values (k)     %
              Sl1_pms(n).d1icnt       , ...   % Initial count vector (x0)    %
              Sl1_pms(n).d1time       , ...   % Vector of times (day)        %
              0                         ...   % Debug flag                   %
        )                                     ;

       Sl1_pms(n).d1lgrc = log10( Sl1_pms(n).d1rcnt ) ;
       nl0_num_rrc       = nl0_num_rrc + 1            ;
    end                                           % End for ( n )                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif ( nc0_mod_num == Sl0_qgd.J_QCME4P )   % Formulation E generation     %
    Sl1_pms     = repmat( Sl0_cps , 1 , jc0_num_psv ) ;
    nl0_num_rrc = 0                                   ;
    nl0_num_tcg = 0                                   ;
    nl1_bad_rcv = zeros( 1 , 4 )                      ;
    r           = 0                                   ;
                                                      % No bounds on Formulation E   %
    for ( n = 1:jc0_num_psv )                         % Generate random solutions    %
       Sl1_pms(n).d1rrcv = oct_mvnrnd                       ...
                           (                                ...
                             Sl1_pms(n).d1orcv'       , ...
                             Sl1_pms(n).d2covm          ...
                           )                          ;
                                                      % Convert to formulation D     %
```

```
Sl1_pms(n).d1rrcv    = exp( Sl1_pms(n).d1rrcv )        ;
Sl1_pms(n).d1rrcv(4) = Sl1_pms(n).d1rrcv(2) - ...
                       Sl1_pms(n).d1rrcv(4)            ;

if ( Sl1_pms(n).d1rrcv(4) <= 0.0 )     % Negative k4: make small      %
   dl0_tm1              = 0.99 * Sl1_pms(n).d1rrcv(2)  ;
   Sl1_pms(n).d1rrcv(4) = min                     ...
                           (                      ...
                               dl0_tm1         , ...
                               1.0e-100          ...
                           )                          ;
end

Sl1_pms(n).d1rrcv = log( Sl1_pms(n).d1rrcv )           ;

Sl1_pms(n).d1rrcv = min                           ... % Infinity limit %
                     (                            ...
                       Sl1_pms(n).d1rrcv       , ...
                       150                         ...
                     )                         ;
Sl1_pms(n).d1rrcv = max                           ... % Infinity limit %
                     (                            ...
                       Sl1_pms(n).d1rrcv       , ...
                       -50                         ...
                     )                         ;

dl0_tm1           = Sl1_pms(n).d1rrcv(2) - ...
                    Sl1_pms(n).d1rrcv(4)       ;

if ( dl0_tm1 < eps )
   Sl1_pms(n).d1rrcv(2) = Sl1_pms(n).d1rrcv(2) + ( eps * 10 )  ;
   r                    = r + 1                                ;

   if ( r >= 50 )                      % Error                       %
      fprintf( jl0_dfi , '\n' )                                 ;

      fprintf( jl0_dfi , '\nFINAL RATE CONSTANTS\n' )           ;

      for ( n = 1:jc0_rcv_len )
         fprintf( jl0_dfi , '%13.6e\t'  , dl2_rcv_mat(1,n) )   ;
      end                              % End for ( n )                %

      fprintf( jl0_dfi , '\t%9.6e\t' , dl1_min_ssr(1) )         ;
      fprintf( jl0_dfi , '\n' )                                 ;

      fprintf( jl0_dfi , '\nFULL COVARIANCE MATRIX\n' )         ;

      for ( r = 1:jc0_rcv_len )
         fprintf( jl0_dfi , '    ' )                            ;
         for ( n = 1:jc0_rcv_len )
            fprintf( jl0_dfi , '%13.6e\t' ,Sl0_cps.d2covm(r,n));
         end                           % End for ( n )                %
         fprintf( jl0_dfi , '\n' )                              ;
      end                              % End for ( r )                %
      error( 'uq2_run_fep: Cannot generate distribution' )      ;
   end
end

nl0_num_tcg = nl0_num_tcg + 1                   ;

if ( mod( nl0_num_tcg , jc0_num_svu/10 ) == 0 )
   fprintf( jl0_dfi , 'Generated variates: %d\n' , nl0_num_tcg ) ;
end

 [                                ...
       Sl1_pms(n).d1rcnt       , ...   % Output count vector (CFU/mL) %
       ~                       , ...   % Count matrix from ODE        %
       nl0_src                   ...   % Status return code           %
```

```
      ] =                                ...
      Dq4_slv_ode                        ...   % Solve ODE formulation D      %
       (                                 ...
            Sl1_pms(n).d1rrcv'       , ...   % Rate constant values (k)     %
            Sl1_pms(n).d1icnt        , ...   % Initial count vector (x0)    %
            Sl1_pms(n).d1time        , ...   % Vector of times (day)        %
            0                          ...   % Debug flag                   %
       )                                                ;

      Sl1_pms(n).d1lgrc = log10( Sl1_pms(n).d1rcnt ) ;
      nl0_num_rrc       = nl0_num_rrc + 1              ;
   end                                         % End for ( n )                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
else                                           % Normal random generation     %
   if ( nc0_mod_num == Sl0_qgd.J_QCMC4P )      % Convert k4 to alpha          %
      Sl0_cps.d1orcv(4) = Sl0_cps.d1orcv(2) - ...
                          Sl0_cps.d1orcv(4)          ;
   end

   Sl1_pms     = repmat( Sl0_cps , 1 , jc0_num_psv ) ;
   nl0_num_rrc = 0                                   ;
   nl0_num_tcg = 0                                   ;
   nl1_bad_rcv = zeros( 1 , 4 )                      ;

   for ( n = 1:jc0_num_psv )                   % Generate random solutions    %
      jl0_flg = 0                     ;        % Test flag                    %
      r       = 0                     ;        % Test counter                 %

      while( ~jl0_flg )                        % Within min/max bounds        %
         Sl1_pms(n).d1rrcv = oct_mvnrnd                    ...
                             (                             ...
                                    Sl1_pms(n).d1orcv' , ...
                                    Sl1_pms(n).d2covm    ...
                             )                                ;
                                                  % Bounds test                %
         dl1_tm1 = Sl1_pms(n).d1rrcv                      ;
         dl1_tm2 = max( dl1_tm1 , Sl1_pms(n).d1mink );
         jl0_flg = isequal( dl1_tm1 , dl1_tm2 )           ;
                                                  % Check if k4 >= 0             %
         if ( ( nc0_mod_num == Sl0_qgd.J_QCMC4P ) && ...
              ( dl1_tm1(4)  >  dl1_tm1(2)       ) )
            jl0_flg = 0                                   ;
         end

         r           = r + 1                              ;
         nl0_num_tcg = nl0_num_tcg + 1                    ;

         if ( mod( nl0_num_tcg , jc0_num_svu ) == 0 )
            fprintf( jl0_dfi , 'Generated variates: %d\n' , nl0_num_tcg ) ;
         end

         if ( ~jl0_flg )
            jl1_tm1     = Sl1_pms(n).d1rrcv < Sl1_pms(n).d1mink      ;
            nl1_bad_rcv = nl1_bad_rcv + double( jl1_tm1 )            ;
         end
      end                                         % End while( jl0_flg )         %

      if ( isempty( Sl1_pms(n).d1rrcv ) )         % No valid random variate      %
         Sl1_pms(n).d1lgrc = []                           ;
      else                                        % Solve with random variate    %
         if ( nc0_mod_num == Sl0_qgd.J_QCMC4P )      % Convert alpha to k4  %
            Sl1_pms(n).d1orcv(4) = Sl1_pms(n).d1orcv(2) - ...
                                   Sl1_pms(n).d1orcv(4)      ;
            Sl1_pms(n).d1rrcv(4) = Sl1_pms(n).d1rrcv(2) - ...
                                   Sl1_pms(n).d1rrcv(4)      ;
         end

          [                                ...
```

```
            Sl1_pms(n).d1rcnt       , ...   % Output count vector (CFU/mL) %
            ~                       , ...   % Count matrix from ODE        %
            nl0_src                   ...   % Status return code           %
         ] =                                  ...
        Bq4_slv_ode                   ...   % Solve ODE formulation B      %
         (                            ...
            Sl1_pms(n).d1rrcv'      , ...   % Rate constant values (k)     %
            Sl1_pms(n).d1icnt       , ...   % Initial count vector (x0)    %
            Sl1_pms(n).d1time       , ...   % Vector of times (day)        %
            0                         ...   % Debug flag                   %
         )                                                ;

        Sl1_pms(n).d1lgrc = log10( Sl1_pms(n).d1rcnt )    ;
        nl0_num_rrc       = nl0_num_rrc + 1               ;
     end
  end                                       % End for ( n )                %
end

dl0_pct_vrv = 100.0                * ...
              double( nl0_num_rrc ) / ...
              double( jc0_num_psv )         ;
dl0_pct_nrm = 100.0                * ...
              double( nl0_num_rrc ) / ...
              double( nl0_num_tcg )         ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                            % Trim output                  %
if ( length( Sl1_pms(1).d1time ) ~= q )     % Check lengths                %
  error( 'uq2_run_fep: inconsistent vector lengths' )     ;
end

dl1_rnd_ssr = ones( nl0_num_rrc , 1 );      % Initialize ssr vec           %
dl1_rnd_ssr = dl1_rnd_ssr * 1.0e+15  ;      % Large number                 %

for ( n = 1:nl0_num_rrc )
  if ( length(Sl1_pms(n).d1lgrc) == length(dl1_lgc_vec) )
     dl1_tm1        = Sl1_pms(n).d1lgrc - dl1_lgc_vec     ;
     dl1_tm1        = dl1_tm1 .^ 2                        ;
     dl1_rnd_ssr(n) = sum( dl1_tm1 ) / double( q )        ;
  end
end                                         % End for ( n )                %

nl0_num_rvp = floor( nl0_num_rrc * dc0_cnf_lim )     ;
nl0_num_rvp = max( nl0_num_rvp , 1 )                 ;

[ ~ , I ]   = sort( dl1_rnd_ssr )                    ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                            % Plot results                 %
if ( jl0_plt_flg )
  dl1_xlm_vec = [ dl0_min_xpl dl0_max_xpl ]          ;
  dl1_xtk_vec = dl0_min_xpl:dl0_tim_inc:dl0_max_xpl ;
  dl1_ylm_vec = [ dl0_min_ypl dl0_max_ypl ]          ;
  dl1_ytk_vec = dl0_min_ypl:1:dl0_max_ypl            ;

  figure(                    ...            % Create figure                %
    'Position'             , ...
    dl1_sc1_pos            , ...
    'Color'                , ...
    [ 1 1 1 ]              )        ;
                                            % Set figure control           %
  axis xy                                   ;
  hold on                                   ;
  box off                                   ;
                                            % Set figure parameters        %
  set( gca , 'XColor'     , [ 0.0 0.0 0.0 ]        )      ;
  set( gca , 'YColor'     , [ 0.0 0.0 0.0 ]        )      ;
  set( gca , 'XLim'       , dl1_xlm_vec            )      ;
```

```
set( gca , 'YLim'       , dl1_ylm_vec             )       ;
set( gca , 'XTick'      , dl1_xtk_vec             )       ;
set( gca , 'YTick'      , dl1_ytk_vec             )       ;
set( gca , 'TickLength' , [ 0.001 , 0.001 ]       )       ;

set( gca , 'GridColor'  , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'GridAlpha'  , 0.5                     )       ;
set( gca , 'Layer'      , 'top'                   )       ;
set( gca , 'XMinorGrid' , 'off'                   )       ;
set( gca , 'YMinorGrid' , 'off'                   )       ;
set( gca , 'FontSize'   , 11                      )       ;
set( gca , 'FontWeight' , 'bold'                  )       ;
grid on                                                   ;
                                          % Add labels                    %
xlabel( 'TIME (days)' , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;
ylabel( 'log CFU/mL'  , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;
title(  cl1_tit_str   , 'FontSize' , 16 , 'FontWeight' , 'bold' ) ;

if ( ~isempty( Sl1_pms ) )                % Draw random solutions        %
   for ( n = 1:nl0_num_rvp )
      r = I(n)                    ;

      if     ( n < nl0_num_rvp * 0.25 )
         dl1_cl2(4) = 0.25        ;
      elseif ( n < nl0_num_rvp * 0.50 )
         dl1_cl2(4) = 0.10        ;
      elseif ( n < nl0_num_rvp * 0.75 )
         dl1_cl2(4) = 0.05        ;
      else
         if ( m == 5 )
            dl1_cl2(4) = 0.03     ;
         else
            dl1_cl2(4) = 0.01     ;
         end
      end

      line(                          ...   % Draw QCM solution            %
          Sl1_pms(r).d1time       , ...
          Sl1_pms(r).d1lgrc       , ...
          'Linewidth' , 1         , ...
          'Color'     , dl1_cl2   )       ;
   end                                    % End for ( n )                 %
end

if     ( m == 1 )                         % Plot the data                 %
   scatter( dl1_xcr , dl1_ycr , 40 , [ 0.0 0.0 1.0 ] , 'filled' , 'd' ) ;
elseif ( m == 2 )
   scatter( dl1_xcr , dl1_ycr , 40 , [ 1.0 0.0 0.0 ] , 'filled' , 's' ) ;
elseif ( m == 3 )
   scatter( dl1_xcr , dl1_ycr , 40 , [ 0.0 1.0 0.0 ] , 'filled' , '^' ) ;
elseif ( m == 4 )
   scatter( dl1_xcr , dl1_ycr , 40 , [ 0.0 0.0 0.0 ] , 'filled' , 'o' ) ;
elseif ( m == 5 )
   scatter( dl1_xcr , dl1_ycr , 40 , [ 1.0 0.0 1.0 ] , 'filled' , 'v' ) ;
elseif ( m == 6 )
   scatter( dl1_xcr , dl1_ycr , 60 , [ 0.5 0.5 0.5 ] , 'filled' , 'h' ) ;
end

if ( isfield( Sl0_nam , 'd1lcnt' ) )
   line(                          ...   % Draw 1EL solution            %
          dl1_ret_tim(1:q)        , ...
          Sl0_nam.d1lcnt          , ...
          'Linewidth' , 2         , ...
          'Color'     , dl1_clr   )       ;
else
   line(                          ...   % Draw QCM solution            %
          dl1_ret_tim(1:q)        , ...
          dl1_lgc_vec(1:q)        , ...
```

```
          'Linewidth' , 2          , ...
          'Color'     , dl1_clr    )       ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                        % Plot s1                      %
dl1_ylm_vec = [ -dl0_max_sen dl0_max_sen ];

figure(                    ...          % Create figure                %
  'Name'                 , ...
  'S1'                   , ...
  'Position'             , ...
  dl1_sc2_pos            , ...
  'Color'                , ...
  [ 1 1 1 ]              )              ;
                                        % Set figure control           %
axis xy                                 ;
hold on                                 ;
box off                                 ;
                                        % Set figure parameters        %
set( gca , 'XColor'     , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'YColor'     , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'XLim'       , dl1_xlm_vec             )       ;
set( gca , 'YLim'       , dl1_ylm_vec             )       ;
set( gca , 'XTick'      , dl1_xtk_vec             )       ;
set( gca , 'TickLength' , [ 0.001 , 0.001 ]       )       ;

set( gca , 'GridColor'  , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'GridAlpha'  , 0.5                     )       ;
set( gca , 'Layer'      , 'top'                   )       ;
set( gca , 'XMinorGrid' , 'off'                   )       ;
set( gca , 'YMinorGrid' , 'off'                   )       ;
set( gca , 'FontSize'   , 11                      )       ;
set( gca , 'FontWeight' , 'bold'                  )       ;
grid on                                                   ;
                                        % Add labels                   %
xlabel( 'TIME (days)'  , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;
ylabel( 'SENSITIVITY'  , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;

for( n = 1:jc0_rcv_len )                % Draw s1 sensitivities        %
   if ( n == 1 ) , dl1_clr = [ 0 0 1 ] ; end      ;
   if ( n == 2 ) , dl1_clr = [ 0 1 0 ] ; end      ;
   if ( n == 3 ) , dl1_clr = [ 1 0 0 ] ; end      ;
   if ( n == 4 ) , dl1_clr = [ 0 0 0 ] ; end      ;

   line(                           ...  % Draw sensitivity s1j         %
          dl1_ret_tim(1:q)         , ...
          dl2_sen_fun(n,1:q)       , ...
          'Linewidth' , 2          , ...
          'Color'     , dl1_clr    )    ;
end                                     % End for ( n )                %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                        % Plot s2                      %
figure(                    ...          % Create figure                %
  'Name'                 , ...
  'S2'                   , ...
  'Position'             , ...
  dl1_sc3_pos            , ...
  'Color'                , ...
  [ 1 1 1 ]              )              ;
                                        % Set figure control           %
axis xy                                 ;
hold on                                 ;
box off                                 ;
                                        % Set figure parameters        %
set( gca , 'XColor'     , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'YColor'     , [ 0.0 0.0 0.0 ]         )       ;
```

```
set( gca , 'XLim'       , dl1_xlm_vec             )       ;
set( gca , 'YLim'       , dl1_ylm_vec             )       ;
set( gca , 'XTick'      , dl1_xtk_vec             )       ;
set( gca , 'TickLength' , [ 0.001 , 0.001 ]       )       ;

set( gca , 'GridColor'  , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'GridAlpha'  , 0.5                     )       ;
set( gca , 'Layer'      , 'top'                   )       ;
set( gca , 'XMinorGrid' , 'off'                   )       ;
set( gca , 'YMinorGrid' , 'off'                   )       ;
set( gca , 'FontSize'   , 11                      )       ;
set( gca , 'FontWeight' , 'bold'                  )       ;
grid on                                                   ;
                                          % Add labels                    %
xlabel( 'TIME (days)'  , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;
ylabel( 'SENSITIVITY'  , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;


for( n = 1:jc0_rcv_len )                  % Draw s2 sensitivities        %
   if ( n == 1 ) , dl1_clr = [ 0 0 1 ] ; end      ;
   if ( n == 2 ) , dl1_clr = [ 0 1 0 ] ; end      ;
   if ( n == 3 ) , dl1_clr = [ 1 0 0 ] ; end      ;
   if ( n == 4 ) , dl1_clr = [ 0 0 0 ] ; end      ;

   p = n + jc0_rcv_len                    ;

   line(                             ...   % Draw sensitivity s1j         %
          dl1_ret_tim(1:q)        , ...
          dl2_sen_fun(p,1:q)      , ...
          'Linewidth' , 2         , ...
          'Color'     , dl1_clr   )       ;
end                                       % End for ( n )                 %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                          % Plot sg                       %
dl1_ylm_vec = [ dl0_sng_mny dl0_sng_mxy ] ;
dl1_xlm_vec = [ dl0_sng_mnx dl0_sng_mxx ] ;

figure(                     ...           % Create figure                 %
  'Name'                  , ...
  'Sg'                    , ...
  'Position'              , ...
  dl1_sc4_pos             , ...
  'Color'                 , ...
  [ 1 1 1 ]               )               ;
                                          % Set figure control            %
axis xy                                   ;
hold on                                   ;
box off                                   ;
                                          % Set figure parameters         %
set( gca , 'XColor'     , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'YColor'     , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'XLim'       , dl1_xlm_vec             )       ;
set( gca , 'YLim'       , dl1_ylm_vec             )       ;
set( gca , 'XTick'      , dl1_xtk_vec             )       ;
set( gca , 'TickLength' , [ 0.001 , 0.001 ]       )       ;

set( gca , 'GridColor'  , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'GridAlpha'  , 0.5                     )       ;
set( gca , 'Layer'      , 'top'                   )       ;
set( gca , 'XMinorGrid' , 'off'                   )       ;
set( gca , 'YMinorGrid' , 'off'                   )       ;
set( gca , 'FontSize'   , 11                      )       ;
set( gca , 'FontWeight' , 'bold'                  )       ;
grid on                                                   ;
                                          % Add labels                    %
xlabel( 'TIME (days)'  , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;
ylabel( 'SENSITIVITY'  , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;
```

```
for( n = 1:jc0_rcv_len )                  % Draw sg sensitivities        %
   if ( n == 1 ) , dl1_clr = [ 0 0 1 ] ; end      ;
   if ( n == 2 ) , dl1_clr = [ 0 1 0 ] ; end      ;
   if ( n == 3 ) , dl1_clr = [ 1 0 0 ] ; end      ;
   if ( n == 4 ) , dl1_clr = [ 0 0 0 ] ; end      ;

   line(                            ...   % Draw sensitivity sgij        %
          dl1_ret_tim(1:q)        , ...
          dl2_sen_mat(n,1:q)      , ...
          'Linewidth' , 2         , ...
          'Color'     , dl1_clr   )                ;
end                                       % End for ( n )                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                          % Plot model reduction         %
dl1_xlm_vec = [ dl0_min_xpl dl0_max_xpl ]         ;
dl1_ylm_vec = [ dl0_min_ypl dl0_max_ypl ]         ;

cl1_buf     = [ 'MODEL REDUCTION: ' , ...
                cl1_tit_str         ]             ;

figure(                     ...           % Create figure                %
  'Position'              , ...
  dl1_sc5_pos             , ...
  'Color'                 , ...
  [ 1 1 1 ]               )               ;
                                          % Set figure control           %
axis xy                                   ;
hold on                                   ;
box off                                   ;
                                          % Set figure parameters        %
set( gca , 'XColor'     , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'YColor'     , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'XLim'       , dl1_xlm_vec             )       ;
set( gca , 'YLim'       , dl1_ylm_vec             )       ;
set( gca , 'XTick'      , dl1_xtk_vec             )       ;
set( gca , 'YTick'      , dl1_ytk_vec             )       ;
set( gca , 'TickLength' , [ 0.001 , 0.001 ]       )       ;

set( gca , 'GridColor'  , [ 0.0 0.0 0.0 ]         )       ;
set( gca , 'GridAlpha'  , 0.5                     )       ;
set( gca , 'Layer'      , 'top'                   )       ;
set( gca , 'XMinorGrid' , 'off'                   )       ;
set( gca , 'YMinorGrid' , 'off'                   )       ;
set( gca , 'FontSize'   , 11                      )       ;
set( gca , 'FontWeight' , 'bold'                  )       ;
grid on                                                   ;
                                          % Add labels                  %
xlabel( 'TIME (days)' , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;
ylabel( 'log CFU/mL'  , 'FontSize' , 11 , 'FontWeight' , 'bold' ) ;
title(  cl1_buf       , 'FontSize' , 16 , 'FontWeight' , 'bold' ) ;

line(                       ...           % Draw QCM solution            %
  dl1_ret_tim(1:q)        , ...
  dl1_lgc_vec(1:q)        , ...
  'Linewidth' , 3         , ...
  'Color'     , [ 0 0 0 ] )       ;

for ( n = 1:length( Sl1_mrs ) )
   if ( n == 1 ) , dl1_clr = [ 0 0 1 ] ; end      ;
   if ( n == 2 ) , dl1_clr = [ 1 0 0 ] ; end      ;
   if ( n == 3 ) , dl1_clr = [ 0 1 0 ] ; end      ;
   if ( n == 4 ) , dl1_clr = [ 1 0 1 ] ; end      ;

   if ( Sl1_mrs(n).j0sigf )
      line(                         ...   % Draw reduced solutions       %
          Sl1_mrs(n).d1timv       , ...
```

```
            Sl1_mrs(n).d1lgcv       , ...
            'Linewidth' , 1         , ...
            'Color'     , dl1_clr   )              ;
      end
   end                                      % End for ( n )              %
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                            % Output results             %
if ( jl0_dbg_flg >= Sl0_qgd.J_DBGLV3 )
   fprintf( jl0_dfi , '\n' )                                ;

   fprintf( jl0_dfi , 'DATASET:     %d\n' , nc0_run_num )   ;
   fprintf( jl0_dfi , 'TIME SERIES: %d\n' , nc0_fil_num )   ;

   if ( nc0_mod_num == Sl0_qgd.J_QCMB4P )
      fprintf( jl0_dfi , 'FORMULATION B\n' )                ;
   elseif ( nc0_mod_num == Sl0_qgd.J_QCMC4P )
      fprintf( jl0_dfi , 'FORMULATION C\n' )                ;
   elseif ( nc0_mod_num == Sl0_qgd.J_QCMD4P )
      fprintf( jl0_dfi , 'FORMULATION D\n' )                ;
   elseif ( nc0_mod_num == Sl0_qgd.J_QCME4P )
      fprintf( jl0_dfi , 'FORMULATION E\n' )                ;
   end

   fprintf( jl0_dfi , '\n' )                                ;
   fprintf( jl0_dfi , 'INITIAL RATE CONSTANTS\n' )          ;

   for ( n = 1:jc0_rcv_len )
      fprintf( jl0_dfi , '%13.6e\t'  , dl2_rcv_ini(1,n) )   ;
   end                                      % End for ( n )              %

   fprintf( jl0_dfi , '\n' )                                ;

   fprintf( jl0_dfi , '\nFINAL RATE CONSTANTS\n' )          ;

   for ( n = 1:jc0_rcv_len )
      fprintf( jl0_dfi , '%13.6e\t'  , dl2_rcv_mat(1,n) )   ;
   end                                      % End for ( n )              %

   fprintf( jl0_dfi , '\t%9.6e\t' , dl1_min_ssr(1) )        ;
   fprintf( jl0_dfi , '\n' )                                ;

   fprintf( jl0_dfi , '\nFULL COVARIANCE MATRIX\n' )        ;

   for ( r = 1:jc0_rcv_len )
      fprintf( jl0_dfi , '    ' )                           ;
      for ( n = 1:jc0_rcv_len )
         fprintf( jl0_dfi , '%13.6e\t' , Sl0_cps.d2covm(r,n));
      end                                   % End for ( n )              %
      fprintf( jl0_dfi , '\n' )                             ;
   end                                      % End for ( r )              %

   fprintf( jl0_dfi , '\nAVERAGE |SENSITIVITY OF g|\n' )    ;

   for ( n = 1:jc0_rcv_len )
      fprintf(                                ...
            jl0_dfi                         , ...
            '%9.6e\t'                       , ...
            mean(abs(dl2_sen_mat(n,1:q)))   )               ;
   end                                      % End for ( n )              %

   fprintf( jl0_dfi , '\nL1 NORM OF SENSITIVITY OF g\n' )   ;

   for ( n = 1:jc0_rcv_len )
      fprintf(                                ...
            jl0_dfi                         , ...
            '%9.6e\t'                       , ...
```

```
          norm(dl2_sen_mat(n,1:q),1)      )                ;
end                                       % End for ( n )                %

fprintf( jl0_dfi , '\nL2 NORM OF SENSITIVITY OF g\n' )    ;

for ( n = 1:jc0_rcv_len )
   fprintf(                                  ...
          jl0_dfi                         , ...
          '%9.6e\t'                       , ...
          norm(dl2_sen_mat(n,1:q),2)      )                ;
end                                       % End for ( n )                %

if ( ~isempty( dl1_std_err ) )
   fprintf( jl0_dfi , '\nSTANDARD ERROR\n' )              ;

   for ( n = 1:jc0_rcv_len )
      fprintf(                          ...
          jl0_dfi                     , ...
          '%9.6e\t'                   , ...
          dl1_std_err(n)              )                   ;
   end                                    % End for ( n )                %
end

if ( ~isempty( dl1_std_er2 ) )
   fprintf( jl0_dfi , '\nALT STANDARD ERROR\n' )          ;

   for ( n = 1:jc0_rcv_len )
      fprintf(                          ...
          jl0_dfi                     , ...
          '%9.6e\t'                   , ...
          dl1_std_er2(n)              )                   ;
   end                                    % End for ( n )                %
end

if ( ( isfield( Sl0_nam , 'd1orcv' )     ) && ... % Alternate model      %
     ( jl0_nam_num == Sl0_qgd.J_NAM1EL ) )        % Two parameter model  %
   fprintf( jl0_dfi , '\n' )                                          ;
   fprintf( jl0_dfi , '\n' )                                          ;

   if ( ( nc0_mod_num == Sl0_qgd.J_QCMD4P ) || ...
        ( nc0_mod_num == Sl0_qgd.J_QCME4P ) )
      fprintf( jl0_dfi , '1-STAGE EXPONENTIAL WITH LAG ' )        ;
      fprintf( jl0_dfi , '(ln(d)/ln(k1)/SSR/R)\n'     )           ;
   else
      fprintf( jl0_dfi , '1-STAGE EXPONENTIAL WITH LAG ' )        ;
      fprintf( jl0_dfi , '(alpha/k1/SSR/R)\n'         )           ;
   end

   for ( n = 1:2 )
      if ( Sl0_nam.d1orcv(n) <= 999.99 )
         fprintf( jl0_dfi , '%13.6f\t' , Sl0_nam.d1orcv(n) )      ;
      else
         fprintf( jl0_dfi , '%10.6e\t'  , Sl0_nam.d1orcv(n) )     ;
      end
   end                                    % End for ( n )                %

   fprintf( jl0_dfi , '\t%9.6e\t'  , Sl0_nam.d0mnsr )             ;
   fprintf( jl0_dfi , '\t%13.6e\t' , dl0_1EL_rat    )             ;
   fprintf( jl0_dfi , '\n' )                                      ;
   fprintf( jl0_dfi , '\n' )                                      ;

   fprintf(                                  ...
          jl0_dfi                         , ...
          '  COV: %13.6e\t%13.6e\n'       , ...
          Sl0_nam.d2covm(1,1)             , ...
          Sl0_nam.d2covm(1,2)             )                      ;
   fprintf(                                  ...
          jl0_dfi                         , ...
```

```
            '       %13.6e\t%13.6e\n'       , ...
            Sl0_nam.d2covm(2,1)             , ...
            Sl0_nam.d2covm(2,2)             )                       ;
      elseif ( isfield( Sl0_nam , 'd1orcv' ) )          % One parameter model  %
         fprintf( jl0_dfi , '\n' )                                          ;
         fprintf( jl0_dfi , '\n' )                                          ;

         fprintf( jl0_dfi , '1-STAGE EXPONENTIAL'           )           ;
         fprintf( jl0_dfi , '(-k1/SSR)\n'                   )           ;

         if ( abs(Sl0_nam.d1orcv(1)) <= 999.99 )
            fprintf( jl0_dfi , '%13.6f\t' , Sl0_nam.d1orcv(1)  )        ;
         else
            fprintf( jl0_dfi , '%10.6e\t'  , Sl0_nam.d1orcv(1) )        ;
         end

         fprintf( jl0_dfi , '\t%9.6e\t' , Sl0_nam.d0mnsr )              ;
         fprintf( jl0_dfi , '\n' )                                          ;
         fprintf( jl0_dfi , '\n' )                                          ;

         fprintf(                                  ...
            jl0_dfi                         , ...
            '  VAR: %13.6e\n'               , ...
            Sl0_nam.d2covm(1,1)             )                       ;
      end

      fprintf( jl0_dfi , '\n' )                                     ;
      fprintf( jl0_dfi , '\n' )                                     ;

      fprintf( jl0_dfi , 'PERCENT VALID RANDOM VECTORS: ' )     ;
      fprintf( jl0_dfi , '%6.2f\t' , dl0_pct_vrv          )     ;

      if ( isempty( Sl1_pms ) )
         fprintf(                           ...
            jl0_dfi                 , ...
            '(covariance not PSD)'  )                         ;
      end

      fprintf( jl0_dfi , '\n' )                                     ;

      fprintf( jl0_dfi , 'LIKELIHOOD MV NORMAL (%%):     ')     ;
      fprintf( jl0_dfi , '%6.2f\t' , dl0_pct_nrm          )     ;

      fprintf( jl0_dfi , '\n\n' )                                   ;

      fprintf( jl0_dfi , 'TOTAL RANDOM VARIATES:     '    )     ;
      fprintf( jl0_dfi , '%d\n' , nl0_num_tcg             )     ;
      fprintf( jl0_dfi , 'TOTAL BY k: \n'                 )     ;

      for ( n = 1:length( nl1_bad_rcv ) )
         fprintf(                           ...
            jl0_dfi                 , ...
            '%d\t'                  , ...
            nl1_bad_rcv(n)          )                         ;
      end                                   % End for ( n )               %

      fprintf( jl0_dfi , '\n\n' )                                   ;
   end
end                                         % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qcm_ini_prm                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           02-DEC-2015                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      08/15/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This  function initializes  the global  parameters  used by the Quasi- %
% Chemical Model program.  A  global structure is returned with the fields set %
% to  values used in the VSA program such  as flags, conversion  codes, status %
% and error codes, default  values, and parameters used for controlling calcu- %
% lations.                                                                     %
%                                                                              %
%       The L_DEBUG flag can be set by passing the  logical value "true" as an %
% argument.  If no  argument, more than one argument, or a single argument not %
% equal to "true" is passed, then the debug flag is set to "false".            %
%                                                                              %
%       The L_EXIST flag can be used to determine if a structure has been ini- %
% tialized when using GLOBAL or PERSISTENT variables.                          %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Sg0_qgd = qcm_ini_prm( varargin )      % Init QCM global data         %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Existence flag               %
   Sg0_qgd.L_EXIST   = true                                                ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % STATUS RETURN CODES          %
                                                % Success code                 %
   Sg0_qgd.S_SUCCESS = uint32( hex2dec( '00000000' ) )                     ;
   Sg0_qgd.C_SUCCESS = { 'QCM-SUCCESS: Successful completion' }            ;

                                                % Information codes            %
   Sg0_qgd.I_UNKINF  = uint32( hex2dec( '81000000' ) )                     ;

   Sg0_qgd.C_INFSTR  = {                                              ...
        'QCM-INFO-UNKINF: Unknown information'                     }       ;

                                                % Fatal error codes            %
   Sg0_qgd.F_UNKFER  = uint32( hex2dec( '41000000' ) )                     ;
   Sg0_qgd.F_FILERR  = uint32( hex2dec( '41000001' ) )                     ;

   Sg0_qgd.C_FERSTR  = {                                              ...
        'QCM-FATAL-UNKFER: Unknown fatal error'                    , ...
        'QCM-FATAL-FILERR: Error opening file'                     }       ;

                                                % Error codes                 %
   Sg0_qgd.E_UNKERR  = uint32( hex2dec( '21000000' ) )                     ;
   Sg0_qgd.E_INVARG  = uint32( hex2dec( '21000001' ) )                     ;
   Sg0_qgd.E_INCARG  = uint32( hex2dec( '21000002' ) )                     ;
```

```
   Sg0_qgd.E_MAXPTS  = uint32( hex2dec( '21000004' ) )                  ;

   Sg0_qgd.C_ERRSTR  = {                                                ...
        'QCM-ERROR-UNKERR: Unknown error'                            , ...
        'QCM-ERROR-INVARG: Invalid argument'                         , ...
        'QCM-ERROR-INCARG: Inconsistent argument'                    , ...
        'QCM-ERROR-MAXPTS: Exceeded maximum number of points'   }       ;

                                                % Warning codes                %
   Sg0_qgd.W_UNKWRN  = uint32( hex2dec( '11000000' ) )                  ;

   Sg0_qgd.C_WRNSTR  = {                                                ...
        'QCM-WARNING-UNKWRN: Unknown warning'                        }       ;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % INTEGER CONSTANTS            %
   Sg0_qgd.J_DBGLV0     = int32(  0 )   ;       % Level 0: No debug            %
   Sg0_qgd.J_DBGLV1     = int32(  1 )   ;       % Level 1: Print function      %
   Sg0_qgd.J_DBGLV2     = int32(  2 )   ;       % Level 2: Print status        %
   Sg0_qgd.J_DBGLV3     = int32(  3 )   ;       % Level 3: Full debug          %
   Sg0_qgd.J_DBGLV4     = int32(  4 )   ;       % Level 4: Special debug       %

   Sg0_qgd.J_QCM4PR     = int32(  1 )   ;       % 4-parameter QCM (QCM-4)      %
   Sg0_qgd.J_QCM5PR     = int32(  2 )   ;       % 5-parameter QCM (QCM-5)      %
   Sg0_qgd.J_QCM6PR     = int32(  3 )   ;       % 6-parameter QCM (QCM-6)      %
   Sg0_qgd.J_QCMB4P     = int32( 11 )   ;       % QCM-4 formulation B          %
   Sg0_qgd.J_QCMC4P     = int32( 12 )   ;       % QCM-4 formulation C          %
   Sg0_qgd.J_QCMD4P     = int32( 13 )   ;       % QCM-4 formulation D          %
   Sg0_qgd.J_QCME4P     = int32( 14 )   ;       % QCM-4 formulation E          %
   Sg0_qgd.J_QCMB4V     = int32( 15 )   ;       % QCM-4 formulation B variable %
   Sg0_qgd.J_QCMC4V     = int32( 16 )   ;       % QCM-4 formulation C variable %
   Sg0_qgd.J_QCMD4V     = int32( 17 )   ;       % QCM-4 formulation D variable %
   Sg0_qgd.J_QCME4V     = int32( 18 )   ;       % QCM-4 formulation E variable %
   Sg0_qgd.J_NAM1EX     = int32( 21 )   ;       % Alt: 1-stage exponential     %
   Sg0_qgd.J_NAM1EL     = int32( 22 )   ;       % Alt: 1-stage exp w/ lag      %
   Sg0_qgd.J_NAM2EX     = int32( 23 )   ;       % Alt: 2-stage exponential     %
   Sg0_qgd.J_NAM2EL     = int32( 24 )   ;       % Alt: 2-stage exp w/ lag      %
   Sg0_qgd.J_NAMHQ4     = int32( 25 )   ;       % Alt: QCM-4 hybrid            %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % DOUBLE CONSTANTS             %
   Sg0_qgd.D_MV         = NaN           ;       % Missing value                %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOGICAL CONSTANTS            %
   if ( nargin == 0 )                           % No argument -> false         %
      Sg0_qgd.L_DEBUG = false           ;
   elseif ( nargin ~= 1 )                       % More than one arg -> false   %
      Sg0_qgd.L_DEBUG = false           ;
   elseif ( varargin{1} == true )               % 1 arg of true -> true        %
      Sg0_qgd.L_DEBUG = true            ;
   else                                         % Anything else -> false       %
      Sg0_qgd.L_DEBUG = false           ;
   end

   return                               ;

end                                             % END OF FUNCTION              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_fit_vqm                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           05-MAY-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      08/15/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       Given a set of measured growth count data, this function uses the non- %
% linear solver MINIMIZE  to find the optimal rate constants (k) for the 4-pa- %
% rameter Quasi-Chemical Model (QCM-4), and is based on function  SD discussed %
% in Reference [a]. The function outputs the optimal rate constants, and model %
% counts at times di1_tim_vec.  This function is designed to find  the parame- %
% ters that minimize the sum of squared  residuals for the following nine mod- %
% els and model formulations                                                   %
%                                                                              %
%               QCM-4 Formulation A (fixed)                                    %
%               QCM-4 Formulation B (fixed and variable)                       %
%               QCM-4 Formulation C (fixed and variable)                       %
%               QCM-4 Formulation D (fixed and variable)                       %
%               QCM-4 Formulation E (fixed and variable)                       %
%               QCM-4 Alternate: 1-stage exponential                           %
%               QCM-4 Alternate: 1-stage exponential with lag                  %
%               QCM-4 Alternate: 2-stage exponential                           %
%               QCM-4 Alternate: 2-stage exponential with lag                  %
%                                                                              %
%       The four  fixed models use a four  element vector to  specify the rate %
% constants in the form [k1 k2 k3 k4].  In the case of Formulation C, the rate %
% constant vector  will be converted to [k1 k2 k3 alpha] within  the function, %
% and the  output will be returned as [k1 k2 k3 k4]. For Formulations D and E, %
% the input  and output rate  constant vector is the natural  logarithm of the %
% rate constants. The minimum and maximum bound vectors for the rate constants %
% in Si0_ops are [k1 k2 k3 k4]  for Formulations A and B, [k1 k2 k3 alpha] for %
% Formulation C, or  [z1 z2 z3 z4] for Formulations D and E where zi = ln(ki). %
%                                                                              %
%       To calculate a variable solution, the value of ni0_mod_num must be set %
% to  .J_QCMB4V, .J_QCMC4V, or .J_QCMD4V; the  initial  rate  constant  vector %
% di1_ini_rte  must have  four values;  and  the rate  constant  index  vector %
% ni1_rsi_vec must contain the list of indexes (1-4) of the rate constants be- %
% ing solved  for.  The minimization will create a rate  constant vector sized %
% to match ni1_rsi_vec, and the  fixed rate constant values will be  filled in %
% using  the initial rate constants in  di1_ini_rte  in the interface function %
% qm4_nlv_int.  The optimal  rate constant  vector will be  returned as either %
% [k1 k2 k3 k4] or [z1 z2 z3 z4].                                              %
%                                                                              %
%       For the  four alternate models which can have one to  four parameters, %
% the incoming vector must have four parameters, and the extra parameters will %
% be ignored as necessary. If X represents an ignored parameter, then the for- %
% mat of the input/output parameter vectors is                                 %
%                                                                              %
%       1-stage exponential             [ a X  X   X  ]                        %
%       1-stage exponential with lag    [ a k1 X   X  ]                        %
%       2-stage exponential             [ a b  tau X  ]                        %
```

```
%       2-stage exponential with lag    [ a b  tau k1 ]                         %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                                   Parameters                                  %
%                                                                               %
%       Parameter       Type            Description                             %
%                                                                               %
% (I)   di1_tim_vec     double(1,n)     This is the time vector associated with%
%                                       the measured counts in days.            %
%                                                                               %
% (I)   di1_cnt_mes     double(1,n)     This is the measured bacterial counts  %
%                                       in CFU/mL which will be fit with the   %
%                                       QCM.                                    %
%                                                                               %
% (I)   di1_ini_rte     double(4,1)     This is the vector of initial rate     %
%                                       constants [k1 k2 k3 k4] for formulation%
%                                       A, B, or C, and [z1 z2 z3 z4] for D or %
%                                       E.  For the alternate models, this vec-%
%                                       tor must still be of length 4, and ex- %
%                                       tra parameters will be ignored.         %
%                                                                               %
% (I)   di1_ini_cnt     double(4,1)     Initial count for x1-x4 in CFU/mL.     %
%                                                                               %
% (I)   ni1_rsi_vec     uint32          Rate constant solution index vector    %
%                                       for variable solutions (optional).  If %
%                                       ni0_mod_num is .J_QCMB4V, .J_QCMC4V, or%
%                                       .J_QCMD4V this vector holds the indexes%
%                                       (1-4) of the parameters to be solved   %
%                                       for.  The vector must be sorted.        %
%                                                                               %
% (I)   di1_evl_tim     double(m,1)     Evaluation time vector (day).  This is %
%                                       an optional parameter.  If blank or    %
%                                       empty, sensitivity calculations will   %
%                                       use di1_tim_vec.                        %
%                                                                               %
% (I)   ni0_mod_num     uint32          Model number defined in qcm_ini_prm.   %
%                                       = .J_QCM4PR QCM-4 formulation A.        %
%                                       = .J_QCMB4P QCM-4 formulation B.        %
%                                       = .J_QCMC4P QCM-4 formulation C.        %
%                                       = .J_QCMD4P QCM-4 formulation D.        %
%                                       = .J_QCMD4P QCM-4 formulation E.        %
%                                       = .J_QCMB4V QCM-4 form B variable.      %
%                                       = .J_QCMC4V QCM-4 form C variable.      %
%                                       = .J_QCMD4V QCM-4 form D variable.      %
%                                       = .J_QCMD4V QCM-4 form E variable.      %
%                                       = .J_NAM1EX Alt: 1-stage exponential    %
%                                       = .J_NAM1EL Alt: 1-stage exp w/ lag     %
%                                       = .J_NAM2EX Alt: 2-stage exponential    %
%                                       = .J_NAM2EL Alt: 2-stage exp w/ lag     %
%                                                                               %
% (I)   Si0_ops         struct          Optimization parameter structure.      %
%                                                                               %
% (I)   Si0_ocs         struct          An optional arg for output control.    %
%                                       Parameters defined in qcm_ini_prm.  If %
%                                       blank or empty, set to no debug.        %
%                                                                               %
% (O)   do1_out_tim     double(m,1)     Output vector of times in days.        %
%                                                                               %
% (O)   do1_out_cnt     double(m,1)     Vector of estimated bacterial counts   %
%                                       in CFU/mL at each value of do1_out_tim.%
%                                                                               %
% (O)   do2_cnt_mat     double(m,4)     Matrix of estimated bacterial counts   %
%                                       in CFU/mL at each value of do1_out_tim %
%                                       and for the four state parameters.      %
%                                                                               %
% (O)   do1_rte_con     double(4,1)     This is the vector of rate constants   %
%                                       (k) calculated for the QCM that best   %
```

```
%                                       fits the data.                          %
%                                                                               %
% (O)   do0_min_ssr     double          Minimum SSR value (CFU/mL).             %
%                                                                               %
% (O)   no0_src         uint32          Status return code.                     %
%                                       = .S_SUCCESS  Successful completion.    %
%                                       = Other - see qcm_ini_prm.              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                               Structure Fields                                %
%                                                                               %
% Si0_ops:                                                                      %
%                                                                               %
%       .d1mink         double(1,4)     Vector of minimum k or alpha values     %
%       .d1maxk         double(1,4)     Vector of maximum k or alpha values     %
%       .d0atol         double          Absolute tolerance value                %
%       .d0rtol         double          Relative tolerance value                %
%                                                                               %
% Si0_ocs:                                                                      %
%                                                                               %
%       .j0dbgf         int32           Debug flag (.J_DBGLV0 to .J_DBGLV4)     %
%       .j0dfid         int32           Debug output file ID number             %
%       .c1dfnm         char            Debug filename                          %
%       .j0pltf         int32           Plot flag (true or false)               %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                                   References                                  %
%                                                                               %
% a. Ross, E.W., et al., The mathematical properties of the quasi-chemical      %
%    model for microorganism growth-death kinetics in foods, International      %
%    Journal of Food Microbiology, Volume 99, pps. 157-171, 2005.               %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % Fit variable QCM to data     %
function                  ...
    [                     ...
        do1_out_tim     , ...                           % Output vector of times (day) %
        do1_out_cnt     , ...                           % Output count vector (CFU/mL) %
        do2_cnt_mat     , ...                           % Count matrix from ODE        %
        do1_rte_con     , ...                           % Final rate constants (k-vals)%
        do0_min_ssr     , ...                           % Minimum SSR (CFU/mL)         %
        no0_src           ...                           % Status return code           %
    ] =                   ...
   qm4_fit_vqm            ...
    (                     ...
        di1_tim_vec     , ...                           % Input time vector (day)      %
        di1_cnt_mes     , ...                           % Input count vector (CFU/mL)  %
        di1_ini_rte     , ...                           % Initial rate constants (k)   %
        di1_ini_cnt     , ...                           % Initial count vector (x0)    %
        ni1_rsi_vec     , ...                           % Rate cnst solution index vec %
        di1_evl_tim     , ...                           % Evaluation time (optional)   %
        ni0_mod_num     , ...                           % Model number                 %
        Si0_ops         , ...                           % Optimization parameters      %
        Si0_ocs           ...                           % Output control structure     %
    )
                                                        % PARAMETER DECLARATIONS       %
   do1_out_tim          = []            ;
   do1_out_cnt          = []            ;
   do2_cnt_mat          = []            ;
   do1_rte_con          = []            ;
   do0_min_ssr          = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOAD GLOBALS                %
   persistent Sc0_qgd                 ;         % Global QCM data             %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )       % Not initialized             %
      Sc0_qgd = qcm_ini_prm()         ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOCAL DECLARATIONS          %
   dl2_iav      = double( 0.0 )       ;         % Inequality constraint A mat  %

   dl1_rat_cnt  = double( 0.0 )       ;         % Optimization rate constants  %
   dl1_ini_rte  = double( 0.0 )       ;         % Initial rate constants (k)   %
   dl1_sol_vec  = double( 0.0 )       ;         % Solution vector              %
   dl1_klb      = double( 0.0 )       ;         % Rate constant lower bounds   %
   dl1_kub      = double( 0.0 )       ;         % Rate constant upper bounds   %
   dl1_ibv      = double( 0.0 )       ;         % Inequality constraint b vec  %

   dl0_min_ssr  = double( 0.0 )       ;         % Minimum SSR value            %
   dl0_tm1      = double( 0.0 )       ;         % Temporary variables          %
   dl0_tm2      = double( 0.0 )       ;
   dl0_tm3      = double( 0.0 )       ;

   m            = uint32( 0 )         ;         % Counters/indexes             %
   n            = uint32( 0 )         ;

   jl0_num_pts  = int32( 0 )          ;         % Number of points             %
   jl0_dbg_flg  = int32( 0 )          ;         % Local debug flag             %
   jl0_rcv_len  = int32( 0 )          ;         % Rate constant vector length  %
   jl0_k2m      = int32( 0 )          ;         % k2 member of ni1_rsi_vec     %
   jl0_alm      = int32( 0 )          ;         % alpha member of ni1_rsi_vec  %
   jl0_dfi      = int32( 0 )          ;         % Debug file ID number         %

   cl1_dsp_opt  = char( '' )          ;         % Display option               %

   Sl1_sop      = []                  ;         % Solver options               %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS         %
   jc0_mxr_len  = int32( 4 )          ;         % Maximum rate constant length %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                 %
   if ( ~exist( 'Si0_ocs' , 'var' ) || ...      % No output control structure  %
        isempty( Si0_ocs )         )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0                      ; % Local debug flag   %
      jl0_dfi     = 1                                     ; % Debug FID          %
   elseif ( Si0_ocs.j0dbgf < Sc0_qgd.J_DBGLV0 ) % Debug must be non-negative   %
      no0_src     = Sc0_qgd.E_INVARG                      ; % Set status code    %
      return                                              ; % Return             %
   elseif ( Si0_ocs.j0dbgf > Sc0_qgd.J_DBGLV0 ) % Valid debug                  %
      jl0_dbg_flg = Si0_ocs.j0dbgf                        ; % Local debug flag   %
      jl0_dfi     = Si0_ocs.j0dfid                        ; % Debug FID          %
      fprintf( jl0_dfi , '*** qm4_fit_vqm ***\n' )        ; % Display function   %
   end

   if ( ( ~isvector( di1_tim_vec ) ) || ...     % Must be vectors              %
        ( ~isvector( di1_cnt_mes ) ) || ...
        ( ~isvector( di1_ini_rte ) ) || ...
        ( ~isvector( di1_ini_cnt ) ) )
      no0_src = Sc0_qgd.E_INCARG      ;
      return                          ;
   end
                                                % Must be correct length       %
   if ( ( length( di1_ini_rte ) ~= jc0_mxr_len ) || ...
```

```
     ( length( di1_ini_cnt ) ~= jc0_mxr_len ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                          % Must be valid model type     %
if ( ( ni0_mod_num ~= Sc0_qgd.J_QCM4PR ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCMB4P ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCMC4P ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCMD4P ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCME4P ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCMB4V ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCMC4V ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCMD4V ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCME4V ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_NAM1EX ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_NAM1EL ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_NAM2EX ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_NAM2EL ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

if ( ~exist( 'di1_evl_tim' , 'var' ) || ...  % No evaluation time           %
     isempty( di1_evl_tim )        )
   di1_evl_tim = di1_tim_vec         ;       % Set to count times           %
elseif ( ~isvector( di1_evl_tim ) )          % Must be a vector             %
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%    The input or global rate constant vector (di1_ini_rte) always has four %
% elements.  The  MINIMIZE function must use a vector with only the parame- %
% ters being minimized over.  This requires  creating a local rate constant %
% vector (dl1_ini_rte). For formulation C (fixed or variable), element four %
% in the global vector is first changed from k4 to alpha.  For fixed formu- %
% lations A, B, C, D, and E the local vector is  set to the  global vector. %
% For the  variable formulations, the local vector is constructed  from the %
% elements in the global vector specified by ni1_rsi_vec. For the alternate %
% models, the  local vector is constructed from  the first one to four ele- %
% ments of the global vector, depending on the model.                      %
%                                                                              %
%    The MINIMIZE  function allows  for specifying constraints.  All models %
% will use both upper and lower bounds specified by the .d1mink and .d1maxk %
% fields in the Si0_ops structure. Formulation C requires linear inequality %
% constraints on  k2 and alpha  to guarantee that k4 is positive.  The ine- %
% quality is  set by specifying a matrix A and vector b such that  A*x <= b %
% where x is the rate  parameter vector.  Two constraints  are required for %
% formulation C:  -k2 + alpha <= 0, and k2 - alpha <= k4max.  For the fixed %
% model, k2 and alpha are variables and are set in A directly. For the var- %
% iable  model, the construction  of A and b depends  on which of k2 and/or %
% alpha is in the set of variable rate constants. Four different conditions %
% are set up in the code to specify A and b. When k2 or alpha is not a var- %
% iable, its value  is extracted from the  global vector since its location %
% is known within the vector.                                              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                          % Set parameters                %
if     ( jl0_dbg_flg == Sc0_qgd.J_DBGLV0 )   % Set MINIMIZE display option  %
   cl1_dsp_opt = 'off'                    ;
elseif ( jl0_dbg_flg == Sc0_qgd.J_DBGLV1 )
   cl1_dsp_opt = 'off'                    ;
elseif ( jl0_dbg_flg == Sc0_qgd.J_DBGLV2 )
   cl1_dsp_opt = 'off'                    ;
elseif ( jl0_dbg_flg == Sc0_qgd.J_DBGLV3 )
   cl1_dsp_opt = 'on'                     ;
elseif ( jl0_dbg_flg == Sc0_qgd.J_DBGLV4 )
```

```
   cl1_dsp_opt = 'iter-detailed'             ;
end
                                             % Convert global k4 to alpha   %
if ( ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCMC4V ) )
   di1_ini_rte(4) = di1_ini_rte(2) - ...             % alpha = k2 - k4      %
                    di1_ini_rte(4)           ;
end
                                             % Set model local parameters   %

if ( ( ni0_mod_num == Sc0_qgd.J_QCMB4V ) || ...      % Variable formulation %
     ( ni0_mod_num == Sc0_qgd.J_QCMC4V ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCMD4V ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCME4V ) )
   if ( ~exist('ni1_rsi_vec','var') || ...           % At least one param   %
        isempty( ni1_rsi_vec )      )
      no0_src = Sc0_qgd.E_INCARG             ;
      return                                 ;
   end

   if ( ~issorted( ni1_rsi_vec ) )                   % Must be sorted       %
      no0_src = Sc0_qgd.E_INCARG             ;
      return                                 ;
   end

   jl0_rcv_len = length( ni1_rsi_vec )       ;
   dl1_ini_rte = di1_ini_rte( ni1_rsi_vec )  ;

   if ( ni0_mod_num == Sc0_qgd.J_QCMC4V )            % Special case         %
      Si0_ops.d1mink(4) = -Si0_ops.d1maxk(4) ;
   end
elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EX )           % Alternate models     %
   jl0_rcv_len = 1                           ;
   dl1_ini_rte = di1_ini_rte(1)              ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EL )
   jl0_rcv_len = 2                           ;
   dl1_ini_rte = di1_ini_rte(1:2)            ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EX )
   jl0_rcv_len = 3                           ;
   dl1_ini_rte = di1_ini_rte(1:3)            ;
else                                                 % Formulation A-D      %
   jl0_rcv_len = 4                           ;
   dl1_ini_rte = di1_ini_rte                 ;
end
                                             % Check vector orientation     %
if ( ~isrow( di1_tim_vec ) ) , di1_tim_vec = di1_tim_vec' ; end ;
if ( ~isrow( di1_cnt_mes ) ) , di1_cnt_mes = di1_cnt_mes' ; end ;
if (  isrow( di1_evl_tim ) ) , di1_evl_tim = di1_evl_tim' ; end ;

jl0_num_pts = length( di1_tim_vec )          ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
% Options for MINIMIZE:                                                        %
%    TolCon                  Tolerance on constraint.                          %
%    TolFun                  Termination tolerance on function value.          %
%    FinDiffType             Type of finite differences used.                  %
%    popsize                 Number of randomized initial values.              %
%    Display                 Algorithm output display level.                   %
%    AlwaysHonorConstraints  Level of constraint enforcement.                  %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Set solver parameters        %
if ( ( ni0_mod_num == Sc0_qgd.J_QCMB4V ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCMC4V ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCMD4V ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCME4V ) )
   dl1_klb = Si0_ops.d1mink( ni1_rsi_vec )   ;       % Min k vals for solver%
```

```
   dl1_kub = Si0_ops.d1maxk( ni1_rsi_vec )   ;       % Max k vals for solver%
else
   dl1_klb = Si0_ops.d1mink                  ;       % Min k vals for solver%
   dl1_kub = Si0_ops.d1maxk                  ;       % Max k vals for solver%
end

Sl1_sop = setoptimoptions(             ...   % Options for MINIMIZE         %
             'TolCon'                , Si0_ops.d0atol       , ...
             'TolFun'                , Si0_ops.d0atol       , ...
             'FinDiffType'           , 'central'            , ...
             'popsize'               , jl0_rcv_len * 50     , ...
             'Display'               , cl1_dsp_opt          , ...
             'AlwaysHonorConstraints' , 'All'                )       ;

                                                 % Model specific conditions    %
if ( ni0_mod_num == Sc0_qgd.J_QCMC4P )           % Constraints for formulation C%
   dl2_iav        = [ 0 -1 0  1  ; ...           % -k2 + alpha                  %
                      0  1 0 -1  ]   ;           % k2 - alpha                   %
   dl1_ibv        = [ 0.0        ; ...           % <= 0.0                       %
                      dl1_kub(4) ]   ;           % <= k4-max                    %
elseif ( ni0_mod_num == Sc0_qgd.J_QCMC4V )       % Constraints for form C var   %
   jl0_k2m = ismember(2,ni1_rsi_vec) ;           % Is k2 in vector              %
   jl0_alm = ismember(4,ni1_rsi_vec) ;           % Is alpha in vector           %

   dl2_iav = zeros( 2 , jl0_rcv_len );           % Define A matrix              %
   dl1_ibv = zeros( 2 , 1           );           % Define b vector              %

   if ( jl0_k2m && jl0_alm )                     % k2 and alpha                 %
      dl0_tm3      = Si0_ops.d1maxk(4)           ;       % k4max                %
      m            = find(ni1_rsi_vec==2)        ;       % k2 index             %
      n            = find(ni1_rsi_vec==4)        ;       % alpha index          %

      dl2_iav(1,m) = -1                          ;       % -k2                  %
      dl2_iav(2,m) =  1                          ;       % k2                   %
      dl2_iav(1,n) =  1                          ;       % alpha                %
      dl2_iav(2,n) = -1                          ;       % -alpha               %

      dl1_ibv(1)   = 0.0                         ;       % <= 0.0               %
      dl1_ibv(2)   = dl0_tm3                     ;       % <= k4max             %
   elseif ( jl0_k2m )                            % k2 only                      %
      dl0_tm3      = Si0_ops.d1maxk(4)           ;       % k4max                %
      dl0_tm1      = -di1_ini_rte(4)             ;       % -alpha               %
      dl0_tm2      = dl0_tm3        + ...                % k4max + alpha        %
                     di1_ini_rte(4)              ;
      m            = find(ni1_rsi_vec==2)        ;       % k2 index             %

      dl2_iav(1,m) = -1                          ;       % -k2                  %
      dl2_iav(2,m) =  1                          ;       % k2                   %

      dl1_ibv(1)   = dl0_tm1                     ;
      dl1_ibv(2)   = dl0_tm2                     ;
   elseif ( jl0_alm )                            % alpha only                   %
      dl0_tm3      = Si0_ops.d1maxk(4)           ;       % k4max                %
      dl0_tm1      = di1_ini_rte(2)              ;       % k2                   %
      dl0_tm2      = dl0_tm3        - ...               % k4max - k2           %
                     di1_ini_rte(2)              ;
      m            = find(ni1_rsi_vec==4)        ;       % alpha index          %

      dl2_iav(1,m) =  1                          ;       % alpha                %
      dl2_iav(2,m) = -1                          ;       % -alpha               %

      dl1_ibv(1)   = dl0_tm1                     ;
      dl1_ibv(2)   = dl0_tm2                     ;
   else                                          % Neither k2 nor alpha         %
      dl2_iav      = []                          ;       % No constraints       %
      dl1_ibv      = []                          ;
   end
else
```

```
   dl2_iav        = []                ;       % Linear inequality constraints%
   dl1_ibv        = []                ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                               % Find optimal k-values        %
 [                                ...
     do1_rte_con             , ...             % Optimal rate constants       %
     dl0_min_ssr             , ...             % Min function value (SSR)     %
 ] =                              ...
minimize                          ...           % Nonlinear solver             %
 (                                ...
     @(dl1_rat_cnt)               ...           % Rate constant control value  %
     qm4_nlv_int                  ...           % Nonlin var solver interface  %
      (                           ...
             dl1_rat_cnt                , ...   % Rate constants               %
             di1_tim_vec                , ...   % Time vector                  %
             di1_cnt_mes                , ...   % Count vector                 %
             di1_ini_cnt                , ...   % Initial count vector         %
             ni1_rsi_vec                , ...   % Rate cnst solution index vec %
             di1_ini_rte                , ...   % Initial rate constants       %
             ni0_mod_num                , ...   % Model number                 %
             []                           ...   % Output control structure     %
      )                           , ...
     dl1_ini_rte'                , ...           % Initial rate constants       %
     dl2_iav , dl1_ibv           , ...           % Linear inequality constraints%
     []      , []                , ...           % Linear equality constraints  %
     dl1_klb , dl1_kub           , ...           % Bounds                       %
     []      , Sl1_sop             ...           % Nonlin constraints; options  %
 )                                   ;
                                               % Adjust variable QCM-4 output %
if ( ( ni0_mod_num == Sc0_qgd.J_QCMB4V ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCMC4V ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCMD4V ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCME4V ) )
   dl1_rat_cnt              = di1_ini_rte    ;
   dl1_rat_cnt(ni1_rsi_vec) = do1_rte_con    ;
   do1_rte_con              = dl1_rat_cnt    ;
end
                                               % Convert alpha to k4          %
if ( ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) || ...
     ( ni0_mod_num == Sc0_qgd.J_QCMC4V ) )
   do1_rte_con(4) = do1_rte_con(2) - ...       % k4 = k2 - alpha              %
                    do1_rte_con(4)             ;
end

do0_min_ssr = dl0_min_ssr            ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                               % Get model output values      %
do1_out_tim = di1_evl_tim            ;         % Output times (days)          %

if ( ni0_mod_num == Sc0_qgd.J_QCM4PR )         % QCM-4 formulation A          %
    [                  ...                     % Solve ODE                    %
     do1_out_cnt     , ...                     % Model count vector           %
     do2_cnt_mat     , ...                     % Model count matrix           %
     no0_src           ...                     % Status return code           %
    ] =                ...
   qm4_slv_ode         ...
    (                  ...
     do1_rte_con     , ...                     % Rate constants (k)           %
     di1_ini_cnt     , ...                     % Initial counts (x0)          %
     do1_out_tim     , ...                     % Time vector (days)           %
     jl0_dbg_flg       ...                     % Debug flag                   %
    )                                          ;
elseif ( ( ni0_mod_num == Sc0_qgd.J_QCMB4P ) || ... % QCM-4 formulation B   %
         ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) || ... % QCM-4 formulation C   %
         ( ni0_mod_num == Sc0_qgd.J_QCMB4V ) || ... % QCM-4 form B variable %
```

```
         ( ni0_mod_num == Sc0_qgd.J_QCMC4V ) )      % QCM-4 form C variable %
    [                  ...                    % Solve ODE                    %
     do1_out_cnt     , ...                    % Model count vector           %
     do2_cnt_mat     , ...                    % Model count matrix           %
     no0_src           ...                    % Status return code           %
    ] =                ...
   Bq4_slv_ode         ...
    (                  ...
     do1_rte_con     , ...                    % Rate constants (k)           %
     di1_ini_cnt     , ...                    % Initial counts (x0)          %
     do1_out_tim     , ...                    % Time vector (days)           %
     jl0_dbg_flg       ...                    % Debug flag                   %
    )                                         ;
elseif ( ( ni0_mod_num == Sc0_qgd.J_QCMD4P ) || ... % QCM-4 formulation D   %
         ( ni0_mod_num == Sc0_qgd.J_QCME4P ) || ... % QCM-4 formulation E   %
         ( ni0_mod_num == Sc0_qgd.J_QCMD4V ) || ... % QCM-4 form D variable %
         ( ni0_mod_num == Sc0_qgd.J_QCME4V ) )      % QCM-4 form E variable %
    [                  ...                    % Solve ODE                    %
     do1_out_cnt     , ...                    % Model count vector           %
     do2_cnt_mat     , ...                    % Model count matrix           %
     no0_src           ...                    % Status return code           %
    ] =                ...
   Dq4_slv_ode         ...
    (                  ...
     do1_rte_con     , ...                    % Rate constants (z)           %
     di1_ini_cnt     , ...                    % Initial counts (x0)          %
     do1_out_tim     , ...                    % Time vector (days)           %
     jl0_dbg_flg       ...                    % Debug flag                   %
    )                                         ;
else                                          % QCM-4 alternate models       %
   if ( ni0_mod_num == Sc0_qgd.J_NAM1EX )
      dl1_sol_vec = [ do1_rte_con(1) , ...           % a parameter          %
                      NaN            , ...
                      NaN            , ...
                      NaN            ]       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EL )
      dl1_sol_vec = [ do1_rte_con(1) , ...           % a parameter          %
                      do1_rte_con(2) , ...           % k1 parameter         %
                      NaN            , ...
                      NaN            ]       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EX )
      dl1_sol_vec = [ do1_rte_con(1) , ...           % a parameter          %
                      do1_rte_con(2) , ...           % b parameter          %
                      do1_rte_con(3) , ...           % tau parameter        %
                      NaN            ]       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EL )
      dl1_sol_vec = [ do1_rte_con(1) , ...           % a parameter          %
                      do1_rte_con(2) , ...           % b parameter          %
                      do1_rte_con(3) , ...           % tau parameter        %
                      do1_rte_con(4) ]       ;       % k1 parameter         %
   end

    [                          ...           % Evaluate alternate model     %
     do1_out_cnt     , ...                    % Model count vector (CFU/mL)  %
     nl0_src           ...                    % Status return code           %
    ] =                ...
   qm4_evl_nam         ...
    (                  ...
     ni0_mod_num     , ...                    % Model number                 %
     dl1_sol_vec     , ...                    % Solution vector              %
     do1_out_tim'    , ...                    % Time vector (day)            %
     di1_ini_cnt(1)  , ...                    % Initial count (CFU/mL)       %
     jl0_dbg_flg       ...                    % Debug flag                   %
    )                                 ;

   do2_cnt_mat = do1_out_cnt         ;       % Only 1 state parameter       %
end
```

```
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Debug                     %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                    ...
        jl0_dfi                                 , ...
        'Time: \t%d\tLog(count): \t%f\n'        , ...
        do1_out_tim(  1)                        , ...
        log10( do1_out_cnt(  1) )               )       ;
      fprintf(                                    ...
        jl0_dfi                                 , ...
        'Time: \t%d\tLog(count): \t%f\n'        , ...
        do1_out_tim(end)                        , ...
        log10( do1_out_cnt(end) )               )       ;

      fprintf( jl0_dfi , '\n' )                         ;
   end
end                                             % END OF FUNCTION           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_nlv_int                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           06-MAY-2015                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      08/15/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function is an interface for the nonlinear solver MINIMIZE. It is %
% called from qm4_fit_vqm as the function argument to the solver MINIMIZE.  If %
% the  optimization  is for one of the  QCM-4 formulations, then  qm4_slv_ode, %
% Bq4_slv_ode, or Dq4_slv_ode  is called with the rate  constants passed in by %
% MINIMIZE.  For the variable  version of  formulations B-E, the rate constant %
% vector  is constructed from the passed rate  constants and the  initial rate %
% constants.  If  one of the  four alternate models is  being  optimized, then %
% qm4_evl_nam is called.  The  solution counts are compared  with the measured %
% counts, and the sum of squared residuals (SSR) is calculated.  The SSR value %
% is  returned  to the solver as the  objective function  value making this an %
% optimization  in the  least-squares sense. If an error occurs in solving the %
% ODEs, the SSR is set to the MATLAB value inf.                                %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di1_rte_con     double(m,1)     Vector of rate constants (k) to evalu- %
%                                       ate the ODEs at for the QCM.  This must%
%                                       be the first input argument when using %
%                                       an optimization solver.  The number of %
%                                       rows (m) is dependent on the model.    %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     This is the time vector associated with%
%                                       the measured counts in days.           %
%                                                                              %
% (I)   di1_cnt_mes     double(n,1)     This is the vector of measured bacter- %
%                                       ial counts in CFU/mL which will be fit %
%                                       with the QCM.                          %
%                                                                              %
% (I)   di1_ini_cnt     double(4,1)     Initial counts of x1-x4 in CFU/mL.     %
%                                                                              %
% (I)   ni1_rsi_vec     uint32          Rate constant solution index vector    %
%                                       for variable solutions (optional).  If %
%                                       ni0_mod_num is .J_QCMB4V, .J_QCMC4V, or%
%                                       .J_QCMD4V this vector holds the indexes%
%                                       (1-4) of the parameters to be solved   %
%                                       for.  The vector must be sorted.       %
%                                                                              %
% (I)   di1_ini_rte     double(4,1)     This is the vector of initial rate     %
%                                       constants (optional).  If ni0_mod_num  %
%                                       is .J_QCMB4V, .J_QCMC4V, or .J_QCMD4V, %
%                                       this vector is used to insert the rate %
```

```
%                                       parameters not being solved for.       %
%                                                                              %
% (I)   ni0_mod_num     uint32          Model number defined in qcm_ini_prm.   %
%                                       = .J_QCM4PR QCM-4 formulation A.       %
%                                       = .J_QCMB4P QCM-4 formulation B.       %
%                                       = .J_QCMC4P QCM-4 formulation C.       %
%                                       = .J_QCMD4P QCM-4 formulation D.       %
%                                       = .J_QCMB4V QCM-4 form B variable.     %
%                                       = .J_QCMC4V QCM-4 form C variable.     %
%                                       = .J_QCMD4V QCM-4 form D variable.     %
%                                       = .J_NAM1EX Alt: 1-stage exponential   %
%                                       = .J_NAM1EL Alt: 1-stage exp w/ lag    %
%                                       = .J_NAM2EX Alt: 2-stage exponential   %
%                                       = .J_NAM2EL Alt: 2-stage exp w/ lag    %
%                                                                              %
% (I)   Si0_ocs         struct          An optional arg for output control.    %
%                                       Parameters defined in qcm_ini_prm.  If %
%                                       blank or empty, set to no debug.       %
%                                                                              %
% (O)   do0_ssr_val     double          Sum of squared residuals between model %
%                                       and measured values.  This is the val- %
%                                       ue of the objective function for the   %
%                                       nonlinear optimization, and must be    %
%                                       the first output parameter.            %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                               Structure Fields                               %
%                                                                              %
% Si0_ocs:                                                                     %
%                                                                              %
%       .j0dbgf         int32           Debug flag (.J_DBGLV0 to .J_DBGLV4)    %
%       .j0dfid         int32           Debug output file ID number            %
%       .c1dfnm         char            Debug filename                         %
%       .j0pltf         int32           Plot flag (true or false)              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Nonlin var solver interface  %
function                  ...
    [                     ...
        do0_ssr_val     , ...                   % SSR value                    %
        no0_src           ...                   % Status return code           %
    ] =                   ...
   qm4_nlv_int            ...
    (                     ...
        di1_rte_con     , ...                   % Rate constants (k)           %
        di1_tim_vec     , ...                   % Input time vector (day)      %
        di1_cnt_mes     , ...                   % Input count vector (CFU/mL)  %
        di1_ini_cnt     , ...                   % Initial count vector (x0)    %
        ni1_rsi_vec     , ...                   % Rate cnst solution index vec %
        di1_ini_rte     , ...                   % Initial rate constants (opt) %
        ni0_mod_num     , ...                   % Model number                 %
        Si0_ocs           ...                   % Output control structure     %
    )
                                                % PARAMETER DECLARATIONS       %
   do0_ssr_val          = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % MATLAB INITIALIZATION        %
   format long                          ;       % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOAD GLOBALS                 %
```

```
   persistent Sc0_qgd                   ;       % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )       % Not initialized             %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOCAL DECLARATIONS          %
   dl2_cnt_mat  = double( 0.0 )         ;       % Full count matrix (CFU/mL)   %

   dl1_rte_con  = double( 0.0 )         ;       % Local rate constant vector   %
   dl1_cnt_mod  = double( 0.0 )         ;       % Model count vector (CFU/mL)  %
   dl1_res_vec  = double( 0.0 )         ;       % Residual vector              %
   dl1_lgc_mod  = double( 0.0 )         ;       % Model log count vector       %
   dl1_lgc_mes  = double( 0.0 )         ;       % Measured log count vector    %
   dl1_sol_vec  = double( 0.0 )         ;       % Solution vector              %

   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag             %
   jl0_rcv_len  = int32( 0 )            ;       % Rate constant vector length  %
   jl0_dfi      = int32( 0 )            ;       % Debug file ID number         %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS        %
   jc0_mxr_len       = 4                ;       % Max rate constant vector len %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                %
   if ( ~exist( 'Si0_ocs' , 'var' ) || ...      % No output control structure  %
        isempty( Si0_ocs )          )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0                    ; % Local debug flag   %
      jl0_dfi     = 1                                   ; % Debug FID          %
   elseif ( Si0_ocs.j0dbgf < Sc0_qgd.J_DBGLV0 ) % Debug must be non-negative   %
      no0_src     = Sc0_qgd.E_INVARG                    ; % Set status code    %
      return                                            ; % Return             %
   elseif ( Si0_ocs.j0dbgf > Sc0_qgd.J_DBGLV0 ) % Valid debug                  %
      jl0_dbg_flg = Si0_ocs.j0dbgf                      ; % Local debug flag   %
      jl0_dfi     = Si0_ocs.j0dfid                      ; % Debug FID          %
      fprintf( jl0_dfi , '*** qm4_nlv_int ***\n' )      ; % Display function   %
   end

   if ( ( ~isvector( di1_tim_vec ) ) || ...     % Must be vectors             %
        ( ~isvector( di1_cnt_mes ) ) || ...
        ( ~isvector( di1_ini_cnt ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Set parameters              %
   if ( ( ni0_mod_num == Sc0_qgd.J_QCMB4V ) || ...      % Variable formulation %
        ( ni0_mod_num == Sc0_qgd.J_QCMC4V ) || ...
        ( ni0_mod_num == Sc0_qgd.J_QCMD4V ) || ...
        ( ni0_mod_num == Sc0_qgd.J_QCME4V ) )
      if ( ~exist('ni1_rsi_vec','var') || ...           % At least one param   %
           isempty( ni1_rsi_vec )      )
         no0_src = Sc0_qgd.E_INCARG             ;
         return                                 ;
      end

      if ( ~exist('di1_ini_rte','var') || ...           % Must exist           %
           isempty( di1_ini_rte )      )
         no0_src = Sc0_qgd.E_INCARG             ;
         return                                 ;
      end

      jl0_rcv_len              = jc0_mxr_len    ;       % Convert to full size %
      dl1_rte_con              = di1_ini_rte    ;       % Init rate constants  %
```

```
   dl1_rte_con(ni1_rsi_vec) = di1_rte_con    ;       % Insert rate constants%
elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EX )           % Alternate models     %
   jl0_rcv_len = 1                           ;
   dl1_rte_con = di1_rte_con                 ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EL )
   jl0_rcv_len = 2                           ;
   dl1_rte_con = di1_rte_con                 ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EX )
   jl0_rcv_len = 3                           ;
   dl1_rte_con = di1_rte_con                 ;
else                                                 % Formulation A-D      %
   jl0_rcv_len = jc0_mxr_len                 ;
   dl1_rte_con = di1_rte_con                 ;
end
                                             % Must be correct length       %
if ( ( length( dl1_rte_con ) ~= jl0_rcv_len ) || ...
     ( length( di1_ini_cnt ) ~= jc0_mxr_len ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

if ( ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) || ...      % QCM-4 formulation C  %
     ( ni0_mod_num == Sc0_qgd.J_QCMC4V ) )           % QCM-4 form C variable%
   dl1_rte_con(4) = dl1_rte_con(2) - ...             % Convert alpha to k4  %
                    dl1_rte_con(4)   ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if ( ni0_mod_num == Sc0_qgd.J_QCM4PR )       % QCM-4 formulation A          %
    [                  ...                   % Solve ODE                    %
     dl1_cnt_mod     , ...                   % Model count vector           %
     dl2_cnt_mat     , ...                   % Model count matrix           %
     no0_src           ...                   % Status return code           %
    ] =                ...
   qm4_slv_ode         ...
    (                  ...
     dl1_rte_con     , ...                   % Rate constants (k)           %
     di1_ini_cnt     , ...                   % Initial counts (x0)          %
     di1_tim_vec     , ...                   % Time vector (days)           %
     jl0_dbg_flg       ...
    )                                        ;
elseif ( ( ni0_mod_num == Sc0_qgd.J_QCMB4P ) || ... % QCM-4 formulation B   %
         ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) || ... % QCM-4 formulation C   %
         ( ni0_mod_num == Sc0_qgd.J_QCMB4V ) || ... % QCM-4 form B variable %
         ( ni0_mod_num == Sc0_qgd.J_QCMC4V ) )      % QCM-4 form C variable %
    [                  ...                   % Solve ODE                    %
     dl1_cnt_mod     , ...                   % Model count vector           %
     dl2_cnt_mat     , ...                   % Model count matrix           %
     no0_src           ...                   % Status return code           %
    ] =                ...
   Bq4_slv_ode         ...
    (                  ...
     dl1_rte_con     , ...                   % Rate constants (k)           %
     di1_ini_cnt     , ...                   % Initial counts (x0)          %
     di1_tim_vec     , ...                   % Time vector (days)           %
     jl0_dbg_flg       ...
    )                                        ;
elseif ( ( ni0_mod_num == Sc0_qgd.J_QCMD4P ) || ... % QCM-4 formulation D   %
         ( ni0_mod_num == Sc0_qgd.J_QCME4P ) || ... % QCM-4 formulation E   %
         ( ni0_mod_num == Sc0_qgd.J_QCMD4V ) || ... % QCM-4 form D variable %
         ( ni0_mod_num == Sc0_qgd.J_QCME4V ) )      % QCM-4 form E variable %
    [                  ...                   % Solve ODE                    %
     dl1_cnt_mod     , ...                   % Model count vector           %
     dl2_cnt_mat     , ...                   % Model count matrix           %
     no0_src           ...                   % Status return code           %
    ] =                ...
   Dq4_slv_ode         ...
```

```
      (                   ...
       dl1_rte_con     , ...                    % Log of rate constants (z)    %
       di1_ini_cnt     , ...                    % Initial counts (x0)           %
       di1_tim_vec     , ...                    % Time vector (days)            %
       jl0_dbg_flg       ...
      )                                          ;
   else                                          % Alternate models              %
      if ( ni0_mod_num == Sc0_qgd.J_NAM1EX )
         dl1_sol_vec = [ dl1_rte_con(1) , ...           % a parameter          %
                         NaN            , ...
                         NaN            , ...
                         NaN            ]       ;
      elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EL )
         dl1_sol_vec = [ dl1_rte_con(1) , ...           % a parameter          %
                         dl1_rte_con(2) , ...           % k1 parameter         %
                         NaN            , ...
                         NaN            ]       ;
      elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EX )
         dl1_sol_vec = [ dl1_rte_con(1) , ...           % a parameter          %
                         dl1_rte_con(2) , ...           % b parameter          %
                         dl1_rte_con(3) , ...           % tau parameter        %
                         NaN            ]       ;
      elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EL )
         dl1_sol_vec = [ dl1_rte_con(1) , ...           % a parameter          %
                         dl1_rte_con(2) , ...           % b parameter          %
                         dl1_rte_con(3) , ...           % tau parameter        %
                         dl1_rte_con(4) ]       ;       % k1 parameter         %
      end

       [                          ...           % Evaluate alternate model     %
        dl1_cnt_mod     , ...                    % Model count vector (CFU/mL)  %
        nl0_src           ...                    % Status return code           %
       ] =                ...
      qm4_evl_nam         ...
       (                  ...
        ni0_mod_num     , ...                    % Model number                 %
        dl1_sol_vec     , ...                    % Solution vector              %
        di1_tim_vec     , ...                    % Time vector (day)            %
        di1_ini_cnt(1)  , ...                    % Initial count (CFU/mL)       %
        jl0_dbg_flg       ...                    % Debug flag                   %
       )                                 ;

      dl2_cnt_mat = dl1_cnt_mod         ;       % Only 1 state parameter       %
   end

   if ( no0_src ~= Sc0_qgd.S_SUCCESS )          % Trap error                   %
      do0_ssr_val = inf                         ;
      return                                    ;
   end
                                                % Calculate SSR of log counts  %
   dl1_lgc_mes = log10( di1_cnt_mes  )          ;       % Log of measured      %
   dl1_lgc_mod = log10( dl1_cnt_mod' )          ;       % Log of model         %
   dl1_res_vec = dl1_lgc_mes - dl1_lgc_mod      ;       % Residuals            %
   do0_ssr_val = sum( dl1_res_vec.^2 )          ;       % SSR                  %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Debug                        %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                    ...
        jl0_dfi                                 , ...
        'SSR: \t%d\n'                           , ...
        do0_ssr_val                             )       ;

      fprintf( jl0_dfi , '\n' )                         ;
   end
end                                             % END OF FUNCTION              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_slv_ode                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           17-NOV-2015                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer                Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      08/15/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function solves the system of ordinary differential equations for %
% the 4-parameter  Quasi-Chemical Model (QCM-4), and  is based on  function ED %
% discussed in Reference [a]. The function uses a MATLAB solver to solve form- %
% ulation A which is a system of four ODEs in the four growth phase state var- %
% iables  M (x1), M* (x2), A (x3), and D (x4).  This function is written to be %
% called by an optimization function such as minimize() or lsqnonlin().        %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of rate constants   %
%                                       (k) for the QCM. This must be the first%
%                                       parameter for the optimization to work.%
%                                                                              %
% (I)   di1_ini_cnt     double(4,1)     This is the initial bacterial counts   %
%                                       (t=0) for the four phases in CFU/mL    %
%                                       (M,M*,A,D or x1,x2,x3,x4).             %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Evaluation time vector in days.  Bac-  %
%                                       terial counts will be estimated at the %
%                                       evaluation times based on the solution %
%                                       to the system of ODEs.                 %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_cnt_vec     double(n,1)     Vector of estimated bacterial counts   %
%                                       in CFU/mL at each value of di1_tim_vec %
%                                       (M+M* or x1+x2).                       %
%                                                                              %
% (O)   do2_cnt_mat     double(n,4)     Matrix of estimated bacterial counts   %
%                                       in CFU/mL at each value of di1_tim_vec %
%                                       and for the four state parameters.     %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%                                                                       %
%                                 References                            %
%                                                                       %
% a. Ross, E.W., et al., The mathematical properties of the quasi-chemical     %
%    model for microorganism growth-death kinetics in foods, International     %
%    Journal of Food Microbiology, Volume 99, pps. 157-171, 2005.              %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % Solve ODE system            %
function                  ...
    [                     ...
        do1_cnt_vec     , ...                           % Count vector from ODE (U)    %
        do2_cnt_mat     , ...                           % Count matrix from ODE (U)    %
        no0_src           ...                           % Status return code           %
    ] =                   ...
   qm4_slv_ode            ...
    (                     ...
        di1_rte_con     , ...                           % Rate constants (k)           %
        di1_ini_cnt     , ...                           % Initial count vector (x0)    %
        di1_tim_vec     , ...                           % Evaluation time vector (td)  %
        ji0_dbg_flg       ...                           % Debug flag                   %
    )
                                                        % PARAMETER DECLARATIONS       %
   do1_cnt_vec          = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;               % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )               % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOCAL DECLARATIONS           %
   dl2_out_cnt  = double( 0.0 )         ;               % Output count matrix          %

   dl1_out_tim  = double( 0.0 )         ;               % Output time vector           %
   dl1_ode_cnt  = double( 0.0 )         ;               % ODE count vector             %

   dl0_ode_tim  = double( 0.0 )         ;               % ODE time value               %

   jl0_dbg_flg  = int32( 0 )            ;               % Local debug flag             %

   Sl0_opt      = []                    ;               % ODE options structure        %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % CONSTANT DEFINITIONS         %
   dc0_abs_tol       = double( 1.0e-12 );               % Absolute tolerance           %
   dc0_rel_tol       = double( 1.0e-4  );               % Relative tolerance           %

   jc0_rcv_len       = 4                ;               % Rate constant vector length  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % BEGINNING OF FUNCTION        %
                                                        % Check inputs                 %
   if ( ~exist('ji0_dbg_flg','var') || ...              % No debug flag                %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )            % Debug must be non-negative   %
      no0_src = Sc0_qgd.E_INVARG        ;               % Set status return code       %
      return                            ;               % Exit function                %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
```

```
   jl0_dbg_flg = ji0_dbg_flg                 ;
   fprintf( 1 , '*** qm4_slv_ode ***\n' )    ;
end

if ( ( ~isvector( di1_rte_con ) ) || ...     % Must be vectors              %
     ( ~isvector( di1_ini_cnt ) ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                             % Must be correct length       %
if ( ( length( di1_rte_con ) ~= jc0_rcv_len ) || ...
     ( length( di1_ini_cnt ) ~= jc0_rcv_len ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

if ( min( di1_rte_con ) < 0.0 )              % Must be non-negative         %
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%    Tests have shown that for the system of differential equations used in %
% the  4-parameter QCM,  that ode45, ode23, ode113, ode15s, ode23s, ode23t, %
% ode23tb return approximately the same results. Since ode45 is the fastest %
% ODE solver, it is used  instead of ode15s.  If an example is  found where %
% the differential equations  appear to be stiff, replace the call to ode45 %
% below with ode15s.  No other change is necessary.                          %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Set up ODE solver            %
     Sl0_opt =                         ...   % Set ODE solver options       %
odeset                                 ...
 (                                     ...
     'Abstol'        , dc0_abs_tol   , ...
     'Reltol'        , dc0_rel_tol     ...
 )                                           ;

 [                              ...           % Run ODE solver               %
     dl1_out_tim              , ...           % Output time vector           %
     dl2_out_cnt                ...           % Output count vector          %
 ] =                            ...
ode45                           ...
 (                              ...
     @( dl0_ode_tim ,           ...           % Solver time and count        %
        dl1_ode_cnt )           ...
     qm4_clc_dif                       ...   % Calculate differentials      %
          (                            ...
             dl0_ode_tim             , ...   % Evaluation time value        %
             dl1_ode_cnt             , ...   % Count vector (x)             %
             di1_rte_con             , ...   % Rate constants (k)           %
             jl0_dbg_flg               ...   % Debug flag                   %
          )                         , ...
     di1_tim_vec                    , ...   % Vector of evaluation times   %
     di1_ini_cnt                    , ...   % Vector of initial counts     %
     Sl0_opt                          ...   % ODE options                  %
 )                                           ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

do1_cnt_vec = dl2_out_cnt(:,1) + ...                 % Total count (M+M*)   %
              dl2_out_cnt(:,2)               ;

do1_cnt_vec = max( do1_cnt_vec , 1.0e-15 )   ;       % Min >= 10^-15        %

do2_cnt_mat = max( dl2_out_cnt , 1.0e-15 )   ;       % Trimmed count matrix %
```

```
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                               % Debug                       %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                   ...
        1                                      , ...
        'Time: \t%d\tLog(count): \t%d\n'       , ...
        dl1_out_tim(  1)                       , ...
        log10( do1_cnt_vec(  1) )              )       ;
      fprintf(                                   ...
        1                                      , ...
        'Time: \t%d\tLog(count): \t%d\n'       , ...
        dl1_out_tim(end)                       , ...
        log10( do1_cnt_vec(end) )              )       ;

      fprintf( 1 , '\n' )                                          ;
   end
end                                            % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_clc_dif                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           17-NOV-2015                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer                Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      11/17/15 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function calculates the  differentials for the 4-parameter Quasi- %
% Chemical Model (QCM-4) and is based on the function  FD  discussed in Refer- %
% ence [a].                                                                    %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di0_tim_val     double          This is the time value to evaluate the %
%                                       differential at. The value is not used %
%                                       in the differential calculations, but  %
%                                       is required to use the ODE solvers.    %
%                                                                              %
% (I)   di1_cnt_vec     double(4,1)     This is the input vector (y) required  %
%                                       by the ODE solvers. It is the bacterial%
%                                       counts for the four phases at the time %
%                                       di0_tim_val in CFU/mL (x(i)).          %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of rate constants   %
%                                       (k) for the QCM.                       %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_dif_vec     double          Vector of calculated differentials.    %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   References                                 %
%                                                                              %
% a. Ross, E.W., et al., The mathematical properties of the quasi-chemical     %
%    model for microorganism growth-death kinetics in foods, International     %
%    Journal of Food Microbiology, Volume 99, pps. 157-171, 2005.              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                      % Calculate ODE differentials  %
```

```
function                    ...
    [                       ...
        do1_dif_vec     , ...                   % Differential vector          %
        no0_src           ...                   % Status return code           %
    ] =                     ...
   qm4_clc_dif              ...
    (                       ...
        di0_tim_val     , ...                   % Evaluation time value        %
        di1_cnt_vec     , ...                   % Count vector (x(t))          %
        di1_rte_con     , ...                   % Rate constants (k)           %
        ji0_dbg_flg       ...                   % Debug flag                   %
    )
                                                % PARAMETER DECLARATIONS       %
   do1_dif_vec          = []            ;       % Initialize output            %
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % MATLAB INITIALIZATION        %
   format long                          ;       % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;       % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )       % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOCAL DECLARATIONS           %
   dl1_vel_vec  = double( 0.0 )         ;       % Velocity vector              %

   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag             %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS         %
   dc0_scl_fac          = 1.0e-9        ;       % Scaling factor               %
   jc0_vec_len          = 4             ;       % Vectors are length 4         %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
   dl1_vel_vec = []                     ;       % Initialize params to empty   %
                                                % Check inputs                 %
   if ( ~exist('ji0_dbg_flg','var') || ...      % No debug flag                %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )    % Debug must be non-negative   %
      no0_src = Sc0_qgd.E_INVARG        ;       % Set status return code       %
      return                            ;       % Exit function                %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
      jl0_dbg_flg = ji0_dbg_flg                 ;
      fprintf( 1 , '*** qm4_clc_dif ***\n' )    ;
   end

   if ( ( ~isvector( di1_cnt_vec ) ) || ...     % Must be vectors              %
        ( ~isvector( di1_rte_con ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct length       %
   if ( ( length( di1_cnt_vec ) ~= jc0_vec_len ) || ...
        ( length( di1_rte_con ) ~= jc0_vec_len ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Set up vectors               %
```

```
   dl1_vel_vec = zeros( jc0_vec_len , 1 )       ;
   do1_dif_vec = zeros( jc0_vec_len , 1 )       ;

                                                % Velocity vectors             %
   dl1_vel_vec(1) = di1_rte_con(1) * di1_cnt_vec(1)     ; % v1 = k1*x1         %
   dl1_vel_vec(2) = di1_rte_con(2) * di1_cnt_vec(2)     ; % v2 = k2*x2         %
   dl1_vel_vec(3) = dc0_scl_fac    * di1_rte_con(3) * ... % v3 = h *k3*x2*x3   %
                    di1_cnt_vec(2) * di1_cnt_vec(3)     ;
   dl1_vel_vec(4) = di1_rte_con(4) * di1_cnt_vec(2)     ; % v4 = k4*x2         %

                                                % Differential vectors         %
   do1_dif_vec(1) =                - dl1_vel_vec(1)     ; % dx1 = -v1          %
   do1_dif_vec(2) = dl1_vel_vec(1) + dl1_vel_vec(2) - ... % dx2 = v1+v2-v3-v4  %
                    dl1_vel_vec(3) - dl1_vel_vec(4)     ;
   do1_dif_vec(3) = dl1_vel_vec(2) - dl1_vel_vec(3)     ; % dx3 = v2-v3        %
   do1_dif_vec(4) = dl1_vel_vec(3) + dl1_vel_vec(4)     ; % dx4 = v3+v4        %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Debug                        %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                    ...
        1                                       , ...
        'Evaluation time:    %f\n'              , ...
        di0_tim_val                             )       ;
      fprintf(                                    ...
        1                                       , ...
        'Count vector:       %e\t%e\t%e\t%e\n'  , ...
        di1_cnt_vec(1)  , di1_cnt_vec(2)        , ...
        di1_cnt_vec(3)  , di1_cnt_vec(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Rate constants:     %f\t%f\t%f\t%f\n'  , ...
        di1_rte_con(1)  , di1_rte_con(2)        , ...
        di1_rte_con(3)  , di1_rte_con(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Velocities:         %e\t%e\t%e\t%e\n'  , ...
        dl1_vel_vec(1)  , dl1_vel_vec(2)        , ...
        dl1_vel_vec(3)  , dl1_vel_vec(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Differentials:      %e\t%e\t%e\t%e\n'  , ...
        do1_dif_vec(1)  , do1_dif_vec(2)        , ...
        do1_dif_vec(3)  , do1_dif_vec(4)        )       ;

      fprintf( 1 , '\n' )                                          ;
   end
end                                             % END OF FUNCTION              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function Bq4_slv_ode                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           21-DEC-2015                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/14/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function solves the system of ordinary differential equations for %
% the  4-parameter  Quasi-Chemical Model (QCM-4)  based on  Formulation B  de- %
% scribed in Reference [a].  It uses a  MATLAB ODE solver to solve for M* (x2) %
% and A (x3).  M (x1) is solved for directly and calculated  after the fact; D %
% (x4) is not solved for and returns values of 10^-15.                         %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of rate constants   %
%                                       (k) for the QCM. This must be the first%
%                                       parameter for the optimization to work.%
%                                                                              %
% (I)   di1_ini_cnt     double(4,1)     This is the initial bacterial counts   %
%                                       (t=0) for the four phases in CFU/mL    %
%                                       (M,M*,A,D or x1,x2,x3,x4).  Only M*(0) %
%                                       and A(0) are used directly in the solv-%
%                                       er.  M(0) is passed to Bq4_clc_dif as  %
%                                       a separate parameter.                  %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Evaluation time vector in days.  Bac-  %
%                                       terial counts will be estimated at the %
%                                       evaluation times based on the solution %
%                                       to the system of ODEs.                 %
%
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_cnt_vec     double(n,1)     Vector of estimated bacterial counts   %
%                                       in CFU/mL at each value of di1_tim_vec.%
%                                       (M+M* or x1+x2).                       %
%                                                                              %
% (O)   do2_cnt_mat     double(n,4)     Matrix of estimated bacterial counts   %
%                                       in CFU/mL at each value of di1_tim_vec %
%                                       and for the four state parameters. The %
%                                       values for x4=D are set to 10^-15.     %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
```

```
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   References                                 %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, NCSU Memorandum, 8 August 2016.                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % Solve ODE system Form B      %
function                    ...
    [                       ...
        do1_cnt_vec     , ...                           % Count vector from ODE (U)    %
        do2_cnt_mat     , ...                           % Count matrix from ODE (U)    %
        no0_src           ...                           % Status return code           %
    ] =                   ...
   Bq4_slv_ode            ...
    (                     ...
        di1_rte_con     , ...                           % Rate constants (k)           %
        di1_ini_cnt     , ...                           % Initial count vector (x0)    %
        di1_tim_vec     , ...                           % Evaluation time vector (td)  %
        ji0_dbg_flg       ...                           % Debug flag                   %
    )
                                                        % PARAMETER DECLARATIONS       %
   do1_cnt_vec          = []            ;
   do2_cnt_mat          = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;               % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )               % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOCAL DECLARATIONS           %
   dl2_out_cnt  = double( 0.0 )         ;               % Output count matrix          %
   dl2_tm1      = double( 0.0 )         ;               % Temporary matrix             %

   dl1_out_tim  = double( 0.0 )         ;               % Output time vector           %
   dl1_ode_cnt  = double( 0.0 )         ;               % ODE count vector             %
   dl1_ini_cnt  = double( 0.0 )         ;               % Init count vector (reduced)  %

   dl0_ode_tim  = double( 0.0 )         ;               % ODE time value               %
   dl0_ix1_cnt  = double( 0.0 )         ;               % Initial x1 count             %
   dl0_tx1_cnt  = double( 0.0 )         ;               % x1 count at time t           %
   dl0_ode_its  = double( 0.0 )         ;               % ODE initial step             %

   m            = uint32( 0 )           ;               % Counters/indexes             %

   jl0_num_tim  = int32( 0 )            ;               % Number of times              %
   jl0_dbg_flg  = int32( 0 )            ;               % Local debug flag             %

   Sl0_opt      = []                    ;               % ODE options structure        %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % CONSTANT DEFINITIONS         %
   dc0_abs_tol       = double( 1.0e-12 );               % Absolute tolerance           %
   dc0_rel_tol       = double( 1.0e-4  );               % Relative tolerance           %

   jc0_ini_tsf       = 5                ;               % Initial time step factor     %
```

```
   jc0_rcv_len       = 4                 ;       % Rate constant vector length  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                 %
   if ( ~exist('ji0_dbg_flg','var') || ...      % No debug flag                %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )    % Debug must be non-negative   %
      no0_src     = Sc0_qgd.E_INVARG    ;       % Set status return code       %
      return                            ;       % Exit function                %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
      jl0_dbg_flg = ji0_dbg_flg                 ;
      fprintf( 1 , '*** Bq4_slv_ode ***\n' )    ;
   end

   if ( ( ~isvector( di1_rte_con ) ) || ...     % Must be vectors              %
        ( ~isvector( di1_ini_cnt ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct length       %
   if ( ( length( di1_rte_con ) ~= jc0_rcv_len ) || ...
        ( length( di1_ini_cnt ) ~= jc0_rcv_len ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   if ( min( di1_rte_con ) < 0.0 )              % Must be non-negative         %
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   dl1_ini_cnt(1) = di1_ini_cnt(2)      ;       % Initial counts (x2 and x3)   %
   dl1_ini_cnt(2) = di1_ini_cnt(3)      ;
   dl0_ix1_cnt    = di1_ini_cnt(1)      ;       % Initial count (x1)           %
   jl0_dbg_flg    = Sc0_qgd.J_DBGLV0    ;       % Turn off debug               %

   dl0_ode_its    = log10( di1_tim_vec(end) )   ;       % Set init time step   %
   dl0_ode_its    = floor( dl0_ode_its )        ;       % Must be 1.0e-X       %
   dl0_ode_its    = dl0_ode_its - jc0_ini_tsf   ;
   dl0_ode_its    = 10^dl0_ode_its              ;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                              %
   %    Tests have shown that for the system of differential equations used in %
   % the  4-parameter QCM,  that ode45, ode23, ode113, ode15s, ode23s, ode23t, %
   % ode23tb return approximately the same results. Since ode45 is the fastest %
   % ODE solver, it is used  instead of ode15s.  If an example is  found where %
   % the differential equations  appear to be stiff, replace the call to ode45 %
   % below with ode15s.  No other change is necessary.                          %
   %                                                                              %
   %    When k1 is  near zero, the initial (and possibly entire) slope  of the %
   % solution is approximately zero.  By default, the ODE solvers choose their %
   % initial  step size in time based on the slope at the initial  time point. %
   % If the initial  slope is zero, this can lead  to problems with step sizes %
   % that are  too large.  To avoid this problem with  small k1, set the value %
   % of InitialStep in the ODE  options to a small value relative to the maxi- %
   % mum time.                                                                   %
   %                                                                              %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Set up ODE solver           %
        Sl0_opt =                          ...  % Set ODE solver options      %
   odeset                                  ...
    (                                      ...
        'Abstol'        , dc0_abs_tol   , ...
        'Reltol'        , dc0_rel_tol   , ...
        'InitialStep'   , dl0_ode_its     ...
```

```
    )                                        ;

    [                              ...         % Run ODE solver             %
        dl1_out_tim              , ...         % Output time vector         %
        dl2_out_cnt                ...         % Output count vector        %
    ] =                            ...
   ode45                           ...
    (                              ...
        @( dl0_ode_tim ,           ...         % Solver time and count      %
           dl1_ode_cnt )           ...
        Bq4_clc_dif                    ...     % Calculate differentials    %
             (                         ...
                dl0_ode_tim          , ...     % Evaluation time value      %
                dl1_ode_cnt          , ...     % Count array                %
                di1_rte_con          , ...     % Rate constants (k)         %
                dl0_ix1_cnt          , ...     % Initial count for x1       %
                jl0_dbg_flg            ...     % Debug flag                 %
             )                   , ...
        di1_tim_vec              , ...         % Vector of evaluation times %
        dl1_ini_cnt              , ...         % Vector of initial counts   %
        Sl0_opt                    ...         % ODE options                %
    )                                        ;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                               % Add in x1 count            %
   jl0_num_tim = length( dl1_out_tim )                 ;
   dl2_tm1     = dl2_out_cnt                           ;
   dl2_out_cnt = zeros( jl0_num_tim , jc0_rcv_len )    ;

   for ( m = 1:jl0_num_tim )
      dl0_tx1_cnt      = dl0_ix1_cnt          * ...
                         exp( -di1_rte_con(1) * ...
                               dl1_out_tim(m) )        ;
      dl2_out_cnt(m,1) = dl0_tx1_cnt                   ; % x1 calculated      %
      dl2_out_cnt(m,2) = dl2_tm1(m,1)                  ; % x2 from ODE solver %
      dl2_out_cnt(m,3) = dl2_tm1(m,2)                  ; % x3 from ODE solver %
      dl2_out_cnt(m,4) = 1.0e-15                       ; % x4 set to zero     %
   end                                         % End for ( m )              %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   do1_cnt_vec = dl2_out_cnt(:,1) + ...                % Total count (M+M*)   %
                 dl2_out_cnt(:,2)            ;

   do1_cnt_vec = max( do1_cnt_vec , 1.0e-15 )   ;      % Min >= 10^-15        %

   do2_cnt_mat = max( dl2_out_cnt , 1.0e-15 )   ;      % Trimmed count matrix %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                               % Debug                      %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                   ...
        1                                      , ...
        'Time: \t%d\tLog(count): \t%d\n'       , ...
        dl1_out_tim(  1)                       , ...
        log10( do1_cnt_vec(  1) )              )       ;
      fprintf(                                   ...
        1                                      , ...
        'Time: \t%d\tLog(count): \t%d\n'       , ...
        dl1_out_tim(end)                       , ...
        log10( do1_cnt_vec(end) )              )       ;

      fprintf( 1 , '\n' )                                          ;
   end
end                                            % END OF FUNCTION            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function Bq4_clc_dif                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           21-DEC-2015                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      12/21/15 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function calculates the  differentials for the 4-parameter Quasi- %
% Chemical Model (QCM-4)  using Formulation B discussed in Reference [a].  The %
% modifications solve for x1 directly, decouple and  remove x4, and only solve %
% for x2 and x3. The calculation of x1 uses the input time which is assumed to %
% be relative to a  start time of zero. If the initial time  is not zero, this %
% should be the offset from the initial time relative to the initial count.    %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di0_tim_val     double          This is the time value to evaluate the %
%                                       differential at. The value is only used%
%                                       to calculate x1 and is assumed to start%
%                                       at a value of zero.                    %
%                                                                              %
% (I)   di1_cnt_vec     double(2,1)     This is the input vector (y) required  %
%                                       by the ODE solvers. It is the bacterial%
%                                       counts for x2 and x3 phases at the time%
%                                       di0_tim_val in CFU/mL (x(i)).          %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of rate constants   %
%                                       (k) for the QCM.                       %
%                                                                              %
% (I)   di0_ix1_cnt     double          This is the initial count for x1.      %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_dif_vec     double(2,1)     Vector of calculated differentials.    %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   References                                 %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
```

```
%    Growth, NCSU Memorandum, 8 August 2016.                                %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                    % Calculate ODE differentials  %
function                    ...
    [                       ...
        do1_dif_vec     , ...                       % Differential vector          %
        no0_src           ...                       % Status return code           %
    ] =                     ...
   Bq4_clc_dif              ...
    (                       ...
        di0_tim_val     , ...                       % Evaluation time value        %
        di1_cnt_vec     , ...                       % Count vector (x(t))          %
        di1_rte_con     , ...                       % Rate constants (k)           %
        di0_ix1_cnt     , ...                       % Initial count for x1         %
        ji0_dbg_flg       ...                       % Debug flag                   %
    )
                                                    % PARAMETER DECLARATIONS       %
   do1_dif_vec          = []            ;           % Initialize output            %
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                    % MATLAB INITIALIZATION        %
   format long                          ;           % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                    % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;           % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )           % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                    % LOCAL DECLARATIONS           %
   dl1_vel_vec  = double( 0.0 )         ;           % Velocity vector              %

   dl0_tx1_cnt  = double( 0.0 )         ;           % x1 count at time t           %

   jl0_dbg_flg  = int32( 0 )            ;           % Local debug flag             %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                    % CONSTANT DEFINITIONS         %
   dc0_scl_fac          = 1.0e-9        ;           % Scaling factor               %
   jc0_vec_len          = 4             ;           % Rate/velocity vectors are 4  %
   jc0_cnt_len          = 2             ;           % Count vector is 2 (x2 and x3)%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                    % BEGINNING OF FUNCTION        %
   dl1_vel_vec = []                     ;           % Initialize params to empty   %
                                                    % Check inputs                 %
   if ( ~exist('ji0_dbg_flg','var') || ...          % No debug flag                %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )        % Debug must be non-negative   %
      no0_src = Sc0_qgd.E_INVARG        ;           % Set status return code       %
      return                            ;           % Exit function                %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
      jl0_dbg_flg = ji0_dbg_flg                     ;
      fprintf( 1 , '*** Bq4_clc_dif ***\n' )        ;
   end

   if ( ( ~isvector( di1_cnt_vec ) ) || ...         % Must be vectors              %
        ( ~isvector( di1_rte_con ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                    % Must be correct length       %
```

```
   if ( ( length( di1_cnt_vec ) ~= jc0_cnt_len ) || ...
        ( length( di1_rte_con ) ~= jc0_vec_len ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Set up vectors              %
   dl1_vel_vec = zeros( jc0_vec_len , 1 )       ;
   do1_dif_vec = zeros( jc0_cnt_len , 1 )       ;
                                                % x1 count at time t          %
   dl0_tx1_cnt    = di0_ix1_cnt    * ...               % x1=x1(0)^e(-k1*t)    %
                    exp( -di1_rte_con(1) * ...
                          di0_tim_val    )              ;
                                                % Velocity vectors            %
   dl1_vel_vec(1) = di1_rte_con(1) * dl0_tx1_cnt        ; % v1 = k1*x1        %
   dl1_vel_vec(2) = di1_rte_con(2) * di1_cnt_vec(1)     ; % v2 = k2*x2        %
   dl1_vel_vec(3) = dc0_scl_fac    * di1_rte_con(3) * ... % v3 = h *k3*x2*x3  %
                    di1_cnt_vec(1) * di1_cnt_vec(2)     ;
   dl1_vel_vec(4) = di1_rte_con(4) * di1_cnt_vec(1)     ; % v4 = k4*x2        %

                                                % Differential vectors        %
   do1_dif_vec(1) = dl1_vel_vec(1) + dl1_vel_vec(2) - ... % dx2 = v1+v2-v3-v4 %
                    dl1_vel_vec(3) - dl1_vel_vec(4)     ;
   do1_dif_vec(2) = dl1_vel_vec(2) - dl1_vel_vec(3)     ; % dx3 = v2-v3       %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Debug                       %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                    ...
        1                                       , ...
        'Evaluation time:   %f\n'               , ...
        di0_tim_val                             )       ;
      fprintf(                                    ...
        1                                       , ...
        'Count vector:      %e\t%e\t%e\n'       , ...
        dl0_tx1_cnt                             , ...
        di1_cnt_vec(1)  , di1_cnt_vec(2)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Rate constants:    %f\t%f\t%f\t%f\n'   , ...
        di1_rte_con(1)  , di1_rte_con(2)        , ...
        di1_rte_con(3)  , di1_rte_con(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Velocities:        %e\t%e\t%e\t%e\n'   , ...
        dl1_vel_vec(1)  , dl1_vel_vec(2)        , ...
        dl1_vel_vec(3)  , dl1_vel_vec(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Differentials:     %e\t%e\n'           , ...
        do1_dif_vec(1)  , do1_dif_vec(2)        )       ;

      fprintf( 1 , '\n' )                                           ;
   end
end                                             % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function Dq4_slv_ode                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           07-JUL-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/25/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This  function is derived  from  Bq4_slv_ode  but solves the system of %
% ordinary  differential equations for Formulations D or E of the  4-parameter %
% Quasi-Chemical Model.                                                        %
%                                                                              %
%       Formulations D and E are assumed to have the same format for input and %
% output when calling this function (i.e. rate constants are [z1 z2 z3 z4]).   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of the logarithm of %
%                                       the rate constants (z) for the QCM.    %
%                                       This must be the first parameter for   %
%                                       the optimization to work.              %
%                                                                              %
% (I)   di1_ini_cnt     double(4,1)     This is the initial bacterial counts   %
%                                       (t=0) for the four phases in CFU/mL    %
%                                       (M,M*,A,D or x1,x2,x3,x4).  Only M*(0) %
%                                       and A(0) are used directly in the solv-%
%                                       er.  M(0) is passed to Dq4_clc_dif as  %
%                                       a separate parameter.                  %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Evaluation time vector in days.  Bac-  %
%                                       terial counts will be estimated at the %
%                                       evaluation times based on the solution %
%                                       to the system of ODEs.                 %
%
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_cnt_vec     double(n,1)     Vector of estimated bacterial counts   %
%                                       in CFU/mL at each value of di1_tim_vec.%
%                                       (M+M* or x1+x2).                       %
%                                                                              %
% (O)   do2_cnt_mat     double(n,4)     Matrix of estimated bacterial counts   %
%                                       in CFU/mL at each value of di1_tim_vec %
%                                       and for the four state parameters. The %
%                                       values for x4=D are set to 10^-15.     %
%                                                                              %
```

```
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % Solve ODE system (Form D/E)  %
function                  ...
    [                     ...
        do1_cnt_vec     , ...                           % Count vector from ODE (U)    %
        do2_cnt_mat     , ...                           % Count matrix from ODE (U)    %
        no0_src           ...                           % Status return code           %
    ] =                   ...
   Dq4_slv_ode            ...
    (                     ...
        di1_rte_con     , ...                           % Log of rate constants (z)    %
        di1_ini_cnt     , ...                           % Initial count vector (x0)    %
        di1_tim_vec     , ...                           % Evaluation time vector (td)  %
        ji0_dbg_flg       ...                           % Debug flag                   %
    )
                                                        % PARAMETER DECLARATIONS       %
   do1_cnt_vec          = []            ;
   do2_cnt_mat          = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;               % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )               % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOCAL DECLARATIONS           %
   dl2_out_cnt  = double( 0.0 )         ;               % Output count matrix          %
   dl2_tm1      = double( 0.0 )         ;               % Temporary matrix             %

   dl1_out_tim  = double( 0.0 )         ;               % Output time vector           %
   dl1_ode_cnt  = double( 0.0 )         ;               % ODE count vector             %
   dl1_ini_cnt  = double( 0.0 )         ;               % Init count vector (reduced)  %
   dl1_rte_con  = double( 0.0 )         ;               % Converted rate constants (k) %

   dl0_ode_tim  = double( 0.0 )         ;               % ODE time value               %
   dl0_ix1_cnt  = double( 0.0 )         ;               % Initial x1 count             %
   dl0_tx1_cnt  = double( 0.0 )         ;               % x1 count at time t           %
   dl0_ode_its  = double( 0.0 )         ;               % ODE initial step             %

   m            = uint32( 0 )           ;               % Counters/indexes             %

   jl0_num_tim  = int32( 0 )            ;               % Number of times              %
   jl0_dbg_flg  = int32( 0 )            ;               % Local debug flag             %

   Sl0_opt      = []                    ;               % ODE options structure        %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % CONSTANT DEFINITIONS         %
   dc0_abs_tol       = double( 1.0e-12 );               % Absolute tolerance           %
   dc0_rel_tol       = double( 1.0e-4  );               % Relative tolerance           %

   jc0_ini_tsf       = 5                ;               % Initial time step factor     %
   jc0_rcv_len       = 4                ;               % Rate constant vector length  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % BEGINNING OF FUNCTION        %
```

```
                                                % Check inputs                 %
if ( ~exist('ji0_dbg_flg','var') || ...         % No debug flag                %
     isempty(ji0_dbg_flg)        )
   jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )       % Debug must be non-negative   %
   no0_src     = Sc0_qgd.E_INVARG    ;          % Set status return code       %
   return                            ;          % Exit function                %
elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
   jl0_dbg_flg = ji0_dbg_flg                    ;
   fprintf( 1 , '*** Dq4_slv_ode ***\n' )       ;
end

if ( ( ~isvector( di1_rte_con ) ) || ...        % Must be vectors              %
     ( ~isvector( di1_ini_cnt ) ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                                % Must be correct length       %
if ( ( length( di1_rte_con ) ~= jc0_rcv_len ) || ...
     ( length( di1_ini_cnt ) ~= jc0_rcv_len ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

dl1_ini_cnt(1) = di1_ini_cnt(2)      ;          % Initial counts (x2 and x3)   %
dl1_ini_cnt(2) = di1_ini_cnt(3)      ;
dl0_ix1_cnt    = di1_ini_cnt(1)      ;          % Initial count (x1)           %
jl0_dbg_flg    = Sc0_qgd.J_DBGLV0    ;          % Turn off debug               %

dl0_ode_its    = log10( di1_tim_vec(end) )      ;       % Set init time step   %
dl0_ode_its    = floor( dl0_ode_its )           ;       % Must be 1.0e-X       %
dl0_ode_its    = dl0_ode_its - jc0_ini_tsf      ;
dl0_ode_its    = 10^dl0_ode_its                 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%    Tests have shown that for the system of differential equations used in %
% the  4-parameter QCM,  that ode45, ode23, ode113, ode15s, ode23s, ode23t, %
% ode23tb return approximately the same results. Since ode45 is the fastest %
% ODE solver, it is used  instead of ode15s.  If an example is  found where %
% the differential equations  appear to be stiff, replace the call to ode45 %
% below with ode15s.  No other change is necessary.                          %
%                                                                              %
%    The forward uncertainty propagation for  formulation D/E is stiff, and %
% ode15s is chosen for solving the ODEs.                                      %
%                                                                              %
%    When k1 is  near zero, the initial (and possibly entire) slope  of the %
% solution is approximately zero.  By default, the ODE solvers choose their %
% initial  step size in time based on the slope at the initial  time point. %
% If the initial  slope is zero, this can lead  to problems with step sizes %
% that are  too large.  To avoid this problem with  small k1, set the value %
% of InitialStep in the ODE  options to a small value relative to the maxi- %
% mum time.                                                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Set up ODE solver            %
     Sl0_opt =                            ...   % Set ODE solver options      %
odeset                                    ...
 (                                        ...
     'Abstol'        , dc0_abs_tol   , ...
     'Reltol'        , dc0_rel_tol   , ...
     'InitialStep'   , dl0_ode_its     ...
 )                                              ;

 [                                ...           % Run ODE solver              %
     dl1_out_tim                , ...           % Output time vector          %
     dl2_out_cnt                  ...           % Output count vector         %
 ] =                              ...
```

```
   ode15s                          ...
    (                              ...
       @( dl0_ode_tim ,            ...           % Solver time and count       %
          dl1_ode_cnt )            ...
       Dq4_clc_dif                         ...   % Calculate differentials     %
            (                              ...
               dl0_ode_tim               , ...   % Evaluation time value       %
               dl1_ode_cnt               , ...   % Count array                 %
               di1_rte_con               , ...   % Log of rate constants (z)   %
               dl0_ix1_cnt               , ...   % Initial count for x1        %
               jl0_dbg_flg                 ...   % Debug flag                  %
            )                      , ...
       di1_tim_vec                 , ...         % Vector of evaluation times  %
       dl1_ini_cnt                 , ...         % Vector of initial counts    %
       Sl0_opt                       ...         % ODE options                 %
    )                                            ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                 % Add in x1 count             %
   jl0_num_tim = length( dl1_out_tim )                   ;
   dl2_tm1     = dl2_out_cnt                             ;
   dl2_out_cnt = zeros( jl0_num_tim , jc0_rcv_len )      ;

   dl1_rte_con = exp( di1_rte_con )                      ; % Convert back      %

   for ( m = 1:jl0_num_tim )
      dl0_tx1_cnt      = dl0_ix1_cnt         * ...
                         exp( -dl1_rte_con(1) * ...
                               dl1_out_tim(m) )          ;
      dl2_out_cnt(m,1) = dl0_tx1_cnt                     ; % x1 calculated     %
      dl2_out_cnt(m,2) = dl2_tm1(m,1)                    ; % x2 from ODE solver %
      dl2_out_cnt(m,3) = dl2_tm1(m,2)                    ; % x3 from ODE solver %
      dl2_out_cnt(m,4) = 1.0e-15                         ; % x4 set to zero    %
   end                                           % End for ( m )               %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   do1_cnt_vec = dl2_out_cnt(:,1) + ...                  % Total count (M+M*)  %
                 dl2_out_cnt(:,2)                ;

   do1_cnt_vec = max( do1_cnt_vec , 1.0e-15 )    ;       % Min >= 10^-15       %

   do2_cnt_mat = max( dl2_out_cnt , 1.0e-15 )    ;       % Trimmed count matrix %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                 % Debug                       %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                     ...
        1                                        , ...
        'Time: \t%d\tLog(count): \t%f\n'         , ...
        dl1_out_tim(  1)                         , ...
        log10( do1_cnt_vec(  1) )                )       ;
      fprintf(                                     ...
        1                                        , ...
        'Time: \t%d\tLog(count): \t%f\n'         , ...
        dl1_out_tim(end)                         , ...
        log10( do1_cnt_vec(end) )                )       ;

      fprintf( 1 , '\n' )                                            ;
   end
end                                              % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function Dq4_clc_dif                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           07-JUL-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/07/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function is a modification of  Bq4_clc_dif  to calculate the dif- %
% ferentials for formulation D of the 4-parameter Quasi-Chemical Model.        %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di0_tim_val     double          This is the time value to evaluate the %
%                                       differential at. The value is only used%
%                                       to calculate x1 and is assumed to start%
%                                       at a value of zero.                    %
%                                                                              %
% (I)   di1_cnt_vec     double(2,1)     This is the input vector (y) required  %
%                                       by the ODE solvers. It is the bacterial%
%                                       counts for x2 and x3 phases at the time%
%                                       di0_tim_val in CFU/mL (x(i)).          %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of log of the rate  %
%                                       constants (z) for QCM-4 formulation D. %
%                                                                              %
% (I)   di0_ix1_cnt     double          This is the initial count for x1.      %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_dif_vec     double(2,1)     Vector of calculated differentials.    %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                   % Calculate ODE differentials  %
function                  ...
    [                     ...
        do1_dif_vec     , ...                      % Differential vector          %
        no0_src           ...                      % Status return code           %
    ] =                   ...
   Dq4_clc_dif            ...
    (                     ...
```

```
        di0_tim_val     , ...                   % Evaluation time value        %
        di1_cnt_vec     , ...                   % Count vector (x(t))          %
        di1_rte_con     , ...                   % Log of rate constants (z)    %
        di0_ix1_cnt     , ...                   % Initial count for x1         %
        ji0_dbg_flg       ...                   % Debug flag                   %
    )
                                                % PARAMETER DECLARATIONS       %
   do1_dif_vec          = []            ;       % Initialize output            %
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % MATLAB INITIALIZATION        %
   format long                          ;       % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;       % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )       % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOCAL DECLARATIONS           %
   dl1_vel_vec  = double( 0.0 )         ;       % Velocity vector              %
   dl1_rte_con  = double( 0.0 )         ;       % Converted rate constants     %

   dl0_tx1_cnt  = double( 0.0 )         ;       % x1 count at time t           %

   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag             %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS         %
   dc0_scl_fac          = 1.0e-9        ;       % Scaling factor               %
   jc0_vec_len          = 4             ;       % Rate/velocity vectors are 4  %
   jc0_cnt_len          = 2             ;       % Count vector is 2 (x2 and x3)%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
   dl1_vel_vec = []                     ;       % Initialize params to empty   %
                                                % Check inputs                 %
   if ( ~exist('ji0_dbg_flg','var') || ...      % No debug flag                %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )    % Debug must be non-negative   %
      no0_src = Sc0_qgd.E_INVARG        ;       % Set status return code       %
      return                            ;       % Exit function                %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
      jl0_dbg_flg = ji0_dbg_flg                 ;
      fprintf( 1 , '*** Dq4_clc_dif ***\n' )    ;
   end

   if ( ( ~isvector( di1_cnt_vec ) ) || ...     % Must be vectors              %
        ( ~isvector( di1_rte_con ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct length       %
   if ( ( length( di1_cnt_vec ) ~= jc0_cnt_len ) || ...
        ( length( di1_rte_con ) ~= jc0_vec_len ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Set up vectors               %
   dl1_vel_vec = zeros( jc0_vec_len , 1 )       ;
   do1_dif_vec = zeros( jc0_cnt_len , 1 )       ;
```

```
   dl1_rte_con = exp( di1_rte_con )                ;
                                                   % x1 count at time t           %
   dl0_tx1_cnt    = di0_ix1_cnt    * ...                 % x1=x1(0)^e(-k1*t)    %
                    exp( -dl1_rte_con(1) * ...
                          di0_tim_val    )                ;
                                                   % Velocity vectors             %
   dl1_vel_vec(1) = dl1_rte_con(1) * dl0_tx1_cnt         ; % v1 = k1*x1         %
   dl1_vel_vec(2) = dl1_rte_con(2) * di1_cnt_vec(1)      ; % v2 = k2*x2         %
   dl1_vel_vec(3) = dc0_scl_fac    * dl1_rte_con(3) * ... % v3 = h *k3*x2*x3   %
                    di1_cnt_vec(1) * di1_cnt_vec(2)      ;
   dl1_vel_vec(4) = dl1_rte_con(4) * di1_cnt_vec(1)      ; % v4 = k4*x2         %

                                                   % Differential vectors         %
   do1_dif_vec(1) = dl1_vel_vec(1) + dl1_vel_vec(2) - ... % dx2 = v1+v2-v3-v4  %
                    dl1_vel_vec(3) - dl1_vel_vec(4)      ;
   do1_dif_vec(2) = dl1_vel_vec(2) - dl1_vel_vec(3)      ; % dx3 = v2-v3        %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                   % Debug                        %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                       ...
        1                                          , ...
        'Evaluation time:   %f\n'                  , ...
        di0_tim_val                                )       ;
      fprintf(                                       ...
        1                                          , ...
        'Count vector:      %e\t%e\t%e\n'          , ...
        dl0_tx1_cnt                                , ...
        di1_cnt_vec(1)  , di1_cnt_vec(2)           )       ;
      fprintf(                                       ...
        1                                          , ...
        'Log of RCVs:       %e\t%e\t%e\t%e\n'      , ...
        di1_rte_con(1)  , di1_rte_con(2)           , ...
        di1_rte_con(3)  , di1_rte_con(4)           )       ;
      fprintf(                                       ...
        1                                          , ...
        'Velocities:        %e\t%e\t%e\t%e\n'      , ...
        dl1_vel_vec(1)  , dl1_vel_vec(2)           , ...
        dl1_vel_vec(3)  , dl1_vel_vec(4)           )       ;
      fprintf(                                       ...
        1                                          , ...
        'Differentials:     %e\t%e\n'              , ...
        do1_dif_vec(1)  , do1_dif_vec(2)           )       ;

      fprintf( 1 , '\n' )                                                ;
   end
end                                                % END OF FUNCTION              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_evl_nam                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           06-APR-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer                Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      04/06/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This  function evaluates the alternate models to the QCM-4. These mod- %
% els are described in Reference [a].                                          %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   ni0_mod_num     uint32          Model number defined in qcm_ini_prm.   %
%                                       Only models .J_NAMxxx are supported.   %
%                                                                              %
% (I)   di1_sol_vec     double(4,1)     Vector of solution parameters.  If     %
%                                       less than 4, fill with NaN. The format %
%                                       of the vector depends on the model     %
%                                       number.                                %
%                                       = .J_NAM1EX     [ a NaN NaN NaN ]      %
%                                       = .J_NAM1EL     [ a k1  NaN NaN ]      %
%                                       = .J_NAM2EX     [ a b   T   NaN ]      %
%                                       = .J_NAM2EL     [ a b   T   k1  ]      %
%                                                                              %
% (I)   di1_tim_vec     double(1,n)     Vector of times in days.               %
%                                                                              %
% (I)   di0_ini_cnt     double          Initial count for M in CFU/mL.         %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_out_cnt     double(n,1)     Vector of estimated bacterial counts   %
%                                       in CFU/mL at each value of di1_tim_vec.%
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   References                                 %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, Memorandum from NCSU, 8 August 2016.                              %
%                                                                              %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % Evaluate alternate models    %
function                 ...
    [                    ...
       do1_out_cnt     , ...                  % Output count vector (CFU/mL) %
       no0_src           ...                  % Status return code           %
    ] =                  ...
   qm4_evl_nam           ...
    (                    ...
       ni0_mod_num     , ...                  % Model number                 %
       di1_sol_vec     , ...                  % Solution vector              %
       di1_tim_vec     , ...                  % Time vector (day)            %
       di0_ini_cnt     , ...                  % Initial count (CFU/mL)       %
       ji0_dbg_flg       ...                  % Debug flag                   %
    )
                                              % PARAMETER DECLARATIONS       %
   do1_out_tim         = []           ;
   do1_out_cnt         = []           ;
   no0_src             = uint32( 0 )  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % MATLAB INITIALIZATION        %
   format long                        ;       % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % LOAD GLOBALS                 %
   persistent Sc0_qgd                 ;       % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )     % Not initialized              %
      Sc0_qgd = qcm_ini_prm()         ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % LOCAL DECLARATIONS           %
   dl1_tim_vec  = double( 0.0 )       ;       % Time vector (day)            %
   dl1_cnt_vec  = double( 0.0 )       ;       % Data count vector (CFU/mL)   %
   dl1_out_cnt  = double( 0.0 )       ;       % Model count vector (CFU/mL)  %
   dl1_twt      = double( 0.0 )       ;       % Time with tau vector         %
   dl1_tm1      = double( 0.0 )       ;       % Temporary vectors            %
   dl1_tm2      = double( 0.0 )       ;
   dl1_tm3      = double( 0.0 )       ;
   dl1_tm4      = double( 0.0 )       ;
   dl1_tm5      = double( 0.0 )       ;
   dl1_tm6      = double( 0.0 )       ;

   dl0_cat      = double( 0.0 )       ;       % M* count at time tau         %
   dl0_tm1      = double( 0.0 )       ;       % Temporary variables          %
   dl0_tm2      = double( 0.0 )       ;
   dl0_tm3      = double( 0.0 )       ;
   dl0_tm4      = double( 0.0 )       ;
   dl0_tm5      = double( 0.0 )       ;
   dl0_tm6      = double( 0.0 )       ;

   q            = uint32( 0 )         ;       % Counters/indexes             %

   jl0_dbg_flg  = int32( 0 )          ;       % Local debug flag             %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % CONSTANT DEFINITIONS         %
   jc0_rcv_len       = 4              ;       % Rate constant vector length  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % BEGINNING OF FUNCTION        %
                                              % Check inputs                 %
   if ( ~exist('ji0_dbg_flg','var') || ...    % No debug flag                %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0  ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )  % Debug must be non-negative   %
```

```
   no0_src = Sc0_qgd.E_INVARG        ;       % Set status return code       %
   return                            ;       % Exit function                %
elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
   jl0_dbg_flg = ji0_dbg_flg                 ;
   fprintf( 1 , '*** qm4_evl_nam ***\n' )    ;
end

if ( ( ~isvector( di1_sol_vec ) ) || ...     % Must be vector               %
     ( ~isvector( di1_tim_vec ) ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                             % Must be correct length       %
if ( length( di1_sol_vec ) ~= jc0_rcv_len )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                             % Supported model              %
if ( ( ni0_mod_num ~= Sc0_qgd.J_NAM1EX ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_NAM1EL ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_NAM2EX ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_NAM2EL ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_NAMHQ4 ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

dl1_tim_vec = di1_tim_vec            ;       % Set time, count, and size    %
jl0_num_pts = length( dl1_tim_vec )  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Get model counts             %
if ( ni0_mod_num == Sc0_qgd.J_NAM1EX )
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                         %
   %                      1-stage Exponential Model                          %
   %                                                                         %
   %         a = di1_sol_vec(1)                                              %
   %         I = di0_ini_cnt                                                 %
   %         t = di1_tim_vec                                                 %
   %         x = do1_out_cnt                                                 %
   %                                                                         %
   %                        x = I * exp( a * t )                             %
   %                                                                         %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   dl1_out_cnt = exp( di1_sol_vec(1) * dl1_tim_vec ) ;
   dl1_out_cnt = dl1_out_cnt * di0_ini_cnt           ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EL )
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                         %
   %                1-stage Exponential Model With Lag                       %
   %                                                                         %
   %         a  = di1_sol_vec(1)                                             %
   %         k1 = di1_sol_vec(2)                                             %
   %         I  = di0_ini_cnt                                                %
   %         t  = di1_tim_vec                                                %
   %         x  = do1_out_cnt                                                %
   %                                                                         %
   %                  f  = ( k1 * I ) / ( a + k1 )                           %
   %                  e1 = exp( -k1 * t )                                    %
   %                  e2 = exp(   a * t )                                    %
   %                  x  = ( I * e1 ) + ( f * ( e2 - e1 ) )                  %
   %                                                                         %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   if ( -di1_sol_vec(1) == di1_sol_vec(2) )  % Check values                 %
      no0_src = Sc0_qgd.E_INCARG     ;
```

```
      return                          ;
   end

   dl0_tm1     = ( di1_sol_vec(2) * di0_ini_cnt    ) / ...     % f          %
                 ( di1_sol_vec(1) + di1_sol_vec(2) )         ;

   dl1_tm1     = exp( -di1_sol_vec(2) * dl1_tim_vec )        ; % e1         %
   dl1_tm2     = exp(  di1_sol_vec(1) * dl1_tim_vec )        ; % e2         %

   dl1_out_cnt = dl1_tm2     - dl1_tm1                       ; % e2 - e1    %
   dl1_out_cnt = dl1_out_cnt * dl0_tm1                       ; % f*(e2-e1)  %
   dl1_out_cnt = dl1_out_cnt + ( dl1_tm1 * di0_ini_cnt )     ; % x          %
elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EX )
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                          %
   %                      2-stage Exponential Model                           %
   %                                                                          %
   %         a  = di1_sol_vec(1)                                              %
   %         b  = di1_sol_vec(2)                                              %
   %         T  = di1_sol_vec(3)                                              %
   %         I  = di0_ini_cnt                                                 %
   %         t  = di1_tim_vec                                                 %
   %         x  = do1_out_cnt                                                 %
   %                                                                          %
   %                 if ( t < T )                                             %
   %                   x1 = I * exp( a * t )                                  %
   %                 else                                                     %
   %                   e1 = -b * ( t - T )                                    %
   %                   e2 =  a * T                                            %
   %                   x2 = I * exp( e1 + e2 )                                %
   %                                                                          %
   %                 x = [ x1 x2 ]                                            %
   %                                                                          %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   q = find( dl1_tim_vec >= di1_sol_vec(3) , 1 , 'first' )   ;

   if ( isempty( dl1_tim_vec(1:q-1) ) )      % Count for t < T              %
      dl1_tm1 = []                                        ;
   else
      dl1_tm1 = di1_sol_vec(1) * dl1_tim_vec(1:q-1)  ;       % a * t        %
      dl1_tm1 = di0_ini_cnt    * exp( dl1_tm1 )      ;       % x1           %
   end

   if ( isempty( dl1_tim_vec(q:end) ) )      % Count for t >= T             %
      dl1_tm2 = []                                        ;
   else
      dl0_tm1 =  di1_sol_vec(1)     * di1_sol_vec(3) ;       % e2           %
      dl1_tm2 =  dl1_tim_vec(q:end) - di1_sol_vec(3) ;
      dl1_tm2 = -di1_sol_vec(2)     * dl1_tm2        ;       % e1           %
      dl1_tm2 =  dl1_tm2            + dl0_tm1        ;       % e1 + e2      %
      dl1_tm2 =  di0_ini_cnt        * exp( dl1_tm2 ) ;       % x2           %
   end

   dl1_out_cnt = [ dl1_tm1 dl1_tm2 ]                  ;       % x            %
elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EL )
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                          %
   %                 2-stage Exponential Model With Lag                       %
   %                                                                          %
   %         a  = di1_sol_vec(1)                                              %
   %         b  = di1_sol_vec(2)                                              %
   %         T  = di1_sol_vec(3)                                              %
   %         k1 = di1_sol_vec(4)                                              %
   %         I  = di0_ini_cnt                                                 %
   %         t  = di1_tim_vec                                                 %
   %         x  = do1_out_cnt                                                 %
   %                                                                          %
```

```
%                  f1 = ( k1 * I ) / ( a + k1 )                            %
%                  f2 = ( k1 * I ) / ( b - k1 )                            %
%                  e1 = exp( -k1 * t )                                     %
%                  e2 = exp(   a * t )                                     %
%                  e3 = exp( b * ( T - t ) )                               %
%                                                                          %
%                  if ( t < T )                                            %
%                    x1 = ( I * e1 ) + ( f1 * ( e2 - e1 ) )                %
%                  else                                                    %
%                    y  = f1 * ( e2(T) - e1(T) )                           %
%                       = x1(T) - ( I * e1(T) )                            %
%                    c1 = I + f2                                           %
%                    c2 = y - ( f2 * e1(T) )                               %
%                    x2 = ( c1 * e1 ) + ( c2 * e3 )                        %
%                                                                          %
%                  x = [ x1 x2 ]                                           %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

q = find( dl1_tim_vec >= di1_sol_vec(3) , 1 , 'first' )   ;

if ( isempty( dl1_tim_vec(1:q-1) ) )      % Count for t < T               %
   no0_src = Sc0_qgd.E_INCARG     ;       % Must have at least 1 t < T    %
   return                         ;
elseif ( -di1_sol_vec(1) == di1_sol_vec(4) )
   no0_src = Sc0_qgd.E_INCARG     ;
   return                         ;
else
   dl1_twt = [ dl1_tim_vec(1:q-1) di1_sol_vec(3) ];       % Add in tau    %

   dl0_tm1 = ( di1_sol_vec(4) * di0_ini_cnt    ) / ...    % f1            %
             ( di1_sol_vec(1) + di1_sol_vec(4) )  ;

   dl1_tm3 =  dl1_twt * di1_sol_vec(1)            ;
   dl1_tm4 = -dl1_twt * di1_sol_vec(4)            ;
   dl1_tm3 = exp( dl1_tm3 )                       ;       % e2            %
   dl1_tm4 = exp( dl1_tm4 )                       ;       % e1            %
   dl1_tm5 = dl1_tm4 * di0_ini_cnt                ;       % I * e1        %

   dl1_tm1 = dl1_tm3 - dl1_tm4                    ;       % e2 - e1       %
   dl1_tm1 = dl0_tm1 * dl1_tm1                    ;       % f1*(e2 - e1) %
   dl0_cat = dl1_tm1(end)                         ;       % Get tau count%
   dl1_tm1 = dl1_tm1 + dl1_tm5                    ;       % x1 + tau cnt %
   dl0_cat = dl1_tm1(end)                         ;       % Get tau count%
   dl1_tm1 = dl1_tm1(1:end-1)                     ;       % Del tau count%
end

if ( isempty( dl1_tim_vec(q:end) ) )      % Count for t >= T              %
   dl1_tm2 = []                                   ;
elseif ( di1_sol_vec(2) == di1_sol_vec(4) )
   no0_src = Sc0_qgd.E_INCARG     ;
   return                         ;
else
   dl0_tm2 = ( di1_sol_vec(4) * di0_ini_cnt    ) / ...    % f2            %
             ( di1_sol_vec(2) - di1_sol_vec(4) )  ;

   dl0_tm3 = -di1_sol_vec(4) * di1_sol_vec(3)     ;       % -k1 * T       %
   dl0_tm4 =  exp( dl0_tm3 )                      ;       % e1(T)         %
   dl0_tm3 =  di0_ini_cnt    * dl0_tm4            ;       % I * e1(T)     %
   dl0_tm3 =  dl0_cat        - dl0_tm3            ;       % y             %
   dl0_tm4 =  dl0_tm2        * dl0_tm4            ;       % f2 * e1(T)    %

   dl0_tm5 =  di0_ini_cnt    + dl0_tm2            ;       % c1            %
   dl0_tm6 =  dl0_tm3        - dl0_tm4            ;       % c2            %

   dl1_tm4 = -di1_sol_vec(4) * dl1_tim_vec(q:end) ;       % -k1 * t       %
   dl1_tm4 =  exp( dl1_tm4 )                      ;       % e1            %
   dl1_tm5 =  di1_sol_vec(3) - dl1_tim_vec(q:end) ;       % T - t         %
```

```
      dl1_tm5 =  di1_sol_vec(2) * dl1_tm5            ;       % b * (T - t)  %
      dl1_tm5 =  exp( dl1_tm5 )                      ;       % e3           %

      dl1_tm2 = ( dl0_tm5 * dl1_tm4 ) + ...                  % x2=(c1*e1) + %
                ( dl0_tm6 * dl1_tm5 )                ;       %    (c2*e3)   %
   end

   dl1_out_cnt = [ dl1_tm1 dl1_tm2 ] ;                       % x            %
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Set output values            %
if ( size(dl1_out_cnt,1) == 1 )                 % If row vector                %
   do1_out_cnt = dl1_out_cnt'        ;
else                                            % If column vector             %
   do1_out_cnt = dl1_out_cnt         ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Debug                        %
if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
   fprintf(                                       ...
     1                                          , ...
     'Time: \t%d\tLog(count): \t%d\n'           , ...
     dl1_tim_vec(  1)                           , ...
     log10( do1_out_cnt(  1) )                  )       ;
   fprintf(                                       ...
     1                                          , ...
     'Time: \t%d\tLog(count): \t%d\n'           , ...
     dl1_tim_vec(end)                           , ...
     log10( do1_out_cnt(end) )                  )       ;

   fprintf( 1 , '\n' )                                          ;
end
end                                             % END OF FUNCTION              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_slv_sBC                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           23-FEB-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/25/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function solves the system of ordinary differential equations for %
% estimating the sensitivity of a QCM-4 solution  to the rate parameters using %
% either formulation B (x1,x2), formulation C (x1,x2, and alpha), formulation  %
% D  (ln(k)), or formulation E  (ln(k1),ln(k2),ln(k3),ln(alpha)).  In  the re- %
% duced  formulations,  x1 is M* and x2 is A.  The function uses a MATLAB  ODE %
% solver to  solve the sensitivity  function described in  Reference [a].  The %
% sensitivity  ODEs are formulated as a matrix which must be linearized to get %
% a  solution which  is described in the  function  qm4_clc_sdB,  qm4_clc_sdC, %
% qm4_clc_sdD, or qm4_clc_sdE.  This function also  calculates the sensitivity %
% of the logarithm of the total bacterial count (g = log10(U)) relative to the %
% rate parameters which is the sensitivity matrix F(theta) in Reference [a].   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of solution rate    %
%                                       constants for the QCM.  It is either   %
%                                       [k1 k2 k3 k4], [k1 k2 k3 alpha], or    %
%                                       [z1 z2 z3 z4] depending on the value   %
%                                       of ni0_mod_num.                        %
%                                                                              %
% (I)   di0_ini_cnt     double          This is the initial bacterial count    %
%                                       of the lag cells (M) in CFU/mL.        %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Evaluation time vector in days.  The   %
%                                       sensitivity function s will be solved  %
%                                       for at the times in di1_tim_vec.       %
%                                                                              %
% (I)   di2_cnt_mat     double(2,n)     Solution bacterial counts for the x1   %
%                                       (activated cells) and x2 (antagonistic)%
%                                       variables in formulation B, C, or D at %
%                                       the times in di1_tim_vec in CFU/mL.    %
%                                                                              %
% (I)   ni0_mod_num     uint32          Model formulation number.  Only values %
%                                       of .J_QCMB4P, .J_QCMC4P, and .J_QCMD4P %
%                                       can be used with this function.        %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
```

```
%                                                                       %
% (O)   do2_sen_fun     double(8,n)     Sensitivity function s of state at     %
%                                       each time in di1_tim_vec. The s values %
%                                       are stored in linearized form as de-   %
%                                       scribed in qm4_clc_sdB, qm4_clc_sdC,   %
%                                       or qm4_clc_sdD.                        %
%                                                                              %
% (O)   do2_cnt_sen     double(4,n)     Sensitivity matrix of count function   %
%                                       U at each time in di1_tim_vec for      %
%                                       theta defined by the formulation.      %
%                                                                              %
% (O)   do2_sen_mat     double(4,n)     Sensitivity matrix of the log10 count  %
%                                       at each time in di1_tim_vec relative   %
%                                       to the rate constants (F(theta)).      %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                  References                                  %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, NCSU Memorandum, 8 August 2016.                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % Solve sensitivities B-E      %
function                  ...
    [                     ...
        do2_sen_fun     , ...                           % Sensitivity function (s)     %
        do2_cnt_sen     , ...                           % Count sensitivity matrix     %
        do2_sen_mat     , ...                           % Sensitivity matrix (F)       %
        no0_src           ...                           % Status return code           %
    ] =                   ...
   qm4_slv_sBC            ...
    (                     ...
        di1_rte_con     , ...                           % Rate constants (k)           %
        di0_ini_cnt     , ...                           % Initial count of M           %
        di1_tim_vec     , ...                           % Evaluation time vector (td)  %
        di2_cnt_mat     , ...                           % Soln count matrix (x1,x2)    %
        ni0_mod_num     , ...                           % Model formulation number     %
        ji0_dbg_flg       ...                           % Debug flag                   %
    )
                                                        % PARAMETER DECLARATIONS       %
   do2_sen_fun          = []            ;
   do2_cnt_sen          = []            ;
   do2_sen_mat          = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOAD GLOBALS                %
   persistent Sc0_qgd                   ;               % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )               % Not initialized             %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOCAL DECLARATIONS           %
   dl2_out_sen  = double( 0.0 )         ;               % Output sensitivity matrix    %

   dl1_out_tim  = double( 0.0 )         ;               % Output time vector           %
   dl1_ode_sen  = double( 0.0 )         ;               % ODE sensitivity vector       %
```

```
   dl1_ini_sen  = double( 0.0 )         ;       % Initial sensitivity vector   %
   dl1_tot_cnt  = double( 0.0 )         ;       % Total count vector (CFU/mL)  %
   dl1_exp_val  = double( 0.0 )         ;       % Exponent value vector        %
   dl1_lag_cnt  = double( 0.0 )         ;       % Lag count vector             %
   dl1_icc_adj  = double( 0.0 )         ;       % Init count adjustment vector %
   dl1_rte_con  = double( 0.0 )         ;       % Local rate constants         %

   dl0_ode_tim  = double( 0.0 )         ;       % ODE time value               %

   m            = uint32( 0 )           ;       % Counters/indexes             %

   jl0_num_tim  = int32( 0 )            ;       % Number of times              %
   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag             %

   Sl0_opt      = []                    ;       % ODE options structure        %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS         %
   dc0_abs_tol       = double( 1.0e-12 );       % Absolute tolerance           %
   dc0_rel_tol       = double( 1.0e-4  );       % Relative tolerance           %
   dc0_log_cor       = log( 10.0 )      ;       % Correction for base 10 log   %

   jc0_ste_len       = 2                ;       % State length [ x1 x2 ]       %
   jc0_rcv_len       = 4                ;       % Rate constant vector length  %
   jc0_sen_len       = 8                ;       % Sensitivity vec/mat length   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                 %
   if ( ~exist('ji0_dbg_flg','var') || ...      % No debug flag                %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )    % Debug must be non-negative   %
      no0_src = Sc0_qgd.E_INVARG        ;       % Set status return code       %
      return                            ;       % Exit function                %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
      jl0_dbg_flg = ji0_dbg_flg                 ;
      fprintf( 1 , '*** qm4_slv_sBC ***\n' )    ;
   end

   if ( ( ~isvector( di1_rte_con ) ) || ...     % Must be vectors              %
        ( ~isvector( di1_tim_vec ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct length/size  %
   if ( ( length( di1_rte_con )   ~= jc0_rcv_len ) || ...
        ( size( di2_cnt_mat , 1 ) ~= jc0_ste_len ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct formulation  %
   if ( ( ni0_mod_num ~= Sc0_qgd.J_QCMB4P ) && ...
        ( ni0_mod_num ~= Sc0_qgd.J_QCMC4P ) && ...
        ( ni0_mod_num ~= Sc0_qgd.J_QCMD4P ) && ...
        ( ni0_mod_num ~= Sc0_qgd.J_QCME4P ) )
      no0_src = Sc0_qgd.E_INVARG        ;
      return                            ;
   end

   dl1_ini_sen = zeros( jc0_sen_len , 1 )       ;       % Initial sensitivity  %
   jl0_dbg_flg = Sc0_qgd.J_DBGLV0               ;       % Turn off debug       %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                              %
   %    Tests have shown that for the system of differential equations used in %
   % the  4-parameter QCM  ode45, ode23, ode113, ode15s,  ode23s, ode23t,  and %
   % ode23tb return approximately the same results. Since ode45 is the fastest %
```

```
% ODE solver, it is used  instead of ode15s.  If an example is  found where %
% the differential equations  appear to be stiff, replace the call to ode45 %
% below with ode15s.  No other change is necessary.                        %
%                                                                          %
%    Formulation D/E is stiff and will use ode15s.                         %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                  % Set up ODE solver       %
     Sl0_opt =                            ...     % Set ODE solver options  %
odeset                                    ...
 (                                        ...
     'Abstol'        , dc0_abs_tol   , ...
     'Reltol'        , dc0_rel_tol     ...
 )                                                ;

if ( ni0_mod_num == Sc0_qgd.J_QCMB4P )            % Formulation B           %
    [                            ...              % Run ODE solver          %
     dl1_out_tim             , ...               % Output time vector      %
     dl2_out_sen               ...               % Output sensitivity matrix %
    ] =                        ...
   ode45                       ...
    (                          ...
     @( dl0_ode_tim ,          ...               % Solver time and sensitivity %
        dl1_ode_sen )          ...
     qm4_clc_sdB                          ...     % Calculate differentials  %
          (                               ...
             dl0_ode_tim               , ...     % Evaluation time value    %
             dl1_ode_sen               , ...     % Count array              %
             di0_ini_cnt               , ...     % Initial concentration of M %
             di1_tim_vec               , ...     % Vector of evaluation times %
             di2_cnt_mat               , ...     % Soln count matrix (x1,x2) %
             di1_rte_con               , ...     % Rate constants (k)       %
             jl0_dbg_flg                 ...     % Debug flag               %
          )                        , ...
     di1_tim_vec               , ...             % Vector of evaluation times %
     dl1_ini_sen               , ...             % Vector of init sensitivities %
     Sl0_opt                     ...             % ODE options             %
    )                                             ;
elseif ( ni0_mod_num == Sc0_qgd.J_QCMC4P )        % Formulation C           %
    [                            ...              % Run ODE solver          %
     dl1_out_tim             , ...               % Output time vector      %
     dl2_out_sen               ...               % Output sensitivity matrix %
    ] =                        ...
   ode45                       ...
    (                          ...
     @( dl0_ode_tim ,          ...               % Solver time and sensitivity %
        dl1_ode_sen )          ...
     qm4_clc_sdC                          ...     % Calculate differentials  %
          (                               ...
             dl0_ode_tim               , ...     % Evaluation time value    %
             dl1_ode_sen               , ...     % Count array              %
             di0_ini_cnt               , ...     % Initial concentration of M %
             di1_tim_vec               , ...     % Vector of evaluation times %
             di2_cnt_mat               , ...     % Soln count matrix (x1,x2) %
             di1_rte_con               , ...     % Rate constants (k)       %
             jl0_dbg_flg                 ...     % Debug flag               %
          )                        , ...
     di1_tim_vec               , ...             % Vector of evaluation times %
     dl1_ini_sen               , ...             % Vector of init sensitivities %
     Sl0_opt                     ...             % ODE options             %
    )                                             ;
elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )        % Formulation D           %
    [                            ...              % Run ODE solver          %
     dl1_out_tim             , ...               % Output time vector      %
     dl2_out_sen               ...               % Output sensitivity matrix %
    ] =                        ...
   ode15s                      ...
    (                          ...
```

```
     @( dl0_ode_tim ,          ...           % Solver time and sensitivity  %
        dl1_ode_sen )          ...
     qm4_clc_sdD                        ...   % Calculate differentials      %
          (                             ...
             dl0_ode_tim             , ...   % Evaluation time value        %
             dl1_ode_sen             , ...   % Count array                  %
             di0_ini_cnt             , ...   % Initial concentration of M   %
             di1_tim_vec             , ...   % Vector of evaluation times   %
             di2_cnt_mat             , ...   % Soln count matrix (x1,x2)    %
             di1_rte_con             , ...   % Rate constants (k)           %
             jl0_dbg_flg               ...   % Debug flag                   %
          )                   , ...
     di1_tim_vec             , ...           % Vector of evaluation times   %
     dl1_ini_sen             , ...           % Vector of init sensitivities %
     Sl0_opt                   ...           % ODE options                  %
    )                                        ;
else                                         % Formulation E                %
    [                          ...           % Run ODE solver               %
     dl1_out_tim             , ...           % Output time vector           %
     dl2_out_sen               ...           % Output sensitivity matrix    %
    ] =                        ...
   ode15s                      ...
    (                          ...
     @( dl0_ode_tim ,          ...           % Solver time and sensitivity  %
        dl1_ode_sen )          ...
     qm4_clc_sdE                        ...   % Calculate differentials      %
          (                             ...
             dl0_ode_tim             , ...   % Evaluation time value        %
             dl1_ode_sen             , ...   % Count array                  %
             di0_ini_cnt             , ...   % Initial concentration of M   %
             di1_tim_vec             , ...   % Vector of evaluation times   %
             di2_cnt_mat             , ...   % Soln count matrix (x1,x2)    %
             di1_rte_con             , ...   % Rate constants (k)           %
             jl0_dbg_flg               ...   % Debug flag                   %
          )                   , ...
     di1_tim_vec             , ...           % Vector of evaluation times   %
     dl1_ini_sen             , ...           % Vector of init sensitivities %
     Sl0_opt                   ...           % ODE options                  %
    )                                        ;
end
                                             % Update output values         %
m           = jc0_rcv_len                    ;
jl0_num_tim = length( dl1_out_tim )          ;
dl2_out_sen = dl2_out_sen'                   ;
do2_sen_fun = dl2_out_sen                    ; % Sensitivity function sij   %
do2_cnt_sen = do2_sen_fun(1:m,:)             ; % Unadjusted count sens      %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
% Calculate the sensitivity matrix: F = dg/dk                                %
%                                                                            %
% k = [k1 k2 k3 k4] for formulation A and B                                  %
% g = ln(U)  -> F = dg/dk = (dU/dk) / (U(t;k) * ln(10))                      %
% U = M + M* = (I * e^(-k1*t)) + x1(t)                                       %
%                                                                            %
% dU/dk1 = s11 - (I * t * e^(-k1*t))         dU/dk2 = s12                    %
% dU/dk3 = s13                               dU/dk4 = s14                    %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Calc/adjust output values    %
if ( ( ni0_mod_num == Sc0_qgd.J_QCMD4P ) || ...      % Formulation D-E      %
     ( ni0_mod_num == Sc0_qgd.J_QCME4P ) )
   dl1_rte_con = exp( di1_rte_con )  ;
else                                                 % Formulation A-C      %
   dl1_rte_con = di1_rte_con         ;
end

dl1_exp_val      = -dl1_rte_con(1)   * dl1_out_tim'  ; % -k1 * t            %
```

```
   dl1_exp_val      =  exp( dl1_exp_val )               ; % Exp value vector   %

   dl1_lag_cnt      =  di0_ini_cnt      * dl1_exp_val   ; % M=I*e^(-k1*t)      %
   dl1_icc_adj      = -dl1_lag_cnt     .* dl1_out_tim'  ; % -I*t*e^(-k1*t)     %

   dl1_tot_cnt      = di2_cnt_mat(1,:)  + dl1_lag_cnt   ; % U = M* + M         %
   do2_cnt_sen(1,:) = do2_cnt_sen(1,:)  + dl1_icc_adj   ; % Adjust dU/dk1      %
   do2_sen_mat      = do2_cnt_sen                       ; % dU/dk              %

   for ( m = 1:jc0_rcv_len )                     % F = (dU/dk)/(U(t;k)*ln(10))  %
      do2_sen_mat(m,:) = do2_sen_mat(m,:) ./ ( dl1_tot_cnt * dc0_log_cor ) ;
   end                                           % End for ( m )                %

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                 % Debug                        %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      if ( ni0_mod_num == Sc0_qgd.J_QCMB4P )
         fprintf( 1 , 'FORM B MIN/MAX SENSITIVITIES\n'  )       ;
      elseif ( ni0_mod_num == Sc0_qgd.J_QCMC4P )
         fprintf( 1 , 'FORM C MIN/MAX SENSITIVITIES\n'  )       ;
      elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )
         fprintf( 1 , 'FORM D MIN/MAX SENSITIVITIES\n'  )       ;
      else
         fprintf( 1 , 'FORM E MIN/MAX SENSITIVITIES\n'  )       ;
      end

      fprintf(                                     ...
        1                                        , ...
        'Sens g(:,1): %13.6e\t%13.6e\n'          , ...
        min(do2_sen_mat(1,:))                    , ...
        max(do2_sen_mat(1,:))                    )       ;
      fprintf(                                     ...
        1                                        , ...
        'Sens g(:,2): %13.6e\t%13.6e\n'          , ...
        min(do2_sen_mat(2,:))                    , ...
        max(do2_sen_mat(2,:))                    )       ;
      fprintf(                                     ...
        1                                        , ...
        'Sens g(:,3): %13.6e\t%13.6e\n'          , ...
        min(do2_sen_mat(3,:))                    , ...
        max(do2_sen_mat(3,:))                    )       ;
      fprintf(                                     ...
        1                                        , ...
        'Sens g(:,4): %13.6e\t%13.6e\n'          , ...
        min(do2_sen_mat(4,:))                    , ...
        max(do2_sen_mat(4,:))                    )       ;

      fprintf( 1 , '\n' )                                ;
   end
end                                              % END OF FUNCTION              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_clc_sdB                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           03-MAR-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      03/03/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function is called by the MATLAB ODE solvers, and calculates  the %
% differentials of the sensitivity function s for the 4-parameter Quasi-Chemi- %
% cal Model (QCM-4) using  Formulation B (2 ODEs) described in  Reference [a]. %
% Formulation B  solves for the two variables x1=M* and x2=A over the rate pa- %
% rameters [ k1 k2 k3 k4 ].  Reference [a] formulates  the sensitivity problem %
% as a matrix  system of ordinary differential equations in s where            %
%                                                                              %
%                       s = | s11 s12 s13 s14 |                                %
%                           | s21 s22 s23 s24 |                                %
%                                                                              %
% In order to use MATLAB's ODE solvers, s must be linearized. The value of the %
% sensitivity vector s is defined as                                           %
%                                                                              %
%               s = [ s11 s12 s13 s14 s21 s22 s23 s24 ]                        %
%                                                                              %
% The output differential vector will be similarly linearized.                 %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di0_tim_val     double          This is the time value to evaluate the %
%                                       differential at.                       %
%                                                                              %
% (I)   di1_sen_vec     double(8,1)     Sensitivity vector.  This is the lin-  %
%                                       earized vector of s values.            %
%                                                                              %
% (I)   di0_ini_cnt     double          This is the initial bacterial count    %
%                                       of the lag cells (M) in CFU/mL.        %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Evaluation time vector (days).         %
%                                                                              %
% (I)   di2_cnt_mat     double(2,n)     Solution bacterial counts for the x1   %
%                                       (activated cells) and x2 (antagonistic)%
%                                       variables in formulation B at the times%
%                                       in di1_tim_vec in CFU/mL.              %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of solution rate    %
%                                       constants (k) for the QCM.             %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
```

```
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_dif_vec     double(8,1)     Vector of calculated differentials.    %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                  References                                  %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, Memorandum from NCSU, 8 August 2016.                              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Calculate ODE differentials  %
function                  ...
    [                     ...
        do1_dif_vec     , ...                   % Differential vector          %
        no0_src           ...                   % Status return code           %
    ] =                   ...
   qm4_clc_sdB            ...
    (                     ...
        di0_tim_val     , ...                   % Evaluation time value        %
        di1_sen_vec     , ...                   % Sensitivity vector (si(t))   %
        di0_ini_cnt     , ...                   % Init concentration in CFU/mL %
        di1_tim_vec     , ...                   % Time vector (days)           %
        di2_cnt_mat     , ...                   % Count matrix (x(t))          %
        di1_rte_con     , ...                   % Rate constants (k)           %
        ji0_dbg_flg       ...                   % Debug flag                   %
    )
                                                % PARAMETER DECLARATIONS       %
   do1_dif_vec          = []            ;       % Initialize output            %
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % MATLAB INITIALIZATION        %
   format long                          ;       % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;       % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )       % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOCAL DECLARATIONS           %
   dl1_cnt_vec  = double( 0.0 )         ;       % Count vector at t            %
   dl1_pfx      = double( 0.0 )         ;       % Partial deriv: df/dx         %
   dl1_pft      = double( 0.0 )         ;       % Partial deriv: df/dtheta     %

   dl0_alp_val  = double( 0.0 )         ;       % Alpha value: k2 - k4         %
   dl0_exp_val  = double( 0.0 )         ;       % Exponent value: -k1 * t      %

   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag             %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS         %
   dc0_scl_fac          = 1.0e-9        ;       % Scaling factor               %
   jc0_ste_len          = 2             ;       % Bacteria count vector length %
   jc0_rte_len          = 4             ;       % Rate constant vectors length %
   jc0_sen_len          = 8             ;       % Sensitivity vector length    %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                 %
if ( ~exist('ji0_dbg_flg','var') || ...         % No debug flag                %
     isempty(ji0_dbg_flg)        )
   jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )       % Debug must be non-negative   %
   no0_src = Sc0_qgd.E_INVARG        ;          % Set status return code       %
   return                            ;          % Exit function                %
elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
   jl0_dbg_flg = ji0_dbg_flg                    ;
   fprintf( 1 , '*** qm4_clc_sdB ***\n' )       ;
end

if ( ( ~isvector( di1_sen_vec ) ) || ...        % Must be vectors              %
     ( ~isvector( di1_rte_con ) ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                                % Must be correct length       %
if ( ( length( di1_sen_vec )   ~= jc0_sen_len ) || ...
     ( length( di1_rte_con )   ~= jc0_rte_len ) || ...
     ( size( di2_cnt_mat , 1 ) ~= jc0_ste_len ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Get counts at di0_tim_val    %
dl1_cnt_vec(1) = interp1(              ...      % Interpolate variable time    %
                     di1_tim_vec     , ...
                     di2_cnt_mat(1,:), ...
                     di0_tim_val     , ...
                     'pchip'         , ...
                     'extrap'        )          ;
dl1_cnt_vec(2) = interp1(              ...
                     di1_tim_vec     , ...
                     di2_cnt_mat(2,:), ...
                     di0_tim_val     , ...
                     'pchip'         , ...
                     'extrap'        )          ;

                                                % Initialize vectors           %
dl1_pfx        = zeros( jc0_ste_len * 2 , 1 )        ;
dl1_pft        = zeros( jc0_rte_len * 2 , 1 )        ;
do1_dif_vec    = zeros( jc0_sen_len     , 1 )        ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                            Partial Derivatives                                %
%                                                                               %
% s1:  (df/dx)(1) =  k2 - k4 - ( h * k3 * x2 )                                  %
%      (df/dx)(2) =     - ( h * k3 * x1 )                                       %
%      (df/dT)(1) =  (1 + (-k1 * t)) * I * e^(-k1 * t)                          %
%      (df/dT)(2) =  x1                                                         %
%      (df/dT)(3) =     - ( h * x1 * x2 )                                       %
%      (df/dT)(4) = -x1                                                         %
%                                                                               %
% s2:  (df/dx)(3) = k2 - ( h * k3 * x2 )                                        %
%      (df/dx)(4) =    - ( h * k3 * x1 )                                        %
%      (df/dT)(5) = 0                                                           %
%      (df/dT)(6) = x1                                                          %
%      (df/dT)(7) =    - ( h * x1 * x2 )                                        %
%      (df/dT)(8) = 0                                                           %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dl0_alp_val =  di1_rte_con(2) - di1_rte_con(4)       ; % a = k2 - k4        %
dl0_exp_val = -di1_rte_con(1) * di0_tim_val          ; % -k1 * t             %
```

```
                                                % Partial derivatives for s1   %
   dl1_pfx(1)  =  dc0_scl_fac    * di1_rte_con(3) * dl1_cnt_vec(2)      ;
   dl1_pfx(1)  =  dl0_alp_val    - dl1_pfx(1)                           ;
   dl1_pfx(2)  = -dc0_scl_fac    * di1_rte_con(3) * dl1_cnt_vec(1)      ;

   dl1_pft(1)  =  1.0            + dl0_exp_val                          ;
   dl1_pft(1)  =  dl1_pft(1)     * di0_ini_cnt * exp( dl0_exp_val )     ;
   dl1_pft(2)  =  dl1_cnt_vec(1)
;
   dl1_pft(3)  = -dc0_scl_fac    * dl1_cnt_vec(1) * dl1_cnt_vec(2)      ;
   dl1_pft(4)  = -dl1_cnt_vec(1)                                        ;
                                                % Partial derivatives for s2   %
   dl1_pfx(3)  =  dc0_scl_fac    * di1_rte_con(3) * dl1_cnt_vec(2)      ;
   dl1_pfx(3)  =  di1_rte_con(2) - dl1_pfx(3)                           ;
   dl1_pfx(4)  = -dc0_scl_fac    * di1_rte_con(3) * dl1_cnt_vec(1)      ;

   dl1_pft(5)  =  0.0                                                   ;
   dl1_pft(6)  =  dl1_cnt_vec(1)                                        ;
   dl1_pft(7)  = -dc0_scl_fac    * dl1_cnt_vec(1) * dl1_cnt_vec(2)      ;
   dl1_pft(8)  =  0.0                                                   ;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Calculate differentials      %
   do1_dif_vec(1) = ( dl1_pfx(1) * di1_sen_vec(1) ) + ... % ds(1) = ds(1,1)    %
                    ( dl1_pfx(2) * di1_sen_vec(5) ) + ...
                    ( dl1_pft(1)                  )     ;

   do1_dif_vec(2) = ( dl1_pfx(1) * di1_sen_vec(2) ) + ... % ds(2) = ds(1,2)    %
                    ( dl1_pfx(2) * di1_sen_vec(6) ) + ...
                    ( dl1_pft(2)                  )     ;

   do1_dif_vec(3) = ( dl1_pfx(1) * di1_sen_vec(3) ) + ... % ds(3) = ds(1,3)    %
                    ( dl1_pfx(2) * di1_sen_vec(7) ) + ...
                    ( dl1_pft(3)                  )     ;

   do1_dif_vec(4) = ( dl1_pfx(1) * di1_sen_vec(4) ) + ... % ds(4) = ds(1,4)    %
                    ( dl1_pfx(2) * di1_sen_vec(8) ) + ...
                    ( dl1_pft(4)                  )     ;

   do1_dif_vec(5) = ( dl1_pfx(3) * di1_sen_vec(1) ) + ... % ds(5) = ds(2,1)    %
                    ( dl1_pfx(4) * di1_sen_vec(5) ) + ...
                    ( dl1_pft(5)                  )     ;

   do1_dif_vec(6) = ( dl1_pfx(3) * di1_sen_vec(2) ) + ... % ds(6) = ds(2,2)    %
                    ( dl1_pfx(4) * di1_sen_vec(6) ) + ...
                    ( dl1_pft(6)                  )     ;

   do1_dif_vec(7) = ( dl1_pfx(3) * di1_sen_vec(3) ) + ... % ds(7) = ds(2,3)    %
                    ( dl1_pfx(4) * di1_sen_vec(7) ) + ...
                    ( dl1_pft(7)                  )     ;

   do1_dif_vec(8) = ( dl1_pfx(3) * di1_sen_vec(4) ) + ... % ds(8) = ds(2,4)    %
                    ( dl1_pfx(4) * di1_sen_vec(8) ) + ...
                    ( dl1_pft(8)                  )     ;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Debug                        %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                    ...
        1                                       , ...
        'Evaluation time:    %f\n'              , ...
        di0_tim_val                             )       ;
      fprintf(                                    ...
        1                                       , ...
        'Count vector:       %e\t%e\n'          , ...
        dl1_cnt_vec(1)  , dl1_cnt_vec(2)        )       ;
      fprintf(                                    ...
        1                                       , ...
```

```
        'Rate constants:     %f\t%f\t%f\t%f\n'  , ...
        di1_rte_con(1)  , di1_rte_con(2)        , ...
        di1_rte_con(3)  , di1_rte_con(4)        )       ;
      fprintf(                                   ...
        1                                       , ...
        'Differentials:      %e\t%e\t%e\t%e\n'  , ...
        do1_dif_vec(1)  , do1_dif_vec(2)        , ...
        do1_dif_vec(3)  , do1_dif_vec(4)        )       ;
      fprintf(                                   ...
        1                                       , ...
        '                    %e\t%e\t%e\t%e\n'  , ...
        do1_dif_vec(5)  , do1_dif_vec(6)        , ...
        do1_dif_vec(7)  , do1_dif_vec(8)        )       ;

      fprintf( 1 , '\n' )                                           ;
   end
end                                              % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_clc_sdC                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           23-FEB-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      02/23/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function is called by the MATLAB ODE solvers, and calculates  the %
% differentials of the sensitivity function s for the 4-parameter Quasi-Chemi- %
% cal Model (QCM-4) using Formulation C (2 ODEs and alpha) described in Refer- %
% ence [a]. Formulation C solves for the two variables x1=M* and x2=A over the %
% rate parameters [ k1 k2 k3 alpha ]. Reference [a] formulates the sensitivity %
% problem as a matrix system of ordinary differential equations in s where     %
%                                                                              %
%                        s = | s11 s12 s13 s14 |                               %
%                            | s21 s22 s23 s24 |                               %
%                                                                              %
% In order to use MATLAB's ODE solvers, s must be linearized. The value of the %
% sensitivity vector s is defined as                                           %
%                                                                              %
%                s = [ s11 s12 s13 s14 s21 s22 s23 s24 ]                       %
%                                                                              %
% The output differential vector will be similarly linearized.                 %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di0_tim_val     double          This is the time value to evaluate the %
%                                       differential at.                       %
%                                                                              %
% (I)   di1_sen_vec     double(8,1)     Sensitivity vector.  This is the lin-  %
%                                       earized vector of s values.            %
%                                                                              %
% (I)   di0_ini_cnt     double          This is the initial bacterial count    %
%                                       of the lag cells (M) in CFU/mL.        %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Evaluation time vector (days).         %
%                                                                              %
% (I)   di2_cnt_mat     double(2,n)     Solution bacterial counts for the x1   %
%                                       (activated cells) and x2 (antagonistic)%
%                                       variables in formulation C at the times%
%                                       in di1_tim_vec in CFU/mL.              %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of solution rate    %
%                                       constants (k and alpha) for the QCM.   %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
```

```
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_dif_vec     double(8,1)     Vector of calculated differentials.    %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                  References                                  %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, NCSU Memorandum, 8 August 2016.                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Calculate ODE differentials  %
function                  ...
    [                     ...
        do1_dif_vec     , ...                   % Differential vector          %
        no0_src           ...                   % Status return code           %
    ] =                   ...
   qm4_clc_sdC            ...
    (                     ...
        di0_tim_val     , ...                   % Evaluation time value        %
        di1_sen_vec     , ...                   % Sensitivity vector (si(t))   %
        di0_ini_cnt     , ...                   % Init concentration in CFU/mL %
        di1_tim_vec     , ...                   % Time vector (days)           %
        di2_cnt_mat     , ...                   % Count matrix (x(t))          %
        di1_rte_con     , ...                   % Rate constants (k)           %
        ji0_dbg_flg       ...                   % Debug flag                   %
    )
                                                % PARAMETER DECLARATIONS       %
   do1_dif_vec          = []            ;       % Initialize output            %
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % MATLAB INITIALIZATION        %
   format long                          ;       % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;       % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )       % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOCAL DECLARATIONS           %
   dl1_cnt_vec  = double( 0.0 )         ;       % Count vector at t            %
   dl1_pfx      = double( 0.0 )         ;       % Partial deriv: df/dx         %
   dl1_pft      = double( 0.0 )         ;       % Partial deriv: df/dtheta     %

   dl0_alp_val  = double( 0.0 )         ;       % Alpha value: k2 - k4         %
   dl0_exp_val  = double( 0.0 )         ;       % Exponent value: -k1 * t      %

   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag             %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS         %
   dc0_scl_fac          = 1.0e-9        ;       % Scaling factor               %
   jc0_ste_len          = 2             ;       % Bacteria count vector length %
   jc0_rte_len          = 4             ;       % Rate constant vectors length %
   jc0_sen_len          = 8             ;       % Sensitivity vector length    %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                 %
if ( ~exist('ji0_dbg_flg','var') || ...         % No debug flag                %
     isempty(ji0_dbg_flg)        )
   jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )       % Debug must be non-negative   %
   no0_src = Sc0_qgd.E_INVARG        ;          % Set status return code       %
   return                            ;          % Exit function                %
elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
   jl0_dbg_flg = ji0_dbg_flg                    ;
   fprintf( 1 , '*** qm4_clc_sdC ***\n' )       ;
end

if ( ( ~isvector( di1_sen_vec ) ) || ...        % Must be vectors              %
     ( ~isvector( di1_rte_con ) ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                                % Must be correct length       %
if ( ( length( di1_sen_vec )   ~= jc0_sen_len ) || ...
     ( length( di1_rte_con )   ~= jc0_rte_len ) || ...
     ( size( di2_cnt_mat , 1 ) ~= jc0_ste_len ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Get counts at di0_tim_val    %
dl1_cnt_vec(1) = interp1(              ...      % Interpolate variable time    %
                     di1_tim_vec     , ...
                     di2_cnt_mat(1,:), ...
                     di0_tim_val     , ...
                     'pchip'         , ...
                     'extrap'        )          ;
dl1_cnt_vec(2) = interp1(              ...
                     di1_tim_vec     , ...
                     di2_cnt_mat(2,:), ...
                     di0_tim_val     , ...
                     'pchip'         , ...
                     'extrap'        )          ;

                                                % Initialize vectors           %
dl1_pfx        = zeros( jc0_ste_len * 2 , 1 )        ;
dl1_pft        = zeros( jc0_rte_len * 2 , 1 )        ;
do1_dif_vec    = zeros( jc0_sen_len     , 1 )        ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                            Partial Derivatives                                %
%                                                                               %
% s1:  (df/dx)(1) = a  - ( h * k3 * x2 )                                        %
%      (df/dx)(2) =    - ( h * k3 * x1 )                                        %
%      (df/dT)(1) = (1 + (-k1 * t)) * I * e^(-k1 * t)                           %
%      (df/dT)(2) = 0                                                           %
%      (df/dT)(3) =    - ( h * x1 * x2 )                                        %
%      (df/dT)(4) = x1                                                          %
%                                                                               %
% s2:  (df/dx)(3) = k2 - ( h * k3 * x2 )                                        %
%      (df/dx)(4) =    - ( h * k3 * x1 )                                        %
%      (df/dT)(5) = 0                                                           %
%      (df/dT)(6) = x1                                                          %
%      (df/dT)(7) =    - ( h * x1 * x2 )                                        %
%      (df/dT)(8) = 0                                                           %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dl0_alp_val =  di1_rte_con(2) - di1_rte_con(4)       ; % a = k2 - k4        %
dl0_exp_val = -di1_rte_con(1) * di0_tim_val          ; % -k1 * t             %
```

```
                                                % Partial derivatives for s1   %
dl1_pfx(1)  =  dc0_scl_fac    * di1_rte_con(3) * dl1_cnt_vec(2)      ;
dl1_pfx(1)  =  dl0_alp_val    - dl1_pfx(1)                           ;
dl1_pfx(2)  = -dc0_scl_fac    * di1_rte_con(3) * dl1_cnt_vec(1)      ;

dl1_pft(1)  =  1.0            + dl0_exp_val                          ;
dl1_pft(1)  =  dl1_pft(1)     * di0_ini_cnt * exp( dl0_exp_val )     ;
dl1_pft(2)  =  0.0                                                   ;
dl1_pft(3)  = -dc0_scl_fac    * dl1_cnt_vec(1) * dl1_cnt_vec(2)      ;
dl1_pft(4)  =  dl1_cnt_vec(1)                                        ;
                                                % Partial derivatives for s2   %
dl1_pfx(3)  =  dc0_scl_fac    * di1_rte_con(3) * dl1_cnt_vec(2)      ;
dl1_pfx(3)  =  di1_rte_con(2) - dl1_pfx(3)                           ;
dl1_pfx(4)  = -dc0_scl_fac    * di1_rte_con(3) * dl1_cnt_vec(1)      ;

dl1_pft(5)  =  0.0                                                   ;
dl1_pft(6)  =  dl1_cnt_vec(1)                                        ;
dl1_pft(7)  = -dc0_scl_fac    * dl1_cnt_vec(1) * dl1_cnt_vec(2)      ;
dl1_pft(8)  =  0.0                                                   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Calculate differentials      %
do1_dif_vec(1) = ( dl1_pfx(1) * di1_sen_vec(1) ) + ... % ds(1) = ds(1,1)    %
                 ( dl1_pfx(2) * di1_sen_vec(5) ) + ...
                 ( dl1_pft(1)                  )     ;

do1_dif_vec(2) = ( dl1_pfx(1) * di1_sen_vec(2) ) + ... % ds(2) = ds(1,2)    %
                 ( dl1_pfx(2) * di1_sen_vec(6) ) + ...
                 ( dl1_pft(2)                  )     ;

do1_dif_vec(3) = ( dl1_pfx(1) * di1_sen_vec(3) ) + ... % ds(3) = ds(1,3)    %
                 ( dl1_pfx(2) * di1_sen_vec(7) ) + ...
                 ( dl1_pft(3)                  )     ;

do1_dif_vec(4) = ( dl1_pfx(1) * di1_sen_vec(4) ) + ... % ds(4) = ds(1,4)    %
                 ( dl1_pfx(2) * di1_sen_vec(8) ) + ...
                 ( dl1_pft(4)                  )     ;

do1_dif_vec(5) = ( dl1_pfx(3) * di1_sen_vec(1) ) + ... % ds(5) = ds(2,1)    %
                 ( dl1_pfx(4) * di1_sen_vec(5) ) + ...
                 ( dl1_pft(5)                  )     ;

do1_dif_vec(6) = ( dl1_pfx(3) * di1_sen_vec(2) ) + ... % ds(6) = ds(2,2)    %
                 ( dl1_pfx(4) * di1_sen_vec(6) ) + ...
                 ( dl1_pft(6)                  )     ;

do1_dif_vec(7) = ( dl1_pfx(3) * di1_sen_vec(3) ) + ... % ds(7) = ds(2,3)    %
                 ( dl1_pfx(4) * di1_sen_vec(7) ) + ...
                 ( dl1_pft(7)                  )     ;

do1_dif_vec(8) = ( dl1_pfx(3) * di1_sen_vec(4) ) + ... % ds(8) = ds(2,4)    %
                 ( dl1_pfx(4) * di1_sen_vec(8) ) + ...
                 ( dl1_pft(8)                  )     ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Debug                        %
if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
   fprintf(                                       ...
     1                                          , ...
     'Evaluation time:    %f\n'                 , ...
     di0_tim_val                                )       ;
   fprintf(                                       ...
     1                                          , ...
     'Count vector:       %e\t%e\n'             , ...
     dl1_cnt_vec(1)  , dl1_cnt_vec(2)           )       ;
   fprintf(                                       ...
     1                                          , ...
     'Rate constants:     %f\t%f\t%f\t%f\n'     , ...
```

```
        di1_rte_con(1)  , di1_rte_con(2)        , ...
        di1_rte_con(3)  , di1_rte_con(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Differentials:      %e\t%e\t%e\t%e\n'  , ...
        do1_dif_vec(1)  , do1_dif_vec(2)        , ...
        do1_dif_vec(3)  , do1_dif_vec(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        '                    %e\t%e\t%e\t%e\n'  , ...
        do1_dif_vec(5)  , do1_dif_vec(6)        , ...
        do1_dif_vec(7)  , do1_dif_vec(8)        )       ;

      fprintf( 1 , '\n' )                                           ;
   end
end                                              % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_clc_sdD                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           11-JUL-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/11/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function is called by the MATLAB ODE solvers, and calculates  the %
% differentials of the sensitivity functions  for the 4-parameter Quasi-Chemi- %
% cal  Model (QCM-4) using  Formulation D (2 ODEs).  Formulation D solves  for %
% the two variables  x1=M* and x2=A  over the  rate parameters [ z1 z2 z3 z4 ] %
% where zi = ln(ki). Reference [a] formulates the sensitivity problem as a ma- %
% trix system of ordinary differential equations in s where                    %
%                                                                              %
%                        s = | s11 s12 s13 s14 |                               %
%                            | s21 s22 s23 s24 |                               %
%                                                                              %
% In order to use MATLAB's ODE solvers, s must be linearized. The value of the %
% sensitivity vector s is defined as                                           %
%                                                                              %
%               s = [ s11 s12 s13 s14 s21 s22 s23 s24 ]                        %
%                                                                              %
% The output differential vector will be similarly linearized.                 %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di0_tim_val     double          This is the time value to evaluate the %
%                                       differential at.                       %
%                                                                              %
% (I)   di1_sen_vec     double(8,1)     Sensitivity vector.  This is the lin-  %
%                                       earized vector of s values.            %
%                                                                              %
% (I)   di0_ini_cnt     double          This is the initial bacterial count    %
%                                       of the lag cells (M) in CFU/mL.        %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Evaluation time vector (days).         %
%                                                                              %
% (I)   di2_cnt_mat     double(2,n)     Solution bacterial counts for the x1   %
%                                       (activated cells) and x2 (antagonistic)%
%                                       variables in formulation D at the times%
%                                       in di1_tim_vec in CFU/mL.              %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of solution rate    %
%                                       constant logarithms (z) for the QCM.   %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
```

```
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (O)   do1_dif_vec     double(8,1)     Vector of calculated differentials.    %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                  References                                  %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, NCSU Memorandum, 8 August 2016.                                 %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                      % Calculate ODE differentials  %
function                  ...
    [                     ...
        do1_dif_vec     , ...                         % Differential vector          %
        no0_src           ...                         % Status return code           %
    ] =                   ...
   qm4_clc_sdD            ...
    (                     ...
        di0_tim_val     , ...                         % Evaluation time value        %
        di1_sen_vec     , ...                         % Sensitivity vector (si(t))   %
        di0_ini_cnt     , ...                         % Init concentration in CFU/mL %
        di1_tim_vec     , ...                         % Time vector (days)           %
        di2_cnt_mat     , ...                         % Count matrix (x(t))          %
        di1_rte_con     , ...                         % Rate constants (z)           %
        ji0_dbg_flg       ...                         % Debug flag                   %
    )
                                                      % PARAMETER DECLARATIONS       %
   do1_dif_vec          = []             ;            % Initialize output            %
   no0_src              = uint32( 0 )    ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                      % MATLAB INITIALIZATION        %
   format long                          ;             % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                      % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;             % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )             % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                      % LOCAL DECLARATIONS           %
   dl1_cnt_vec  = double( 0.0 )         ;             % Count vector at t            %
   dl1_pfx      = double( 0.0 )         ;             % Partial deriv: df/dx         %
   dl1_pft      = double( 0.0 )         ;             % Partial deriv: df/dtheta     %

   dl0_exp_vz1  = double( 0.0 )         ;             % exp( z1 )                    %
   dl0_exp_vz2  = double( 0.0 )         ;             % exp( z2 )                    %
   dl0_exp_vz3  = double( 0.0 )         ;             % exp( z3 )                    %
   dl0_exp_vz4  = double( 0.0 )         ;             % exp( z4 )                    %
   dl0_tm1      = double( 0.0 )         ;             % Temporary variables          %
   dl0_tm2      = double( 0.0 )         ;

   jl0_dbg_flg  = int32( 0 )            ;             % Local debug flag             %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                      % CONSTANT DEFINITIONS         %
   dc0_scl_fac          = 1.0e-9        ;             % Scaling factor               %
   jc0_ste_len          = 2             ;             % Bacteria count vector length %
```

```
   jc0_rte_len          = 4             ;       % Rate constant vectors length %
   jc0_sen_len          = 8             ;       % Sensitivity vector length    %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                 %
   if ( ~exist('ji0_dbg_flg','var') || ...      % No debug flag                %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )    % Debug must be non-negative   %
      no0_src = Sc0_qgd.E_INVARG        ;       % Set status return code       %
      return                            ;       % Exit function                %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
      jl0_dbg_flg = ji0_dbg_flg                 ;
      fprintf( 1 , '*** qm4_clc_sdD ***\n' )    ;
   end

   if ( ( ~isvector( di1_sen_vec ) ) || ...     % Must be vectors              %
        ( ~isvector( di1_rte_con ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct length       %
   if ( ( length( di1_sen_vec )   ~= jc0_sen_len ) || ...
        ( length( di1_rte_con )   ~= jc0_rte_len ) || ...
        ( size( di2_cnt_mat , 1 ) ~= jc0_ste_len ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Get counts at di0_tim_val    %
   dl1_cnt_vec(1) = interp1(              ...   % Interpolate variable time    %
                       di1_tim_vec     , ...
                       di2_cnt_mat(1,:), ...
                       di0_tim_val     , ...
                       'pchip'         , ...
                       'extrap'        )       ;
   dl1_cnt_vec(2) = interp1(              ...
                       di1_tim_vec     , ...
                       di2_cnt_mat(2,:), ...
                       di0_tim_val     , ...
                       'pchip'         , ...
                       'extrap'        )       ;

                                                % Initialize vectors           %
   dl1_pfx        = zeros( jc0_ste_len * 2 , 1 )        ;       % df/dx        %
   dl1_pft        = zeros( jc0_rte_len * 2 , 1 )        ;       % df/dT        %
   do1_dif_vec    = zeros( jc0_sen_len     , 1 )        ;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                              %
   %                            Partial Derivatives                               %
   %                                                                              %
   % s1:  (df/dx)(1) =  e^(z2) - e^(z4) - ( h * e^(z3) * x2 )                     %
   %      (df/dx)(2) =                  - ( h * e^(z3) * x1 )                     %
   %      (df/dT)(1) =  I * (1-e^(z1)*t) * e^(z1-e^(z1)*t)                        %
   %      (df/dT)(2) =  e^(z2)*x1                                                 %
   %      (df/dT)(3) =                  - ( h * e^(z3) * x1 * x2 )                %
   %      (df/dT)(4) = -e^(z4)*x1                                                 %
   %                                                                              %
   % s2:  (df/dx)(3) =  e^(z2)          - ( h * e^(z3) * x2 )                    %
   %      (df/dx)(4) =                  - ( h * e^(z3) * x1 )                    %
   %      (df/dT)(5) = 0                                                          %
   %      (df/dT)(6) =  e^(z2)*x1                                                 %
   %      (df/dT)(7) =                  - ( h * e^(z3) * x1 * x2 )                %
   %      (df/dT)(8) = 0                                                          %
   %                                                                              %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dl0_exp_vz1 = exp( di1_rte_con(1) )          ;       % e^z1                 %
dl0_exp_vz2 = exp( di1_rte_con(2) )          ;       % e^z2                 %
dl0_exp_vz3 = exp( di1_rte_con(3) )          ;       % e^z3                 %
dl0_exp_vz4 = exp( di1_rte_con(4) )          ;       % e^z4                 %

                                             % Partial derivatives for s1   %
dl0_tm1     =  dc0_scl_fac    * dl0_exp_vz3    * dl1_cnt_vec(2)      ;
dl1_pfx(1)  =  dl0_exp_vz2    - dl0_exp_vz4    - dl0_tm1             ;
dl1_pfx(2)  = -dc0_scl_fac    * dl0_exp_vz3    * dl1_cnt_vec(1)      ;

dl0_tm1     =  dl0_exp_vz1    * di0_tim_val                          ;
dl0_tm1     =  1.0            - dl0_tm1                              ;
dl0_tm2     =  dl0_exp_vz1    * di0_tim_val                          ;
dl0_tm2     =  di1_rte_con(1) - dl0_tm2                              ;
dl0_tm2     =  exp( dl0_tm2 )                                        ;
dl1_pft(1)  =  di0_ini_cnt    * dl0_tm1        * dl0_tm2             ;
dl1_pft(2)  =  dl0_exp_vz2    * dl1_cnt_vec(1)                       ;
dl0_tm1     =  dl1_cnt_vec(1) * dl1_cnt_vec(2)                       ;
dl1_pft(3)  = -dc0_scl_fac    * dl0_exp_vz3    * dl0_tm1             ;
dl1_pft(4)  = -dl0_exp_vz4    * dl1_cnt_vec(1)                       ;
                                             % Partial derivatives for s2   %
dl0_tm1     =  dc0_scl_fac    * dl0_exp_vz3    * dl1_cnt_vec(2)      ;
dl1_pfx(3)  =  dl0_exp_vz2                     - dl0_tm1             ;
dl1_pfx(4)  =  dl1_pfx(2)                                            ;

dl1_pft(5)  =  0.0                                                   ;
dl1_pft(6)  =  dl1_pft(2)                                            ;
dl1_pft(7)  =  dl1_pft(3)                                            ;
dl1_pft(8)  =  0.0                                                   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Calculate differentials      %
do1_dif_vec(1) = ( dl1_pfx(1) * di1_sen_vec(1) ) + ... % ds(1) = ds(1,1)    %
                 ( dl1_pfx(2) * di1_sen_vec(5) ) + ...
                 ( dl1_pft(1)                  )     ;

do1_dif_vec(2) = ( dl1_pfx(1) * di1_sen_vec(2) ) + ... % ds(2) = ds(1,2)    %
                 ( dl1_pfx(2) * di1_sen_vec(6) ) + ...
                 ( dl1_pft(2)                  )     ;

do1_dif_vec(3) = ( dl1_pfx(1) * di1_sen_vec(3) ) + ... % ds(3) = ds(1,3)    %
                 ( dl1_pfx(2) * di1_sen_vec(7) ) + ...
                 ( dl1_pft(3)                  )     ;

do1_dif_vec(4) = ( dl1_pfx(1) * di1_sen_vec(4) ) + ... % ds(4) = ds(1,4)    %
                 ( dl1_pfx(2) * di1_sen_vec(8) ) + ...
                 ( dl1_pft(4)                  )     ;

do1_dif_vec(5) = ( dl1_pfx(3) * di1_sen_vec(1) ) + ... % ds(5) = ds(2,1)    %
                 ( dl1_pfx(4) * di1_sen_vec(5) ) + ...
                 ( dl1_pft(5)                  )     ;

do1_dif_vec(6) = ( dl1_pfx(3) * di1_sen_vec(2) ) + ... % ds(6) = ds(2,2)    %
                 ( dl1_pfx(4) * di1_sen_vec(6) ) + ...
                 ( dl1_pft(6)                  )     ;

do1_dif_vec(7) = ( dl1_pfx(3) * di1_sen_vec(3) ) + ... % ds(7) = ds(2,3)    %
                 ( dl1_pfx(4) * di1_sen_vec(7) ) + ...
                 ( dl1_pft(7)                  )     ;

do1_dif_vec(8) = ( dl1_pfx(3) * di1_sen_vec(4) ) + ... % ds(8) = ds(2,4)    %
                 ( dl1_pfx(4) * di1_sen_vec(8) ) + ...
                 ( dl1_pft(8)                  )     ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Debug                        %
```

```
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                    ...
        1                                       , ...
        'Evaluation time:     %f\n'             , ...
        di0_tim_val                             )       ;
      fprintf(                                    ...
        1                                       , ...
        'Count vector:        %e\t%e\n'         , ...
        dl1_cnt_vec(1)  , dl1_cnt_vec(2)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Rate constants (z): %f\t%f\t%f\t%f\n'  , ...
        di1_rte_con(1)  , di1_rte_con(2)        , ...
        di1_rte_con(3)  , di1_rte_con(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Differentials:       %e\t%e\t%e\t%e\n' , ...
        do1_dif_vec(1)  , do1_dif_vec(2)        , ...
        do1_dif_vec(3)  , do1_dif_vec(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        '                     %e\t%e\t%e\t%e\n' , ...
        do1_dif_vec(5)  , do1_dif_vec(6)        , ...
        do1_dif_vec(7)  , do1_dif_vec(8)        )       ;

      fprintf( 1 , '\n' )                                           ;
   end
end                                              % END OF FUNCTION           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_clc_sdE                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           25-JUL-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/25/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function is called by the MATLAB ODE solvers, and calculates  the %
% differentials of the sensitivity functions  for the 4-parameter Quasi-Chemi- %
% cal  Model (QCM-4) using  Formulation E (2 ODEs).  Formulation E solves  for %
% the two variables  x1=M* and x2=A  over the  rate parameters [ z1 z2 z3 z4 ] %
% where zi = ln(ki) for i=1-3, and z4=ln(alpha).  Reference [a] formulates the %
% sensitivity problem as a matrix system of ordinary differential equations in %
% s where                                                                      %
%                                                                              %
%                       s = | s11 s12 s13 s14 |                                %
%                           | s21 s22 s23 s24 |                                %
%                                                                              %
% In order to use MATLAB's ODE solvers, s must be linearized. The value of the %
% sensitivity vector s is defined as                                           %
%                                                                              %
%               s = [ s11 s12 s13 s14 s21 s22 s23 s24 ]                        %
%                                                                              %
% The output differential vector will be similarly linearized.                 %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di0_tim_val     double          This is the time value to evaluate the %
%                                       differential at.                       %
%                                                                              %
% (I)   di1_sen_vec     double(8,1)     Sensitivity vector.  This is the lin-  %
%                                       earized vector of s values.            %
%                                                                              %
% (I)   di0_ini_cnt     double          This is the initial bacterial count    %
%                                       of the lag cells (M) in CFU/mL.        %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Evaluation time vector (days).         %
%                                                                              %
% (I)   di2_cnt_mat     double(2,n)     Solution bacterial counts for the x1   %
%                                       (activated cells) and x2 (antagonistic)%
%                                       variables in formulation E at the times%
%                                       in di1_tim_vec in CFU/mL.              %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of solution rate    %
%                                       constant logarithms (z) for the QCM.   %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
```

```
%                                       empty, set to no debug.                 %
%                                       = .J_DBGLV0 No debug info is shown.     %
%                                       = Other - see qcm_ini_prm.              %
%                                                                               %
% (O)   do1_dif_vec     double(8,1)     Vector of calculated differentials.     %
%                                                                               %
% (O)   no0_src         uint32          Status return code.                     %
%                                       = .S_SUCCESS  Successful completion.    %
%                                       = Other - see qcm_ini_prm.              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                                  References                                   %
%                                                                               %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial      %
%    Growth, NCSU Memorandum, 8 August 2016.                                 %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % Calculate ODE differentials  %
function                  ...
    [                     ...
        do1_dif_vec     , ...                           % Differential vector          %
        no0_src           ...                           % Status return code           %
    ] =                   ...
   qm4_clc_sdE            ...
    (                     ...
        di0_tim_val     , ...                           % Evaluation time value        %
        di1_sen_vec     , ...                           % Sensitivity vector (si(t))   %
        di0_ini_cnt     , ...                           % Init concentration in CFU/mL %
        di1_tim_vec     , ...                           % Time vector (days)           %
        di2_cnt_mat     , ...                           % Count matrix (x(t))          %
        di1_rte_con     , ...                           % Rate constants (z)           %
        ji0_dbg_flg       ...                           % Debug flag                   %
    )
                                                        % PARAMETER DECLARATIONS       %
   do1_dif_vec          = []            ;               % Initialize output            %
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOAD GLOBALS                %
   persistent Sc0_qgd                   ;               % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )               % Not initialized             %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOCAL DECLARATIONS           %
   dl1_cnt_vec  = double( 0.0 )         ;               % Count vector at t            %
   dl1_pfx      = double( 0.0 )         ;               % Partial deriv: df/dx         %
   dl1_pft      = double( 0.0 )         ;               % Partial deriv: df/dtheta     %

   dl0_exp_vz1  = double( 0.0 )         ;               % exp( z1 )                    %
   dl0_exp_vz2  = double( 0.0 )         ;               % exp( z2 )                    %
   dl0_exp_vz3  = double( 0.0 )         ;               % exp( z3 )                    %
   dl0_exp_vz4  = double( 0.0 )         ;               % exp( z4 )                    %
   dl0_tm1      = double( 0.0 )         ;               % Temporary variables          %
   dl0_tm2      = double( 0.0 )         ;

   jl0_dbg_flg  = int32( 0 )            ;               % Local debug flag             %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % CONSTANT DEFINITIONS         %
   dc0_scl_fac          = 1.0e-9        ;               % Scaling factor               %
```

```
   jc0_ste_len          = 2             ;       % Bacteria count vector length %
   jc0_rte_len          = 4             ;       % Rate constant vectors length %
   jc0_sen_len          = 8             ;       % Sensitivity vector length    %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                 %
   if ( ~exist('ji0_dbg_flg','var') || ...      % No debug flag                %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )    % Debug must be non-negative   %
      no0_src = Sc0_qgd.E_INVARG        ;       % Set status return code       %
      return                            ;       % Exit function                %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
      jl0_dbg_flg = ji0_dbg_flg                 ;
      fprintf( 1 , '*** qm4_clc_sdE ***\n' )    ;
   end

   if ( ( ~isvector( di1_sen_vec ) ) || ...     % Must be vectors              %
        ( ~isvector( di1_rte_con ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct length       %
   if ( ( length( di1_sen_vec )   ~= jc0_sen_len ) || ...
        ( length( di1_rte_con )   ~= jc0_rte_len ) || ...
        ( size( di2_cnt_mat , 1 ) ~= jc0_ste_len ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Get counts at di0_tim_val    %
   dl1_cnt_vec(1) = interp1(              ...   % Interpolate variable time    %
                       di1_tim_vec     , ...
                       di2_cnt_mat(1,:), ...
                       di0_tim_val     , ...
                       'pchip'         , ...
                       'extrap'        )       ;
   dl1_cnt_vec(2) = interp1(              ...
                       di1_tim_vec     , ...
                       di2_cnt_mat(2,:), ...
                       di0_tim_val     , ...
                       'pchip'         , ...
                       'extrap'        )       ;

                                                % Initialize vectors           %
   dl1_pfx        = zeros( jc0_ste_len * 2 , 1 )        ;       % df/dx        %
   dl1_pft        = zeros( jc0_rte_len * 2 , 1 )        ;       % df/dT        %
   do1_dif_vec    = zeros( jc0_sen_len     , 1 )        ;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                              %
   %                            Partial Derivatives                               %
   %                                                                              %
   % s1:  (df/dx)(1) =  e^(z4)          - ( h * e^(z3) * x2 )                    %
   %      (df/dx)(2) =                  - ( h * e^(z3) * x1 )                    %
   %      (df/dT)(1) =  I * (1-e^(z1)*t) * e^(z1-e^(z1)*t)                       %
   %      (df/dT)(2) =  0                                                        %
   %      (df/dT)(3) =                  - ( h * e^(z3) * x1 * x2 )               %
   %      (df/dT)(4) =  e^(z4)*x1                                                %
   %                                                                              %
   % s2:  (df/dx)(3) =  e^(z2)          - ( h * e^(z3) * x2 )                    %
   %      (df/dx)(4) =                  - ( h * e^(z3) * x1 )                    %
   %      (df/dT)(5) = 0                                                         %
   %      (df/dT)(6) =  e^(z2)*x1                                                %
   %      (df/dT)(7) =                  - ( h * e^(z3) * x1 * x2 )               %
   %      (df/dT)(8) = 0                                                         %
```

```
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dl0_exp_vz1 = exp( di1_rte_con(1) )          ;       % e^z1                   %
dl0_exp_vz2 = exp( di1_rte_con(2) )          ;       % e^z2                   %
dl0_exp_vz3 = exp( di1_rte_con(3) )          ;       % e^z3                   %
dl0_exp_vz4 = exp( di1_rte_con(4) )          ;       % e^z4                   %

                                             % Partial derivatives for s1   %
dl0_tm1     =  dc0_scl_fac    * dl0_exp_vz3    * dl1_cnt_vec(2)      ;
dl1_pfx(1)  =  dl0_exp_vz4                     - dl0_tm1             ;
dl1_pfx(2)  = -dc0_scl_fac    * dl0_exp_vz3    * dl1_cnt_vec(1)      ;

dl0_tm1     =  dl0_exp_vz1    * di0_tim_val                          ;
dl0_tm1     =  1.0            - dl0_tm1                              ;
dl0_tm2     =  dl0_exp_vz1    * di0_tim_val                          ;
dl0_tm2     =  di1_rte_con(1) - dl0_tm2                              ;
dl0_tm2     =  exp( dl0_tm2 )                                        ;
dl1_pft(1)  =  di0_ini_cnt    * dl0_tm1        * dl0_tm2             ;
dl1_pft(2)  =  0.0                                                   ;
dl0_tm1     =  dl1_cnt_vec(1) * dl1_cnt_vec(2)                       ;
dl1_pft(3)  = -dc0_scl_fac    * dl0_exp_vz3    * dl0_tm1             ;
dl1_pft(4)  =  dl0_exp_vz4    * dl1_cnt_vec(1)                       ;
                                             % Partial derivatives for s2   %
dl0_tm1     =  dc0_scl_fac    * dl0_exp_vz3    * dl1_cnt_vec(2)      ;
dl1_pfx(3)  =  dl0_exp_vz2                     - dl0_tm1             ;
dl1_pfx(4)  =  dl1_pfx(2)                                            ;

dl1_pft(5)  =  0.0                                                   ;
dl1_pft(6)  =  dl0_exp_vz2    * dl1_cnt_vec(1)                       ;
dl1_pft(7)  =  dl1_pft(3)                                            ;
dl1_pft(8)  =  0.0                                                   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Calculate differentials      %
do1_dif_vec(1) = ( dl1_pfx(1) * di1_sen_vec(1) ) + ... % ds(1) = ds(1,1)    %
                 ( dl1_pfx(2) * di1_sen_vec(5) ) + ...
                 ( dl1_pft(1)                  )     ;

do1_dif_vec(2) = ( dl1_pfx(1) * di1_sen_vec(2) ) + ... % ds(2) = ds(1,2)    %
                 ( dl1_pfx(2) * di1_sen_vec(6) ) + ...
                 ( dl1_pft(2)                  )     ;

do1_dif_vec(3) = ( dl1_pfx(1) * di1_sen_vec(3) ) + ... % ds(3) = ds(1,3)    %
                 ( dl1_pfx(2) * di1_sen_vec(7) ) + ...
                 ( dl1_pft(3)                  )     ;

do1_dif_vec(4) = ( dl1_pfx(1) * di1_sen_vec(4) ) + ... % ds(4) = ds(1,4)    %
                 ( dl1_pfx(2) * di1_sen_vec(8) ) + ...
                 ( dl1_pft(4)                  )     ;

do1_dif_vec(5) = ( dl1_pfx(3) * di1_sen_vec(1) ) + ... % ds(5) = ds(2,1)    %
                 ( dl1_pfx(4) * di1_sen_vec(5) ) + ...
                 ( dl1_pft(5)                  )     ;

do1_dif_vec(6) = ( dl1_pfx(3) * di1_sen_vec(2) ) + ... % ds(6) = ds(2,2)    %
                 ( dl1_pfx(4) * di1_sen_vec(6) ) + ...
                 ( dl1_pft(6)                  )     ;

do1_dif_vec(7) = ( dl1_pfx(3) * di1_sen_vec(3) ) + ... % ds(7) = ds(2,3)    %
                 ( dl1_pfx(4) * di1_sen_vec(7) ) + ...
                 ( dl1_pft(7)                  )     ;

do1_dif_vec(8) = ( dl1_pfx(3) * di1_sen_vec(4) ) + ... % ds(8) = ds(2,4)    %
                 ( dl1_pfx(4) * di1_sen_vec(8) ) + ...
                 ( dl1_pft(8)                  )     ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
                                                % Debug                        %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      fprintf(                                    ...
        1                                       , ...
        'Evaluation time:    %f\n'              , ...
        di0_tim_val                             )       ;
      fprintf(                                    ...
        1                                       , ...
        'Count vector:       %e\t%e\n'          , ...
        dl1_cnt_vec(1)  , dl1_cnt_vec(2)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Rate constants (z): %f\t%f\t%f\t%f\n'  , ...
        di1_rte_con(1)  , di1_rte_con(2)        , ...
        di1_rte_con(3)  , di1_rte_con(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        'Differentials:      %e\t%e\t%e\t%e\n'  , ...
        do1_dif_vec(1)  , do1_dif_vec(2)        , ...
        do1_dif_vec(3)  , do1_dif_vec(4)        )       ;
      fprintf(                                    ...
        1                                       , ...
        '                    %e\t%e\t%e\t%e\n'  , ...
        do1_dif_vec(5)  , do1_dif_vec(6)        , ...
        do1_dif_vec(7)  , do1_dif_vec(8)        )       ;

      fprintf( 1 , '\n' )                                           ;
   end
end                                             % END OF FUNCTION              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_cse_sBC                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           15-MAR-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/26/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function calculates the standard error  for the sensitivity  of a %
% QCM-4  solution using formulation B, C, D, or E.  It uses an  existing solu- %
% tion  and count data to solve  for the standard  error (SE) as  described in %
% Reference [a].                                                               %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of solution rate    %
%                                       constants (k) for the QCM. For formu-  %
%                                       lation E, alpha is positive.           %
%                                       B: [k1 k2 k3 k4]                       %
%                                       C: [k1 k2 k3 alpha]                    %
%                                       D: [ln(k1) ln(k2) ln(k3) ln(k4)]       %
%                                       E: [ln(k1) ln(k2) ln(k3) ln(alpha)]    %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Vector of measured count times (day).  %
%                                       Must be of length greater than 4.      %
%                                                                              %
% (I)   di1_cnt_vec     double(n,1)     Measured bacterial counts at the times %
%                                       in di1_tim_vec in CFU/mL.              %
%                                                                              %
% (I)   ni0_mod_num     uint32          Model formulation number.  Only values %
%                                       of .J_QCMB4P, .J_QCMC4P, .J_QCMD4P and %
%                                       .J_QCME4P can be used in this function.%
%                                                                              %
% (I)   Si0_ocs         struct          An optional arg for output control.    %
%                                       Parameters defined in qcm_ini_prm.  If %
%                                       blank or empty, set to no debug.       %
%                                                                              %
% (O)   do1_std_err     double(4,1)     Vector of calculated standard error    %
%                                       values for rate constants (k).         %
%                                                                              %
% (O)   do2_cov_mat     double(4,4)     Covariance matrix from sensitivity.    %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Structure Fields                              %
```

```
%                                                                       %
% Si0_ocs:                                                              %
%                                                                       %
%       .j0dbgf         int32           Debug flag (.J_DBGLV0 to .J_DBGLV4)    %
%       .j0dfid         int32           Debug output file ID number            %
%       .c1dfnm         char            Debug filename                         %
%       .j0pltf         int32           Plot flag (true or false)              %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                       %
%                                   References                          %
%                                                                       %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, NCSU Memorandum, 8 August 2016.                            %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % Calc standard err for B-E    %
function                    ...
    [                       ...
        do1_std_err     , ...                           % Calculated standard error    %
        do2_cov_mat     , ...                           % Covariance matrix            %
        no0_src           ...                           % Status return code           %
    ] =                     ...
   qm4_cse_sBC              ...
    (                       ...
        di1_rte_con     , ...                           % Rate constants (k or z)      %
        di1_tim_vec     , ...                           % Vector of count times        %
        di1_cnt_vec     , ...                           % Vector of measured counts    %
        ni0_mod_num     , ...                           % Model formulation number     %
        Si0_ocs           ...                           % Output control structure     %
    )
                                                        % PARAMETER DECLARATIONS       %
   do1_std_err          = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;               % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )               % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOCAL DECLARATIONS           %
   dl2_cnt_mat  = double( 0.0 )         ;               % Count matrix from ODE        %
   dl2_sen_cnt  = double( 0.0 )         ;               % Counts for sensitivity       %
   dl2_sen_fun  = double( 0.0 )         ;               % Sensitivity function (s)     %
   dl2_cnt_sen  = double( 0.0 )         ;               % Count sensitivity matrix     %
   dl2_sen_mat  = double( 0.0 )         ;               % Sensitivity matrix (F)       %
   dl2_prd_mat  = double( 0.0 )         ;               % Product matrix               %
   dl2_cov_mat  = double( 0.0 )         ;               % Covariance matrix            %

   dl1_ini_cnt  = double( 0.0 )         ;               % Initial count vector (x0)    %
   dl1_ret_cnt  = double( 0.0 )         ;               % Output count vector (CFU/mL) %
   dl1_ssr_vec  = double( 0.0 )         ;               % SSR vector                   %
   dl1_rte_con  = double( 0.0 )         ;               % Local rate constants (k)     %

   dl0_dat_var  = double( 0.0 )         ;               % Data variance                %
   dl0_mat_con  = double( 0.0 )         ;               % Matrix condition number      %
   dl0_mat_det  = double( 0.0 )         ;               % Matrix determinant           %

   jl0_num_tim  = int32( 0 )            ;               % Number of times              %
   jl0_dat_dof  = int32( 0 )            ;               % Data degrees of freedom      %
```

```
   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag             %
   jl0_mat_len  = int32( 0 )            ;       % Product matrix length        %
   jl0_mat_rnk  = int32( 0 )            ;       % Product matrix rank          %
   jl0_dfi      = int32( 0 )            ;       % Debug file ID number         %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS         %
   jc0_rcv_len       = 4                ;       % Rate constant vector length  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                 %
   if ( ~exist( 'Si0_ocs' , 'var' ) || ...      % No output control structure  %
        isempty( Si0_ocs )          )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0                    ; % Local debug flag   %
      jl0_dfi     = 1                                   ; % Debug FID          %
   elseif ( Si0_ocs.j0dbgf < Sc0_qgd.J_DBGLV0 ) % Debug must be non-negative   %
      no0_src = Sc0_qgd.E_INVARG                        ; % Set status code    %
      return                                            ; % Return             %
   elseif ( Si0_ocs.j0dbgf > Sc0_qgd.J_DBGLV0 ) % Valid debug                  %
      jl0_dbg_flg = Si0_ocs.j0dbgf                      ; % Local debug flag   %
      jl0_dfi     = Si0_ocs.j0dfid                      ; % Debug FID          %
      fprintf( jl0_dfi , '*** qm4_cse_sBC ***\n' )      ; % Display function   %
   end

   if ( ( ~isvector( di1_tim_vec ) ) || ...     % Must be vectors              %
        ( ~isvector( di1_cnt_vec ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct length/size  %
   if ( length( di1_tim_vec ) ~= length( di1_cnt_vec ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   if ( ( length( di1_rte_con ) ~= jc0_rcv_len ) || ...
        ( length( di1_tim_vec ) <= jc0_rcv_len ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct formulation  %
   if ( ( ni0_mod_num ~= Sc0_qgd.J_QCMB4P ) && ...
        ( ni0_mod_num ~= Sc0_qgd.J_QCMC4P ) && ...
        ( ni0_mod_num ~= Sc0_qgd.J_QCMD4P ) && ...
        ( ni0_mod_num ~= Sc0_qgd.J_QCME4P ) )
      no0_src = Sc0_qgd.E_INVARG        ;
      return                            ;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Calculate model counts       %
   jl0_num_tim    = length( di1_tim_vec )       ;       % Number of counts     %
   jl0_dat_dof    = jl0_num_tim - jc0_rcv_len   ;       % Degrees of freedom   %
   di1_tim_vec(1) = 0.0                         ;       % First time is zero   %
   dl1_ini_cnt    = [ di1_cnt_vec(1) , ...              % Set initial counts   %
                      0              , ...
                      0              , ...
                      0              ]         ;

   if ( ( ni0_mod_num == Sc0_qgd.J_QCMB4P ) || ...      % Formulation B        %
        ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) )           % Formulation C        %
      dl1_rte_con = di1_rte_con                 ;

      if ( ni0_mod_num == Sc0_qgd.J_QCMC4P )
         dl1_rte_con(4) = dl1_rte_con(2) - ...          % Convert alpha to k4  %
                          dl1_rte_con(4)        ;
      end
```

```
    [                   ...
     dl1_ret_cnt     , ...                    % Output count vector (CFU/mL) %
     dl2_cnt_mat     , ...                    % Count matrix from ODE        %
     no0_src           ...                    % Status return code           %
    ] =                ...
   Bq4_slv_ode         ...                    % Solve ODE formulation B      %
    (                  ...
     dl1_rte_con     , ...                    % Rate constant values (k)     %
     dl1_ini_cnt     , ...                    % Initial count vector (x0)    %
     di1_tim_vec     , ...                    % Vector of times (day)        %
     jl0_dbg_flg       ...                    % Debug flag                   %
    )                                 ;
elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )           % Formulation D        %
    [                   ...
     dl1_ret_cnt     , ...                    % Output count vector (CFU/mL) %
     dl2_cnt_mat     , ...                    % Count matrix from ODE        %
     no0_src           ...                    % Status return code           %
    ] =                ...
   Dq4_slv_ode         ...                    % Solve ODE formulation D      %
    (                  ...
     di1_rte_con     , ...                    % Rate constant values (k)     %
     dl1_ini_cnt     , ...                    % Initial count vector (x0)    %
     di1_tim_vec     , ...                    % Vector of times (day)        %
     jl0_dbg_flg       ...                    % Debug flag                   %
    )                                 ;
else                                                 % Formulation E        %
   dl1_rte_con    = di1_rte_con               ;      % Convert to form D    %
   dl1_rte_con(4) = exp( dl1_rte_con(2) ) - ...
                    exp( dl1_rte_con(4) )    ;
   dl1_rte_con(4) = log( dl1_rte_con(4) )    ;

    [                   ...
     dl1_ret_cnt     , ...                    % Output count vector (CFU/mL) %
     dl2_cnt_mat     , ...                    % Count matrix from ODE        %
     no0_src           ...                    % Status return code           %
    ] =                ...
   Dq4_slv_ode         ...                    % Solve ODE formulation D      %
    (                  ...
     dl1_rte_con     , ...                    % Rate constant values (k)     %
     dl1_ini_cnt     , ...                    % Initial count vector (x0)    %
     di1_tim_vec     , ...                    % Vector of times (day)        %
     jl0_dbg_flg       ...                    % Debug flag                   %
    )                                 ;
end
                                              % Calculate data variance      %
dl1_ssr_vec = log10( di1_cnt_vec  ) - log10( dl1_ret_cnt' )  ; % (y-g)      %
dl1_ssr_vec = dl1_ssr_vec .^2                                ; % (y-g)^2    %
dl0_dat_var = sum( dl1_ssr_vec )    / double( jl0_dat_dof )  ; % (y-g)^2/dof%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%    The standard error calculations from Reference [a] requires calculat-  %
% ing the inverse of a Gram matrix (Tr(A)*A) to estimate the covariance.    %
% Using the NSRDEC datasets, this Gram matrix is found to often be ill-con- %
% ditioned with condition numbers as high as 10^47.  These ill-conditioned  %
% matrices could lead to poor estimates for the covariance.                %
%                                                                              %
%    In the reduced QCM formulations (B-E), the sensitivity is only depend- %
% ent on the counts for  M* and A  which are the only values solved  for in %
% the  system of ODEs.  The count matrix dl2_cnt_sen holds count values for %
% M, M*, A, and D  where D is set to 10e-15.  The  values needed for sensi- %
% tivity calculations are then dl2_cnt_mat(:,2:3).                          %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % Sensitivity at measurements  %
dl2_sen_cnt = zeros( 2 , jl0_num_tim )        ;
dl2_sen_cnt = dl2_cnt_mat(:,2:3)'             ;
```

```
 [                      ...                    % Calc QCM solution sensitivity%
     dl2_sen_fun     , ...                     % Sensitivity function (s)     %
     dl2_cnt_sen     , ...                     % Count sensitivity matrix     %
     dl2_sen_mat     , ...                     % Sensitivity matrix (F)       %
     no0_src           ...                     % Status return code           %
 ] =                   ...
qm4_slv_sBC            ...
 (                     ...
     di1_rte_con     , ...                     % Solution rate constants      %
     dl1_ini_cnt(1)  , ...                     % Initial count of M           %
     di1_tim_vec     , ...                     % Time vector (day)            %
     dl2_sen_cnt     , ...                     % Solution counts (x1 and x2)  %
     ni0_mod_num     , ...                     % Model formulation number     %
     jl0_dbg_flg       ...                     % Debug flag                   %
 )                            ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                               % Calculate standard error     %
dl2_prd_mat = dl2_sen_mat * ...                % Calculate product matrix     %
              dl2_sen_mat'           ;        % [4,n] x [n,4]                %

jl0_mat_len = length( dl2_prd_mat )  ;         % Matrix characteristics       %
jl0_mat_rnk = rank( dl2_prd_mat )    ;
dl0_mat_con = cond( dl2_prd_mat )    ;
dl0_mat_det = det( dl2_prd_mat )     ;

if ( jl0_mat_rnk < jl0_mat_len )               % Not full rank                %
   dl2_cov_mat = pinv( dl2_prd_mat ) * dl0_dat_var   ;
else                                           % Full rank                    %
   dl2_cov_mat =  inv( dl2_prd_mat ) * dl0_dat_var   ;
end
                                               % Set output values            %
do1_std_err = diag( dl2_cov_mat )    ;         % Standard error               %
do1_std_err = sqrt( do1_std_err )    ;
do2_cov_mat = dl2_cov_mat            ;         % Covariance matrix            %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                               % Debug                        %
if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
   if ( ni0_mod_num == Sc0_qgd.J_QCMB4P )            % Formulation B        %
      fprintf( jl0_dfi , 'FORM B MIN/MAX STANDARD ERRORS\n'  )       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCMC4P )        % Formulation C        %
      fprintf( jl0_dfi , 'FORM C MIN/MAX STANDARD ERRORS\n'  )       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )        % Formulation D        %
      fprintf( jl0_dfi , 'FORM D MIN/MAX STANDARD ERRORS\n'  )       ;
   else
      fprintf( jl0_dfi , 'FORM E MIN/MAX STANDARD ERRORS\n'  )       ;
   end

   fprintf( jl0_dfi , 'PRODUCT MATRIX CHARACTERISTICS\n' )           ;

   fprintf( jl0_dfi , ' Matrix length:    %d\n' , jl0_mat_len )      ;
   fprintf( jl0_dfi , ' Matrix rank:      %d\n' , jl0_mat_rnk )      ;
   fprintf( jl0_dfi , ' Condition number: %e\n' , dl0_mat_con )      ;
   fprintf( jl0_dfi , ' Determinant:      %e\n' , dl0_mat_det )      ;

   fprintf( jl0_dfi , '\n' )                                   ;

   fprintf( jl0_dfi , 'STANDARD ERROR\n' )                     ;
   fprintf(                    ...
     jl0_dfi                                        , ...
     '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'             , ...
     do1_std_err(1)          , do1_std_err(2)       , ...
     do1_std_err(3)          , do1_std_err(4)       )       ;

   fprintf( jl0_dfi , '\n' )                                   ;
```

```
      fprintf( jl0_dfi , 'COVARIANCE MATRIX\n' )                ;
      fprintf(                                             ...
        jl0_dfi                                          , ...
        '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'               , ...
        dl2_cov_mat(1,1)        , dl2_cov_mat(1,2)       , ...
        dl2_cov_mat(1,3)        , dl2_cov_mat(1,4)       )       ;
      fprintf(                                             ...
        jl0_dfi                                          , ...
        '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'               , ...
        dl2_cov_mat(2,1)        , dl2_cov_mat(2,2)       , ...
        dl2_cov_mat(2,3)        , dl2_cov_mat(2,4)       )       ;
      fprintf(                                             ...
        jl0_dfi                                          , ...
        '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'               , ...
        dl2_cov_mat(3,1)        , dl2_cov_mat(3,2)       , ...
        dl2_cov_mat(3,3)        , dl2_cov_mat(3,4)       )       ;
      fprintf(                                             ...
        jl0_dfi                                          , ...
        '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'               , ...
        dl2_cov_mat(4,1)        , dl2_cov_mat(4,2)       , ...
        dl2_cov_mat(4,3)        , dl2_cov_mat(4,4)       )       ;

      fprintf( jl0_dfi , '\n' )                                 ;
   end

end                                              % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_cse_aBC                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           15-MAR-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/26/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function calculates the standard error  for the sensitivity  of a %
% QCM-4  solution using formulation B, C, D, or E.  It uses an  existing solu- %
% tion  and count data to solve  for the standard  error (SE) as  described in %
% Reference [a].                                                               %
%                                                                              %
%       This is an altered version of the function qm4_cse_sBC that will eval- %
% uate the standard error across the entire experiment time with uniform spac- %
% ing.  The measured data is only used to evaluate the measurement variance.   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of solution rate    %
%                                       constants (k) for the QCM. For formu-  %
%                                       lation E, alpha is positive.           %
%                                       B: [k1 k2 k3 k4]                       %
%                                       C: [k1 k2 k3 alpha]                    %
%                                       D: [ln(k1) ln(k2) ln(k3) ln(k4)]       %
%                                       E: [ln(k1) ln(k2) ln(k3) ln(alpha)]    %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Vector of measured count times (day).  %
%                                       Must be of length greater than 4.      %
%                                                                              %
% (I)   di1_cnt_vec     double(n,1)     Measured bacterial counts at the times %
%                                       in di1_tim_vec in CFU/mL.              %
%                                                                              %
% (I)   di1_evl_tim     double(m,1)     Evaluation time vector (day).          %
%                                                                              %
% (I)   ni0_mod_num     uint32          Model formulation number.  Only values %
%                                       of .J_QCMB4P, .J_QCMC4P, .J_QCMD4P and %
%                                       .J_QCME4P can be used in this function.%
%                                                                              %
% (I)   Si0_ocs         struct          An optional arg for output control.    %
%                                       Parameters defined in qcm_ini_prm.  If %
%                                       blank or empty, set to no debug.       %
%                                                                              %
% (O)   do1_std_err     double(4,1)     Vector of calculated standard error    %
%                                       values for rate constants (k).         %
%                                                                              %
% (O)   do2_cov_mat     double(4,4)     Covariance matrix from sensitivity.    %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
```

```
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                 Structure Fields                             %
%                                                                              %
% Si0_ocs:                                                                     %
%                                                                              %
%       .j0dbgf         int32           Debug flag (.J_DBGLV0 to .J_DBGLV4)    %
%       .j0dfid         int32           Debug output file ID number            %
%       .c1dfnm         char            Debug filename                         %
%       .j0pltf         int32           Plot flag (true or false)              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                    References                                %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, NCSU Memorandum, 8 August 2016.                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % ALT: Calc std err for B-E    %
function                    ...
    [                       ...
        do1_std_err     , ...                           % Calculated standard error    %
        do2_cov_mat     , ...                           % Covariance matrix            %
        no0_src           ...                           % Status return code           %
    ] =                     ...
   qm4_cse_aBC              ...
    (                       ...
        di1_rte_con     , ...                           % Rate constants (k or z)      %
        di1_tim_vec     , ...                           % Vector of count times        %
        di1_cnt_vec     , ...                           % Vector of measured counts    %
        di1_evl_tim     , ...                           % Evaluation times             %
        ni0_mod_num     , ...                           % Model formulation number     %
        Si0_ocs           ...                           % Output control structure     %
    )
                                                        % PARAMETER DECLARATIONS       %
   do1_std_err          = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;               % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )               % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOCAL DECLARATIONS           %
   dl2_cnt_mat  = double( 0.0 )         ;               % Count matrix from ODE        %
   dl2_sen_cnt  = double( 0.0 )         ;               % Counts for sensitivity       %
   dl2_sen_fun  = double( 0.0 )         ;               % Sensitivity function (s)     %
   dl2_cnt_sen  = double( 0.0 )         ;               % Count sensitivity matrix     %
   dl2_sen_mat  = double( 0.0 )         ;               % Sensitivity matrix (F)       %
   dl2_prd_mat  = double( 0.0 )         ;               % Product matrix               %
   dl2_cov_mat  = double( 0.0 )         ;               % Covariance matrix            %

   dl1_ini_cnt  = double( 0.0 )         ;               % Initial count vector (x0)    %
   dl1_ret_cnt  = double( 0.0 )         ;               % Output count vector (CFU/mL) %
   dl1_ssr_vec  = double( 0.0 )         ;               % SSR vector                   %
   dl1_rte_con  = double( 0.0 )         ;               % Local rate constants (k)     %
```

```
   dl0_dat_var  = double( 0.0 )         ;       % Data variance                %
   dl0_mat_con  = double( 0.0 )         ;       % Matrix condition number      %
   dl0_mat_det  = double( 0.0 )         ;       % Matrix determinant           %

   jl0_num_tim  = int32( 0 )            ;       % Number of times              %
   jl0_dat_dof  = int32( 0 )            ;       % Data degrees of freedom      %
   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag             %
   jl0_mat_len  = int32( 0 )            ;       % Product matrix length        %
   jl0_mat_rnk  = int32( 0 )            ;       % Product matrix rank          %
   jl0_dfi      = int32( 0 )            ;       % Debug file ID number         %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS         %
   jc0_rcv_len       = 4                ;       % Rate constant vector length  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION        %
                                                % Check inputs                 %
   if ( ~exist( 'Si0_ocs' , 'var' ) || ...      % No output control structure  %
        isempty( Si0_ocs )          )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0                    ; % Local debug flag   %
      jl0_dfi     = 1                                   ; % Debug FID          %
   elseif ( Si0_ocs.j0dbgf < Sc0_qgd.J_DBGLV0 ) % Debug must be non-negative   %
      no0_src = Sc0_qgd.E_INVARG                        ; % Set status code    %
      return                                            ; % Return             %
   elseif ( Si0_ocs.j0dbgf > Sc0_qgd.J_DBGLV0 ) % Valid debug                  %
      jl0_dbg_flg = Si0_ocs.j0dbgf                      ; % Local debug flag   %
      jl0_dfi     = Si0_ocs.j0dfid                      ; % Debug FID          %
      fprintf( jl0_dfi , '*** qm4_cse_aBC ***\n' )      ; % Display function   %
   end

   if ( ( ~isvector( di1_tim_vec ) ) || ...     % Must be vectors             %
        ( ~isvector( di1_cnt_vec ) ) || ...
        ( ~isvector( di1_evl_tim ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct length/size  %
   if ( length( di1_tim_vec ) ~= length( di1_cnt_vec ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end

   if ( ( length( di1_rte_con ) ~= jc0_rcv_len ) || ...
        ( length( di1_tim_vec ) <= jc0_rcv_len ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct formulation  %
   if ( ( ni0_mod_num ~= Sc0_qgd.J_QCMB4P ) && ...
        ( ni0_mod_num ~= Sc0_qgd.J_QCMC4P ) && ...
        ( ni0_mod_num ~= Sc0_qgd.J_QCMD4P ) && ...
        ( ni0_mod_num ~= Sc0_qgd.J_QCME4P ) )
      no0_src = Sc0_qgd.E_INVARG        ;
      return                            ;
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Calculate data variance      %
   jl0_num_tim    = length( di1_tim_vec )      ;       % Number of counts     %
   jl0_dat_dof    = jl0_num_tim - jc0_rcv_len  ;       % Degrees of freedom   %
   di1_tim_vec(1) = 0.0                        ;       % First time is zero   %
   dl1_ini_cnt    = [ di1_cnt_vec(1) , ...             % Set initial counts   %
                      0               , ...
                      0               , ...
                      0               ]         ;
```

```
if ( ( ni0_mod_num == Sc0_qgd.J_QCMB4P ) || ...      % Formulation B        %
     ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) )           % Formulation C        %
   dl1_rte_con = di1_rte_con                  ;

   if ( ni0_mod_num == Sc0_qgd.J_QCMC4P )
      dl1_rte_con(4) = dl1_rte_con(2) - ...          % Convert alpha to k4  %
                       dl1_rte_con(4)        ;
   end

    [                  ...                    % Solve ODEs at data times     %
     dl1_ret_cnt     , ...                    % Output count vector (CFU/mL) %
     dl2_cnt_mat     , ...                    % Count matrix from ODE        %
     no0_src           ...                    % Status return code           %
    ] =                ...
   Bq4_slv_ode         ...                    % Solve ODE formulation B      %
    (                  ...
     dl1_rte_con     , ...                    % Rate constant values (k)     %
     dl1_ini_cnt     , ...                    % Initial count vector (x0)    %
     di1_tim_vec     , ...                    % Vector of times (day)        %
     jl0_dbg_flg       ...                    % Debug flag                   %
    )                                ;
elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )           % Formulation D        %
    [                  ...                    % Solve ODEs at data times     %
     dl1_ret_cnt     , ...                    % Output count vector (CFU/mL) %
     dl2_cnt_mat     , ...                    % Count matrix from ODE        %
     no0_src           ...                    % Status return code           %
    ] =                ...
   Dq4_slv_ode         ...                    % Solve ODE formulation D      %
    (                  ...
     di1_rte_con     , ...                    % Rate constant values (k)     %
     dl1_ini_cnt     , ...                    % Initial count vector (x0)    %
     di1_tim_vec     , ...                    % Vector of times (day)        %
     jl0_dbg_flg       ...                    % Debug flag                   %
    )                                ;
else                                                 % Formulation E        %
   dl1_rte_con    = di1_rte_con               ;      % Convert to form D    %
   dl1_rte_con(4) = exp( dl1_rte_con(2) ) - ...
                    exp( dl1_rte_con(4) )    ;
   dl1_rte_con(4) = log( dl1_rte_con(4) )    ;

    [                  ...                    % Solve ODEs at data times     %
     dl1_ret_cnt     , ...                    % Output count vector (CFU/mL) %
     dl2_cnt_mat     , ...                    % Count matrix from ODE        %
     no0_src           ...                    % Status return code           %
    ] =                ...
   Dq4_slv_ode         ...                    % Solve ODE formulation D      %
    (                  ...
     dl1_rte_con     , ...                    % Rate constant values (k)     %
     dl1_ini_cnt     , ...                    % Initial count vector (x0)    %
     di1_tim_vec     , ...                    % Vector of times (day)        %
     jl0_dbg_flg       ...                    % Debug flag                   %
    )                                ;
end
                                              % Calculate data variance      %
dl1_ssr_vec = log10( di1_cnt_vec  ) - log10( dl1_ret_cnt' )  ;
dl1_ssr_vec = dl1_ssr_vec .^2                                ;
dl0_dat_var = sum( dl1_ssr_vec )    / double( jl0_dat_dof )  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % Calculate evaluation counts  %
jl0_num_tim    = length( di1_evl_tim )        ;       % Number of counts     %
jl0_dat_dof    = jl0_num_tim - jc0_rcv_len    ;       % Degrees of freedom   %
di1_evl_tim(1) = 0.0                          ;       % First time is zero   %
dl1_ini_cnt    = [ di1_cnt_vec(1) , ...               % Set initial counts   %
                   0              , ...
                   0              , ...
                   0              ]          ;
```

```
if ( ( ni0_mod_num == Sc0_qgd.J_QCMB4P ) || ...      % Formulation B        %
     ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) )           % Formulation C        %
    [                  ...                   % Solve ODEs at eval times     %
     dl1_ret_cnt     , ...                   % Count vector U (CFU/mL)      %
     dl2_cnt_mat     , ...                   % Count matrix from ODE        %
     no0_src           ...                   % Status return code           %
    ] =                ...
   Bq4_slv_ode         ...                   % Solve ODE formulation B      %
    (                  ...
     dl1_rte_con     , ...                   % Rate constant values (k)     %
     dl1_ini_cnt     , ...                   % Initial count vector (x0)    %
     di1_evl_tim     , ...                   % Vector of times (day)        %
     jl0_dbg_flg       ...                   % Debug flag                   %
    )                                ;
elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )           % Formulation D        %
    [                  ...                   % Solve ODEs at eval times     %
     dl1_ret_cnt     , ...                   % Count vector U (CFU/mL)      %
     dl2_cnt_mat     , ...                   % Count matrix from ODE        %
     no0_src           ...                   % Status return code           %
    ] =                ...
   Dq4_slv_ode         ...                   % Solve ODE formulation D      %
    (                  ...
     di1_rte_con     , ...                   % Rate constant values (k)     %
     dl1_ini_cnt     , ...                   % Initial count vector (x0)    %
     di1_evl_tim     , ...                   % Vector of times (day)        %
     jl0_dbg_flg       ...                   % Debug flag                   %
    )                                ;
else                                                 % Formulation E        %
    [                  ...                   % Solve ODEs at eval times     %
     dl1_ret_cnt     , ...                   % Count vector U (CFU/mL)      %
     dl2_cnt_mat     , ...                   % Count matrix from ODE        %
     no0_src           ...                   % Status return code           %
    ] =                ...
   Dq4_slv_ode         ...                   % Solve ODE formulation D      %
    (                  ...
     dl1_rte_con     , ...                   % Rate constant values (k)     %
     dl1_ini_cnt     , ...                   % Initial count vector (x0)    %
     di1_evl_tim     , ...                   % Vector of times (day)        %
     jl0_dbg_flg       ...                   % Debug flag                   %
    )                                ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%    The standard error calculations from Reference [a] requires calculat-    %
% ing the inverse of a Gram matrix (Tr(A)*A) to estimate the covariance.      %
% Using the NSRDEC datasets, this Gram matrix is found to often be ill-con-   %
% ditioned with condition numbers as high as 10^47.  These ill-conditioned     %
% matrices could lead to poor estimates for the covariance.                   %
%                                                                              %
%    In the reduced QCM formulations (B-E), the sensitivity is only depend-   %
% ent on the counts for  M* and A  which are the only values solved  for in   %
% the  system of ODEs.  The count matrix dl2_cnt_sen holds count values for   %
% M, M*, A, and D  where D is set to 10e-15.  The  values needed for sensi-   %
% tivity calculations are then dl2_cnt_mat(:,2:3).                             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                     % Sensitivity at measurements  %
dl2_sen_cnt = zeros( 2 , jl0_num_tim )               ;
dl2_sen_cnt = dl2_cnt_mat(:,2:3)'                    ;

 [                 ...                   % Calc QCM solution sensitivity%
     dl2_sen_fun     , ...                   % Sensitivity function (s)     %
     dl2_cnt_sen     , ...                   % Count sensitivity matrix     %
     dl2_sen_mat     , ...                   % Sensitivity matrix (F)       %
     no0_src           ...                   % Status return code           %
 ] =                ...
qm4_slv_sBC            ...
```

```
 (                       ...
     di1_rte_con     , ...                   % Solution rate constants      %
     dl1_ini_cnt(1)  , ...                   % Initial count of M            %
     di1_evl_tim     , ...                   % Time vector (day)            %
     dl2_sen_cnt     , ...                   % Solution counts (x1 and x2)  %
     ni0_mod_num     , ...                   % Model formulation number     %
     jl0_dbg_flg       ...                   % Debug flag                   %
 )                           ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Calculate standard error     %
dl2_prd_mat = dl2_sen_mat * ...              % Calculate product matrix     %
              dl2_sen_mat'           ;      % [4,n] x [n,4]                %

jl0_mat_len = length( dl2_prd_mat )  ;       % Matrix characteristics       %
jl0_mat_rnk = rank( dl2_prd_mat )    ;
dl0_mat_con = cond( dl2_prd_mat )    ;
dl0_mat_det = det( dl2_prd_mat )     ;

if ( jl0_mat_rnk < jl0_mat_len )             % Not full rank                %
   dl2_cov_mat = pinv( dl2_prd_mat ) * dl0_dat_var   ;
else                                         % Full rank                    %
   dl2_cov_mat =  inv( dl2_prd_mat ) * dl0_dat_var   ;
end
                                             % Set output values            %
do1_std_err = diag( dl2_cov_mat )    ;       % Standard error               %
do1_std_err = sqrt( do1_std_err )    ;
do2_cov_mat = dl2_cov_mat            ;       % Covariance matrix            %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Debug                        %
if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
   if ( ni0_mod_num == Sc0_qgd.J_QCMB4P )            % Formulation B        %
      fprintf( jl0_dfi , 'FORM B MIN/MAX STANDARD ERRORS\n'  )       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCMC4P )        % Formulation C        %
      fprintf( jl0_dfi , 'FORM C MIN/MAX STANDARD ERRORS\n'  )       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )        % Formulation D        %
      fprintf( jl0_dfi , 'FORM D MIN/MAX STANDARD ERRORS\n'  )       ;
   else                                              % Formulation E        %
      fprintf( jl0_dfi , 'FORM E MIN/MAX STANDARD ERRORS\n'  )       ;
   end

   fprintf( jl0_dfi , 'PRODUCT MATRIX CHARACTERISTICS\n' )           ;

   fprintf( jl0_dfi , ' Matrix length:    %d\n' , jl0_mat_len )      ;
   fprintf( jl0_dfi , ' Matrix rank:      %d\n' , jl0_mat_rnk )      ;
   fprintf( jl0_dfi , ' Condition number: %e\n' , dl0_mat_con )      ;
   fprintf( jl0_dfi , ' Determinant:      %e\n' , dl0_mat_det )      ;

   fprintf( jl0_dfi , '\n' )                                   ;

   fprintf( jl0_dfi , 'STANDARD ERROR\n' )                     ;
   fprintf(                    ...
     jl0_dfi                                           , ...
     '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'                , ...
     do1_std_err(1)          , do1_std_err(2)          , ...
     do1_std_err(3)          , do1_std_err(4)          )       ;

   fprintf( jl0_dfi , '\n' )                                   ;

   fprintf( jl0_dfi , 'COVARIANCE MATRIX\n' )                  ;
   fprintf(                                                ...
     jl0_dfi                                           , ...
     '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'                , ...
     dl2_cov_mat(1,1)        , dl2_cov_mat(1,2)        , ...
     dl2_cov_mat(1,3)        , dl2_cov_mat(1,4)        )       ;
   fprintf(                                                ...
     jl0_dfi                                           , ...
```

```
        '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'                  , ...
        dl2_cov_mat(2,1)        , dl2_cov_mat(2,2)          , ...
        dl2_cov_mat(2,3)        , dl2_cov_mat(2,4)          )        ;
      fprintf(                                                ...
        jl0_dfi                                             , ...
        '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'                  , ...
        dl2_cov_mat(3,1)        , dl2_cov_mat(3,2)          , ...
        dl2_cov_mat(3,3)        , dl2_cov_mat(3,4)          )        ;
      fprintf(                                                ...
        jl0_dfi                                             , ...
        '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'                  , ...
        dl2_cov_mat(4,1)        , dl2_cov_mat(4,2)          , ...
        dl2_cov_mat(4,3)        , dl2_cov_mat(4,4)          )        ;

      fprintf( jl0_dfi , '\n' )                                      ;
   end

end                                                % END OF FUNCTION              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_qmr_fBC                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           23-FEB-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer                Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/26/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function performs model reduction for  Formulations B, C, D and E %
% of the QCM-4 model.                                                          %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of solution rate    %
%                                       parameters for the QCM.  It is set to  %
%                                       [k1 k2 k3 k4] for formulation B and C, %
%                                       and [z1 z2 z3 z4] for formulation D/E. %
%                                       The function will reduce the rate para-%
%                                       meters to the minimal set that can be  %
%                                       reliably estimated.                    %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Vector of measured count times (day).  %
%                                       Must be of length greater than 4.      %
%                                                                              %
% (I)   di1_cnt_vec     double(n,1)     Measured bacterial counts at the times %
%                                       in di1_tim_vec in CFU/mL.              %
%                                                                              %
% (I)   di1_evl_tim     double(m,1)     Evaluation time vector (day).  This is %
%                                       an optional parameter.  If blank or    %
%                                       empty, sensitivity calculations will   %
%                                       use di1_tim_vec.                       %
%                                                                              %
% (I)   ni0_mod_num     uint32          Model formulation number.  Only values %
%                                       of .J_QCMB4P, .J_QCMC4P, .J_QCMD4P or  %
%                                       .J_QCME4P are used with this function. %
%                                                                              %
% (I)   ji0_rfd_flg     int32           An optional argument for forcing re-   %
%                                       duction for death.  If true, the func- %
%                                       tion forces a two parameter solution   %
%                                       for k1 and k4.  If blank, empty, or    %
%                                       false, it is ignored.                  %
%                                                                              %
% (I)   Si0_ops         struct          Optimization parameter structure.      %
%                                                                              %
% (I)   Si0_ocs         struct          An optional arg for output control.    %
%                                       Parameters defined in qcm_ini_prm.  If %
%                                       blank or empty, set to no debug.       %
%                                                                              %
% (O)   So1_mrs         struct()        Model reduction structure vector. This %
```

```
%                                       structure array is of size 1 to 4 de-  %
%                                       pending on the number of parameters    %
%                                       that are significant.                  %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                               Structure Fields                               %
%                                                                              %
% Si0_ops:                                                                     %
%                                                                              %
%       .d1mink         double(1,4)     Vector of minimum k or alpha values    %
%       .d1maxk         double(1,4)     Vector of maximum k or alpha values    %
%       .d0atol         double          Absolute tolerance value               %
%       .d0rtol         double          Relative tolerance value               %
%                                                                              %
% Si0_ocs:                                                                     %
%                                                                              %
%       .j0dbgf         int32           Debug flag (.J_DBGLV0 to .J_DBGLV4)    %
%       .j0dfid         int32           Debug output file ID number            %
%       .c1dfnm         char            Debug filename                         %
%       .j0pltf         int32           Plot flag (true or false)              %
%                                                                              %
% So1_mrs(k):                                                                  %
%                                                                              %
%       .d0mssr         double          Minimum SSR value                      %
%       .d1srcv         double(1,4)     Solution rate constant values          %
%       .d1timv         double(n,1)     Count time vector (day)                %
%       .d1cntv         double(n,1)     Bacterial count vector (CFU/mL)        %
%       .d1lgcv         double(n,1)     Log10 of count vector                  %
%       .n1rciv         uint32(1,k)     RCV solution index vector (k=1 to 4)   %
%       .d0tsta         double          Test statistic T                       %
%       .d0chis         double          Chi-square value                       %
%       .j0sigf         logical         Significant flag (true or false)       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % QCM-4 model reduction: B-E   %
function                  ...
    [                     ...
        So1_mrs         , ...                           % Model reduction struct vector%
        no0_src           ...                           % Status return code           %
    ] =                   ...
   qm4_qmr_fBC            ...
    (                     ...
        di1_rte_con     , ...                           % Solution rate constants (k)  %
        di1_tim_vec     , ...                           % Vector of count times        %
        di1_cnt_vec     , ...                           % Vector of measured counts    %
        di1_evl_tim     , ...                           % Evaluation times (optional)  %
        ni0_mod_num     , ...                           % Model formulation number     %
        ji0_rfd_flg     , ...                           % Reduction for death          %
        Si0_ops         , ...                           % Optimization parameters      %
        Si0_ocs           ...                           % Output control structure     %
    )
                                                        % PARAMETER DECLARATIONS       %
   So1_mrs              = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;               % Global QCM data              %
```

```
   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )       % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%
```

```
     isempty(ji0_rfd_flg)        )
   jl0_rfd_flg = false               ;
else
   jl0_rfd_flg = ji0_rfd_flg         ;
end

if ( ( ~isvector( di1_rte_con ) ) || ...     % Must be vectors              %
     ( ~isvector( di1_tim_vec ) ) || ...
     ( ~isvector( di1_cnt_vec ) ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                             % Must be correct length/size  %
if ( ( length( di1_rte_con ) ~= jc0_rcv_len           ) || ...
     ( length( di1_tim_vec ) ~= length( di1_cnt_vec ) ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                             % Must be correct formulation  %
if     ( ni0_mod_num == Sc0_qgd.J_QCMB4P )
   nl0_mod_num = Sc0_qgd.J_QCMB4V    ;
   dl1_rte_con = di1_rte_con         ;
elseif ( ni0_mod_num == Sc0_qgd.J_QCMC4P )
   nl0_mod_num = Sc0_qgd.J_QCMC4V    ;
   dl1_rte_con = di1_rte_con         ;
elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )
   nl0_mod_num = Sc0_qgd.J_QCMD4V    ;
   dl1_rte_con = di1_rte_con         ;
elseif ( ni0_mod_num == Sc0_qgd.J_QCME4P )
   nl0_mod_num    = Sc0_qgd.J_QCME4V         ;
   dl1_rte_con    = di1_rte_con              ;       % Convert to form D    %
   dl1_rte_con(4) = exp( dl1_rte_con(2) ) - ...
                    exp( dl1_rte_con(4) )    ;
   dl1_rte_con(4) = log( dl1_rte_con(4) )    ;
else
   no0_src = Sc0_qgd.E_INVARG        ;
   return                            ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Initialize values            %
dl1_ini_cnt    = [ di1_cnt_vec(1) , ...      % Set initial counts (x(0))    %
                   0              , ...
                   0              , ...
                   0              ]         ;
di1_tim_vec(1) = 0.0                         ;       % First t must be zero %
jl0_num_pts    = length( di1_tim_vec ) - 1   ;       % N - 1                %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%    The ranking of the rate constants for parameter reduction is found by  %
% first calculating the sensitivity matrix for a set of rate constants, and %
% then performing a QR decomposition on the sensitivity matrix.  The MATLAB %
% qr function is used to perform an economy-size decomposition, and the re- %
% sulting  permutation matrix holds the  ranking of the rate constants. For %
% model reduction, the sensitivity is calculated using the data times.      %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Calculate sensitivities      %
 [                     ...                   % Calc QCM solution sensitivity%
     dl2_sen_fun     , ...                   % Sensitivity function (s)     %
     dl2_cnt_sen     , ...                   % Count sensitivity matrix     %
     dl2_sen_mat     , ...                   % Sensitivity matrix (F)       %
     dl2_cov_mat     , ...                   % Covariance matrix            %
     dl1_std_err     , ...                   % Standard error vector (SE)   %
     dl0_ssr_val     , ...                   % Full SSR value               %
     no0_src           ...                   % Status return code           %
 ] =                   ...
```

```
qm4_cso_fBC             ...
 (                      ...
     di1_rte_con      , ...                    % Rate constants (k)           %
     di1_tim_vec      , ...                    % Vector of count times        %
     di1_cnt_vec      , ...                    % Vector of measured counts    %
     []               , ...                    % Evaluation times             %
     ni0_mod_num      , ...                    % Model formulation number     %
     Si0_ocs            ...                    % Debug flag                   %
 )                                     ;

 [                      ...                    % Economy-size QR decomposition%
     dl2_uni_mat      , ...                    % Unitary matrix (Q)           %
     dl2_utr_mat      , ...                    % Upper triangular matrix (R)  %
     dl1_prm_vec        ...                    % Permutation vector (P)       %
 ] =                    ...
qr                      ...
 (                      ...
     dl2_sen_mat'     , ...                    % Sensitivity matrix (F)       %
     0                  ...                    % Specify economy-size decomp  %
 )                                     ;

if ( jl0_rfd_flg )                              % Reduction for death          %
   dl1_prm_vec(1) = 1                 ;         % k1                           %
   dl1_prm_vec(2) = 4                 ;         % k4                           %
   dl1_prm_vec(3) = 2                 ;         % Ignored                      %
   dl1_prm_vec(4) = 3                 ;         % Ignored                      %
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % 1-parameter solution         %
dl1_ssr_vec = [ NaN NaN NaN NaN ]     ;         % Initialize SSR vector        %
nl1_rsi_vec = dl1_prm_vec(1)          ;         % Rate constant soln index vec %

 [                      ...
     dl1_out_tim      , ...                    % Output vector of times (day) %
     dl1_out_cnt      , ...                    % Output count vector (CFU/mL) %
     ~                , ...                    % Count matrix from ODE        %
     dl1_sol_rcv      , ...                    % Final rate constants (k-vals)%
     dl1_ssr_vec(1)   , ...                    % Minimum SSR (CFU/mL)         %
     no0_src            ...                    % Status return code           %
 ] =                    ...
qm4_fit_vqm             ...
 (                      ...
     di1_tim_vec      , ...                    % Input time vector (day)      %
     di1_cnt_vec      , ...                    % Input count vector (CFU/mL)  %
     dl1_rte_con      , ...                    % Initial rate constants (k)   %
     dl1_ini_cnt      , ...                    % Initial count vector (x0)    %
     nl1_rsi_vec      , ...                    % Rate cnst solution index vec %
     di1_evl_tim      , ...                    % Evaluation times             %
     nl0_mod_num      , ...                    % Model number                 %
     Si0_ops          , ...                    % Optimization parameters      %
     Si0_ocs            ...                    % Debug flag                   %
 )                                     ;

Sl1_mrs(1).d0mssr = dl1_ssr_vec(1)            ;       % Minimum SSR          %
Sl1_mrs(1).d1srcv = dl1_sol_rcv               ;       % Soln rate const vals %
Sl1_mrs(1).d1timv = dl1_out_tim               ;       % Count time vec (day) %
Sl1_mrs(1).d1cntv = dl1_out_cnt               ;       % Count vector (CFU/mL)%
Sl1_mrs(1).d1lgcv = log10( dl1_out_cnt )      ;       % Log count vector     %
Sl1_mrs(1).n1rciv = nl1_rsi_vec               ;       % RCV soln index vector%
Sl1_mrs(1).d0tsta = []                        ;       % Test statistic T     %
Sl1_mrs(1).d0chis = []                        ;       % Chi-square value     %
Sl1_mrs(1).j0sigf = true                      ;       % Significant flag     %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % 2-parameter solution         %
nl1_rsi_vec = dl1_prm_vec(1:2)        ;         % Rate constant soln index vec %
nl1_rsi_vec = sort( nl1_rsi_vec )     ;
```

```
 [                      ...
     dl1_out_tim     , ...                   % Output vector of times (day) %
     dl1_out_cnt     , ...                   % Output count vector (CFU/mL) %
     ~               , ...                   % Count matrix from ODE        %
     dl1_sol_rcv     , ...                   % Final rate constants (k-vals)%
     dl1_ssr_vec(2)  , ...                   % Minimum SSR (CFU/mL)         %
     no0_src           ...                   % Status return code           %
 ] =                   ...
qm4_fit_vqm            ...
 (                     ...
     di1_tim_vec     , ...                   % Input time vector (day)      %
     di1_cnt_vec     , ...                   % Input count vector (CFU/mL)  %
     dl1_rte_con     , ...                   % Initial rate constants (k)   %
     dl1_ini_cnt     , ...                   % Initial count vector (x0)    %
     nl1_rsi_vec     , ...                   % Rate cnst solution index vec %
     di1_evl_tim     , ...                   % Evaluation times             %
     nl0_mod_num     , ...                   % Model number                 %
     Si0_ops         , ...                   % Optimization parameters      %
     Si0_ocs           ...                   % Debug flag                   %
 )                                   ;

dl0_tst_sta = dl1_ssr_vec(1) - dl1_ssr_vec(2)        ;       % T            %
dl0_chi_val = dl0_tst_sta    * double( jl0_num_pts ) ;       % nT           %
dl0_chi_val = dl0_chi_val    / dl1_ssr_vec(2)        ;       % U = nT/J     %

Sl1_mrs(2).d0mssr = dl1_ssr_vec(2)           ;       % Minimum SSR          %
Sl1_mrs(2).d1srcv = dl1_sol_rcv              ;       % Soln rate const vals %
Sl1_mrs(2).d1timv = dl1_out_tim              ;       % Count time vec (day) %
Sl1_mrs(2).d1cntv = dl1_out_cnt              ;       % Count vector (CFU/mL)%
Sl1_mrs(2).d1lgcv = log10( dl1_out_cnt )     ;       % Log count vector     %
Sl1_mrs(2).n1rciv = nl1_rsi_vec              ;       % RCV soln index vector%
Sl1_mrs(2).d0tsta = dl0_tst_sta              ;       % Test statistic T     %
Sl1_mrs(2).d0chis = dl0_chi_val              ;       % Chi-square value     %

if ( dl0_chi_val > dc0_chi_95p )             % Check if enough              %
   jl0_tst_flg       = true                  ;
   Sl1_mrs(2).j0sigf = true                  ;
else
   jl0_tst_flg       = false                 ;
   Sl1_mrs(2).j0sigf = false                 ;
end

if ( jl0_rfd_flg )                           % Force termination            %
   jl0_tst_flg       = false                 ;
   Sl1_mrs(2).j0sigf = false                 ;
end

if ( ni0_mod_num == Sc0_qgd.J_QCME4P )       % Force termination for now    %
   jl0_tst_flg       = false                 ;
   Sl1_mrs(2).j0sigf = false                 ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % 3-parameter solution         %
if ( jl0_tst_flg )
   nl1_rsi_vec = dl1_prm_vec(1:3)    ;       % Rate constant soln index vec %
   nl1_rsi_vec = sort( nl1_rsi_vec ) ;

    [                    ...
     dl1_out_tim     , ...                   % Output vector of times (day) %
     dl1_out_cnt     , ...                   % Output count vector (CFU/mL) %
     ~               , ...                   % Count matrix from ODE        %
     dl1_sol_rcv     , ...                   % Final rate constants (k-vals)%
     dl1_ssr_vec(3)  , ...                   % Minimum SSR (CFU/mL)         %
     no0_src           ...                   % Status return code           %
    ] =                ...
   qm4_fit_vqm         ...
```

```
    (                   ...
     di1_tim_vec     , ...                    % Input time vector (day)      %
     di1_cnt_vec     , ...                    % Input count vector (CFU/mL)  %
     dl1_rte_con     , ...                    % Initial rate constants (k)   %
     dl1_ini_cnt     , ...                    % Initial count vector (x0)    %
     nl1_rsi_vec     , ...                    % Rate cnst solution index vec %
     di1_evl_tim     , ...                    % Evaluation times             %
     nl0_mod_num     , ...                    % Model number                 %
     Si0_ops         , ...                    % Optimization parameters      %
     Si0_ocs           ...                    % Debug flag                   %
    )                                 ;

   dl0_tst_sta = dl1_ssr_vec(2) - dl1_ssr_vec(3)             ; % T          %
   dl0_chi_val = dl0_tst_sta    * double( jl0_num_pts )      ; % nT         %
   dl0_chi_val = dl0_chi_val    / dl1_ssr_vec(3)             ; % U = nT/J   %

   Sl1_mrs(3).d0mssr = dl1_ssr_vec(3)        ;       % Minimum SSR          %
   Sl1_mrs(3).d1srcv = dl1_sol_rcv           ;       % Soln rate const vals %
   Sl1_mrs(3).d1timv = dl1_out_tim           ;       % Count time vec (day) %
   Sl1_mrs(3).d1cntv = dl1_out_cnt           ;       % Count vector (CFU/mL)%
   Sl1_mrs(3).d1lgcv = log10( dl1_out_cnt )  ;       % Log count vector     %
   Sl1_mrs(3).n1rciv = nl1_rsi_vec           ;       % RCV soln index vector%
   Sl1_mrs(3).d0tsta = dl0_tst_sta           ;       % Test statistic T     %
   Sl1_mrs(3).d0chis = dl0_chi_val           ;       % Chi-square value     %

   if ( dl0_chi_val > dc0_chi_95p )
      jl0_tst_flg       = true               ;
      Sl1_mrs(3).j0sigf = true               ;
   else
      jl0_tst_flg       = false              ;
      Sl1_mrs(3).j0sigf = false              ;
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % 4-parameter solution         %
if ( jl0_tst_flg )
   nl1_rsi_vec = sort( dl1_prm_vec ) ;       % Rate constant soln index vec %

    [                   ...
     dl1_out_tim     , ...                    % Output vector of times (day) %
     dl1_out_cnt     , ...                    % Output count vector (CFU/mL) %
     ~               , ...                    % Count matrix from ODE        %
     dl1_sol_rcv     , ...                    % Final rate constants (k-vals)%
     dl1_ssr_vec(4)  , ...                    % Minimum SSR (CFU/mL)         %
     no0_src           ...                    % Status return code           %
    ] =                ...
   qm4_fit_vqm         ...
    (                  ...
     di1_tim_vec     , ...                    % Input time vector (day)      %
     di1_cnt_vec     , ...                    % Input count vector (CFU/mL)  %
     dl1_rte_con     , ...                    % Initial rate constants (k)   %
     dl1_ini_cnt     , ...                    % Initial count vector (x0)    %
     nl1_rsi_vec     , ...                    % Rate cnst solution index vec %
     di1_evl_tim     , ...                    % Evaluation times             %
     nl0_mod_num     , ...                    % Model number                 %
     Si0_ops         , ...                    % Optimization parameters      %
     Si0_ocs           ...                    % Debug flag                   %
    )                                 ;

   dl0_tst_sta = dl1_ssr_vec(3) - dl1_ssr_vec(4)             ; % T          %
   dl0_chi_val = dl0_tst_sta    * double( jl0_num_pts )      ; % nT         %
   dl0_chi_val = dl0_chi_val    / dl1_ssr_vec(4)             ; % U = nT/J   %

   Sl1_mrs(4).d0mssr = dl1_ssr_vec(4)        ;       % Minimum SSR          %
   Sl1_mrs(4).d1srcv = dl1_sol_rcv           ;       % Soln rate const vals %
   Sl1_mrs(4).d1timv = dl1_out_tim           ;       % Count time vec (day) %
   Sl1_mrs(4).d1cntv = dl1_out_cnt           ;       % Count vector (CFU/mL)%
```

```
   Sl1_mrs(4).d1lgcv = log10( dl1_out_cnt )  ;       % Log count vector     %
   Sl1_mrs(4).n1rciv = nl1_rsi_vec           ;       % RCV soln index vector%
   Sl1_mrs(4).d0tsta = dl0_tst_sta           ;       % Test statistic T     %
   Sl1_mrs(4).d0chis = dl0_chi_val           ;       % Chi-square value     %

   if ( dl0_chi_val > dc0_chi_95p )
      jl0_tst_flg       = true               ;
      Sl1_mrs(4).j0sigf = true               ;
   else
      jl0_tst_flg       = false              ;
      Sl1_mrs(4).j0sigf = false              ;
   end
end

So1_mrs = Sl1_mrs                            ;       % Set output           %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Debug                        %
if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
   if ( ni0_mod_num == Sc0_qgd.J_QCMB4P )            % Formulation B        %
      fprintf( jl0_dfi , 'FORM B MODEL REDUCTION\n'  )       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCMC4P )        % Formulation C        %
      fprintf( jl0_dfi , 'FORM C MODEL REDUCTION\n'  )       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )        % Formulation D        %
      fprintf( jl0_dfi , 'FORM D MODEL REDUCTION\n'  )       ;
   else
      fprintf( jl0_dfi , 'FORM E MODEL REDUCTION\n'  )       ;
   end

   if ( ni0_mod_num == Sc0_qgd.J_QCMD4P )            % Formulation D        %
      dl1_rte_con = exp( di1_rte_con )               ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCME4P )        % Formulation E        %
      dl1_rte_con = exp( di1_rte_con )               ;
   else                                              % Formulation B and C  %
      dl1_rte_con = di1_rte_con                      ;
   end

   fprintf( jl0_dfi , 'INITIAL RATE CONSTANTS\n' )                   ;
   fprintf(                   ...
     jl0_dfi                                         , ...
     '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'              , ...
     dl1_rte_con(1)          , dl1_rte_con(2)        , ...
     dl1_rte_con(3)          , dl1_rte_con(4)        )               ;

   fprintf( jl0_dfi , '\n' )                                         ;

   fprintf( jl0_dfi , 'REDUCED MODEL RATE CONSTANTS (SIGNIFICANT) ') ;

   if ( jl0_rfd_flg )
      fprintf( jl0_dfi , '(FORCED FOR DEATH)\n' )                    ;
   else
      fprintf( jl0_dfi , '\n' )                                      ;
   end

   for ( m = 1:length( Sl1_mrs ) )
      if ( Sl1_mrs(m).j0sigf )
         if ( ni0_mod_num == Sc0_qgd.J_QCMD4P )      % Formulation D        %
            dl1_rte_con = exp( Sl1_mrs(m).d1srcv )   ;
         elseif ( ni0_mod_num == Sc0_qgd.J_QCME4P )  % Formulation E        %
            dl1_rte_con = exp( Sl1_mrs(m).d1srcv )   ;
         else                                        % Formulation B and C  %
            dl1_rte_con = Sl1_mrs(m).d1srcv          ;
         end

         cl1_buf = num2str( Sl1_mrs(m).n1rciv )                      ;

         fprintf(                                             ...
             jl0_dfi                                        , ...
```

```
                '%12s '                                             , ...
                cl1_buf                                             )       ;
            fprintf(                                                  ...
                jl0_dfi                                             , ...
                '%13.6e %13.6e %13.6e %13.6e   %13.6e'              , ...
                dl1_rte_con(1)       , dl1_rte_con(2)               , ...
                dl1_rte_con(3)       , dl1_rte_con(4)               , ...
                Sl1_mrs(m).d0mssr                                   )       ;

            if ( m > 1 )
               fprintf(                                               ...
                jl0_dfi                                             , ...
                '   %13.6e   %13.6e'                                , ...
                Sl1_mrs(m).d0tsta    , Sl1_mrs(m).d0chis            )       ;
            end

            fprintf( jl0_dfi , '\n' )                                       ;
         end
      end

      fprintf( jl0_dfi , '\n' )                                             ;
   end
end                                                % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_cso_fBC                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           05-MAY-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer                Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/26/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function calculates the  sensitivity output of a QCM-4  solution  %
% using formulations B-E.  It uses an  existing QCM-4 solution and  count data %
% to calculate  the sensitivity function (s), the  sensitivity matrix (F), and %
% the  standard error.  Sensitivity  can be  evaluated  at the  measured  data %
% points or at a set of specified times to allow for uniform sensitivity anal- %
% ysis.  This function is based on the methodology discussed in  Reference [a] %
% and the formulations coded in qm4_cse_sBC.m and qm4_slv_sBC.                 %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   di1_rte_con     double(4,1)     This is the vector of solution rate    %
%                                       constants for the QCM.  It is either   %
%                                       [k1 k2 k3 k4], [k1 k2 k3 alpha], or    %
%                                       [z1 z2 z3 z4] depending on the value   %
%                                       of ni0_mod_num.                        %
%                                                                              %
% (I)   di1_tim_vec     double(n,1)     Vector of measured count times (day).  %
%                                       Must be of length greater than 4.      %
%                                                                              %
% (I)   di1_cnt_vec     double(n,1)     Measured bacterial counts at the times %
%                                       in di1_tim_vec in CFU/mL.              %
%                                                                              %
% (I)   di1_evl_tim     double(m,1)     Evaluation time vector (day).  This is %
%                                       an optional parameter.  If blank or    %
%                                       empty, sensitivity calculations will   %
%                                       use di1_tim_vec.                       %
%                                                                              %
% (I)   ni0_mod_num     uint32          Model formulation number.  Only values %
%                                       of .J_QCMB4P, .J_QCMC4P, .J_QCMD4P and %
%                                       .J_QCME4P are used with this function. %
%                                                                              %
% (I)   Si0_ocs         struct          An optional arg for output control.    %
%                                       Parameters defined in qcm_ini_prm.  If %
%                                       blank or empty, set to no debug.       %
%                                                                              %
% (O)   do2_sen_fun     double(8,n)     Sensitivity function s of state at     %
%                                       each time in di1_tim_vec.  The s val-  %
%                                       ues are stored in linearized form as   %
%                                       described in qm4_clc_sdB, qm4_clc_sdC, %
%                                       or qm4_clc_sdD.                        %
%                                                                              %
```

```
% (O)   do2_cnt_sen     double(4,n)     Sensitivity matrix of count function   %
%                                       U at each time in di1_tim_vec for      %
%                                       theta equal to [ k1 k2 k3 k4 ] for     %
%                                       B, [ k1 k2 k3 alpha ] for C, or        %
%                                       [ z1 z2 z3 z4 ] for D and E. For D, z4 %
%                                       is ln(k4); for E, z4 is ln(alpha).     %
%                                                                              %
% (O)   do2_sen_mat     double(4,n)     Sensitivity matrix of the log10 count  %
%                                       at each time in di1_tim_vec relative   %
%                                       to the rate constants (F(theta)).      %
%                                                                              %
% (O)   do2_cov_mat     double(4,4)     Covariance matrix from sensitivity.    %
%                                                                              %
% (O)   do1_std_err     double(4,1)     Vector of calculated standard error    %
%                                       values for rate constants (k).         %
%                                                                              %
% (O)   do0_ssr_val     double          SSR value of QCM-4 solution.           %
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                               Structure Fields                               %
%                                                                              %
% Si0_ocs:                                                                     %
%                                                                              %
%       .j0dbgf         int32           Debug flag (.J_DBGLV0 to .J_DBGLV4)    %
%       .j0dfid         int32           Debug output file ID number            %
%       .c1dfnm         char            Debug filename                         %
%       .j0pltf         int32           Plot flag (true or false)              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                  References                                  %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, NCSU Memorandum, 8 August 2016.                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % Calc sensitivity output B-E  %
function                  ...
    [                     ...
        do2_sen_fun     , ...                           % Sensitivity function (s)     %
        do2_cnt_sen     , ...                           % Count sensitivity matrix     %
        do2_sen_mat     , ...                           % Sensitivity matrix (F)       %
        do2_cov_mat     , ...                           % Covariance matrix            %
        do1_std_err     , ...                           % Calculated standard error    %
        do0_ssr_val     , ...                           % SSR value of QCM-4 solution  %
        no0_src           ...                           % Status return code           %
    ] =                   ...
   qm4_cso_fBC            ...
    (                     ...
        di1_rte_con     , ...                           % Rate constants (k or z)      %
        di1_tim_vec     , ...                           % Vector of count times        %
        di1_cnt_vec     , ...                           % Vector of measured counts    %
        di1_evl_tim     , ...                           % Evaluation times             %
        ni0_mod_num     , ...                           % Model formulation number     %
        Si0_ocs           ...                           % Output control structure     %
    )
                                                        % PARAMETER DECLARATIONS       %
   do2_sen_fun          = []            ;
   do2_cnt_sen          = []            ;
   do2_sen_mat          = []            ;
   do2_cov_mat          = []            ;
   do1_std_err          = []            ;
   do0_ssr_val          = []            ;
```

```
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % MATLAB INITIALIZATION       %
   format long                          ;       % Set default format to long  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOAD GLOBALS                %
   persistent Sc0_qgd                   ;       % Global QCM data             %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )       % Not initialized             %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOCAL DECLARATIONS          %
   dl2_cnt_mat  = double( 0.0 )         ;       % Count matrix from ODE       %
   dl2_sen_cnt  = double( 0.0 )         ;       % Counts for sensitivity      %
   dl2_sen_fun  = double( 0.0 )         ;       % Sensitivity function (s)    %
   dl2_cnt_sen  = double( 0.0 )         ;       % Count sensitivity matrix    %
   dl2_sen_mat  = double( 0.0 )         ;       % Sensitivity matrix (F)      %
   dl2_prd_mat  = double( 0.0 )         ;       % Product matrix              %
   dl2_cov_mat  = double( 0.0 )         ;       % Covariance matrix           %

   dl1_ini_cnt  = double( 0.0 )         ;       % Initial count vector (x0)   %
   dl1_ret_cnt  = double( 0.0 )         ;       % Output count vector (CFU/mL) %
   dl1_ssr_vec  = double( 0.0 )         ;       % SSR vector                  %
   dl1_rte_con  = double( 0.0 )         ;       % Local rate constants        %

   dl0_dat_var  = double( 0.0 )         ;       % Data variance               %
   dl0_mat_con  = double( 0.0 )         ;       % Matrix condition number     %
   dl0_mat_det  = double( 0.0 )         ;       % Matrix determinant          %
   dl0_ssr_val  = double( 0.0 )         ;       % SSR value                   %

   jl0_num_tim  = int32( 0 )            ;       % Number of times             %
   jl0_dat_dof  = int32( 0 )            ;       % Data degrees of freedom     %
   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag            %
   jl0_mat_len  = int32( 0 )            ;       % Product matrix length       %
   jl0_mat_rnk  = int32( 0 )            ;       % Product matrix rank         %
   jl0_dot_flg  = int32( 0 )            ;       % Data or time flag           %
   jl0_dfi      = int32( 0 )            ;       % Debug file ID number        %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % CONSTANT DEFINITIONS        %
   jc0_rcv_len       = 4                ;       % Rate constant vector length %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION       %
                                                % Check inputs                %
   if ( ~exist( 'Si0_ocs' , 'var' ) || ...      % No output control structure %
        isempty( Si0_ocs )          )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0                      ; % Local debug flag  %
      jl0_dfi     = 1                                     ; % Debug FID         %
   elseif ( Si0_ocs.j0dbgf < Sc0_qgd.J_DBGLV0 ) % Debug must be non-negative  %
      no0_src     = Sc0_qgd.E_INVARG                      ; % Set status code   %
      return                                              ; % Return            %
   elseif ( Si0_ocs.j0dbgf > Sc0_qgd.J_DBGLV0 ) % Valid debug                %
      jl0_dbg_flg = Si0_ocs.j0dbgf                        ; % Local debug flag  %
      jl0_dfi     = Si0_ocs.j0dfid                        ; % Debug FID         %
      fprintf( jl0_dfi , '*** qm4_cso_fBC ***\n' )        ; % Display function  %
   end

   if ( ( ~isvector( di1_tim_vec ) ) || ...     % Must be vectors             %
        ( ~isvector( di1_cnt_vec ) ) )
      no0_src = Sc0_qgd.E_INCARG        ;
      return                            ;
   end
                                                % Must be correct length/size %
```

```
if ( length( di1_tim_vec ) ~= length( di1_cnt_vec ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

if ( ( length( di1_rte_con ) ~= jc0_rcv_len ) || ...
     ( length( di1_tim_vec ) <= jc0_rcv_len ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end

                                                % Must be correct formulation  %
if ( ( ni0_mod_num ~= Sc0_qgd.J_QCMB4P ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCMC4P ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCMD4P ) && ...
     ( ni0_mod_num ~= Sc0_qgd.J_QCME4P ) )
   no0_src = Sc0_qgd.E_INVARG        ;
   return                            ;
end

if ( ~exist( 'di1_evl_tim' , 'var' ) || ...  % No evaluation time           %
     isempty( di1_evl_tim )        )
   jl0_dot_flg = 0                   ;       % Sensitivity at data times    %
   di1_evl_tim = di1_tim_vec         ;
elseif ( ~isvector( di1_evl_tim ) )          % Must be a vector             %
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
else
   jl0_dot_flg = 1                   ;       % Sensitivity at eval times    %
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Calculate data variance      %
jl0_num_tim    = length( di1_tim_vec )       ;       % Number of counts     %
jl0_dat_dof    = jl0_num_tim - jc0_rcv_len   ;       % Degrees of freedom   %
jl0_dat_dof    = jl0_dat_dof - 1             ;       % Remove first point   %
di1_tim_vec(1) = 0.0                         ;       % First time is zero   %
dl1_ini_cnt    = [ di1_cnt_vec(1) , ...              % Set initial counts   %
                   0              , ...
                   0              , ...
                   0              ]         ;

if ( ( ni0_mod_num == Sc0_qgd.J_QCMB4P ) || ...      % Formulation B        %
     ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) )           % Formulation C        %
   dl1_rte_con = di1_rte_con                 ;

   if ( ni0_mod_num == Sc0_qgd.J_QCMC4P )
      dl1_rte_con(4) = dl1_rte_con(2) - ...          % Convert alpha to k4  %
                       dl1_rte_con(4)        ;
   end

    [                  ...                   % Solve ODEs at data times     %
     dl1_ret_cnt     , ...                   % Output count vector (CFU/mL) %
     dl2_cnt_mat     , ...                   % Count matrix from ODE        %
     no0_src           ...                   % Status return code           %
    ] =                ...
   Bq4_slv_ode         ...                   % Solve ODE formulation B      %
    (                  ...
     dl1_rte_con     , ...                   % Rate constant values (k)     %
     dl1_ini_cnt     , ...                   % Initial count vector (x0)    %
     di1_tim_vec     , ...                   % Vector of times (day)        %
     jl0_dbg_flg       ...                   % Debug flag                   %
    )                                ;
elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )           % Formulation D        %
    [                  ...                   % Solve ODEs at data times     %
     dl1_ret_cnt     , ...                   % Output count vector (CFU/mL) %
     dl2_cnt_mat     , ...                   % Count matrix from ODE        %
     no0_src           ...                   % Status return code           %
```

```
    ] =                 ...
   Dq4_slv_ode          ...                    % Solve ODE formulation D      %
    (                   ...
     di1_rte_con      , ...                    % Rate constant values (k)     %
     dl1_ini_cnt      , ...                    % Initial count vector (x0)    %
     di1_tim_vec      , ...                    % Vector of times (day)        %
     jl0_dbg_flg        ...                    % Debug flag                   %
    )                                 ;
else                                                   % Formulation E        %
   dl1_rte_con    = di1_rte_con                ;       % Convert to form D    %
   dl1_rte_con(4) = exp( dl1_rte_con(2) ) - ...
                    exp( dl1_rte_con(4) )      ;
   dl1_rte_con(4) = log( dl1_rte_con(4) )      ;

    [                   ...                    % Solve ODEs at data times     %
     dl1_ret_cnt      , ...                    % Output count vector (CFU/mL) %
     dl2_cnt_mat      , ...                    % Count matrix from ODE        %
     no0_src            ...                    % Status return code           %
    ] =                 ...
   Dq4_slv_ode          ...                    % Solve ODE formulation D      %
    (                   ...
     dl1_rte_con      , ...                    % Rate constant values (k)     %
     dl1_ini_cnt      , ...                    % Initial count vector (x0)    %
     di1_tim_vec      , ...                    % Vector of times (day)        %
     jl0_dbg_flg        ...                    % Debug flag                   %
    )                                 ;
end
                                                % Calculate data variance      %
dl1_ssr_vec = log10( di1_cnt_vec  ) - log10( dl1_ret_cnt' )  ;
dl1_ssr_vec = dl1_ssr_vec .^2                                ;
dl0_ssr_val = sum( dl1_ssr_vec )                             ;
dl0_dat_var = dl0_ssr_val           / double( jl0_dat_dof )  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Calculate evaluation counts  %
if ( jl0_dot_flg == 1 )
   jl0_num_tim    = length( di1_evl_tim )    ;       % Number of counts     %
   jl0_dat_dof    = jl0_num_tim - jc0_rcv_len;       % Degrees of freedom   %
   di1_evl_tim(1) = 0.0                      ;       % First time is zero   %
   dl1_ini_cnt    = [ di1_cnt_vec(1) , ...           % Set initial counts   %
                      0               , ...
                      0               , ...
                      0               ]       ;

   if ( ( ni0_mod_num == Sc0_qgd.J_QCMB4P ) || ...   % Formulation B        %
        ( ni0_mod_num == Sc0_qgd.J_QCMC4P ) )        % Formulation C        %

       [                        ...           % Solve ODEs at eval times     %
             dl1_ret_cnt      , ...           % Count vector U (CFU/mL)      %
             dl2_cnt_mat      , ...           % Count matrix from ODE        %
             no0_src            ...           % Status return code           %
       ] =                      ...
      Bq4_slv_ode               ...           % Solve ODE formulation B      %
       (                        ...
             dl1_rte_con      , ...           % Rate constant values (k)     %
             dl1_ini_cnt      , ...           % Initial count vector (x0)    %
             di1_evl_tim      , ...           % Vector of times (day)        %
             jl0_dbg_flg        ...           % Debug flag                   %
       )                                ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )        % Formulation D        %
       [                        ...           % Solve ODEs at eval times     %
             dl1_ret_cnt      , ...           % Count vector U (CFU/mL)      %
             dl2_cnt_mat      , ...           % Count matrix from ODE        %
             no0_src            ...           % Status return code           %
       ] =                      ...
      Dq4_slv_ode               ...           % Solve ODE formulation D      %
       (                        ...
             di1_rte_con      , ...           % Rate constant values (k)     %
```

```
            dl1_ini_cnt     , ...           % Initial count vector (x0)    %
            di1_evl_tim     , ...           % Vector of times (day)        %
            jl0_dbg_flg       ...           % Debug flag                   %
       )                                ;
   else                                            % Formulation E        %
       [                      ...           % Solve ODEs at eval times     %
            dl1_ret_cnt     , ...           % Count vector U (CFU/mL)      %
            dl2_cnt_mat     , ...           % Count matrix from ODE        %
            no0_src           ...           % Status return code           %
       ] =                    ...
      Dq4_slv_ode             ...           % Solve ODE formulation D      %
       (                      ...
            dl1_rte_con     , ...           % Rate constant values (k)     %
            dl1_ini_cnt     , ...           % Initial count vector (x0)    %
            di1_evl_tim     , ...           % Vector of times (day)        %
            jl0_dbg_flg       ...           % Debug flag                   %
       )                                ;
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%    The standard error calculations from Reference [a] requires calculat-  %
% ing the inverse of a Gram matrix (Tr(A)*A) to estimate the covariance.    %
% Using the NSRDEC datasets, this Gram matrix is found to often be ill-con- %
% ditioned with condition numbers as high as 10^47.  These ill-conditioned  %
% matrices could lead to poor estimates for the covariance.                %
%                                                                              %
%    In the reduced QCM formulations (B-E), the sensitivity is only depend- %
% ent on the counts for  M* and A  which are the only values solved  for in %
% the  system of ODEs.  The count matrix dl2_cnt_sen holds count values for %
% M, M*, A, and D  where D is set to 10e-15.  The  values needed for sensi- %
% tivity calculations are then dl2_cnt_mat(:,2:3).                          %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                 % Sensitivity at measurements  %
dl2_sen_cnt = zeros( 2 , jl0_num_tim )           ;
dl2_sen_cnt = dl2_cnt_mat(:,2:3)'                ;

 [                     ...                       % Calc QCM solution sensitivity%
     dl2_sen_fun     , ...                       % Sensitivity function (s)     %
     dl2_cnt_sen     , ...                       % Count sensitivity matrix     %
     dl2_sen_mat     , ...                       % Sensitivity matrix (F)       %
     no0_src           ...                       % Status return code           %
 ] =                   ...
qm4_slv_sBC            ...
 (                     ...
     di1_rte_con     , ...                       % Solution rate constants      %
     dl1_ini_cnt(1)  , ...                       % Initial count of M           %
     di1_evl_tim     , ...                       % Time vector (day)            %
     dl2_sen_cnt     , ...                       % Solution counts (x1 and x2)  %
     ni0_mod_num     , ...                       % Model formulation number     %
     jl0_dbg_flg       ...                       % Debug flag                   %
 )                           ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                 % Calculate standard error     %
dl2_prd_mat = dl2_sen_mat * ...                  % Calculate product matrix     %
              dl2_sen_mat'             ;         % [4,n] x [n,4]                %

jl0_mat_len = length( dl2_prd_mat )  ;           % Matrix characteristics       %
jl0_mat_rnk = rank( dl2_prd_mat )    ;
dl0_mat_con = cond( dl2_prd_mat )    ;
dl0_mat_det = det( dl2_prd_mat )     ;

if ( jl0_mat_rnk < jl0_mat_len )                 % Not full rank                %
   dl2_cov_mat = pinv( dl2_prd_mat ) * dl0_dat_var   ;
else                                             % Full rank                    %
```

```
   dl2_cov_mat =  inv( dl2_prd_mat ) * dl0_dat_var   ;
end
                                                % Set output values            %
do1_std_err = diag( dl2_cov_mat )    ;          % Standard error               %
do1_std_err = sqrt( do1_std_err )    ;
do2_cov_mat = dl2_cov_mat            ;          % Covariance matrix            %
do2_sen_fun = dl2_sen_fun            ;          % Sensitivity function (s)     %
do2_cnt_sen = dl2_cnt_sen            ;          % Count sensitivity            %
do2_sen_mat = dl2_sen_mat            ;          % Sensitivity matrix (F)       %

do0_ssr_val = dl0_ssr_val            ;          % SSR value                    %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Debug                        %
if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
   if ( ni0_mod_num == Sc0_qgd.J_QCMB4P )            % Formulation B        %
      fprintf( jl0_dfi , 'FORM B MIN/MAX STANDARD ERRORS\n'  )       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCMC4P )        % Formulation C        %
      fprintf( jl0_dfi , 'FORM C MIN/MAX STANDARD ERRORS\n'  )       ;
   elseif ( ni0_mod_num == Sc0_qgd.J_QCMD4P )        % Formulation D        %
      fprintf( jl0_dfi , 'FORM D MIN/MAX STANDARD ERRORS\n'  )       ;
   else                                             % Formulation E        %
      fprintf( jl0_dfi , 'FORM D MIN/MAX STANDARD ERRORS\n'  )       ;
   end

   fprintf( jl0_dfi , 'PRODUCT MATRIX CHARACTERISTICS\n' )           ;

   fprintf( jl0_dfi , ' Matrix length:    %d\n' , jl0_mat_len )      ;
   fprintf( jl0_dfi , ' Matrix rank:      %d\n' , jl0_mat_rnk )      ;
   fprintf( jl0_dfi , ' Condition number: %e\n' , dl0_mat_con )      ;
   fprintf( jl0_dfi , ' Determinant:      %e\n' , dl0_mat_det )      ;

   fprintf( jl0_dfi , '\n' )                                         ;

   fprintf( jl0_dfi , 'STANDARD ERROR\n' )                           ;
   fprintf(                    ...
     jl0_dfi                                          , ...
     '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'               , ...
     do1_std_err(1)          , do1_std_err(2)         , ...
     do1_std_err(3)          , do1_std_err(4)         )              ;

   fprintf( jl0_dfi , '\n' )                                         ;

   fprintf( jl0_dfi , 'COVARIANCE MATRIX\n' )                        ;
   fprintf(                                                 ...
     jl0_dfi                                          , ...
     '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'               , ...
     dl2_cov_mat(1,1)        , dl2_cov_mat(1,2)       , ...
     dl2_cov_mat(1,3)        , dl2_cov_mat(1,4)       )              ;
   fprintf(                                                 ...
     jl0_dfi                                          , ...
     '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'               , ...
     dl2_cov_mat(2,1)        , dl2_cov_mat(2,2)       , ...
     dl2_cov_mat(2,3)        , dl2_cov_mat(2,4)       )              ;
   fprintf(                                                 ...
     jl0_dfi                                          , ...
     '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'               , ...
     dl2_cov_mat(3,1)        , dl2_cov_mat(3,2)       , ...
     dl2_cov_mat(3,3)        , dl2_cov_mat(3,4)       )              ;
   fprintf(                                                 ...
     jl0_dfi                                          , ...
     '%13.6e\t%13.6e\t%13.6e\t%13.6e\n'               , ...
     dl2_cov_mat(4,1)        , dl2_cov_mat(4,2)       , ...
     dl2_cov_mat(4,3)        , dl2_cov_mat(4,4)       )              ;

   fprintf( jl0_dfi , '\n' )                                         ;

   fprintf( jl0_dfi , ' SSR value:        %e\n' , do0_ssr_val )      ;
```

```
      fprintf( jl0_dfi , '\n' )                                          ;
   end

end                                               % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_nam_cse                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           18-APR-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      04/18/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function calculates the sensitivities and standard errors for the %
% alternate  models  (NAMs) to the QCM-4.  The four alternate  models are  de- %
% scribed in Reference [a]. This function can only be  called after an alter-  %
% nate model has been fit to a set of measured count data.                     %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   ni0_mod_num     uint32          Model number defined in qcm_ini_prm.   %
%                                       Only models .J_NAMxxx are supported.   %
%                                                                              %
% (I)   Si0_nam         struct          NAM structure that stores the output   %
%                                       from a fit to measured count data.     %
%                                                                              %
% (I)   di1_tim_vec     double(1,n)     Vector of times in days.               %
%                                                                              %
% (I)   di0_ini_cnt     double          Initial count for M in CFU/mL.         %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (I)   Si0_ocs         struct          An optional arg for output control.    %
%                                       Parameters defined in qcm_ini_prm.  If %
%                                       blank or empty, set to no debug.       %
%                                       Overrides ji0_dbg_flg.                 %
%                                                                              %
% (O)   So0_nam         struct          Updated NAM structure with standard    %
%                                       error values and additional statistics.%
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Structure Fields                                %
%                                                                              %
% Si0_nam/So0_nam:                                                             %
%                                                                              %
```

```
%       .d1mink         double(4)       Minimum rate constant vector.          %
%       .d1maxk         double(4)       Maximum rate constant vector.          %
%       .d1ircv         double(4)       Initial rate constant vector.          %
%       .d1rtim         double(n)       Return time from model fits.           %
%       .d1rcnt         double(n)       Return counts from model fits.         %
%       .d2cmat         double(n,4)     Return count matrix.                   %
%       .d1orcv         double(n)       Optimal rate constant vector.          %
%       .d0mnsr         double          Minimum SSR value of fit.              %
%       .d1lcnt         double(n)       Log of return counts from model fits.  %
%       .d2lcnt         double(n,4)     Log of return count matrix.            %
%       .d1serr         double(n)       Standard error vector.                 %
%       .d0dvar         double          Data variance.                         %
%       .d0aicc         double          Corrected AIC value (AICc).            %
%       .n0nump         uint32          Number of points.                      %
%       .j0mlen         int32           Matrix length.                         %
%       .j0mrnk         int32           Matrix rank.                           %
%       .d0mcon         double          Matrix condition number.               %
%       .d0mdet         double          Matrix determinant.                    %
%       .d2covm         double(n,k)     Covariance matrix                      %
%                                                                              %
% Si0_ocs:                                                                     %
%                                                                              %
%       .j0dbgf         int32           Debug flag (.J_DBGLV0 to .J_DBGLV4)    %
%       .j0dfid         int32           Debug output file ID number            %
%       .c1dfnm         char            Debug filename                         %
%       .j0pltf         int32           Plot flag (true or false)              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                  References                                  %
%                                                                              %
% a. Banks, H.T., et al., Analysis of a Quasi-Chemical Model for Bacterial     %
%    Growth, NCSU Memorandum, 8 August 2016.                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % Calc Standard Error for NAMs %
function                  ...
    [                     ...
        So0_nam         , ...                           % Output NAM structure         %
        no0_src           ...                           % Status return code           %
    ] =                   ...
   qm4_nam_cse            ...
    (                     ...
        ni0_mod_num     , ...                           % Model number                 %
        Si0_nam         , ...                           % Input NAM structure          %
        di1_tim_vec     , ...                           % Time vector (day)            %
        di0_ini_cnt     , ...                           % Initial count (CFU/mL)       %
        ji0_dbg_flg     , ...                           % Debug flag                   %
        Si0_ocs           ...                           % Output control structure     %
    )
                                                        % PARAMETER DECLARATIONS       %
   So0_nam              = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % MATLAB INITIALIZATION        %
   format long                          ;               % Set default format to long   %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOAD GLOBALS                 %
   persistent Sc0_qgd                   ;               % Global QCM data              %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )               % Not initialized              %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                        % LOCAL DECLARATIONS           %
```

```
   dl2_prd_mat  = double( 0.0 )         ;       % Product matrix                %
   dl2_cov_mat  = double( 0.0 )         ;       % Covariance matrix             %

   dl1_sol_vec  = double( 0.0 )         ;       % Solution vector               %
   dl1_out_cnt  = double( 0.0 )         ;       % Output count                  %
   dl1_std_err  = double( 0.0 )         ;       % Standard error                %
   dl1_s11      = double( 0.0 )         ;       % Sensitivity matrix values     %
   dl1_s12      = double( 0.0 )         ;
   dl1_s13      = double( 0.0 )         ;
   dl1_s14      = double( 0.0 )         ;
   dl1_s21      = double( 0.0 )         ;
   dl1_s22      = double( 0.0 )         ;
   dl1_s23      = double( 0.0 )         ;
   dl1_s24      = double( 0.0 )         ;
   dl1_e01      = double( 0.0 )         ;       % Exponential vectors           %
   dl1_e02      = double( 0.0 )         ;
   dl1_e03      = double( 0.0 )         ;
   dl1_e04      = double( 0.0 )         ;
   dl1_e05      = double( 0.0 )         ;
   dl1_e06      = double( 0.0 )         ;
   dl1_tm1      = double( 0.0 )         ;       % Temporary vectors             %
   dl1_tm2      = double( 0.0 )         ;
   dl1_tm3      = double( 0.0 )         ;
   dl1_tm4      = double( 0.0 )         ;
   dl1_tm5      = double( 0.0 )         ;
   dl1_tm6      = double( 0.0 )         ;
   dl1_tm7      = double( 0.0 )         ;
   dl1_tm8      = double( 0.0 )         ;

   dl0_dat_var  = double( 0.0 )         ;       % Data variance                 %
   dl0_mat_con  = double( 0.0 )         ;       % Matrix condition number       %
   dl0_mat_det  = double( 0.0 )         ;       % Matrix determinant            %
   dl0_aic_val  = double( 0.0 )         ;       % AIC value                     %
   dl0_f01      = double( 0.0 )         ;       % Fraction values               %
   dl0_f02      = double( 0.0 )         ;
   dl0_f03      = double( 0.0 )         ;
   dl0_f04      = double( 0.0 )         ;
   dl0_tm1      = double( 0.0 )         ;       % Temporary variables           %
   dl0_tm2      = double( 0.0 )         ;
   dl0_tm3      = double( 0.0 )         ;

   p            = uint32( 0 )           ;       % Counters/indexes              %

   jl0_num_pts  = int32( 0 )            ;       % Number of points              %
   jl0_mat_len  = int32( 0 )            ;       % Matrix length                 %
   jl0_mat_rnk  = int32( 0 )            ;       % Matrix rank                   %
   jl0_rcv_len  = int32( 0 )            ;       % Rate constant vector length   %
   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag              %
   jl0_dfi      = int32( 0 )            ;       % Debug file ID number          %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION         %
                                                % Check inputs                  %
   if ( ~exist('ji0_dbg_flg','var') || ...      % No debug flag                 %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
      jl0_dfi     = 1                   ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )    % Debug must be non-negative    %
      no0_src = Sc0_qgd.E_INVARG        ;       % Set status return code        %
      return                            ;       % Exit function                 %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
      jl0_dbg_flg = ji0_dbg_flg                 ;
      jl0_dfi     = 1                           ;
      fprintf( 1 , '*** qm4_nam_cse ***\n' )    ;
   end

   if ( ~exist( 'Si0_ocs' , 'var' ) || ...      % No output control structure   %
        isempty( Si0_ocs )          )
```

```
   jl0_dbg_flg = Sc0_qgd.J_DBGLV0                    ; % Local debug flag   %
   jl0_dfi     = 1                                   ; % Debug FID          %
elseif ( Si0_ocs.j0dbgf < Sc0_qgd.J_DBGLV0 ) % Debug must be non-negative   %
   no0_src = Sc0_qgd.E_INVARG                        ; % Set status code    %
   return                                            ; % Return             %
elseif ( Si0_ocs.j0dbgf > Sc0_qgd.J_DBGLV0 ) % Valid debug                  %
   jl0_dbg_flg = Si0_ocs.j0dbgf                      ; % Local debug flag   %
   jl0_dfi     = Si0_ocs.j0dfid                      ; % Debug FID          %
   fprintf( jl0_dfi , '+++ qm4_nam_cse +++\n' )      ; % Display function   %
end

if ( ni0_mod_num == Sc0_qgd.J_NAM1EX )       % Number of rate constants     %
   jl0_rcv_len = 1                    ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EL )
   jl0_rcv_len = 2                    ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EX )
   jl0_rcv_len = 3                    ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EL )
   jl0_rcv_len = 4                    ;
else
   no0_src = Sc0_qgd.E_INCARG         ;
   return                             ;
end

if ( ( ~isvector( Si0_nam.d1orcv ) ) || ...  % Must be vector               %
     ( ~isvector( di1_tim_vec    ) ) )
   no0_src = Sc0_qgd.E_INCARG         ;
   return                             ;
end
                                              % Must be correct length       %
if ( length( Si0_nam.d1orcv ) ~= jl0_rcv_len )
   no0_src = Sc0_qgd.E_INCARG         ;
   return                             ;
end
                                              % Init structure and variables %
So0_nam        = Si0_nam                      ;
So0_nam.d1serr = []                           ;
So0_nam.d0aicc = []                           ;
jl0_num_pts    = length( di1_tim_vec )        ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % Calculate standard error     %
if ( ni0_mod_num == Sc0_qgd.J_NAM1EX )
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                          %
   %                     1-stage Exponential Model                            %
   %                                                                          %
   %         a = di1_sol_vec(1)                                               %
   %         I = di0_ini_cnt                                                  %
   %         t = di1_tim_vec                                                  %
   %         x = dl1_out_cnt                                                  %
   %                                                                          %
   % Define:                                                                  %
   %                                                                          %
   %         e1      exp(a*t)                                                 %
   %                                                                          %
   % The sensitivity matrix is (1x1) and given by                             %
   %                                                                          %
   %         s1      I*t*e1                                                   %
   %                                                                          %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                              % Calculate data variance      %
   dl0_tm1     = double( jl0_num_pts - jl0_rcv_len ) ;
   dl0_dat_var = Si0_nam.d0mnsr / dl0_tm1             ;
                                              % Set solution vector          %
   dl1_sol_vec = [                              ...
                     Si0_nam.d1orcv(1)        , ...   % a parameter          %
                     NaN                      , ...
```

```
                  NaN                     , ...
                  NaN                       ...
                ]                         ;

     [               ...                  % Evaluate NAM at data        %
      dl1_out_cnt     , ...               % Model count vector (CFU/mL)  %
      no0_src          ...                % Status return code          %
     ] =               ...
    qm4_evl_nam        ...
     (                 ...
      ni0_mod_num     , ...               % Model number                %
      dl1_sol_vec     , ...               % Solution vector             %
      di1_tim_vec     , ...               % Time vector (day)           %
      di0_ini_cnt     , ...               % Initial count (CFU/mL)      %
      0                ...                % Debug flag                  %
     )                                ;

    dl1_out_cnt = dl1_out_cnt'        ;   % Flip to make x(t;q)         %

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    dl1_tm1 =  Si0_nam.d1orcv(1) * di1_tim_vec       ;      % a*t          %

    dl1_e01 = exp( dl1_tm1 )                         ;      % Exponentials %
                                          % Calculate sensitivities     %
    dl1_s11 = zeros( 1 , jl0_num_pts )               ;
                                          % a sensitivity               %
    dl1_s11 = ( di0_ini_cnt        *   ...        % I *                %
                di1_tim_vec       .*   ...        % t *                %
                dl1_e01            )          ;   % e^(a*t)            %
                                          % Adjust for log10 of counts  %
    dl1_s11 = dl1_s11 ./ ( dl1_out_cnt * log( 10.0 ) )       ;
                                          % Calc product and gram matrix %
    dl2_prd_mat = [ dl1_s11 ]                                ;
    dl2_prd_mat = dl2_prd_mat * dl2_prd_mat'                 ;
                                          % Matrix characteristics      %
    jl0_mat_len = length( dl2_prd_mat )                      ;
    jl0_mat_rnk = rank( dl2_prd_mat )                        ;
    dl0_mat_con = cond( dl2_prd_mat )                        ;
    dl0_mat_det = det( dl2_prd_mat )                         ;

    if ( jl0_mat_rnk < jl0_mat_len )      % Not full rank               %
       dl2_cov_mat = pinv( dl2_prd_mat ) * dl0_dat_var       ;
    else                                  % Full rank                   %
       dl2_cov_mat =  inv( dl2_prd_mat ) * dl0_dat_var       ;
    end
                                          % Standard error              %
    dl1_std_err    = diag( dl2_cov_mat )  ;
    dl1_std_err    = sqrt( dl1_std_err )  ;
    dl1_std_err(2) = NaN                  ;
    dl1_std_err(3) = NaN                  ;
    dl1_std_err(4) = NaN                  ;
 elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EL )
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                                                       %
    %                1-stage Exponential Model With Lag                     %
    %                                                                       %
    %         a  = di1_sol_vec(1)                                           %
    %         k1 = di1_sol_vec(2)                                           %
    %         I  = di0_ini_cnt                                              %
    %         t  = di1_tim_vec                                              %
    %         x  = dl1_out_cnt                                              %
    %                                                                       %
    % Define:                                                               %
    %                                                                       %
    %         f1      I*k1 / (a+k1)                                         %
    %         f2      I*k1 / (a+k1)^2                                       %
    %         f3      I*a  / (a+k1)^2                                       %
```

```
%                                                                       %
%         e1      exp(a*t)                                              %
%         e2      exp(-k1*t)                                            %
%                                                                       %
% The sensitivity matrix is (1x2) and given by                          %
%                                                                       %
%         s1      f1*t*e1 - f2*(e1-e2)                                  %
%         s2      -I*t*e2 + f1*t*e2 + f3*(e1-e2)                        %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                        % Calculate data variance      %
dl0_tm1     = double( jl0_num_pts - jl0_rcv_len ) ;
dl0_dat_var = Si0_nam.d0mnsr / dl0_tm1          ;
                                        % Set solution vector          %
dl1_sol_vec = [                           ...
                  Si0_nam.d1orcv(1)     , ...   % a parameter          %
                  Si0_nam.d1orcv(2)     , ...   % b parameter          %
                  NaN                   , ...
                  NaN                     ...
              ]                         ;

 [                  ...                 % Evaluate NAM at data         %
  dl1_out_cnt     , ...                 % Model count vector (CFU/mL)  %
  no0_src           ...                 % Status return code           %
 ] =                ...
qm4_evl_nam         ...
 (                  ...
  ni0_mod_num     , ...                 % Model number                 %
  dl1_sol_vec     , ...                 % Solution vector              %
  di1_tim_vec     , ...                 % Time vector (day)            %
  di0_ini_cnt     , ...                 % Initial count (CFU/mL)       %
  0                 ...                 % Debug flag                   %
 )                              ;

dl1_out_cnt = dl1_out_cnt'        ;     % Flip to make x(t;q)          %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dl0_tm1 =  Si0_nam.d1orcv(1) + Si0_nam.d1orcv(2)  ;       % a+k1         %
dl0_tm2 =  di0_ini_cnt       * Si0_nam.d1orcv(2)  ;       % I*k1         %
dl0_tm3 =  di0_ini_cnt       * Si0_nam.d1orcv(1)  ;       % I*a          %
dl0_f01 =  dl0_tm2           / dl0_tm1            ;       % I*k1/(a+k1)  %
dl0_f02 =  dl0_f01           / dl0_tm1            ;       % I*k1/(a+k1)^2%
dl0_f03 =  dl0_tm3           / dl0_tm1            ;       % I*a/(a+k1)   %
dl0_f03 =  dl0_f03           / dl0_tm1            ;       % I*a/(a+k1)^2 %

dl1_tm1 =  Si0_nam.d1orcv(1) * di1_tim_vec        ;       % a*t          %
dl1_tm2 = -Si0_nam.d1orcv(2) * di1_tim_vec        ;       % -k1*t        %

dl1_e01 = exp( dl1_tm1 )                          ;       % Exponentials %
dl1_e02 = exp( dl1_tm2 )                          ;
                                        % Calculate sensitivities      %
dl1_s11 = zeros( 1 , jl0_num_pts )                ;
dl1_s12 = zeros( 1 , jl0_num_pts )                ;

dl1_tm1 = dl1_e01      - dl1_e02                  ;       % e1-e2        %
                                        % a sensitivity                %
dl1_s11 = ( dl0_f01            *   ...            % f1 *                 %
            di1_tim_vec       .*   ...            % t *                  %
            dl1_e01            ) - ...            % e^(a*t)              %
          ( dl0_f02            *   ...            % f2 *                 %
            dl1_tm1            )          ;       % e1-e2                %
dl1_s12 = (-di0_ini_cnt        *   ...            % -I *                 %
            di1_tim_vec       .*   ...            % t *                  %
            dl1_e02            ) + ...            % e^(-k1*t)            %
          ( dl0_f01            *   ...            % f1 *                 %
            di1_tim_vec       .*   ...            % t *                  %
            dl1_e02            ) + ...            % e^(-k1*t)            %
```

```
            ( dl0_f03            *   ...            % f3 *                 %
              dl1_tm1            )          ;       % e1-e2                 %
                                            % Adjust for log10 of counts   %
   dl1_s11 = dl1_s11 ./ ( dl1_out_cnt * log( 10.0 ) )        ;
   dl1_s12 = dl1_s12 ./ ( dl1_out_cnt * log( 10.0 ) )        ;
                                            % Calc product and gram matrix %
   dl2_prd_mat = [ dl1_s11 ; dl1_s12 ]                       ;
   dl2_prd_mat = dl2_prd_mat * dl2_prd_mat'                  ;
                                            % Matrix characteristics       %
   jl0_mat_len = length( dl2_prd_mat )                       ;
   jl0_mat_rnk = rank( dl2_prd_mat )                         ;
   dl0_mat_con = cond( dl2_prd_mat )                         ;
   dl0_mat_det = det( dl2_prd_mat )                          ;

   if ( jl0_mat_rnk < jl0_mat_len )         % Not full rank                %
      dl2_cov_mat = pinv( dl2_prd_mat ) * dl0_dat_var        ;
   else                                     % Full rank                    %
      dl2_cov_mat =  inv( dl2_prd_mat ) * dl0_dat_var        ;
   end
                                            % Standard error               %
   dl1_std_err    = diag( dl2_cov_mat )      ;
   dl1_std_err    = sqrt( dl1_std_err )      ;
   dl1_std_err(3) = NaN                      ;
   dl1_std_err(4) = NaN                      ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EX )
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                         %
   %                     2-stage Exponential Model (2EX)                     %
   %                                                                         %
   % For the 2EX model:                                                      %
   %                                                                         %
   %         a  = di1_sol_vec(1)                                             %
   %         b  = di1_sol_vec(2)                                             %
   %         T  = di1_sol_vec(3)                                             %
   %         I  = di0_ini_cnt                                                %
   %         t  = di1_tim_vec                                                %
   %         x  = dl1_out_cnt                                                %
   %                                                                         %
   % x(t;q) and s(t;q) are conditional functions.  If T is the model time   %
   % constant tau and p is the index in di1_tim_vec such that                %
   %                                                                         %
   %            T > di1_tim_vec(1:p-1) and T <= di1_tim_vec(p:end)           %
   %                                                                         %
   % then x = [x1(1:p-1),x2(p:end)] and s = [s1(1:p-1),s2(p:end)].           %
   %                                                                         %
   % Define:                                                                 %
   %                                                                         %
   %         e1      exp(a*t)                                                %
   %         e2      exp((b*(T-t))-(a*T))                                    %
   %                                                                         %
   % The sensitivity matrix is (2x3) and given by                            %
   %                                                                         %
   %         s11     I*t*e1                                                  %
   %         s12     0                                                       %
   %         s13     0                                                       %
   %         s21     I*T*e2                                                  %
   %         s22     I*(T-t)*e2                                              %
   %         s23     I*(a+b)*e2                                              %
   %                                                                         %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                            % Calculate data variance      %
   dl0_tm1     = double( jl0_num_pts - jl0_rcv_len ) ;
   dl0_dat_var = Si0_nam.d0mnsr / dl0_tm1            ;
                                            % Set solution vector          %
   dl1_sol_vec = [                            ...
                   Si0_nam.d1orcv(1)       , ...   % a parameter          %
                   Si0_nam.d1orcv(2)       , ...   % b parameter          %
                   Si0_nam.d1orcv(3)       , ...   % tau parameter        %
```

```
                 NaN                          ...
             ]                            ;

 [                 ...                    % Evaluate NAM at data         %
  dl1_out_cnt     , ...                   % Model count vector (CFU/mL)  %
  no0_src           ...                   % Status return code           %
 ] =                ...
qm4_evl_nam         ...
 (                  ...
  ni0_mod_num     , ...                   % Model number                 %
  dl1_sol_vec     , ...                   % Solution vector              %
  di1_tim_vec     , ...                   % Time vector (day)            %
  di0_ini_cnt     , ...                   % Initial count (CFU/mL)       %
  0                 ...                   % Debug flag                   %
 )                                 ;

dl1_out_cnt = dl1_out_cnt'        ;       % Flip to make x(t;q)          %
                                          % Index of tau (T)             %
p = find( di1_tim_vec >= dl1_sol_vec(3) , 1 , 'first' )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dl1_tm1 =  Si0_nam.d1orcv(1) * di1_tim_vec        ;       % a*t          %
dl1_tm3 =  Si0_nam.d1orcv(3) - di1_tim_vec        ;       % T-t          %
dl1_tm3 =  Si0_nam.d1orcv(2) * dl1_tm3            ;       % b*(T-t)      %
dl1_tm4 =  Si0_nam.d1orcv(1) * Si0_nam.d1orcv(3)  ;       % a*T          %
dl1_tm2 =  dl1_tm3           + dl1_tm4            ;       % b*(T-t)+a*T  %

dl1_e01 = exp( dl1_tm1 )                          ;       % Exponentials %
dl1_e02 = exp( dl1_tm2 )                          ;
                                          % Calculate sensitivities      %
dl1_s11 = zeros( 1 , jl0_num_pts )                ;
dl1_s12 = zeros( 1 , jl0_num_pts )                ;
dl1_s13 = zeros( 1 , jl0_num_pts )                ;

dl0_tm1 = Si0_nam.d1orcv(1) + Si0_nam.d1orcv(2)   ;       % a+b          %
dl1_tm1 = Si0_nam.d1orcv(3) - di1_tim_vec         ;       % T-t          %
                                          % a sensitivity before tau     %
dl1_s11 = ( di0_ini_cnt        *   ...            % I *                  %
            di1_tim_vec       .*   ...            % t *                  %
            dl1_e01            )          ;       % e1                   %
                                          % Zero out after tau           %
dl1_s11(p:end) = 0.0                      ;

                                          % a sensitivity after tau      %
dl1_s21 = ( di0_ini_cnt        *   ...            % I *                  %
            Si0_nam.d1orcv(3)  *   ...            % T *                  %
            dl1_e02            )          ;       % e2                   %
                                          % b sensitivity after tau      %
dl1_s22 = ( di0_ini_cnt        *   ...            % I *                  %
            dl1_tm1           .*   ...            % (T-t) *              %
            dl1_e02            )          ;       % e2                   %
                                          % T sensitivity after tau      %
dl1_s23 = ( di0_ini_cnt        *   ...            % I *                  %
            dl0_tm1            *   ...            % (a+b) *              %
            dl1_e02            )          ;       % e2                   %
                                          % Zero out before tau          %
dl1_s21(1:p-1) = 0.0                      ;
dl1_s22(1:p-1) = 0.0                      ;
dl1_s23(1:p-1) = 0.0                      ;
                                          % Combine sensitivities        %
dl1_s11 = dl1_s11 + dl1_s21               ;
dl1_s12 = dl1_s12 + dl1_s22               ;
dl1_s13 = dl1_s13 + dl1_s23               ;
                                          % Adjust for log10 of counts   %
dl1_s11 = dl1_s11 ./ ( dl1_out_cnt * log( 10.0 ) )        ;
dl1_s12 = dl1_s12 ./ ( dl1_out_cnt * log( 10.0 ) )        ;
dl1_s13 = dl1_s13 ./ ( dl1_out_cnt * log( 10.0 ) )        ;
```

```
                                                % Calc product and gram matrix %
   dl2_prd_mat = [ dl1_s11 ; dl1_s12 ; dl1_s13 ]              ;
   dl2_prd_mat = dl2_prd_mat * dl2_prd_mat'                   ;
                                                % Matrix characteristics       %
   jl0_mat_len = length( dl2_prd_mat )                        ;
   jl0_mat_rnk = rank( dl2_prd_mat )                          ;
   dl0_mat_con = cond( dl2_prd_mat )                          ;
   dl0_mat_det = det( dl2_prd_mat )                           ;

   if ( jl0_mat_rnk < jl0_mat_len )             % Not full rank                %
      dl2_cov_mat = pinv( dl2_prd_mat ) * dl0_dat_var        ;
   else                                         % Full rank                    %
      dl2_cov_mat =  inv( dl2_prd_mat ) * dl0_dat_var        ;
   end
                                                % Standard error               %
   dl1_std_err    = diag( dl2_cov_mat )         ;
   dl1_std_err    = sqrt( dl1_std_err )         ;
   dl1_std_err(4) = NaN                         ;
elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EL )
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %                                                                          %
   %                2-stage Exponential Model With Lag (2EL)                  %
   %                                                                          %
   % For the 2EL model:                                                       %
   %                                                                          %
   %         a  = Si0_nam.d1orcv(1)                                           %
   %         b  = Si0_nam.d1orcv(2)                                           %
   %         T  = Si0_nam.d1orcv(3)                                           %
   %         k1 = Si0_nam.d1orcv(4)                                           %
   %         I  = di0_ini_cnt                                                 %
   %         t  = di1_tim_vec                                                 %
   %         x  = dl1_out_cnt                                                 %
   %                                                                          %
   % x(t;q) and s(t;q) are conditional functions.  If T is the model time     %
   % constant tau and p is the index in di1_tim_vec such that                 %
   %                                                                          %
   %            T > di1_tim_vec(1:p-1) and T <= di1_tim_vec(p:end)            %
   %                                                                          %
   % then x = [x1(1:p-1),x2(p:end)] and s = [s1(1:p-1),s2(p:end)].            %
   %                                                                          %
   % Define:                                                                  %
   %                                                                          %
   %         f1      I / (a+k1)                                               %
   %         f2      I / (a+k1)^2                                             %
   %         f3      I / (b-k1)                                               %
   %         f4      I / (b-k1)^2                                             %
   %                                                                          %
   %         e1      exp(a*T)                                                 %
   %         e2      exp(-k1*T)                                               %
   %         e3      exp(b*(T-t))                                             %
   %         e4      exp(-k1*t)                                               %
   %         e5      exp((b*(T-t))-(k1*T))                                    %
   %         e6      exp(a*t)                                                 %
   %                                                                          %
   % The sensitivity matrix is (2x4) and given by                             %
   %                                                                          %
   %         s11     k1*f1*t*e6 - k1*f2*(e6-e4)                               %
   %         s12     0                                                        %
   %         s13     0                                                        %
   %         s14     -I*t*e4 + k1*f1*t*e4 + a*f2*(e6-e4)                      %
   %         s21     (T*f1*e1 - f2*(e1-e2)) * k1*e3                           %
   %         s22     k1*[f4*(e5-e4) + (f1*(e01-e02) - f3*e2))*(T-t)*e3]       %
   %         s23     k1*f1*(e1-e2)*(a+b)*e3                                   %
   %         s24     q1*e3 + q2*e4                                            %
   %          q1     f1*((e1-e2) + k1*T*e2) - f2*k1*(e1-e2) +                 %
   %                 f3*(k1*T-1)*e2 - f4*k1*e2                                %
   %          q2     f4*k1 - f3*(k1*t-1) - I*t                                %
   %                                                                          %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                          % Calculate data variance     %
dl0_tm1     = double( jl0_num_pts - jl0_rcv_len ) ;
dl0_dat_var = Si0_nam.d0mnsr / dl0_tm1         ;
                                          % Set solution vector         %
dl1_sol_vec = [                             ...
                  Si0_nam.d1orcv(1)       , ...   % a parameter         %
                  Si0_nam.d1orcv(2)       , ...   % b parameter         %
                  Si0_nam.d1orcv(3)       , ...   % tau parameter       %
                  Si0_nam.d1orcv(4)         ...   % k1 parameter        %
              ]                           ;

 [                  ...                   % Evaluate NAM at data        %
  dl1_out_cnt     , ...                   % Model count vector (CFU/mL)  %
  no0_src           ...                   % Status return code          %
 ] =                ...
qm4_evl_nam         ...
 (                  ...
  ni0_mod_num     , ...                   % Model number                %
  dl1_sol_vec     , ...                   % Solution vector             %
  di1_tim_vec     , ...                   % Time vector (day)           %
  di0_ini_cnt     , ...                   % Initial count (CFU/mL)      %
  0                 ...                   % Debug flag                  %
 )                                ;

dl1_out_cnt = dl1_out_cnt'        ;       % Flip to make x(t;q)         %
                                          % Index of tau (T)            %
p = find( di1_tim_vec >= dl1_sol_vec(3) , 1 , 'first' )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dl0_tm1 =  Si0_nam.d1orcv(1) + Si0_nam.d1orcv(4)  ;       % a+k1          %
dl0_tm2 =  Si0_nam.d1orcv(2) - Si0_nam.d1orcv(4)  ;       % b-k1          %
dl0_f01 =  di0_ini_cnt       / dl0_tm1            ;       % I/(a+k1)      %
dl0_f02 =  dl0_f01           / dl0_tm1            ;       % I/(a+k1)^2    %
dl0_f03 =  di0_ini_cnt       / dl0_tm2            ;       % I/(b-k1)      %
dl0_f04 =  dl0_f03           / dl0_tm2            ;       % I/(b-k1)^2    %

dl1_tm1 =  Si0_nam.d1orcv(1) * Si0_nam.d1orcv(3)  ;       % a*T           %
dl1_tm2 = -Si0_nam.d1orcv(4) * Si0_nam.d1orcv(3)  ;       % -k1*T         %
dl1_tm3 =  Si0_nam.d1orcv(3) - di1_tim_vec        ;       % T-t           %
dl1_tm3 =  Si0_nam.d1orcv(2) * dl1_tm3            ;       % b*(T-t)       %
dl1_tm4 = -Si0_nam.d1orcv(4) * di1_tim_vec        ;       % -k1*t         %
dl1_tm5 =  dl1_tm3           + dl1_tm2            ;       % b*(T-t)-k1*T %
dl1_tm6 =  Si0_nam.d1orcv(1) * di1_tim_vec        ;       % a*t           %

dl1_e01 = exp( dl1_tm1 )                          ;       % Exponentials %
dl1_e02 = exp( dl1_tm2 )                          ;
dl1_e03 = exp( dl1_tm3 )                          ;
dl1_e04 = exp( dl1_tm4 )                          ;
dl1_e05 = exp( dl1_tm5 )                          ;
dl1_e06 = exp( dl1_tm6 )                          ;
                                          % Calculate x1 sensitivities   %
dl1_s11 = zeros( 1 , jl0_num_pts )                ;
dl1_s12 = zeros( 1 , jl0_num_pts )                ;
dl1_s13 = zeros( 1 , jl0_num_pts )                ;
dl1_s14 = zeros( 1 , jl0_num_pts )                ;

dl1_tm1 = di1_tim_vec .* dl1_e06          ;       % t * e^(a*t)         %
dl1_tm2 = dl1_e06      - dl1_e04          ;       % e6 - e4             %
dl1_tm3 = di1_tim_vec .* dl1_e04          ;       % t * e^(-k1*t)       %
                                          % a sensitivity before tau     %
dl1_s11 = ( Si0_nam.d1orcv(4)  *   ...            % k1 *                %
            dl0_f01           .*   ...            % f1 *                %
            dl1_tm1            ) - ...            % t * e^(a*t)         %
          ( Si0_nam.d1orcv(4)  *   ...            % k1 *                %
            dl0_f02           .*   ...            % f2 *                %
            dl1_tm2            )          ;       % e6-e4               %
```

```
                                          % k1 sensitivity before tau    %
dl1_s14 = (-di0_ini_cnt       .*   ...            % -I *                 %
            dl1_tm3            ) + ...            % t * e^(-k1*t)        %
          ( Si0_nam.d1orcv(4)  *   ...            % k1 *                 %
            dl0_f01           .*   ...            % f1                   %
            dl1_tm3            ) + ...            % t * e^(-k1*t)        %
          ( Si0_nam.d1orcv(1)  *   ...            % a *                  %
            dl0_f02           .*   ...            % f2 *                 %
            dl1_tm2            )          ;       % e6-e4                 %
                                          % Zero out after tau           %
dl1_s11(p:end) = 0.0                      ;
dl1_s14(p:end) = 0.0                      ;
                                          % Calculate x2 sensitivities   %
dl1_s21 = zeros( 1 , jl0_num_pts )                ;
dl1_s22 = zeros( 1 , jl0_num_pts )                ;
dl1_s23 = zeros( 1 , jl0_num_pts )                ;
dl1_s24 = zeros( 1 , jl0_num_pts )                ;

dl1_tm1 = dl1_e01           - dl1_e02             ;       % e1 - e2      %
dl1_tm2 = dl1_e05           - dl1_e04             ;       % e5 - e4      %
dl1_tm3 = Si0_nam.d1orcv(3) - di1_tim_vec         ;       % T  - t       %
dl1_tm4 = Si0_nam.d1orcv(1) + Si0_nam.d1orcv(2)   ;       % a + b        %
dl1_tm5 = Si0_nam.d1orcv(3) * Si0_nam.d1orcv(4)   ;       % T * k1       %
dl1_tm5 = dl1_tm5           - 1.0                 ;       % (T*k1) - 1   %
dl1_tm6 = di1_tim_vec       * Si0_nam.d1orcv(4)   ;       % t * k1       %
dl1_tm6 = dl1_tm6           - 1.0                 ;       % (t*k1) - 1   %
                                          % a sensitivity after tau      %
dl1_s21 = ( Si0_nam.d1orcv(3)  *   ...            % T *                  %
            dl0_f01           .*   ...            % f1 *                 %
            dl1_e01            ) - ...            % e^(a*T)              %
          ( dl0_f02           .*   ...            % f2 *                 %
            dl1_tm1            )          ;       % e1-e2                 %
dl1_s21 = ( dl1_s21           .*   ...
            Si0_nam.d1orcv(4) .*   ...            % k1 *                 %
            dl1_e03            )          ;       % e^(b*(T-t))          %
                                          % b sensitivity after tau      %
dl1_s22 = ( dl0_f01            *   ...            % f1 *                 %
            dl1_tm1            ) - ...            % e1-e2                %
          ( dl0_f03            *   ...            % f3 *                 %
            dl1_e02            )          ;       % e2                   %
dl1_s22 = ( dl1_s22           .*   ...
            dl1_tm3           .*   ...            % T-t *                %
            dl1_e03            ) + ...            % e3                   %
          ( dl0_f04           .*   ...            % f4 *                 %
            dl1_tm2            )          ;       % e5-e4                %
dl1_s22 = ( dl1_s22            *   ...
            Si0_nam.d1orcv(4)  )          ;       % k1                   %
                                          % tau sensitivity after tau    %
dl1_s23 = ( Si0_nam.d1orcv(4)  *   ...            % k1 *                 %
            dl0_f01           .*   ...            % f1 *                 %
            dl1_tm1           .*   ...            % e1-e2                %
            dl1_tm4           .*   ...            % a+b                  %
            dl1_e03            )          ;       % e3                   %
                                          % k1 sensitivity after tau     %
dl1_s24 = ( dl1_tm1            ) + ...            % e1-e2                %
          ( Si0_nam.d1orcv(4)  *   ...            % k1 *                 %
            Si0_nam.d1orcv(3) .*   ...            % T *                  %
            dl1_e02            )          ;       % e2                   %
dl1_tm7 = ( dl1_s24            *   ...
            dl0_f01            ) - ...            % f1                   %
          ( dl0_f02            *   ...            % f2 *                 %
            Si0_nam.d1orcv(4) .*   ...            % k1                   %
            dl1_tm1            ) + ...            % e1-e2                %
          ( dl0_f03            *   ...            % f3                   %
            dl1_tm5           .*   ...            % (T*k1) - 1           %
            dl1_e02            ) - ...            % e2                   %
          ( dl0_f04            *   ...            % f4 *                 %
            Si0_nam.d1orcv(4) .*   ...            % k1 *                 %
```

```
                dl1_e02            )          ;       % e2                    %
   dl1_tm8 = ( dl0_f04            *   ...             % f4 *                  %
               Si0_nam.d1orcv(4)  ) - ...             % k1                    %
             ( dl0_f03            *   ...             % f3 *                  %
               dl1_tm6            ) - ...             % (t*k1) - 1            %
             ( di0_ini_cnt       .*   ...             % I *                   %
               di1_tim_vec        )          ;       % t                     %
   dl1_s24 = ( dl1_tm7           .*   ...
               dl1_e03            ) + ...             % e3                    %
             ( dl1_tm8           .*   ...
               dl1_e04            )          ;       % e4                    %
                                             % Zero out before tau           %
   dl1_s21(1:p-1) = 0.0                      ;
   dl1_s22(1:p-1) = 0.0                      ;
   dl1_s23(1:p-1) = 0.0                      ;
   dl1_s24(1:p-1) = 0.0                      ;
                                             % Combine sensitivities         %
   dl1_s11 = dl1_s11 + dl1_s21               ;
   dl1_s12 = dl1_s12 + dl1_s22               ;
   dl1_s13 = dl1_s13 + dl1_s23               ;
   dl1_s14 = dl1_s14 + dl1_s24               ;
                                             % Adjust for log10 of counts    %
   dl1_s11 = dl1_s11 ./ ( dl1_out_cnt * log( 10.0 ) )        ;
   dl1_s12 = dl1_s12 ./ ( dl1_out_cnt * log( 10.0 ) )        ;
   dl1_s13 = dl1_s13 ./ ( dl1_out_cnt * log( 10.0 ) )        ;
   dl1_s14 = dl1_s14 ./ ( dl1_out_cnt * log( 10.0 ) )        ;
                                             % Calc product and gram matrix %
   dl2_prd_mat = [ dl1_s11 ; dl1_s12 ; dl1_s13 ; dl1_s14 ]   ;
   dl2_prd_mat = dl2_prd_mat * dl2_prd_mat'                  ;
                                             % Matrix characteristics        %
   jl0_mat_len = length( dl2_prd_mat )                       ;
   jl0_mat_rnk = rank( dl2_prd_mat )                         ;
   dl0_mat_con = cond( dl2_prd_mat )                         ;
   dl0_mat_det = det( dl2_prd_mat )                          ;

   if ( jl0_mat_rnk < jl0_mat_len )          % Not full rank                 %
      dl2_cov_mat = pinv( dl2_prd_mat ) * dl0_dat_var        ;
   else                                      % Full rank                     %
      dl2_cov_mat =  inv( dl2_prd_mat ) * dl0_dat_var        ;
   end

   dl1_std_err = diag( dl2_cov_mat ) ;       % Standard error                %
   dl1_std_err = sqrt( dl1_std_err ) ;
else
   return                            ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Calculate AICc value          %
dl0_tm1     = Si0_nam.d0mnsr / double( jl0_num_pts ) ; % SSR / n             %
dl0_tm1     = log( dl0_tm1 ) * double( jl0_num_pts ) ; % n * ln( SSR / n )   %

dl0_tm2     = double( jl0_rcv_len )                  ; % k                   %
dl0_tm2     = dl0_tm2        + 1.0                   ; % k + 1               %
dl0_tm2     = dl0_tm2        * 2.0                   ; % 2 * (k + 1)         %

dl0_tm3     = double( jl0_num_pts - jl0_rcv_len )    ; % n - k               %
dl0_tm3     = dl0_tm3        - 2.0                   ; % n - k - 2           %
dl0_tm3     = 2.0                        * ...         % 2*                  %
              double( jl0_rcv_len + 1 ) * ...          % (k+1)*              %
              double( jl0_rcv_len + 2 ) / ...          % (k+2)/              %
              dl0_tm3                                ; % (n-k-2)             %

dl0_aic_val = dl0_tm1        + dl0_tm2      + dl0_tm3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Set output values            %
So0_nam.d1serr = dl1_std_err         ;       % Update output structure      %
```

```
   So0_nam.d0dvar = dl0_dat_var         ;
   So0_nam.d0aicc = dl0_aic_val         ;
   So0_nam.n0nump = jl0_num_pts         ;
   So0_nam.j0mlen = jl0_mat_len         ;
   So0_nam.j0mrnk = jl0_mat_rnk         ;
   So0_nam.d0mcon = dl0_mat_con         ;
   So0_nam.d0mdet = dl0_mat_det         ;
   So0_nam.d2covm = dl2_cov_mat         ;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Debug                        %
   if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
      if ( ni0_mod_num == Sc0_qgd.J_NAM1EX )
         fprintf( jl0_dfi , '1-stage Exponential\n' )                  ;
         fprintf( jl0_dfi , ' Std error a:   \t%e\n' , dl1_std_err(1) ) ;
      elseif ( ni0_mod_num == Sc0_qgd.J_NAM1EL )
         fprintf( jl0_dfi , '1-stage Exponential with Lag\n' )         ;
         fprintf( jl0_dfi , ' Std error a:   \t%e\n' , dl1_std_err(1) ) ;
         fprintf( jl0_dfi , ' Std error k1:  \t%e\n' , dl1_std_err(2) ) ;
      elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EX )
         fprintf( jl0_dfi , '2-stage Exponential\n' )                  ;
         fprintf( jl0_dfi , ' Std error a:   \t%e\n' , dl1_std_err(1) ) ;
         fprintf( jl0_dfi , ' Std error b:   \t%e\n' , dl1_std_err(2) ) ;
         fprintf( jl0_dfi , ' Std error T:   \t%e\n' , dl1_std_err(3) ) ;
      elseif ( ni0_mod_num == Sc0_qgd.J_NAM2EL )
         fprintf( jl0_dfi , '2-stage Exponential with Lag\n' )         ;
         fprintf( jl0_dfi , ' Std error a:   \t%e\n' , dl1_std_err(1) ) ;
         fprintf( jl0_dfi , ' Std error b:   \t%e\n' , dl1_std_err(2) ) ;
         fprintf( jl0_dfi , ' Std error T:   \t%e\n' , dl1_std_err(3) ) ;
         fprintf( jl0_dfi , ' Std error k1:  \t%e\n' , dl1_std_err(4) ) ;
      end

      fprintf( jl0_dfi , ' Number of points:  %d\n' , jl0_num_pts )     ;
      fprintf( jl0_dfi , ' AICc:              %e\n' , dl0_aic_val )     ;
      fprintf( jl0_dfi , ' Data variance:     %e\n' , dl0_dat_var )     ;
      fprintf( jl0_dfi , ' Matrix length:     %d\n' , jl0_mat_len )     ;
      fprintf( jl0_dfi , ' Matrix rank:       %d\n' , jl0_mat_rnk )     ;
      fprintf( jl0_dfi , ' Condition number:  %e\n' , dl0_mat_con )     ;
      fprintf( jl0_dfi , ' Determinant:       %e\n' , dl0_mat_det )     ;
      fprintf( jl0_dfi , '\n' )                                         ;
   end
end                                          % END OF FUNCTION              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                              Function qm4_dor_cnv                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%       Start date:           19-JUL-2016                                      %
%       Written by:           Applied Mathematics, Inc.                        %
%       Version number:       V1.0.0                                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                Revision History                              %
%                                                                              %
% ------------- -------- -------------- ------------------------------------- %
% Revision No.    Date     Programmer               Description                %
% ------------- -------- -------------- ------------------------------------- %
% V1.0.0.0      07/19/16 T.E. Wood      Initial version.                       %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Description                                %
%                                                                              %
%       This function is used for death-only parameter reduction.  It converts %
% a 1-stage exponential  with lag (1EL) solution  into  exponential values and %
% covariance that  can be used for  uncertainty propagation.  The 1EL solution %
% has parameters a and k1 where a is negative and k1 is positive.  The conver- %
% ted solution has parameters z1 and z2 where                                  %
%                                                                              %
%               d  = -a                                                        %
%               z1 = ln( d )                                                   %
%               z2 = ln( k1 )                                                  %
%                                                                              %
% The covariance matrix is formed from sensitivities for z1 and z2.            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                                   Parameters                                 %
%                                                                              %
%       Parameter       Type            Description                            %
%                                                                              %
% (I)   Si0_nam         struct          NAM structure that stores the output   %
%                                       from a fit to measured count data.     %
%                                                                              %
% (I)   di1_tim_vec     double(1,n)     Vector of times in days.               %
%                                                                              %
% (I)   di0_ini_cnt     double          Initial count for M in CFU/mL.         %
%                                                                              %
% (I)   ji0_dbg_flg     int32           An optional argument for a debug flag. %
%                                       Defined in qcm_ini_prm.  If blank or   %
%                                       empty, set to no debug.                %
%                                       = .J_DBGLV0 No debug info is shown.    %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
% (I)   Si0_ocs         struct          An optional arg for output control.    %
%                                       Parameters defined in qcm_ini_prm.  If %
%                                       blank or empty, set to no debug.       %
%                                       Overrides ji0_dbg_flg.                 %
%                                                                              %
% (O)   So0_nam         struct          Updated NAM structure with standard    %
%                                       error values and additional statistics.%
%                                                                              %
% (O)   no0_src         uint32          Status return code.                    %
%                                       = .S_SUCCESS  Successful completion.   %
%                                       = Other - see qcm_ini_prm.             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
```

```
%                              Structure Fields                              %
%                                                                            %
% Si0_nam/So0_nam:                                                           %
%                                                                            %
%       .d1mink         double(4)       Minimum rate constant vector.        %
%       .d1maxk         double(4)       Maximum rate constant vector.        %
%       .d1ircv         double(4)       Initial rate constant vector.        %
%       .d1rtim         double(n)       Return time from model fits.         %
%       .d1icnt         double(4)       Initial count in CFU/mL.             %
%       .d1rcnt         double(n)       Return counts from model fits.       %
%       .d2cmat         double(n,4)     Return count matrix.                 %
%       .d1orcv         double(n)       Optimal rate constant vector.        %
%       .d0mnsr         double          Minimum SSR value of fit.            %
%       .d1lcnt         double(n)       Log of return counts from model fits. %
%       .d2lcnt         double(n,4)     Log of return count matrix.          %
%       .d1serr         double(n)       Standard error vector.               %
%       .d0dvar         double          Data variance.                       %
%       .d0aicc         double          Corrected AIC value (AICc).          %
%       .n0nump         uint32          Number of points.                    %
%       .j0mlen         int32           Matrix length.                       %
%       .j0mrnk         int32           Matrix rank.                         %
%       .d0mcon         double          Matrix condition number.             %
%       .d0mdet         double          Matrix determinant.                  %
%       .d2covm         double(n,k)     Covariance matrix                    %
%                                                                            %
% Si0_ocs:                                                                   %
%                                                                            %
%       .j0dbgf         int32           Debug flag (.J_DBGLV0 to .J_DBGLV4)  %
%       .j0dfid         int32           Debug output file ID number          %
%       .c1dfnm         char            Debug filename                       %
%       .j0pltf         int32           Plot flag (true or false)            %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Death-only Reduction Conv   %
function                  ...
    [                     ...
        So0_nam         , ...                   % Output NAM structure        %
        no0_src           ...                   % Status return code          %
    ] =                   ...
   qm4_dor_cnv            ...
    (                     ...
        Si0_nam         , ...                   % Input NAM structure         %
        di1_tim_vec     , ...                   % Time vector (day)           %
        di0_ini_cnt     , ...                   % Initial count (CFU/mL)      %
        ji0_dbg_flg     , ...                   % Debug flag                  %
        Si0_ocs           ...                   % Output control structure    %
    )
                                                % PARAMETER DECLARATIONS      %
   So0_nam              = []            ;
   no0_src              = uint32( 0 )   ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % MATLAB INITIALIZATION       %
   format long                          ;       % Set default format to long  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOAD GLOBALS                %
   persistent Sc0_qgd                   ;       % Global QCM data             %

   if ( ~isfield( Sc0_qgd , 'L_EXIST' ) )       % Not initialized             %
      Sc0_qgd = qcm_ini_prm()           ;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % LOCAL DECLARATIONS          %
   dl2_prd_mat  = double( 0.0 )         ;       % Product matrix              %
   dl2_cov_mat  = double( 0.0 )         ;       % Covariance matrix           %
```

```
   dl1_sol_vec  = double( 0.0 )         ;       % Solution vector             %
   dl1_out_cnt  = double( 0.0 )         ;       % Output count                %
   dl1_std_err  = double( 0.0 )         ;       % Standard error              %
   dl1_s11      = double( 0.0 )         ;       % Sensitivity matrix values   %
   dl1_s12      = double( 0.0 )         ;
   dl1_s13      = double( 0.0 )         ;
   dl1_s14      = double( 0.0 )         ;
   dl1_s21      = double( 0.0 )         ;
   dl1_s22      = double( 0.0 )         ;
   dl1_s23      = double( 0.0 )         ;
   dl1_s24      = double( 0.0 )         ;
   dl1_e01      = double( 0.0 )         ;       % Exponential vectors         %
   dl1_e02      = double( 0.0 )         ;
   dl1_e03      = double( 0.0 )         ;
   dl1_e04      = double( 0.0 )         ;
   dl1_e05      = double( 0.0 )         ;
   dl1_e06      = double( 0.0 )         ;
   dl1_f01      = double( 0.0 )         ;       % Function vectors            %
   dl1_f02      = double( 0.0 )         ;
   dl1_f03      = double( 0.0 )         ;
   dl1_tm1      = double( 0.0 )         ;       % Temporary vectors           %
   dl1_tm2      = double( 0.0 )         ;
   dl1_tm3      = double( 0.0 )         ;
   dl1_tm4      = double( 0.0 )         ;
   dl1_tm5      = double( 0.0 )         ;
   dl1_tm6      = double( 0.0 )         ;
   dl1_tm7      = double( 0.0 )         ;
   dl1_tm8      = double( 0.0 )         ;

   dl0_dat_var  = double( 0.0 )         ;       % Data variance               %
   dl0_mat_con  = double( 0.0 )         ;       % Matrix condition number     %
   dl0_mat_det  = double( 0.0 )         ;       % Matrix determinant          %
   dl0_aic_val  = double( 0.0 )         ;       % AIC value                   %
   dl0_e01      = double( 0.0 )         ;       % Exponential scalars         %
   dl0_e02      = double( 0.0 )         ;
   dl0_f01      = double( 0.0 )         ;       % Fraction values             %
   dl0_f02      = double( 0.0 )         ;
   dl0_f03      = double( 0.0 )         ;
   dl0_f04      = double( 0.0 )         ;
   dl0_tm1      = double( 0.0 )         ;       % Temporary variables         %
   dl0_tm2      = double( 0.0 )         ;
   dl0_tm3      = double( 0.0 )         ;

   p            = uint32( 0 )           ;       % Counters/indexes            %

   jl0_num_pts  = int32( 0 )            ;       % Number of points            %
   jl0_mat_len  = int32( 0 )            ;       % Matrix length               %
   jl0_mat_rnk  = int32( 0 )            ;       % Matrix rank                 %
   jl0_rcv_len  = int32( 0 )            ;       % Rate constant vector length %
   jl0_dbg_flg  = int32( 0 )            ;       % Local debug flag            %
   jl0_dfi      = int32( 0 )            ;       % Debug file ID number        %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % BEGINNING OF FUNCTION       %
                                                % Check inputs                %
   if ( ~exist('ji0_dbg_flg','var') || ...      % No debug flag               %
        isempty(ji0_dbg_flg)        )
      jl0_dbg_flg = Sc0_qgd.J_DBGLV0    ;
      jl0_dfi     = 1                   ;
   elseif ( ji0_dbg_flg < Sc0_qgd.J_DBGLV0 )    % Debug must be non-negative  %
      no0_src = Sc0_qgd.E_INVARG        ;       % Set status return code      %
      return                            ;       % Exit function               %
   elseif ( ji0_dbg_flg > Sc0_qgd.J_DBGLV0 )
      jl0_dbg_flg = ji0_dbg_flg                 ;
      jl0_dfi     = 1                           ;
      fprintf( 1 , '*** qm4_dor_cnv ***\n' )    ;
   end
```

```
if ( ~exist( 'Si0_ocs' , 'var' ) || ...      % No output control structure  %
     isempty( Si0_ocs )          )
   jl0_dbg_flg = Sc0_qgd.J_DBGLV0                     ; % Local debug flag   %
   jl0_dfi     = 1                                    ; % Debug FID          %
elseif ( Si0_ocs.j0dbgf < Sc0_qgd.J_DBGLV0 ) % Debug must be non-negative   %
   no0_src = Sc0_qgd.E_INVARG                         ; % Set status code    %
   return                                             ; % Return             %
elseif ( Si0_ocs.j0dbgf > Sc0_qgd.J_DBGLV0 ) % Valid debug                  %
   jl0_dbg_flg = Si0_ocs.j0dbgf                       ; % Local debug flag   %
   jl0_dfi     = Si0_ocs.j0dfid                       ; % Debug FID          %
   fprintf( jl0_dfi , '+++ qm4_dor_cnv +++\n' )       ; % Display function   %
end

jl0_rcv_len = 2                      ;       % Number of rate constants     %

if ( ( ~isvector( Si0_nam.d1orcv ) ) || ...  % Must be vector               %
     ( ~isvector( di1_tim_vec    ) ) )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                             % Must be correct length       %
if ( length( Si0_nam.d1orcv ) ~= jl0_rcv_len )
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                             % Solution must be good        %
if ( ( Si0_nam.d1orcv(1) >= 0 ) || ...       % a < 0                        %
     ( Si0_nam.d1orcv(2) <= 0 ) )            % k1 > 0                       %
   no0_src = Sc0_qgd.E_INCARG        ;
   return                            ;
end
                                             % Init structure and variables %
So0_nam        = Si0_nam                     ;
So0_nam.d1serr = []                          ;
So0_nam.d0aicc = []                          ;
jl0_num_pts    = length( di1_tim_vec )       ;

                                             % Run evaluation at least once %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%                   1-stage Exponential Model With Lag                         %
%                                                                              %
%            a  = di1_sol_vec(1)                                               %
%            k1 = di1_sol_vec(2)                                               %
%            I  = di0_ini_cnt                                                  %
%            t  = di1_tim_vec                                                  %
%            x  = dl1_out_cnt                                                  %
%                                                                              %
% Define:                                                                      %
%                                                                              %
%            f1      I*k1 / (a+k1)                                             %
%            f2      I*k1 / (a+k1)^2                                           %
%            f3      I*a  / (a+k1)^2                                           %
%                                                                              %
%            e1      exp(a*t)                                                  %
%            e2      exp(-k1*t)                                                %
%                                                                              %
% The sensitivity matrix is (1x2) and given by                                 %
%                                                                              %
%            s1      f1*t*e1 - f2*(e1-e2)                                      %
%            s2      -I*t*e2 + f1*t*e2 + f3*(e1-e2)                            %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                             % Calculate data variance     %
dl0_tm1     = double( jl0_num_pts - jl0_rcv_len )    ;
dl0_dat_var = Si0_nam.d0mnsr / dl0_tm1               ;
                                             % Set solution vector         %
dl1_sol_vec = [                                ...
```

```
                    Si0_nam.d1orcv(1)       , ...   % a parameter          %
                    Si0_nam.d1orcv(2)       , ...   % b parameter          %
                    NaN                     , ...
                    NaN                       ...
              ]                             ;

 [                    ...                   % Evaluate NAM at data         %
     dl1_out_cnt    , ...                   % Model count vector (CFU/mL)  %
     no0_src          ...                   % Status return code           %
 ] =                  ...
qm4_evl_nam           ...
 (                    ...
     Sc0_qgd.J_NAM1EL, ...                   % Model number                 %
     dl1_sol_vec    , ...                   % Solution vector              %
     di1_tim_vec    , ...                   % Time vector (day)            %
     di0_ini_cnt    , ...                   % Initial count (CFU/mL)       %
     0                ...                   % Debug flag                   %
 )                                  ;

dl1_out_cnt = dl1_out_cnt'          ;       % Flip to make x(t;q)          %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dl0_tm1 =  Si0_nam.d1orcv(1) + Si0_nam.d1orcv(2)     ;       % a+k1         %
dl0_tm2 =  di0_ini_cnt       * Si0_nam.d1orcv(2)     ;       % I*k1         %
dl0_tm3 =  di0_ini_cnt       * Si0_nam.d1orcv(1)     ;       % I*a          %
dl0_f01 =  dl0_tm2           / dl0_tm1               ;       % I*k1/(a+k1)  %
dl0_f02 =  dl0_f01           / dl0_tm1               ;       % I*k1/(a+k1)^2%
dl0_f03 =  dl0_tm3           / dl0_tm1               ;       % I*a/(a+k1)   %
dl0_f03 =  dl0_f03           / dl0_tm1               ;       % I*a/(a+k1)^2 %

dl1_tm1 =  Si0_nam.d1orcv(1) * di1_tim_vec           ;       % a*t          %
dl1_tm2 = -Si0_nam.d1orcv(2) * di1_tim_vec           ;       % -k1*t        %

dl1_e01 = exp( dl1_tm1 )                             ;       % Exponentials %
dl1_e02 = exp( dl1_tm2 )                             ;
                                                     % Calculate sensitivities      %
dl1_s11 = zeros( 1 , jl0_num_pts )                   ;
dl1_s12 = zeros( 1 , jl0_num_pts )                   ;

dl1_tm1 = dl1_e01      - dl1_e02                     ;       % e1-e2        %
                                                     % a sensitivity                %
dl1_s11 = ( dl0_f01            *   ...               % f1 *                 %
            di1_tim_vec       .*   ...               % t *                  %
            dl1_e01            ) - ...               % e^(a*t)              %
          ( dl0_f02            *   ...               % f2 *                 %
            dl1_tm1            )                     ;       % e1-e2                %
                                                     % k1 sensitivity               %
dl1_s12 = (-di0_ini_cnt        *   ...               % -I *                 %
            di1_tim_vec       .*   ...               % t *                  %
            dl1_e02            ) + ...               % e^(-k1*t)            %
          ( dl0_f01            *   ...               % f1 *                 %
            di1_tim_vec       .*   ...               % t *                  %
            dl1_e02            ) + ...               % e^(-k1*t)            %
          ( dl0_f03            *   ...               % f3 *                 %
            dl1_tm1            )                     ;       % e1-e2                %
                                                     % Adjust for log10 of counts   %
dl1_s11 = dl1_s11 ./ ( dl1_out_cnt * log( 10.0 ) )   ;
dl1_s12 = dl1_s12 ./ ( dl1_out_cnt * log( 10.0 ) )   ;
                                                     % Calc product and gram matrix %
dl2_prd_mat = [ dl1_s11 ; dl1_s12 ]                  ;
dl2_prd_mat = dl2_prd_mat * dl2_prd_mat'             ;
                                                     % Matrix characteristics       %
jl0_mat_len = length( dl2_prd_mat )                  ;
jl0_mat_rnk = rank( dl2_prd_mat )                    ;
dl0_mat_con = cond( dl2_prd_mat )                    ;
dl0_mat_det = det( dl2_prd_mat )                     ;
```

```
if ( jl0_mat_rnk < jl0_mat_len )            % Not full rank               %
   dl2_cov_mat = pinv( dl2_prd_mat ) * dl0_dat_var   ;
else                                  % Full rank                   %
   dl2_cov_mat =  inv( dl2_prd_mat ) * dl0_dat_var   ;
end
                                            % Standard error              %
dl1_std_err    = diag( dl2_cov_mat ) ;
dl1_std_err    = sqrt( dl1_std_err ) ;
dl1_std_err(3) = NaN                 ;
dl1_std_err(4) = NaN                 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                            % Calculate AICc value        %
dl0_tm1     = Si0_nam.d0mnsr / double( jl0_num_pts ) ; % SSR / n          %
dl0_tm1     = log( dl0_tm1 ) * double( jl0_num_pts ) ; % n * ln( SSR / n ) %

dl0_tm2     = double( jl0_rcv_len )                  ; % k                %
dl0_tm2     = dl0_tm2       + 1.0                    ; % k + 1            %
dl0_tm2     = dl0_tm2       * 2.0                    ; % 2 * (k + 1)      %

dl0_tm3     = double( jl0_num_pts - jl0_rcv_len )    ; % n - k            %
dl0_tm3     = dl0_tm3       - 2.0                    ; % n - k - 2        %
dl0_tm3     = 2.0                 * ...                % 2*               %
              double( jl0_rcv_len + 1 ) * ...          % (k+1)*           %
              double( jl0_rcv_len + 2 ) / ...          % (k+2)/           %
              dl0_tm3                                ; % (n-k-2)          %

dl0_aic_val = dl0_tm1       + dl0_tm2      + dl0_tm3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                            % Set output values           %
So0_nam.d1serr = dl1_std_err         ;      % Update output structure     %
So0_nam.d0dvar = dl0_dat_var         ;
So0_nam.d0aicc = dl0_aic_val         ;
So0_nam.n0nump = jl0_num_pts         ;
So0_nam.j0mlen = jl0_mat_len         ;
So0_nam.j0mrnk = jl0_mat_rnk         ;
So0_nam.d0mcon = dl0_mat_con         ;
So0_nam.d0mdet = dl0_mat_det         ;
So0_nam.d2covm = dl2_cov_mat         ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                         %
%    z1 = ln( d ) = ln( -a )                                              %
%    z2 = ln( k1 )                                                        %
%                                                                         %
%    e01 = exp( z1 )                                                      %
%    e02 = exp( z2 )                                                      %
%    e11 = -e01 * t                                                       %
%    e12 = -e02 * t                                                       %
%                                                                         %
%    f01 = I*exp(z1+z2)/(e02-e01)^2                                       %
%    f11 = exp(e11) - exp(e12)                                            %
%    f12 = (exp(z1+e11) - exp(z2+e11)) .* t                               %
%    f13 = (exp(z1+e12) - exp(z2+e12)) .* t                               %
%                                                                         %
%    s11 =  f01 * [ f11 + f12 ]                                           %
%    s12 = -f01 * [ f11 + f13 ]                                           %
%                                                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                            % Convert to logarithms       %
So0_nam.d1orcv(1) = log( -So0_nam.d1orcv(1) )        ;
So0_nam.d1orcv(2) = log(  So0_nam.d1orcv(2) )        ;

dl0_e01 = exp( So0_nam.d1orcv(1) )                   ; % e^(z1)           %
dl0_e02 = exp( So0_nam.d1orcv(2) )                   ; % e^(z2)           %

dl1_e01 = -dl0_e01           *  di1_tim_vec          ; % -e^(z1)*t        %
```

```
dl1_e02 = -dl0_e02           *  di1_tim_vec          ; % -e^(z2)*t          %

dl0_tm1 =  dl0_e02           -  dl0_e01              ; % e^(z2)-e^(z1)      %
dl0_tm1 =  dl0_tm1^2                                 ; % (e^(z2)-e^(z1))^2  %
dl0_tm2 =  dl0_e01           *  dl0_e02              ; % e^(z1+z2)          %
dl0_tm2 =  di0_ini_cnt       *  dl0_tm2              ; % I*e^(z1+z2)        %
dl0_f01 =  dl0_tm2           /  dl0_tm1              ; % f01                 %

dl1_f01 =  exp( dl1_e01 )    -  exp( dl1_e02 )       ; % e^(e11) - e^(e12)  %

dl1_tm1 =  So0_nam.d1orcv(1) +  dl1_e01              ; % t1=z1 - e^(z1)*t   %
dl1_tm2 =  So0_nam.d1orcv(2) +  dl1_e01              ; % t2=z2 - e^(z1)*t   %
dl1_tm3 =  exp( dl1_tm1 )    -  exp( dl1_tm2 )       ; % e^t1 - e^t2        %
dl1_f02 =  dl1_tm3           .* di1_tim_vec          ; % f12                 %

dl1_tm1 =  So0_nam.d1orcv(1) +  dl1_e02              ; % t1=z1 - e^(z2)*t   %
dl1_tm2 =  So0_nam.d1orcv(2) +  dl1_e02              ; % t2=z2 - e^(z2)*t   %
dl1_tm3 =  exp( dl1_tm1 )    -  exp( dl1_tm2 )       ; % e^t1 - e^t2        %
dl1_f03 =  dl1_tm3           .* di1_tim_vec          ; % f13                 %

dl1_tm1 =  dl1_f01           +  dl1_f02              ; % f11 + f12          %
dl1_s11 =  dl0_f01           *  dl1_tm1              ; % f01 * [f11+f12]    %

dl1_tm2 =  dl1_f01           +  dl1_f03              ; % f11 + f13          %
dl1_s12 = -dl0_f01           *  dl1_tm2              ; % -f01 * [f11+f13]   %
                                                % Adjust for log10 of counts   %
dl1_s11 = dl1_s11 ./ ( dl1_out_cnt * log( 10.0 ) )   ;
dl1_s12 = dl1_s12 ./ ( dl1_out_cnt * log( 10.0 ) )   ;
                                                % Calc product and gram matrix %
dl2_prd_mat = [ dl1_s11 ; dl1_s12 ]                  ;
dl2_prd_mat = dl2_prd_mat * dl2_prd_mat'             ;
                                                % Matrix characteristics       %
jl0_mat_len = length( dl2_prd_mat )                  ;
jl0_mat_rnk = rank( dl2_prd_mat )                    ;
dl0_mat_con = cond( dl2_prd_mat )                    ;
dl0_mat_det = det( dl2_prd_mat )                     ;

if ( jl0_mat_rnk < jl0_mat_len )                % Not full rank                %
   dl2_cov_mat = pinv( dl2_prd_mat ) * dl0_dat_var   ;
else                                          % Full rank                   %
   dl2_cov_mat =  inv( dl2_prd_mat ) * dl0_dat_var   ;
end
                                                % Standard error               %
dl1_std_err    = diag( dl2_cov_mat ) ;
dl1_std_err    = sqrt( dl1_std_err ) ;
dl1_std_err(3) = NaN                 ;
dl1_std_err(4) = NaN                 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Update output values         %
So0_nam.d1serr = dl1_std_err         ;          % Update output structure      %
So0_nam.d0dvar = dl0_dat_var         ;
So0_nam.d0aicc = dl0_aic_val         ;
So0_nam.n0nump = jl0_num_pts         ;
So0_nam.j0mlen = jl0_mat_len         ;
So0_nam.j0mrnk = jl0_mat_rnk         ;
So0_nam.d0mcon = dl0_mat_con         ;
So0_nam.d0mdet = dl0_mat_det         ;
So0_nam.d2covm = dl2_cov_mat         ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                % Debug                        %
if ( jl0_dbg_flg >= Sc0_qgd.J_DBGLV3 )
   fprintf( jl0_dfi , '1-stage Exponential with Lag\n' )            ;
   fprintf( jl0_dfi , ' Std error a:   \t%e\n' , dl1_std_err(1) )    ;
   fprintf( jl0_dfi , ' Std error k1:  \t%e\n' , dl1_std_err(2) )    ;

   fprintf( jl0_dfi , ' Number of points:  %d\n' , jl0_num_pts )     ;
```

```
      fprintf( jl0_dfi , ' AICc:              %e\n' , dl0_aic_val )     ;
      fprintf( jl0_dfi , ' Data variance:     %e\n' , dl0_dat_var )     ;
      fprintf( jl0_dfi , ' Matrix length:     %d\n' , jl0_mat_len )     ;
      fprintf( jl0_dfi , ' Matrix rank:       %d\n' , jl0_mat_rnk )     ;
      fprintf( jl0_dfi , ' Condition number:  %e\n' , dl0_mat_con )     ;
      fprintf( jl0_dfi , ' Determinant:       %e\n' , dl0_mat_det )     ;
      fprintf( jl0_dfi , '\n' )                                         ;
   end
end                                                 % END OF FUNCTION             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

B.2 BAYESIAN SOURCE CODE

The Python code that implements the Bayesian methods was developed under Python 2.7. The code requires the installation of three opensource packages (NumPy, SciPy, and Matplotlib) obtained from the SciPy.org repository (Reference [gg]), and the vuq opensource package from the github repository (Reference [hh]). The four opensource packages are listed in Table B-3. The code fits the datasets discussed in Appendix A using the Variational Bayesian Inference with a single Gaussian family as described in Section 2.2. The code is run by activating the main script file quasi_chemical_variational.pbs which sets up file locations and run options for the do_variational.py script. The file quasi_chemical_variational.pbs must be edited to specify the local directory holding the Python code. Data for the datasets is given in Figures A-11 through A-49, and should be entered into a $n \times 2$ NumPy array of time and base 10 logarithm of the bacterial concentration, and output as a NumPy array binary file (.npy). After a solution has been calculated and output, the output filename in the script file plot_uq_propag_F1.py can be edited to plot the QCM-4 model data and fit.

Table B-3. Opensource Python Packages

Function	Description
NumPy	Base N-dimensional array package
SciPy	Fundamental library for scientific computing
Matplotlib	Comprehensive 2D plotting
vuq	Variational reformulation of Bayesian inverse problems

The Python source code for the Bayesian method consists of 1 Python bash script file (.bps) and 8 Python code script files (.py). A description of each script file is provided in Table B-4. A listing of the code for the 9 files is given on pages B-159 through B-176.

Table B-4. Bayesian UQ Python Script Files

Function	Description
quasi_chemical_variational.pbs	Main driver script for Bayesian UQ code
do_variational.py	Control script for variational method
common.py	Defines utility functions common to variational method
quasi_chemical_model.py	Sets up methods and structures for QCM-4 model
quasi_chemical_common.py	Provides common functions for entering QCM-4 datasets and plotting output
QCM_solve.py	Runs the solver for the system of ODEs defined by the QCM-4 model
examples_common.py	Sets common file and directory locations
_forward_QCM_model.py	Accesses the QCM-4 fit function and differentials for to calculate the bacterial growth as a function of time
plot_uq_propag_F1.py	Plots data and QCM-4 solutions for datasets from Figure 1 of Reference [b]. Can be edited for other datasets

```
#!/bin/sh -l
#PBS -V
#PBS -q ibilion
#PBS -l nodes=1:ppn=16
#PBS -l walltime=01:00:00
#PBS -N variational_catalysis

# Setting up everything
PROJECT_DIR=/Users/panostsilifis/catalysis_code/catalysis-problem/code
SCRIPT_DIR=$PROJECT_DIR
PAPER_DIR=$PROJECT_DIR

# Model specific data
MODEL_FILE=$PAPER_DIR/quasi_chemical_model.py

cd $PAPER_DIR

python \
    $SCRIPT_DIR/do_variational.py \
        -m $MODEL_FILE \
        --entropy-approximation="EntropyLowerBound" \
      --mu-lower-bound="[-10] * 5" \
      --mu-upper-bound="[10] * 5" \
      --C-lower-bound="[1.e-8] * 5" \
      --C-upper-bound="[10] * 5" \
        --num-comp=1 \
        --optimize-full-mu \
        -f

"""
Solve the inverse problem using the variational methodology.

Author:
    Ilias Bilionis

Date:
    9/10/2014

"""


from collections import Iterable
from common import *
from vuq import *


def parse_expr_callback(option, opt, value, parser):
    """
    This is not very good for security, but it is a fast
    way to pass options.
    """
    if value is not None:
        setattr(parser.values, option.dest, eval(value))


def parse_list_callback(option, opt, value, parser):
    if value is not None:
        value = [float(x) for x in value.split(',')]
        if len(value) == 0:
            value = value[0]
    setattr(parser.values, option.dest, value)


def convert_to_list(l, n):
    if not isinstance(l, Iterable):
        l = [l] * n
    return l
```

```
def initialize_bounds(l, u, n):
    l = convert_to_list(l, n)
    u = convert_to_list(u, n)
    b = tuple((l[i], u[i]) for i in xrange(n))
    return b


def main(options):
    """
    The main function.
    """
    # Load the model
    model = initialize_native_model(options.model)
    # Get the target
    log_p = model['log_p']
    # Get the prior
    log_prior = model['log_prior']
    # Construct the initial approximation
    comp = [MultivariateNormal(log_prior.sample().flatten())
            for i in xrange(options.num_comp)]
    log_q = MixtureOfMultivariateNormals(comp)
    if options.mu_init is not None:
        log_q.mu = options.mu_init
        print str(log_q)
    # The entropy approximation
    entropy = eval(options.entropy_approximation + '()')
    # The expectation functional
    expectation_functional = eval(options.expectation_functional + '(log_p)')
    # The Evidence-Lower-BOund
    elbo = EvidenceLowerBound(entropy, expectation_functional)
    # The optimizer
    optimizer = Optimizer(elbo)
    # The upper and lower bounds for everything
    mu_bounds = initialize_bounds(options.mu_lower_bound, options.mu_upper_bound,
log_q.num_dim)
    C_bounds = initialize_bounds(options.C_lower_bound, options.C_upper_bound,
log_q.num_dim)
    print 'mu_bounds', mu_bounds
    print 'C_bounds', C_bounds
    # The output file
    output_file = options.output
    if output_file is None:
        output_file = os.path.abspath(os.path.splitext(options.model)[0] + '_'
                                      + 'num_comp=' + str(options.num_comp) +
'.pcl')
    # Delete the output file if it exists and you want to force calculations
    if os.path.exists(output_file) and options.force:
        print '-', output_file, 'exists'
        print '- I am removing it'
        os.remove(output_file)
    # If the file exists at this point, then this is not a forced calculation
    # and we should just load it
    if os.path.exists(output_file):
        print '- I am not repeating the calculations'
        with open(output_file, 'rb') as fd:
            results = pickle.load(fd)
            L = results['L']
            log_q = results['log_q']
            nfev = results['nfev']
    else:
        # Do the optimization
        L, nfev = optimizer.optimize(log_q,
                                     tol=options.tol,
                                     max_it=options.max_it,
                                     mu_bounds=mu_bounds,
                                     C_bounds=C_bounds,
                                     full_mu=options.optimize_full_mu)
```

```
        #print str(elbo)
        # Save the results
        results = {}
        results['L'] = L
        results['log_q'] = log_q
        results['nfev'] = nfev
        with open(output_file, 'wb') as fd:
            pickle.dump(results, fd, protocol=pickle.HIGHEST_PROTOCOL)
    # Print a salary of the statistics
    w = log_q.w
    mu = log_q.mu
    print mu, mu.shape, w.shape
    C = log_q.C
    c = np.vstack([np.diag(C[i, :, :]) for i in xrange(log_q.num_comp)])
    x_m = np.dot(w,mu)
    #x_m = np.mean(w * mu, axis=0)
    # The variance of the mixture can only be approximated via sampling...
    samples = log_q.sample(options.std_samples)
    x_s = np.std(samples, axis=0)
    x_05 = np.percentile(samples, 5, axis=0)
    x_95 = np.percentile(samples, 95, axis=0)
    print '{0:10s} {1:10s} {2:10s}'.format('Parameter', 'Mean', 'Std.')
    print '-' * 32
    for i in xrange(log_q.num_dim):
        print '{0:10s} {1:4.6f} +-{2:2.6f}'.format('x_' + str(i+1), x_m[i], 1.96
* x_s[i])
    for i in xrange(log_q.num_dim):
        print '(%1.3f, %1.3f)' % (x_05[i], x_95[i])
    print 'Number of evaluations:', nfev
    print log_q


if __name__ == '__main__':
    parser = OptionParser()
    parser.add_option('-m', '--model', metavar='FILE', dest='model',
                      help='specify the file containing the native model')
    parser.add_option('--entropy-approximation', type=str,
dest='entropy_approximation',
                      default='FirstOrderEntropyApproximation',
                      help='specify the entropy approximation to be used')
    parser.add_option('--expectation-functional', type=str,
dest='expectation_functional',
                      default='ThirdOrderExpectationFunctional',
                      help='specify the expectation functional to be used')
    parser.add_option('--num-comp', type=int, dest='num_comp', default=1,
                      help='specify the number of components that you want to
use')
    parser.add_option('--mu-lower-bound', type=str, dest='mu_lower_bound',
action='callback',
                      callback=parse_expr_callback, default=None,
                      help='specify the lower bound for mu')
    parser.add_option('--mu-upper-bound', type=str, dest='mu_upper_bound',
action='callback',
                      callback=parse_expr_callback, default=None,
                      help='specify the upper bound for mu')
    parser.add_option('--C-lower-bound', type=str, dest='C_lower_bound',
action='callback',
                      callback=parse_expr_callback, default=1e-6,
                      help='specify the lower bound for C')
    parser.add_option('--C-upper-bound', type=str, dest='C_upper_bound',
action='callback',
                      callback=parse_expr_callback, default=100.,
                      help='specify the upper bound for C')
    parser.add_option('--max-it', type=int, dest='max_it', default=100,
                      help='specify the maximum number of iterations')
    parser.add_option('--tol', type=float, dest='tol', default=1e-2,
                      help='specify the tolerance')
```

```
    parser.add_option('--std-samples', type=int, dest='std_samples',
default=10000,
                      help='specify the number of samples used in the estimation
of the std')
    parser.add_option('--optimize-full-mu', action='store_true',
dest='optimize_full_mu',
                      help='optimize all the mus simultaneously')
    parser.add_option('-f', '--force', action='store_true', dest='force',
                      help='force the calculation')
    parser.add_option('-o', '--output', metavar='FILE', dest='output',
                      help='specify the output file')
    parser.add_option('--mu-init', type=str, dest='mu_init', action='callback',
                      callback=parse_expr_callback,
                      help='specify the initial point for mu num_comp x num_dim')
    options, args = parser.parse_args()
    main(options)

"""
Some routines that are common to all scripts.

Author:
    Ilias Bilionis

Date:
    9/9/2014

"""


import sys
import os
import imp
import cPickle as pickle
import numpy as np
from optparse import OptionParser


# Make sure that we can see the path that shows to the base of this project
project_dir = os.path.abspath(os.path.join(os.path.split(__file__)[0], '..'))
# And add this to our path
sys.path.insert(0, project_dir)

# Now try to load vuq
import vuq


# Now some useful function definitions

def init_mpi(use_mpi):
    """
    Initialize mpi (if requested).
    """
    if use_mpi is not None:
        import mpi4py.MPI as mpi
        comm = mpi.COMM_WORLD
        rank = comm.Get_rank()
        size = comm.Get_size()
        return mpi, comm, rank, size
    else:
        return None, None, 0, 1


def get_rank_size(comm):
    """
    Get the rank and size from a communicator.
    """
    if comm is None:
        return 0, 1
```

```
    rank = comm.Get_rank()
    size = comm.Get_size()
    return rank, size


def print_once(msg, comm=None):
    """
    Use to print a message once when using mpi.
    """
    if comm is None:
        sys.stdout.write(msg)
    else:
        rank = comm.Get_rank()
        if rank == 0:
            sys.stdout.write(msg)
            sys.stdout.flush()
        comm.barrier()


def initialize_model(model_file, model_name='Model', comm=None):
    """
    Initialize the ``model_file`` which should be a proper python module.
    """
    rank, size = get_rank_size(comm)
    print_once('Initializing the %s.\n' % model_name, comm=comm)
    print_once('-------------------------\n', comm=comm)
    for i in xrange(size):
        if rank == i:
            try:
                model = imp.load_source('', model_file)
            except Exception as e:
                signal_fatal_error('I couldn\'t load the %s.\n' % model_name,
                                   comm=comm, e=e)
            print_rank(i, 'initialized %s.\n' % model_name)
        mpi_wait(comm)
    print_once('Done.\n'
               '-------------------------\n', comm=comm)
    return model


def initialize_pymc_model(model_file, model_name='PyMC Model', comm=None):
    """
    Initialize a PyMC model.
    """
    return initialize_model(model_file, model_name=model_name,
comm=comm).make_model()


def initialize_native_model(model_file, model_name='Native Model', comm=None):
    """
    Initialize a native model.
    """
    return initialize_model(model_file, model_name=model_name,
comm=comm).make_model()


def signal_fatal_error(err, err_code=1, comm=None, e=None):
    """
    Signal a fatal error. The program will exit.
    """
    sys.stderr.write(err)
    if comm is None:
        if e is not None:
            print e
        sys.exit(1)
    else:
        if comm.Get_rank() == 0:
            print e
```

```
        comm.Abort(1)


def print_rank(i, msg):
    """
    Use to print information from a particular rank.
    This is an mpi only function.
    """
    sys.stdout.write('rank %d: %s' % (i, msg))
    sys.stdout.flush()


def mpi_wait(comm):
    """
    Wait if comm is not ``None``.
    """
    if comm is not None:
        comm.barrier()


def get_running_statistics(data):
    """
    Returns the running mean and standard deviation of the data.
    """
    x_m = np.cumsum(data, axis=0) / np.arange(1, data.shape[0] + 1)[:, None]
    x2_m = np.cumsum(data ** 2., axis=0) / np.arange(1, data.shape[0] + 1)[:,
None]
    x_s = np.sqrt(x2_m - x_m ** 2.)
    return x_m, x_s

"""
The native model for the dimensionless catalysis problem.

Author:
    Panagiotis Tsilifis
Date:
    12/03/2015

"""


import sys
import os
sys.path.insert(0, os.path.abspath(os.path.split(__file__)[0]))
sys.path.insert(0, os.path.abspath(os.path.join(os.path.split(__file__)[0],
'../..')))
from quasi_chemical_common import *
from vuq import MultivariateNormal
from vuq import PDFCollection
from vuq import Joint
from vuq import IsotropicGaussianLikelihood
from quasi_chemical import QuasiChemicalModel


def make_model():
    """
    The native version of the Quasi-Chemical model.
    """
    # The number of dimensions for this particular problem
    num_dim = 5

    # The log prior
    #mu_pr = np.array([0., 4.,  5., 5.])
    #mu_pr = np.array([0.20317747,  1.51182072 , 1.22204306, -0.3173649])
    #mu_pr = np.array([0.04203951, 5.4587158, 5.45635107])
    #mu_pr = np.log(np.array([0.37, 40.50, 187.30, 40.08])) # 0.84
    #mu_pr = np.log(np.array([1.04, 238.82, 314.01, 238.26])) # 0.79
    #prior_mu = np.hstack([mu_pr, -1.])
```

```
    prior_mu = np.hstack([np.zeros(num_dim - 1), -1.])
    prior_C = 1* np.eye(num_dim)
    #prior_C[0,0] *= 10
#    prior_C[2,2] *=
    log_prior = MultivariateNormal(prior_mu, C=prior_C)

    # The data
    y = load_QCM_data()

    # The forward model
    qc_model = QuasiChemicalModel()

    # The log likelihood
    log_like = IsotropicGaussianLikelihood(y[0, :], qc_model)

    # The joint
    log_p = Joint(log_like, log_prior)

    return locals()


if __name__ == '__main__':
    model = make_model()
    print model

"""
Things that are common to the QCM problem (no matter how you solve it).

Author:
    Panagiotis Tsilifis

Date:
    11/29/2015

"""


from examples_common import *
import numpy as np


#data = os.path.join(example_dir, 'F1_aw_91.npy')


def load_QCM_data():
    """
    Loads the quasi-chemical data and casts them to the right format.
    """
#    print example_dir
    data = np.load(os.path.join(example_dir, 'F1_aw_91.npy'))
#    data = np.load('../quasi_chem_data/F2_25.npy')
    print data.shape
    y = data[:, 1]
    return y.reshape((1, y.shape[0]))


#def load_dimensionless_catalysis_data():
#    """
#    Loads a dimensionless version of the catalysis data.
#    """
#    return load_catalysis_data() / 500.


def plot_QCM_output(fig_name,
                            Y,
                            y=load_QCM_data(),
                            t = np.array([  0.        ,   0.98420413,   1.98055893,
3.00121507,
```

```
                                        4.0218712 ,   5.018226  ,   6.03888214,
7.0109356 ,
                                        8.03159174,   9.05224787,  10.02430134,
11.04495747,
                                        11.9927096 ,  13.03766707,  13.9854192 ,
15.03037667,
                                        16.9744836 ,  20.]),
                           colors=['b'],
                           linestyles=['--'],
                           markerstyles=['^'],
                           legend=False,
                           title=None):
    """
    Draw the output of the catalysis problem.

    :param fig_name:    A name for the figure.
    :param Y:           The samples observed.
    :param y:           The observations.
    :parma t:           The times of observations.
    """
    shape = (t.shape[0], Y.shape[1] / t.shape[0])
    y = y.reshape(shape)
    Y_m = np.percentile(Y, 50, axis=0).reshape(shape)
    Y_p05 = np.percentile(Y, 5, axis=0).reshape(shape)
    Y_p95 = np.percentile(Y, 95, axis=0).reshape(shape)
    Y_p05[Y_p05 <= 0.] = 0.
    import matplotlib.pyplot as plt
    fig = plt.figure()
    ax = fig.add_subplot(111)
    for i in xrange(5):
        ax.plot(t, Y_m[:, i], colors[i] + linestyles[i], linewidth=2,
markersize=5.)
        ax.fill_between(t, Y_p05[:, i], Y_p95[:, i], color=colors[i], alpha=0.5)
    for i in xrange(5):
        ax.plot(t, y[:, i], colors[i] + markerstyles[i], markersize=10)
    ax.set_xlabel('Time ($\\tau$)', fontsize=26)
    ax.set_ylabel('Concentration', fontsize=26)
    ax.set_title(title, fontsize=26)
    plt.setp(ax.get_xticklabels(), fontsize=26)
    plt.setp(ax.get_yticklabels(), fontsize=26)
    if legend:
        leg = plt.legend(['$\operatorname{NO}_3^-$', '$\operatorname{NO}_2^-$',
                          '$\operatorname{N}_2$',
'$\operatorname{N}_2\operatorname{O}$',
                          '$\operatorname{NH}_3$'], loc='best')
        plt.setp(leg.get_texts(), fontsize=26)
    ax.set_ylim([0., 1.5])
    plt.tight_layout()
    print 'Writing:', fig_name
    plt.savefig(fig_name)

""" This script contains a solver of the Quasi-chemical kinetics model (QCM) for
    bacterial growth (Doona et al.). The functions below, solve numerically the
    original ODE system and the augmented ODE's that return the gradients and
    Hessians of the solution with respect to the model parameters.


Author:
    Panagiotis Tsilifis

Date:
    11/29/2015


REFERENCE: Doona, Christopher J., Florence E. Feeherry, and Edward W. Ross.
           "A quasi-chemical model for the growth and death of microorganisms
           in foods by non-thermal and high-pressure processing." International
           journal of food microbiology 100.1 (2005): 21-32.
```

```
"""

import numpy as np
from scipy.integrate import ode
import matplotlib.pyplot as plt
import math


def make_A(kappa):
    """
    Make the matrix of the dynamical system from ``kappa``.
    """
    A = np.array([[-kappa[0], 0, 0, 0],
              [kappa[0], kappa[1]-kappa[3], 0, 0],
              [0, kappa[1], 0, 0],
              [0, kappa[3], 0, 0]])
    # The derivative of A with respect to kappa
    d = A.shape[0]
    s = kappa.shape[0]
    dA = np.zeros((d, d, s))
    dA[0, 0, 0] = -1.
    dA[1, 0, 0] = 1.
    dA[1, 1, 1] = 1.
    dA[2, 1, 1] = 1.
    dA[1, 1, 3] = -1.
    dA[3, 1, 3] = 1.
    return A, dA


def f0(t, y, kappa):
    #print y.shape
    #assert y.shape == (4,1)
    assert kappa.shape[0] == 4
    f = np.zeros((4,1))
    f[0,0] = -kappa[0] * y[0]
    f[1,0] = kappa[0] * y[0] + kappa[1] * y[1] - 1.e-9*kappa[2] * y[1] * y[2] -
kappa[3] * y[1]
    f[2,0] = kappa[1] * y[1] - 1.e-9 * kappa[2] * y[1] * y[2]
    f[3,0] = 1.e-9 * kappa[2] * y[1] * y[2] + kappa[3] * y[1]
    return f

def f1(t, y, kappa):
    assert kappa.shape[0] == 4
    f1 = np.zeros((20,1))
    y_0 = y[:4]
    f1[:4,0] = f0(t, y_0, kappa)[:,0]
    y_1 = y[4:y.shape[0]]
    f1[4:8,0] = -kappa[0] * y_1[:4]
    f1[4,0] -= y_0[0]
    f1[8,0] = y_0[0] + kappa[0] * y_1[0] + kappa[1] * y_1[4] - 1.e-9*kappa[2]
*(y_1[4] * y_0[2] + y_0[1] * y_1[8]) - kappa[3] * y_1[4]
    f1[9,0] = y_0[1] + kappa[0] * y_1[1] + kappa[1] * y_1[5] - 1.e-9*kappa[2]
*(y_1[5] * y_0[2] + y_0[1] * y_1[9]) - kappa[3] * y_1[5]
    f1[10,0] = -1.e-9 * y_0[1]*y_0[2] +kappa[0]*y_1[2] + kappa[1] * y_1[6] - 1.e-
9 * kappa[2] * (y_1[6] * y_0[2] + y_0[1] * y_1[10]) - kappa[3]*y_1[6]
    f1[11,0] = -y_0[1] + kappa[0]*y_1[3] + kappa[1]*y_1[7] - 1.e-9*kappa[2] *
(y_1[7]*y_0[2] + y_0[1]*y_1[11]) - kappa[3] * y_1[7]
    f1[12,0] = kappa[1]*y_1[4] - 1.e-9*kappa[2]*(y_1[4]*y_0[2] + y_0[1]*y_1[8])
    f1[13,0] = y_0[1] + kappa[1]*y_1[5] - 1.e-9*kappa[2]*(y_1[5]*y_0[2] +
y_0[1]*y_1[9])
    f1[14,0] = kappa[1] * y_1[6] - 1.e-9 * y_0[1]*y_0[2] - 1.e-9 * kappa[2] *
(y_1[6]*y_0[2] + y_0[1]*y_1[10])
    f1[15,0] = kappa[1] * y_1[7] - 1.e-9 * kappa[2] * (y_1[7]*y_0[2] +
y_0[1]*y_1[11])
    f1[16,0] = 1.e-9 * kappa[2] * (y_1[4]*y_0[2] + y_0[1]*y_1[8]) + kappa[3] *
y_1[4]
```

```
    f1[17,0] = 1.e-9 * kappa[2] * (y_1[5]*y_0[2] + y_0[1]*y_1[9]) + kappa[3] *
y_1[5]
    f1[18,0] = 1.e-9 * y_0[1] * y_0[2] + 1.e-9*kappa[2]*(y_1[6]*y_0[2] +
y_0[1]*y_1[10]) + kappa[3]*y_1[6]
    f1[19,0] = 1.e-9 * kappa[2] * (y_1[7] * y_0[2] + y_0[1] * y_1[11]) + kappa[3]
* y_1[7] + y_0[1]
    return f1

def f2(t, y, kappa):
    assert kappa.shape[0] == 4
    f2 = np.zeros((84,1))
    y_1 = y[:20]
    f2[:20,0] = f1(t, y_1, kappa)[:,0]
    y_2 = y[20:y.shape[0]]
    #f2[20:24,0] = -y_1[4:8] -kappa[0] * y_2[:4] # Checked
    f2[20,0] = -2*y_1[4] - kappa[0] * y_2[0]
    f2[21,0] = -y_1[5] - kappa[0] * y_2[1]
    f2[22,0] = -y_1[6] - kappa[0] * y_2[2]
    f2[23,0] = -y_1[7] - kappa[0] * y_2[3]
    #f2[20,0] -= y_1[4]
    f2[24:28,0] = -kappa[0] * y_2[4:8] ### --- Checked ! ! ! ---
    f2[24,0] -= y_1[5]
    f2[28:32,0] = -kappa[0] * y_2[8:12] ### --- Checked ! ! ! ---
    f2[28,0] -= y_1[6]
    f2[32:36,0] = -kappa[0] * y_2[12:16] ### --- Checked ! ! ! ---
    f2[32,0] -= y_1[7]
    f2[36:40,0] = y_1[4:8] + kappa[0] * y_2[:4] + kappa[1] * y_2[16:20] - 1.e-
9*kappa[2]*(y_2[16:20]*y_1[2] + y_1[8]*y_1[12:16] + y_1[8:12]*y_1[12] +
y_1[1]*y_2[32:36]) - kappa[3]*y_2[16:20] ### --- Checked ! ! ! ---
    f2[36,0] += y_1[4]
    f2[37,0] += y_1[8]
    f2[38,0] -= 1.e-9 * (y_1[8]*y_1[2] + y_1[1]*y_1[12])
    f2[39,0] -= y_1[8]
    f2[40:44,0] = y_1[8:12] + kappa[0]* y_2[4:8] + kappa[1]* y_2[20:24] - 1.e-
9*kappa[2]*(y_2[20:24]*y_1[2] + y_1[9]*y_1[12:16] + y_1[8:12]*y_1[13] +
y_1[1]*y_2[36:40]) - kappa[3]*y_2[20:24] ### --- Checked ! ! ! ---
    f2[40,0] += y_1[5]
    f2[41,0] += y_1[9]
    f2[42,0] -= 1.e-9 * (y_1[9]*y_1[2] + y_1[1]*y_1[13])
    f2[43,0] -= y_1[9]
    f2[44:48,0] = kappa[0] * y_2[8:12] + kappa[1] * y_2[24:28] - 1.e-9 *
(y_1[8:12]*y_1[2] + y_1[1]*y_1[12:16] + kappa[2]*(y_2[24:28]*y_1[2] +
y_1[10]*y_1[12:16] + y_1[8:12]*y_1[14] + y_1[1]*y_2[40:44]) ) -
kappa[3]*y_2[24:28] ### ---- Checked ! ! ! ---
    f2[44,0] += y_1[6]
    f2[45,0] += y_1[10]
    f2[46,0] -= 1.e-9 * (y_1[10]*y_1[2] + y_1[1]*y_1[14])
    f2[47,0] -= y_1[10]
    f2[48:52,0] = -y_1[8:12] + kappa[0]* y_2[12:16] + kappa[1]*y_2[28:32] - 1.e-
9*kappa[2]*(y_2[28:32]*y_1[2] + y_1[11]*y_1[12:16] + y_1[8:12]*y_1[15] + y_1[1] *
y_2[44:48]) - kappa[3]*y_2[28:32] ### --- Checked ! ! ! ---
    f2[48,0] += y_1[7]
    f2[49,0] += y_1[11]
    f2[50,0] -= 1.e-9 * (y_1[11]*y_1[2] + y_1[1]*y_1[15])
    f2[51,0] -= y_1[11]
    f2[52:56,0] = kappa[1] * y_2[16:20] - 1.e-9*kappa[2]*(y_2[16:20]*y_1[2] +
y_1[8]*y_1[12:16] + y_1[8:12]*y_1[12] + y_1[1]*y_2[32:36])
    f2[53,0] += y_1[8]
    f2[54,0] -= 1.e-9 * (y_1[8]*y_1[2] + y_1[1]*y_1[12])
    f2[56:60,0] = y_1[8:12] + kappa[1]*y_2[20:24] - 1.e-9*kappa[2] *
(y_2[20:24]*y_1[2] + y_1[9]*y_1[12:16] + y_1[8:12]*y_1[13] + y_1[1]*y_2[36:40])
    f2[57,0] += y_1[9]
    f2[58,0] -= 1.e-9 * (y_1[9]*y_1[2] + y_1[1]*y_1[13])
    f2[60:64,0] = kappa[1] * y_2[24:28] - 1.e-9 * (y_1[8:12]*y_1[2] +
y_1[1]*y_1[12:16]) - 1.e-9*kappa[2]* (y_2[24:28]*y_1[2] + y_1[10]*y_1[12:16] +
y_1[8:12]*y_1[14] + y_1[1]*y_2[40:44])
    f2[61,0] += y_1[10]
    f2[62,0] -= 1.e-9 * (y_1[10]*y_1[2] + y_1[1]*y_1[14])
```

```
    f2[64:68,0] = kappa[1] * y_2[28:32] - 1.e-9*kappa[2]*(y_2[28:32]*y_1[2] +
y_1[11]*y_1[12:16] + y_1[8:12]*y_1[15] + y_1[1]*y_2[44:48])
    f2[65,0] += y_1[11]
    f2[66,0] -= 1.e-9 * (y_1[11]*y_1[2] + y_1[1]*y_1[15])
    f2[68:72,0] = kappa[3] * y_2[16:20] + 1.e-9*kappa[2]*(y_2[16:20]*y_1[2] +
y_1[8]*y_1[12:16] + y_1[8:12]*y_1[12] + y_1[1]*y_2[32:36])
    f2[71,0] += y_1[8]
    f2[70,0] += 1.e-9 * (y_1[8]*y_1[2] + y_1[1]*y_1[12])
    f2[72:76,0] = kappa[3]*y_2[20:24] + 1.e-9*kappa[2] * (y_2[20:24]*y_1[2] +
y_1[9]*y_1[12:16] + y_1[8:12]*y_1[13] + y_1[1]*y_2[36:40])
    f2[75,0] += y_1[9]
    f2[74,0] += 1.e-9 * (y_1[9]*y_1[2] + y_1[1]*y_1[13])
    f2[76:80,0] = kappa[3] * y_2[24:28] + 1.e-9 * (y_1[8:12]*y_1[2] +
y_1[1]*y_1[12:16]) + 1.e-9*kappa[2]* (y_2[24:28]*y_1[2] + y_1[10]*y_1[12:16] +
y_1[8:12]*y_1[14] + y_1[1]*y_2[40:44])
    f2[79,0] += y_1[10]
    f2[78,0] += 1.e-9 * (y_1[10]*y_1[2] + y_1[1]*y_1[14])
    f2[80:84,0] = y_1[8:12] + kappa[3] * y_2[28:32] + 1.e-
9*kappa[2]*(y_2[28:32]*y_1[2] + y_1[11]*y_1[12:16] + y_1[8:12]*y_1[15] +
y_1[1]*y_2[44:48])
    f2[83,0] += y_1[11]
    f2[82,0] += 1.e-9 * (y_1[11]*y_1[2] + y_1[1]*y_1[15])
    return f2


def f(kappa, y0, t):
    """
    Evaluate the model at ''kappa'' and at times ''t'' with initial
    conditions ''y0''.

    It returns a flatten version of the system, i.e.:
    y_1(t_1)
    ...
    y_d(t_1)
    ...
    y_1(t_K)
    ...
    y_d(t_K)
    """
    r = ode(f0)
    r.set_initial_value(y0, 0).set_f_params(kappa)
    y_m = [y0[None, :]]
    for tt in t[1:]:
        r.integrate(tt)
        y_m.append(r.y[None, :])
    y_m = np.vstack(y_m)
    return y_m.flatten()

def df(kappa, y0, t):
    """
    Evaluate the Jacobian with respect to the model derivatives at ''kappa'' and
    at times ''t'' with initial conditions ''y0''.

    This returns a matrix of the following form:
    dy_1(t_1) / dkappa_1 ... dy_1(t_1) / dkappa_s
    ...
    dy_d(t_1) / dkappa_1 ... dy_d(t_1) / dkappa_s
    ...
    dy_1(t_K) / dkappa_1 ... dy_d(t_K) / dkappa_s
    ...
    dy_d(t_K) / dkappa_1 ... dy_d(t_K) / dkappa_s
    """
    r = ode(f1)
    r.set_initial_value(np.hstack([y0, np.zeros((16, )) ])).set_f_params(kappa)
    y_m = [r.y]
    for tt in t[1:]:
        r.integrate(tt)
        y_m.append(r.y[None, :])
```

```
    y_m = np.vstack(y_m)
    J = y_m[:, 4:].reshape((t.shape[0], 4, 4))
    return J.reshape((4 * t.shape[0], kappa.shape[0]))

def df2(kappa, y0, t):
    """
    Evaluate the Hessian with respect to the model derivatives at ''kappa'' and
    at times ''t'' with initial conditions ''y0''.

    This returns a matrix of the following form:
    dy_1(t_1) / dkappa_1 ... dy_1(t_1) / dkappa_s
    ...
    dy_d(t_1) / dkappa_1 ... dy_d(t_1) / dkappa_s
    ...
    dy_1(t_K) / dkappa_1 ... dy_d(t_K) / dkappa_s
    ...
    dy_d(t_K) / dkappa_1 ... dy_d(t_K) / dkappa_s
    """
    r = ode(f2) # Look at system.py: this should be the f of the adjoint
    r.set_initial_value(np.hstack([y0, np.zeros((80, ))])).set_f_params(kappa)
    y_m = [r.y]
    for tt in t[1:]:
        r.integrate(tt)
        y_m.append(r.y[None, :])
    y_m = np.vstack(y_m)
    #print y_m.shape
    #J = y_m[:, d:d+d**3].reshape((t.shape[0], d, d, d))
    J2 = y_m[:,20:].reshape((t.shape[0], 4, 4, 4))
    return J2.reshape((4 * t.shape[0], kappa.shape[0], kappa.shape[0]))

def ndf(kappa0, y0, t, h=1e-3):
    """
    Numerically compute the Jacobian of f at x0.
    """
    y = f(kappa0, y0, t)
    kappa = kappa0.copy()
    J = np.zeros((y.shape[0], kappa0.shape[0]))
    for i in xrange(kappa0.shape[0]):
        kappa[i] += h
        py = f(kappa, y0, t)
        kappa[i] -= h
        J[:, i] = (py - y) / h
    return J


def ndf2(kappa0, y0 ,t, h=1e-2):

    y = f(kappa0, y0, t)
    kappa = kappa0.copy()
    J = np.zeros((y.shape[0], kappa0.shape[0], kappa0.shape[0]))
    for i in xrange(kappa0.shape[0]):
        for j in xrange(kappa0.shape[0]):
            kappa[i] += h
            kappa[j] += h
            py1 = f(kappa, y0, t)
            kappa[j] -= 2*h
            py2 = f(kappa, y0 ,t)
            kappa[j] += 2*h
            kappa[i] -= 2*h
            py3 = f(kappa, y0, t)
            kappa[j] -= 2*h
            py4 = f(kappa, y0, t)
            kappa[i] += h
            kappa[j] += h
            J[:, i, j] = (py1 - py2 - py3 + py4) / (4*h**2)
    return J
```

```
"""
Some methods common to all examples.

Author:
    Ilias Bilionis

Date:
    9/9/2014

"""

import sys
import os

# The examples directory
example_dir = os.path.abspath(os.path.join(os.path.split(__file__)[0], '.'))
project_dir = os.path.abspath(os.path.join(os.path.split(__file__)[0], '../..'))
demo_dir = os.path.join(project_dir, 'demos')
sys.path.insert(0, demo_dir)

# And load everything from the common.py of the scripts dir
scripts_dir= os.path.join(project_dir, 'scripts')
sys.path.insert(0, scripts_dir)
from common import *

"""
QCM Forward model

Author:
    Panagiotis Tsilifis
Date:
    11/29/2015

"""


import numpy as np
from QCM_solve import *
import sys
sys.path.insert(0,'../')
from vuq import Model
from vuq import view_as_column



class QuasiChemicalModel(Model):
    """
    A class representing the forward model of the Quasi-chemical model for
    bacterial growth.
    """

    def __init__(self, name='Quasi Chemical model'):
        """
        Initialize the object
        """
        #self._kappa = x
        super(QuasiChemicalModel, self).__init__(4, 26,
name=name,compute_hessian=False)

    def _eval(self, z):
        """
        Solves the dynamical system for given parameters x.
        """
       # z0 = np.array([z[0],  z[1],   5.75878015,  z[2]])
        z0 = view_as_column(z)
        x = np.exp(z0)
        # Points where the solution will be evaluated
        #t = np.array([0., 1./6, 1./3, 1./2, 2./3, 5./6, 1.])
```

```
        t = np.load('../quasi_chem_data/F2_25.npy')[:,0]
        y_init = np.load('../quasi_chem_data/F2_25.npy')[0,1]
        #t = np.array([  0.        ,   0.98420413,   1.98055893,   3.00121507,
        #              4.0218712 ,   5.018226  ,   6.03888214,   7.0109356 ,
        #              8.03159174,   9.05224787,  10.02430134,  11.04495747,
        #              11.9927096 ,  13.03766707,  13.9854192 ,  15.03037667,
        #              16.9744836 ,  20.])
        t_len = t.shape[0]
#        t = np.load('F1_aw_91.npy')[:,0]
        t = view_as_column(t)
        # Initial condition
        y0 = np.array([10 ** y_init, 0., 0., 0.])
        y0 = view_as_column(y0)
        assert x.shape[0] == 4
        sol = f(x[:,0], y0[:,0], t[:,0])
        J = df(x[:,0], y0[:,0], t[:,0])
        H = df2(x[:,0], y0[:,0], t[:,0])
        y = (sol.reshape((t.shape[0],4))[:,0] + sol.reshape((t.shape[0],
4))[:,1]).flatten() #
        dy = (J.reshape((t.shape[0],4,4))[:,0,:] +
J.reshape((t.shape[0],4,4))[:,1,:])
        d2y = np.zeros((t_len, H.shape[1], H.shape[2]))
        for i in range(H.shape[1]):
            for j in range(H.shape[2]):
                d2y[:,i,j]= (H.reshape((t_len,4,4,4))[:,0,i,j] +
H.reshape((t_len,4,4,4))[:,1,i,j]) / (y*np.log(10.)) -  dy[:,i] * dy[:,j] / (y**2
* np.log(10.))#
        for i in range(4):
            dy[:,i] = x[i,0] * dy[:,i] / (y*np.log(10.))
        for i in range(4):
            for j in range(4):
                if i == j:
                    d2y[:,i,j] = d2y[:,i,j] * x[j,0] * x[i,0] + dy[:,i]
                else:
                    d2y[:,i,j] = d2y[:,i,j] * x[i,0] * x[j,0]
        #state2d = np.zeros((t.shape[0], 3,3))
        #state2d[:,0,0] = d2y[:,0,0]
        #state2d[:,0,1] = d2y[:,0,1]
        #state2d[:,0,2] = d2y[:,0,3]
        #state2d[:,1,0] = d2y[:,1,0]
        #state2d[:,1,1] = d2y[:,1,1]
        #state2d[:,1,2] = d2y[:,1,3]
        #state2d[:,2,0] = d2y[:,3,0]
        #state2d[:,2,1] = d2y[:,3,1]
        #state2d[:,2,2] = d2y[:,3,3]

        state = {}
        state['f'] = np.log10(y)
        state['f_grad'] = dy#[:,[0,1,3]].reshape(t.shape[0],3) #* x.T
        #xx = np.kron(x, x.T)
        state['f_grad_2'] = d2y#[:,0,0].reshape(t.shape[0],1,1) #* xx
        #state['f_grad_2'] = state2d
        return state
"""
Forward problem

Author:
    Panagiotis Tsilifis
Date:
    11/29/2015

"""



import numpy as np
#from vuq import MultivariateNormal
import scipy.stats as st
```

```
from numpy import random
import matplotlib.pyplot as plt
import pickle

import sys
sys.path.insert(0,'demos/quasi_chemical/')

from QCM_solve import *

num_sam = 500
t = np.linspace(0.,15., 100)

data_79 = np.load('./quasi_chem_data/F1_aw_79.npy')
data_84 = np.load('./quasi_chem_data/F1_aw_84.npy')
#print data_84[:,0]
data_87 = np.load('./quasi_chem_data/F1_aw_87.npy')
data_91 = np.load('./quasi_chem_data/F2_25.npy')
print data_91.shape

### --- Initial con.
y_79 = np.array([10**data_79[0,1], 0., 0., 0.])
y_84 = np.array([10**data_84[0,1], 0., 0., 0.])
y_87 = np.array([10**data_87[0,1], 0., 0., 0.])
y_91 = np.array([10**data_91[0,1], 0., 0., 0.])


opt_params = pickle.load( open( "./quasi_chem_results/F1_newrun.p", "rb" ) )
#mu = np.array(opt_params['F1_aw_79']['opt_mean'][:4])
#mu = np.array([-0.19221667, -0.22810135, -0.0039794,  0.44954051])
#mu = np.log(np.array([1.04, 238.82, 314.01, 238.26]))
#print mu
#mu = np.array([0.05452692,  5.47245035,  5.7428491 ,  5.47021633])

#mu = np.array([0.04203951,  5.4587158,   5.75878015,  5.45635107])
#y = f(np.exp(mu), y_79, t).reshape((t.shape[0],4))[:,0] + f(np.exp(mu), y_79,
t).reshape((t.shape[0],4))[:,1]
#plt.plot(t, np.log10(y))
#plt.plot(data_79[:,0], data_79[:,1], '.')
#plt.show()

#dy_79 = df(np.exp(mu), y_79, t).reshape((t.shape[0],4,4))[:,0,:] +
df(np.exp(mu), y_79, t).reshape((t.shape[0],4,4))[:,1,:]
#for i in range(4):
#    dy_79[:,i] = dy_79[:,i] / (y*np.log(10.))
#C = np.diag(np.array(opt_params['F1_aw_79']['opt_var'][:4])**2)
#C = np.diag(np.array( [1.00036040e-16, 1.00004547e-16, 1.00000445e-16,
1.00000593e-16] ) )
#kappa_79_samples = random.multivariate_normal(mu, np.sqrt(C), size = (num_sam,))

#mu2 = np.array([-1.11617586e+00, -4.13228755e-01, 2.06251807e-04, -1.31479757e-
01])
#mu1 = np.array([-1.12240434e+00, 1.87967476e-01, 3.32168773e-04, 3.52896388e-
01])
#mu3 = np.array([-1.12182819e+00, 5.91030308e-02, 5.93635681e-04, 2.44648500e-
01])

#mu = np.array(opt_params['F1_aw_84']['opt_mean'][:4])

#mu = np.log(np.array([0.37, 40.50, 187.30, 40.08])) ## report
#print mu
#mu = np.array([-0.99083946 , 3.71723459 , 5.21660158 , 3.70711096])
#mu = np.array([-1.94404756,  2.93185984,  5.80684752 , 3.65948223])

#y = f(np.exp(mu), y_84, t).reshape((t.shape[0],4))[:,0] + f(np.exp(mu), y_84,
t).reshape((t.shape[0],4))[:,1]
#plt.plot(t, np.log10(y))
#plt.plot(data_84[:,0], data_84[:,1], '.')
#plt.show()
```

```
#dy_84 = df(np.exp(mu), y_84, t).reshape((t.shape[0],4,4))[:,0,:] +
df(np.exp(mu), y_84, t).reshape((t.shape[0],4,4))[:,1,:]
#for i in range(4):
#    dy_84[:,i] = dy_84[:,i] / (y*np.log(10.))

#C = np.diag(np.array([ 0.04389012,  0.23601138, 0.95329917,  0.60662732]))

#C = np.diag(np.array(opt_params['F1_aw_84']['opt_var'][:4])**2)
#C1 = np.diag(np.array([ 7.83279792e-02, 1.22179254e-03, 9.99741989e-01,
8.78546456e-04]))
#C2 = np.diag(np.array([ 0.07957315, 0.00388346, 0.99920959, 0.00221923]))
#C3 = np.diag(np.array([ 0.07845885,  0.00154094, 0.99976148,  0.00106475]))
#C = np.diag(np.array([  1.30446880e-02, 1.57903027e-06, 1.30290335e-02,
1.64474629e-06]))

#kappa_84_samples = random.multivariate_normal(mu, C, size = (num_sam,))

mu = np.array(opt_params['F1_aw_87']['opt_mean'][:4])
print mu
#mu = np.array([-1.48231339,  0.31780191,  2.68286168, -1.38814406 ])

C = np.diag(np.array(opt_params['F1_aw_87']['opt_var'][:4])**2)
print np.sqrt(C)
C = np.diag(np.array([0.5471, 0.0254, 0.6360, 0.0421])**2)
y = f(np.exp(mu), y_87, t).reshape((t.shape[0],4))[:,0] + f(np.exp(mu), y_87,
t).reshape((t.shape[0],4))[:,1]
#dy_87 = df(np.exp(mu), y_87, t).reshape((t.shape[0],4,4))[:,0,:] +
df(np.exp(mu), y_87, t).reshape((t.shape[0],4,4))[:,1,:]
#for i in range(4):
#    dy_87[:,i] = dy_87[:,i] / (y*np.log(10.))
kappa_87_samples = random.multivariate_normal(mu, C, size = (num_sam,))

#mu = np.array(opt_params['F1_aw_91']['opt_mean'][:4])
#print mu

#mu = np.array([-0.49759826,  0.31892818,  0.40055088, -0.76767675 ])

#mu = np.array([0.07876292 , 0.34721592 , 0.23821628, -0.60814052]) # comp 2
#mu = np.array([-0.07271478,  0.24874828,  0.53192293, -0.9192804])
#mu = np.array([-0.08836538,  0.30782519,  0.36213571, -0.74058935]) # maybe use
it as comp 1

"""Mixture of Gaussians"""

#mu1 = np.array([-0.57598059,  0.3130815,   0.48927411, -0.8119115]) # comp 3
#mu2 = np.array([-0.24830812,  0.34791964,  0.23141949, -0.64387795])
#mu3 = np.array([-0.84231204,  0.33795978,  0.41195735, -0.77015655])
#mu4 = np.array([-0.47935876,  0.38617573,  0.21738956, -0.57196846])

#C1 = np.array([1.71071834e-02, 7.37222226e-05, 1.11004609e-02, 7.45055528e-04])
#C2 = np.array([ 2.10909258e-02, 6.25775624e-05, 1.11255446e-02, 4.57430436e-04])
#C3 = np.array([1.28689674e-02,  6.62485448e-05, 1.05727284e-02, 6.33046252e-04])
#C4 = np.array([1.87164674e-02,  6.35655510e-05, 1.14997535e-02, 4.40016551e-04])


#all_mu = [mu1, mu2, mu3, mu4]
#all_C = [C1, C2, C3, C4]
#w = st.randint.rvs(0,4, size = (50000,))
#x1 = np.zeros(w.shape)
#x2 = np.zeros(w.shape)
#x3 = np.zeros(w.shape)
#x4 = np.zeros(w.shape)
#for i in range(w.shape[0]):
#    x1[i] = st.norm.rvs(loc = all_mu[w[i]][0], scale = np.sqrt(all_C[w[i]][0]))
#    x2[i] = st.norm.rvs(loc = all_mu[w[i]][1], scale = np.sqrt(all_C[w[i]][1]))
#    x3[i] = st.norm.rvs(loc = all_mu[w[i]][2], scale = np.sqrt(all_C[w[i]][2]))
#    x4[i] = st.norm.rvs(loc = all_mu[w[i]][3], scale = np.sqrt(all_C[w[i]][3]))
```

```
#kappa_91_samples = np.vstack([x1, x2, x3, x4]).T


#C = np.diag(np.array([1.80806989e-02, 7.36426324e-05, 1.13096013e-02,
6.45864416e-04]))
#print np.sqrt(C)
#C = np.diag(np.array(opt_params['F1_aw_91']['opt_var'][:4])**2)
#y = f(np.exp(mu), y_91, t).reshape((t.shape[0],4))[:,0] + f(np.exp(mu), y_91,
t).reshape((t.shape[0],4))[:,1]
#dy_91 = df(np.exp(mu), y_91, t).reshape((t.shape[0],4,4))[:,0,:] +
df(np.exp(mu), y_91, t).reshape((t.shape[0],4,4))[:,1,:]
#for i in range(4):
#    dy_91[:,i] = dy_91[:,i] / (y*np.log(10.))
#kappa_91_samples = random.multivariate_normal(mu, np.sqrt(C), size = (num_sam,))



#samples_out_79 = np.zeros((t.shape[0],num_sam))
#samples_out_84 = np.zeros((t.shape[0],num_sam))
samples_out_87 = np.zeros((t.shape[0],num_sam))
#samples_out_91 = np.zeros((t.shape[0],num_sam))


for i in range(num_sam):
#    print 'Taking sample no: ' + str(i)
#    kappa_79 = np.exp(kappa_79_samples[i,:])
#    samples_out_79[:,i] = np.log10(f(kappa_79, y_79,
t).reshape((t.shape[0],4))[:,0] + f(kappa_79, y_79,
t).reshape((t.shape[0],4))[:,1])
#    kappa_84 = np.exp(kappa_84_samples[i,:])
#    samples_out_84[:,i] = np.log10(f(kappa_84, y_84,
t).reshape((t.shape[0],4))[:,0] + f(kappa_84, y_84,
t).reshape((t.shape[0],4))[:,1])
    kappa_87 = np.exp(kappa_87_samples[i,:])
    samples_out_87[:,i] = np.log10(f(kappa_87, y_87,
t).reshape((t.shape[0],4))[:,0] + f(kappa_87, y_87,
t).reshape((t.shape[0],4))[:,1])
#    kappa_91 = np.exp(kappa_91_samples[i,:])
#    samples_out_91[:,i] = np.log10(f(kappa_91, y_91,
t).reshape((t.shape[0],4))[:,0] + f(kappa_91, y_91,
t).reshape((t.shape[0],4))[:,1])


#Y_m_79 = np.percentile(samples_out_79, 50, axis=1)
#Y_p05_79 = np.percentile(samples_out_79, 5, axis=1)
#Y_p95_79 = np.percentile(samples_out_79, 95, axis=1)

#Y_m_84 = np.percentile(samples_out_84, 50, axis=1)
#Y_p05_84 = np.percentile(samples_out_84, 5, axis=1)
#Y_p95_84 = np.percentile(samples_out_84, 95, axis=1)


#Y_m_91 = np.percentile(samples_out_91, 50, axis=1)
#Y_p05_91 = np.percentile(samples_out_91, 5, axis=1)
#Y_p95_91 = np.percentile(samples_out_91, 95, axis=1)

#np.save('Y_m_84_3.npy', Y_m_84)
#np.save('Y_p05_84_3.npy', Y_p05_84)
#np.save('Y_p95_84_3.npy', Y_p95_84)

#Y_m_84_1 = np.load('./quasi_chem_results/Y_m_84_1.npy')
#Y_m_84_2 = np.load('./quasi_chem_results/Y_m_84_2.npy')
#Y_m_84_3 = np.load('./quasi_chem_results/Y_m_84_3.npy')
#Y_p05_84_1 = np.load('./quasi_chem_results/Y_p05_84_1.npy')
#Y_p05_84_2 = np.load('./quasi_chem_results/Y_p05_84_2.npy')
#Y_p05_84_3 = np.load('./quasi_chem_results/Y_p05_84_3.npy')
#Y_p95_84_1 = np.load('./quasi_chem_results/Y_p95_84_1.npy')
#Y_p95_84_2 = np.load('./quasi_chem_results/Y_p95_84_2.npy')
```

```
#Y_p95_84_3 = np.load('./quasi_chem_results/Y_p95_84_3.npy')

Y_m_87 = np.percentile(samples_out_87, 50, axis=1)
Y_p05_87 = np.percentile(samples_out_87, 5, axis=1)
Y_p95_87 = np.percentile(samples_out_87, 95, axis=1)

#Y_m_91 = np.percentile(samples_out_91, 50, axis=1)
#Y_p05_91 = np.percentile(samples_out_91, 5, axis=1)
#Y_p95_91 = np.percentile(samples_out_91, 95, axis=1)

fig = plt.figure()
ax = fig.add_subplot(111)

#ax.plot(t, Y_m_79, 'b', linewidth=2, markersize=5., label = 'Mean prediction')
#ax.plot(t, Y_p05_79, 'b', linewidth = 0.5)
#ax.plot(t, Y_p95_79, 'b', linewidth = 0.5)
#ax.fill_between(t, Y_p05_79, Y_p95_79, color = 'gray', alpha=0.4)
#ax.plot(data_79[:,0], data_79[:,1], 'k.', markersize=8, label = 'Data')
#ax.plot(t, Y_m_84, 'b', linewidth=1.5, markersize=5., label = 'Mean prediction')
#ax.plot(t, Y_p05_84, 'b', linewidth = 0.2)
#ax.plot(t, Y_p95_84, 'b', linewidth = 0.2)
#ax.fill_between(t, Y_p05_84, Y_p95_84, color = 'gray', alpha=0.4)
#ax.plot(t, Y_m_84_2, 'b', linewidth=1.5, markersize=5., label = 'Mean
prediction')
#ax.plot(t, Y_p05_84_2, 'b', linewidth = 0.2)
#ax.plot(t, Y_p95_84_2, 'b', linewidth = 0.2)
#ax.fill_between(t, Y_p05_84_2, Y_p95_84_2, color = 'gray', alpha=0.4)
#ax.plot(t, Y_m_84_3, 'b', linewidth=1.5, markersize=5., label = 'Mean
prediction')
#ax.plot(t, Y_p05_84_3, 'b', linewidth = 0.2)
#ax.plot(t, Y_p95_84_3, 'b', linewidth = 0.2)
#ax.fill_between(t, Y_p05_84_3, Y_p95_84_3, color = 'gray', alpha=0.4)
#ax.plot(data_84[:,0], data_84[:,1], 'k.', markersize=8, label = 'Data')
ax.plot(t, Y_m_87, 'b', linewidth=2, markersize=5., label = 'Mean prediction')
ax.plot(t, Y_p05_87, 'b', linewidth = 0.5)
ax.plot(t, Y_p95_87, 'b', linewidth = 0.5)
ax.fill_between(t, Y_p05_87, Y_p95_87, color = 'gray', alpha=0.4)
ax.plot(data_87[:,0], data_87[:,1], 'k.', markersize=8, label = 'Data')
#ax.plot(t, Y_m_91, 'b', linewidth=2, markersize=5., label = 'Mean prediction')
#ax.plot(t, Y_p05_91, 'b', linewidth = 0.5)
#ax.plot(t, Y_p95_91, 'b', linewidth = 0.5)
#ax.fill_between(t, Y_p05_91, Y_p95_91, color = 'gray', alpha=0.4)
#ax.plot(data_91[:,0], data_91[:,1], 'k.', markersize=8, label = 'Data')

## ------------ Sensitivities ----------------
#ax.plot(t, dy_91[:,0], label = 'k1')
#ax.plot(t, dy_91[:,1], label = 'k2')
#ax.plot(t, dy_91[:,2], label = 'k3')
#ax.plot(t, dy_91[:,3], label = 'k4')
ax.set_title('Aw = 0.87', fontsize = 18)
ax.set_xlabel('Time (days)', fontsize=18)
ax.set_ylabel('Log-concentration', fontsize=18)
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.get_xaxis().tick_bottom()
ax.get_yaxis().tick_left()
for tick in ax.xaxis.get_major_ticks():
    tick.label.set_fontsize(16)
for tick in ax.yaxis.get_major_ticks():
    tick.label.set_fontsize(16)
plt.legend(loc = 2)
plt.show()
```

APPENDIX C

BACTERIA COUNT VARIABILITY

A cause of uncertainty in the results from a microbiological experiment is uncertainty in organism enumeration; that is, variability in estimating the number of bacteria in a sample. Bacteria within an inoculated host are too small, and often too numerous to count directly; the true number of bacteria within an experimental sample is never known. Instead, microbiologists have developed methods and procedures for estimating the number of bacteria in a sample or host.

An example of an industry standard bacterial test is the American Association of Textile Chemists and Colorists (AATCC) Test Method 100 (TM 100) described in Reference [ii]. The TM 100 is a test for evaluating antibacterial treatments to fabrics, and is regularly conducted by the Biological Sciences Team (BST) at NSRDEC. The TM 100 test as conducted by the BST consists of inoculum preparation, sample preparation, and enumeration. A sample of microorganism stock obtained from the American Type Culture Collection (ATCC) is grown in a nutrient broth, then diluted to a concentration of approximately 10^7 colony forming units per milliliter (CFU/mL) and resuspended to form the test inoculum. Control and/or antimicrobial treated fabric swatches are inoculated and allowed to grow in a controlled environment for a period of time ranging from 0h to 30h. After the growth stage, the microorganisms on a fabric swatch are recovered into a solution using agitation, and a sample of the solution is extracted and prepared for enumeration. The extracted samples are diluted to two or three concentrations and then used to either inoculate an agar plate for spread plate enumeration, or injected into a test card for machine enumeration. Each enumeration method has limits in which there can be either too few or too many organisms to count; multiple dilutions are used to increase the likelihood that samples fall between these limits. After plate/card preparation, the samples are incubated for approximately 24 hours and then enumerated. Two enumeration methods are used by the BST.

An enumeration method defines both how a sample is prepared for enumeration as well as how the number of microorganisms are counted. **Spread plate** enumeration involves growing bacterial colonies on an agar-filled plate. Each bacteria in the sample is assumed capable of growing into a colony, and the type of agar is chosen to support growth of the specific microorganism. A known amount of the sample solution is deposited in the center of the plate, and a bent glass rod is used by the tester to distribute the bacteria-filled liquid uniformly across the agar. The plate is then sealed, numbered, and placed in an incubator. After incubation, the BST uses an Advanced Instruments, Inc. QCount Colony Counter for the enumeration counts. The QCount takes an illuminated image of the plate face and displays it on a computer screen. The tester then uses a cursor to identify each colony in the image, and the QCount outputs the total count and microorganism concentration.

The **Tempo machine** is an instrument manufactured by bioMérieux that provides a method of enumeration that is almost completely automated. A bacterial sample is added to a microorganism specific reagent that fluoresces in the presence of the microorganisms. The reagent is then agitated and used to fill a Tempo card which has 16 wells at three different

dilutions. The cards are then incubated in the Tempo machine. After incubation, the Tempo machine counts the number of wells that fluoresce (i.e. positive wells), and calculates a quantity called the Most Probable Number (MPN) which is related to the microorganism concentration.

Although the accuracy of enumeration results is difficult to quantify, variability in the results has been studied. Reference [jj] describes the results from a study of standard plate counting errors when counted twice by one analyst and once by each of two analysts. In the study, 16 samples of pasteurized milk were prepared and plated in duplicate. Four analysts visually counted each of the 32 plates twice. The analysis looked at the variability of one analyst counting the same plate twice; two analysts counting the same plate; and one analyst counting duplicate plates. Variability was based on absolute percentage errors of the base 10 logarithm of the recorded counts. The results of the study showed that at a 0.1 level of significance, the variability of one analyst counting the same plate twice was 6%; the variability of two analysts counting the same plate was 8%; and the variability of an analyst counting the duplicate plates was 31%.

Data collected by BST using both spread plate and Tempo enumerations were obtained by Applied Mathematics, Inc. The data were obtained from three different TM 100 tests conducted at NSRDEC. For spread plate results, 22 samples were enumerated twice with a mean percentage variability between the enumerations of 12%. For Tempo results, 119 samples were enumerated twice with a mean percentage variability between the enumerations of 41%.