Naval Research Laboratory

Stennis Space Center, MS 39529-5004

NRL/MR/7320--10-9209

Software Design Description for the Tidal Open-boundary Prediction System (TOPS)

Scott R. Smith
Pamela G. Posey
Paul J. Martin
Gretchen M. Dawson
Clark Rowley

Ocean Dynamics and Prediction Branch
Oceanography Division

Suzanne N. Carroll

QinetiQ North America
Stennis Space Center, Mississippi

May 4, 2010

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
04-05-2010

2. REPORT TYPE
Memorandum Report

3. DATES COVERED *(From - To)*

4. TITLE AND SUBTITLE
Software Design Description for the Tidal Open-boundary Prediction System (TOPS)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
0601153N

6. AUTHOR(S)
Scott R. Smith, Pamela G. Posey, Paul J. Martin, Gretchen M. Dawson, Clark Rowley, and Suzanne N. Carroll*

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
73-5097-B9-5

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Research Laboratory
Oceanography Division
Stennis Space Center, MS 39529-5004

8. PERFORMING ORGANIZATION REPORT NUMBER
NRL/MR/7320--10-9209

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Space & Naval Warfare Systems Command
2451 Crystal Drive
Arlington, VA 22245-5200

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

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

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
*QinetiQ North America, Stennis Space Center, Mississippi

14. ABSTRACT
The Tidal Open-boundary Prediction System (TOPS) is a tool that provides tidal predictions for the open boundaries of the Navy Coastal Ocean Model (NCOM). This system predicts tides by assimilating International Hydrographic Office (IHO) tide gauge and TOPEX altimetry data with shallow water dynamics using four-dimensional variational analysis (4DVAR). TOPS is based on the Oregon State University (OSU) Tidal Inversion Software (OTIS) package, which was simplified, automated, and transitioned to operate within the setup of the Relocatable NCOM system. This report describes the design of the TOPS software package.

15. SUBJECT TERMS
Tidal prediction, Satellite altimetry, 4DVAR assimilation, Ocean modeling

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
Unclassified	Unclassified	Unclassified	UL	63	Scott Smith
					19b. TELEPHONE NUMBER *(include area code)* (228) 688-4630

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

TABLE OF CONTENTS

TABLE OF FIGURES

1.0 SCOPE

1.1 Identification

Tides are an integral part of the variability of sea-surface height (SSH) and currents in coastal and shallow-water areas and are, therefore, an important component of coastal ocean modeling. Oregon State University (OSU) has made available a global database and a number of regional tidal databases, with grid resolutions of 1/4° for the global database and 1/12° for most of the regional databases. The databases were created using the OSU Tidal Inversion Software (OTIS), which assimilates SSH derived from satellite altimetry into a solution of the barotropic, shallow-water equations for a specified domain (Egbert et al., (1994); Egbert and Erofeeva, (2002)), using four-dimensional variational analysis (4D-VAR) (Egbert and Ray, 2003). OTIS consists of three fundamental components: the data, the ocean dynamics, and the assimilation tools to optimally combine the data and dynamics. The OTIS package includes software for generating grids, prior model covariances, and boundary conditions. It also has formulations for time stepping the non-linear shallow water equations and for tidal processing of TOPEX/POSEIDON altimeter data. OTIS was initially developed in 1994 by Gary Egbert at OSU (Egbert et al., 1994) and has since gone through several revisions and upgrades (Egbert, 1997; Egbert and Ray, 2001; Egbert and Erofeeva, 2002; Egbert and Ray, 2003; Erofeeva et al., 2003). Despite OTIS's longevity and relative robustness, the system is fairly cumbersome and complicated to use. OTIS contains a significant number of options and parameters that need to be specified by the user, and an understanding of 4D-VAR assimilation is necessary to properly set them.

The goal of developing the Naval Research Laboratory's Tidal Open-boundary Prediction System (TOPS) is to use OTIS to improve the tidal solutions generated for the regional modeling areas of Navy interest. These include using (1) higher grid resolution, (2) better bathymetry, (3) additional tidal data for assimilation, and (4) inclusion of additional tidal constituents. TOPS has improved upon the OTIS approach by automating, and thus simplifying, most of the user-required inputs, thus reducing the amount of effort required to run OTIS. The TOPS will provide tidal boundary conditions (TBCs) for the relocatable Navy Coastal Ocean Model (RELO NCOM) instead of extracting and interpolating tides from a larger and coarser database. The RELO NCOM system automatically runs the modified OTIS package and computes the tides at the same resolution as the specified NCOM domain, providing more accurate tides within the RELO NCOM domain. Since the RELO NCOM system will be run operationally in shallow marginal seas and littoral regions, accurate and high-resolution TBCs will be critical. Running OTIS at higher resolutions for these types of regions has been demonstrated to significantly improve the estimation of tides (Martin et al., 2009). Portions of the OTIS code were modified and new code added to automatically compute optimal values of several parameters that vary with the type of region and resolution used. TOPS also has improved the grid resolution from the 1/12° resolution typically used by OTIS for regional areas to the same resolution set for RELO NCOM, which can be up to 1/160° . This serves to better resolve the spatial variation of the tide in many coastal areas relative to the original OTIS tidal databases.

Although OTIS relies greatly on data assimilation, the propagation of the tides within the OTIS tidal model depends strongly on the bathymetry. Therefore, tidal solutions in shallow coastal areas, where there may not be much data to assimilate, are dependent on accurate bathymetry. Most

Manuscript approved October 15, 2009.

widely available bathymetry databases do not provide accurate bathymetry for many coastal areas of the world's oceans. Therefore TOPS will use the same bathymetry as the RELO NCOM domain, which will most likely come from the highest-resolution bathymetry database available. The prinary bathymetry database currently used at NRL is DBDB2, where the "2" refers to the database's 2-minute (1/30°) resolution. For even higher-resolution grids, the National Oceanic and Atmospheric Administration (NOAA) has gridded bathymetries up to 3-seconds for U.S. coastal areas, and the World Vector Shoreline (WVS) offers high resolution sounding data. Note that TOPS will be linked to RELO NCOM, so if RELO NCOM is run on the classified systems at the Naval Oceanographic Office (NAVOCEANO), then TOPS will have access to the NAVOCEANO classified bathymetry databases. Bathymetry data are usually referenced to low tide and the NRL ocean models are referenced to mean SSH, so a correction must be made to the bathymetry before a simulation is run. This can be done using a tidal database (TDB) to add the depth due to the amplitude of the local low tide to the bathymetry values.

TOPS has incorporated the use of tidal SSH data in coastal areas from the tide station database of the International Hydrographic Office (IHO). This database consists of tidal SSH data from over 4500 tide gauge stations scattered about the coastal areas of the world's oceans. While Relo NCOM can use TDBs with 10 constituents for the global domain and four to eight for regional seas, the current version of TOPS will generally use just the four largest tidal constituents, because a large number of the tide gauges that are assimilated into the system do not have some of the smaller constituents.

1.2 Document Overview

The purpose of this Software Design Description (SDD) is to describe the model layout and code of the Tidal Open-boundary Prediction System (TOPS). It includes descriptions of the TOPS physics, programs, subprograms, and common blocks. This document is accompanied by a Validation Test Report (VTR) (Smith et al., 2009). Instructions for TOPS use will be incorporated into the User's Manual for the Relocatable Navy Coastal Ocean Model (RELO NCOM), to be published in the near future. However, since the TOPS is mostly automated, instructions in the RELO NCOM User's Manual will be minimal.

2.0 REFERENCED DOCUMENTS

2.1 TOPS Software Documentation

Martin, P.J., Smith, S.R., Posey, P.G., Dawson, G.M., Riedlinger, S.H., (2009). Use Of OTIS To Generate Improved Tidal Predictions In The EAS., *NRL Memo. Rpt*., NRL/MR/7320-09-9176.

Smith, S.R., Posey, P.G., Martin, P.J., Dawson, G.M., Rowley, C.D., Carroll, S.N., (2009). Validation Test Report for the Tactical Open-boundary Prediction System (TOPS) Version 1.2., *NRL Memo. Rpt.*, NRL/MR/7320—09-xxxx.

2.2 General Technical References

Bennett, A.F., (2002). Inverse Modeling of the Ocean and Atmosphere. Cambridge University Press, Cambridge.

Egbert, G., Bennett, A., Foreman, M. (1994). TOPEX/Poseidon tides estimated using a global inverse model. *J. of Geophys Res*., 99(C12): 24821-24852.

Egbert, G. (1997). Tidal data inversion: interpolation and inference. *Prog. Oceanog.*, 40: 53-80.

Egbert, G., and Ray, R., (2001). Estimates of M2 tidal energy dissipation from TOPEXPOSEIDON altimeter data. *J. Geophys. Res*., 106(C10): 22,475-22,502.

Egbert, G.D. and Erofeeva, S. (2002). Efficient inverse modeling of barotropic ocean tides, *J. of Ocean and Atmos. Tech*, 19(2): 183–204.

Egbert, G., and Ray, R., (2003). Semi-diurnal and diurnal tidal dissipation from TOPEXPOSEIDON altimetry. *Geo. Res. Let*., 30(17).

Erofeeva, S., Egbert, G., and Kosro, P., (2003). Tidal currents on the central Oregon shelf: Models, data and assimilation. *J. Geophy. Res*., 108(C5): 3148.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P. (1986). Numerical Recipes. Cambridge, Mass: Cambridge Univ. Press.

Zaron, E.D. and Egbert, G.D. (2006). Estimating Open-Ocean Barotropic Tidal Dissipation: The Hawaiian Ridge. *J. of Phys. Oc*., 36: 1019-1035.

3.0 TOPS SOFTWARE SUMMARY

3.1 Memory Allocation and Code Specifications

OTIS was originally written in Fortran 77, Fortran 90, and Matlab and runs in a UNIX environment. The MatLab programs are not currently used in TOPS. TOPS is currently running on single-processor, 64-Bit Opterons with 16GB of memory. However, the software can be configured and compiled for other platforms. A standard benchmark OTIS run, such as in the East Asian Seas (EAS), uses a domain extending from -17.5 to 53°N and 97.5 to 159°E, with 700 x 507 grid points, and solves for four tidal constituents. This run takes approximately 26 hours to run using a LINUX workstation (a dual-core, Advanced Micro Devices (AMD), 2.4 gHz Opteron with 16 GB of RAM). In this document, the references to OTIS and TOPS are virtually interchangeable, as TOPS is essentially a simplified, automated version of the original OTIS software.

3.2 Code Modifications

Many code modifications have been made from the original OTIS programs. The following is a brief summary:

- Updated the TOPEX altimetry database to encompass the newly processed NRL database that includes the entire TOPEX database and about six years of data from the follow-on Jason-1 satellite. Jason-1 has the same orbit as TOPEX, so their data are in alignment and can easily be appended. This brings the time series of data to 16 years, thus improving the accuracy of extracting the tidal amplitudes and phases through harmonic analysis.
- Updated the default global bathymetry database to the NRL standard DBDB2.
- Included the assimilation of tide gauge data from the global IHO database. A module was created to automatically search this database and extract and use the appropriate tide gauge data that fall within the specified domain and pass certain quality control checks.
- All original Matlab GUIs were replaced with automated Fortran code. These include:
  - GridEdit: This Matlab GUI allowed for manual specification of domain dimensions and resolution, as well as selection of open boundary points and conditions. Unwanted water grid points (such as land-locked water mases and small bays) could also be manually masked out. GridEdit's functions were automated into a subroutine called **otis_setup.x**, which gathers all necessary grid information from NCOM, selects the appropriate open boundary points, and masks out bodies of water that are too negligible to be resolved by the model.
  - RepEdit: This feature allowed manual selection of data locations and representer locations (See Section 5.2.2 for an explanation of representers.) It has been replaced with **lat_lon.f,** which extracts all available altimetry and IHO data from their global database and determines representer locations using a sophisticated algorithm to maximize the influence that the data will have in the assimilation. The **lat_lon.f** algorithm determines the optimal

locations of representers to be: 1) all IHO station locations, 2) all TOPEX cross-over points where the ascending and descending altimeter passes meet, and 3) TOPEX along-track points in shallow water.

- Invert: This Matlab GUI allowed users to examine performance curves associated with the assimilation, manipulate two critical assimilation parameters, and rerun the assimilation. The process was cumbersome, inefficient, and required numerous iterations before obtaining an appropriate solution. The first assimilation parameter to be automated was the truncation parameter. The truncation parameter first ranks representer functions using eigenvalue decomposition according to their level of influence and then specifies how many of the top influencial representers should be retained in the assimilation. The rest are discarded. A primary limiting factor of how large the truncation parameter can be is based on available RAM. Code was written to assess the amount of available RAM and uses this value to set the maximum size of the truncation parameter. The second assimilation parameter that was automated is the damping parameter, which controls the relative fit between the dynamics and data within the cost function. It was automated using the generalized cross-validation function of Zaron and Egbert (2006). This parameter is discussed in Section 5.2.7.

4.0 TOPS SOFTWARE INVENTORY

4.1 TOPS Components

4.1.1 TOPS Routines

atgf.f, blockgen.f, BSI_weights.f, CDG.f, checklim.f, constit.f, covsc_in.f, create_A.f, crossdat.f, dcomb.f, def_cid.f, def_form.f, delta.f, diffuse.f, ds_subs.f, filter_outliers.f, fwd_fac.f, glob_case.f, glob_case_c.f, gsmooth.f, h_uv.f, height.f, iflag16.f, inner.f, interp_rpx.f, interpSAL.f, ipshft.f, j_days.f, lat_lon.f, loadModel.f, loadModel_uv.f, lp_tide.f, lteco.f, make_a.f, makeB.f, makedat.f, makeE.f, makeE_fwd.f, mix_ave.f, mkwts.f, modelcov.f, nodal.f, obstime.f, openfiles.f, otis_comp_iho2.f, otis_ setup.f, out_file_init.f, param_subs.f, pe_subs.f, r_sites.f, rd_c_alpha.f, rd_com_line.f, rd_num.f, read_adcp.f, read_b.f, read_cm.f, read_rad.f, read_tg.f, readTpxo.f, reduce_b.f, repx.f, rlc.f, rpx_to_p.f, rtloadtopex.f, SALset.f, Sfac.f, topexinit.f, varest.f, write_cm.f, write_rad.f, write_tg.f, writeTpxo.f, wrt_uvsc.f.

4.1.2 TOPS Common Blocks

common/cflag, common/cfnamersr, common/cmission, common/constrsr8, common/constsi2, common/constsi4, common/cunits, common/datablk, common/rmultblk

4.2 TOPS Software Organization and Implementation

4.2.1 Directory Structure

The model code directory contains all of the files needed to generate the TOPS executable.

OTIS/

- bin/- Directory containing several of the TOPS executable(s) that must be compiled prior to running the code, initialization routines and the subroutines used to create these executables. Executables cannot be made for all source code before running.

- crd- Main runscript
- DB/- Databases, including the global OTIS solution (for OBCs), a global tidal forcing correction for ocean self-attraction/loading (the forward tide model), and a correction for radial deformation load tides for (for altimetry data).
  - topex/ - Global TOPEX/Jason-1 database used for assimilation.
- do_not_touch/- A makefile and initial header and parameter files. Every time the runscript (`crd`) is executed, a run folder is created in the 'local' directory and the contents of this 'do_not_touch' directory are copied into the run folder and used in the runscript.
- local/-
  - Experiment_Name/ - Created by the runscript. Its name is specified with the calling of the runscript (‘`crd[experiment_name][number of representers]`’).
    - dat/ - temporary data files created throughout the operation of the assimilation experiment.

exe/- Executables compiled explicitly for the designated experiment, the primary parameter file, and the log files generated by the executables.

include/ - Header files automatically copied to this directory from the ‘do not touch’ directory. They are periodically updated through the execution of the experiment.

out/ - Contains tidal solutions from the initial forward model and the final assimilation solution.

prm/ - Parameter files both copied from the ‘do not touch’ directory and created through the execution of the experiment.

repx/ - Data files containing the representer solutions.

src/- TOPS source code.

fwd_ts/ - Code that handles the operation of the forward tide model.

mkb/ - Code that handles and prepares the data to be assimilated.

rp_dp/ - Code that primarily handles the assimilation.

4.2.2 Concept of Execution

In the top OTIS folder there is a UNIX script **crd** that when called goes through and executes all of the various pieces of the software. This script is automatically called within the setup of RELO NCOM. All of the input parameters, such as grid information, bathymetry, etc., are directly pulled from RELO NCOM.

The **crd** runstream script requires two inputs when called (`crd[experiment_name][number of representers]`) and is outlined below:

1) Create experiment directory (with associated subdirectories) in ‘local’ and copy the necessary files from ‘do_not_touch’ into this directory hierarchy.
2) run `/bin/otis setup.1x` This is the set-up program that defines the domain to be run, sets up the grid and the bathymetry, defines the tidal constituents to be used, defines the IHO data to be used for assimilation, and sets up the BCs for running the forward model.
3) run `lat_lon`. This sets up altimetry representer and data lists.
4) run `make all.` This step makes the OTIS programs for the local domain, which are hard-wired for the grid dimensions and tidal constituents selected (../exe/). Therefore, this make needs to be redone every time a new region is set up.
5) run `fwd fac` to get a prior solution by running forward tidal model.
6) run `diffuse.` It computes the error covariance scales.

7) run `varest.` This step makes the OTIS covariance file (../prm/covsc).

8) run `repx` to compute the representers.

9) run `makedat` to make the altimetry data set.

10) run `makeB` to perform a harmonic analysis on the TOPEX data and create the “reduced” altimetry data set for assimilation (../dat/B.dat).

11) run `makeB –a –D../prm/iho data dat –t` to append the B.dat data file with the IHO data set.

12) run `rpx to p` and `rpx to p -r` to make the spatial representer matrices.

13) run `reduce_b` to calculate the representer coefficients and other data files needed for the assimilation.

14) `reduce_b -b –q` is rerun with a different option in order to compute the optimal damping parameter.

15) run `rlc`, which executes the tidal data assimilation model and creates optimal solutions.

16) run `/bin/otis_comp_iho2.1x`. This inspects the OTIS output and computes mean and RMS errors with respect to the tidal data at the IHO stations.

5.0 TOPS DETAILED DESIGN

All routines are written in FORTRAN 90.

The following paragraphs give a detailed description of the purpose, variables, logic, and constraints for the TOPS. Descriptions of the common blocks are found in Section 7.0. Argument definitions for some of the most common subroutine variables are found in Section 8.0.

5.1 Constraints and Limitations

TOPS is a fully automated system, thus minimizing limations to its functionality. Code exists to automatically adjust many of the assimilation parameters to ensure that the system completes operation successfully and in a timely manner, regardless of the domain size. However, there are a few limitations of the model. Two issues that may impact the accuracy of this system are:

1) The size of the domain. As the size of the domain increases, the grid resolution decreases and the estuaries, bays, and near-shore locations where the majority of tide gauges are located are not as well defined.
2) The accuracy of the bathymetry has a significant impact on the accuracy of this system (this has been verified through experiments). Currently DBDB2 bathymetry is the default, but if a higher resolution data set is available it should be used.

5.2 Logic and Basic Equations for OTIS/TOPS

A report by Martin et al., (2009) gives an overview of the physics and design of OTIS and TOPS. The code is currently being updated to be more automated and require much less user interaction.

5.2.1 OTIS Assmimilation Method

OTIS has the potential for assimilating data from a wide variety of sources, including satellite altimetry, tide gauges, current meters, Coastal Ocean Dynamics Application Radar (CODAR), and Acoustic Doppler Current Profilers (ADCPs). In this TOPS version, however, only SSH data from TOPEX/Poseidon and IHO tide gauge databases are used. This is primarily because these two databases are global, therefore simplifying the overall use of OTIS within the relocatable NCOM system.

Because domains are almost always under-sampled, the OTIS assimilation method includes ocean dynamics to disseminate information from the data locations to the entire domain. Thus, the assimilation method establishes the optimal tidal solution for the full domain that complies with the tidal dynamics and simultaneously gives the best overall fit to the assimilated observations. To describe the dynamics of the tides, OTIS employs the linearized shallow water equations:

$$\frac{\partial U}{\partial t} - fV + gH\frac{\partial(\zeta - \zeta_{SAL})}{\partial x} + \kappa U = \mathrm{f}_U, \tag{1}$$

$$\frac{\partial V}{\partial t} - fU + gH\frac{\partial(\zeta - \zeta_{SAL})}{\partial y} + \kappa V = \mathrm{f}_V, \tag{2}$$

$$\frac{\partial \zeta}{\partial t} = -\left(\frac{\partial U}{\partial x}, + \frac{\partial V}{\partial y}\right), \tag{3}$$

where *U* and *V* are the two components of the barotropic transport (i.e., the depth-averaged velocity times the depth H), *f* is the Coriolis parameter, t is the time, x and y are the distance in the two horizontal coordinate directions, *g* is the acceleration of gravity, ζ is the SSH, and ζ_{SAL} represents the tidal loading and self-attraction. The last term on the left-hand side (LHS) of the first two equations is the linearized bottom drag (κ is a dissipation coefficient) and the terms on the right-hand side (RHS) represent the earth's body tide.

TOPS can also use nonlinear dynamics, which include advection and nonlinear bottom drag. Comparisons with the use of the linearized equations have shown that the inclusion of nonlinear physics significantly increases the computation cost and the resulting accuracy is only slightly, if at all, better. The linearized OTIS dynamics have a simple transformation from the time domain into the frequency domain using Fourier Transforms. With modest manipulation, the equations above can be expressed with the following time-independent equations:

$$\nabla \cdot gH\,\Omega^{-1}\nabla\zeta - i\omega\zeta = \nabla \cdot \Omega^{-1}\mathrm{f}_U - \mathrm{f}\zeta \tag{4}$$

$$\mathrm{U} = -\,gH\,\Omega^{-1}\nabla\zeta + \Omega^{-1}\mathrm{f}_U, \tag{5}$$

where $\Omega = \begin{bmatrix} i\omega+\kappa & -f \\ f & i\omega+\kappa \end{bmatrix}$ *and* $\mathrm{U}=\begin{bmatrix} U \\ V \end{bmatrix}$. When OTIS solves the forward shallow water equations, equation 4 is used in conjunction with open boundary conditions from an OTIS TDB and bathymetry from DBDB2 to compute the phase and amplitude of SSH (ζ) at each grid point. With SSH known, the momentum equation (Eq. 5) can then be used to solve for the phase and amplitude of both transport components (*U* and *V*) at each grid point.

5.2.2 The Representer Method

OTIS uses a modified or reduced basis representer approach to solve variational assimilation issues, as in Egbert et al., (1994). That is, representers are calculated for a subset of data locations and a solution to the variational problem is sought within the space of linear combinations of calculated representers. With this approach it is quite feasible to fit all available altimetry data in a modest sized area (e.g., 20 degrees by 20 degrees, with data at several thousand locations) using a few hundred representers per constituent. In this version, representers are calculated by solving the linearized frequency domain shallow water equations (SWE) after factoring the matrix of coefficients for the elevation wave equation (Egbert and Erofeeva, 2002). This approach allows for a decrease in computational time by a factor of 100 or more. Combined with the reduced basis representer approach, this allows for the solution of moderate size problems on a high-end workstation.

A variational assimilation problem requires finding a solution ($\mathbf{u}$) that minimizes a cost function ($J[\mathbf{u}]$).

$$J[\mathbf{u}] = (\mathrm{L}\mathbf{u} - \mathbf{d})^T \textstyle\sum_e^{-1} (\mathbf{Lu} - \mathbf{d}) + (\mathbf{Su} - \mathbf{f}_0)^T \sum_f^{-1} (\mathbf{Su} - \mathbf{f}_0), \tag{6}$$

where

- $\mathbf{L}$: Measurement functional that maps variables from data space to state space
- $\mathbf{d}$: Data
- $\sum_e^{-1}$: Measurement error covariance
- $\mathbf{S}$: Model
- $\mathbf{f}_0$: Forcing to model
- $\sum_f^{-1}$ Model error covariance

The solution that minimizes this cost function also best satisfies all data (1st term) and all dynamics (2nd term) simultaneously.

OTIS can alternate between the time domain and the frequency domain. Most computations, including the representers, are performed in the frequency domain. In describing the time domain, assume that there is a measurement of SSH at location (X,Y) and at time T. This measurement must be assimilated into an ocean model whose domain and time period encompass this measurement (X0 <= X <= Xm, Y0<=Y<=Yn, T0<=T<=Tt). The first step is to take the measurement's location (X,Y,T) (the actual measurement isn't used yet), and insert an impulse of SSH (L_{SSH}) there. The impulse maps the distance from data space to the discretized model space. If the measurement's location is simply moved to the nearest grid point, then the measurement functional can just be a 1 (OTIS does this). The influence of this impulse is then propagated backwards in space and time via the transpose of the model dynamics

$$\mathbf{S}^T \lambda_k = L_k, \tag{7}$$

which is referred to as the adjoint. In the above equation, $\mathbf{S}$ and λ_k represent the model and adjoint variables, respectively. As the information from this impulse propagates backwards to the initial condition of the model, the impulse's influence spreads throughout the domain and into other state variables via the dynamics of the model. At the initial condition, all influence information is convolved with the model's error covariance ($\sum_f$). Finally, the convolved influence is propagated forward through the original model's dynamics from the initial conditions to the end of the model's time period, creating a representer ($\mathbf{r}_k$) for this potential measurement:

$$\mathbf{S}\mathbf{r}_k = \textstyle\sum_f \lambda_k. \qquad (8)$$

This representer is a large array providing the convolved influence that this single measurement will have on the entire domain (x,y,t) for all state variables (SSH, U, V, …). This process is repeated and representers are calculated for all potential measurements to be assimilated. In its most straightforward application, the representer method takes these K representer functions ($\mathbf{r}_k$ for $k = 1 \rightarrow K$, where K is the number of measurements), a first-guess of the model solution ($\mathbf{u}_0$), the K measurement values (**d),** along with the measurement covariances ($\sum_e$) and calculates the best estimated solution:

$$\hat{\mathbf{u}} = \mathbf{u}_0 + \sum_{k=1}^{K} \beta_k \mathbf{r}_k, \qquad (9)$$

where β_k represents the representer coefficients and are calculated as:

$$\beta = \left(\mathbf{R} + \textstyle\sum_e\right)^{-1}\left(\mathbf{d} - \mathbf{L}\mathbf{u}_\mathbf{0}\right). \qquad (10)$$

R is a symmetric $K \times K$ representer matrix that contains values from all representers estimated at all measurement locations ($R_{jk} = L_j \mathbf{r}_k$). $\sum_e$ is the measurement error covariance. A few different techniques can be used to accelerate the representer method. One is the Reduced Basis Approach (used by OTIS), in which representers are calculated for only a sub-sampled portion of data (this will be discussed later). If representers are left out, the solution is only an approximation and not a full solution. Research has shown, though, that the savings in computational cost far outweighs the loss of accuracy. This is primarily because the saved computational cost is put towards a finer resolution grid and/or the assimilation of more data.

5.2.3 TOPS Functionality

TOPS has two main components: data and dynamics. The data consist of both IHO tide gauge data and TOPEX altimetry. While the tide gauge data have been decomposed into harmonic tidal constituents already, the TOPEX data have not. For each data location, the time series of the TOPEX data is long enough that it can be separated into all major tidal constituents using least squares fitting. After performing this step and appending the IHO data, a data file (**B.dat**) is generated. It lists the latitude, longitude, SSH amplitude, SSH phase, and associated estimated variances for each specified tidal constituent at each distinct data location. For setup at NAVOCEANO, all available TOPEX and IHO data may be used. OTIS uses the Reduced Basis Approach, so representers do not need to be calculated for all of the data. Representers will be computed for all IHO stations and a portion of the TOPEX data with a new sampling strategy described in Section 5.2.8 and the accompanying figure.

5.2.4 Spectral Dynamics

OTIS uses linear shallow water dynamics (OTIS has an option to use nonlinear dynamics, which will not be discussed here or used at NAVOCEANO). See Equations 1-3. With these dynamics being linear, they can be transformed from the time domain into the frequency domain using Fourier Transforms.

$$\int_T \left[\frac{\partial U}{\partial t} - fV + gH\frac{\partial \zeta}{\partial x} + \kappa U - \mathbf{f}_U \right] e^{-i\omega t} dt = 0 \tag{11}$$

$$\int_T \left[\frac{\partial V}{\partial t} + fU + gH\frac{\partial \zeta}{\partial y} + \kappa V - \mathbf{f}_V \right] e^{-i\omega t} dt = 0 \tag{12}$$

$$\int_T \left[\left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} \right) + \frac{\partial \zeta}{\partial t} - \mathbf{f}_\zeta \right] e^{-i\omega t} dt = 0. \tag{13}$$

In the above equations, ζ_{SAL} is grouped in with the given tidal forcing, and the integrations span the time period of the model. By using integration, the above equations become:

$$Ue^{-i\omega t}\Big|_0^T + \int_T \left[i\omega U - fV + gH\frac{\partial \zeta}{\partial x} + \kappa U - \mathbf{f}_U \right] e^{-i\omega t} dt = 0 \tag{14}$$

$$Ve^{-i\omega t}\Big|_0^T + \int_T \left[i\omega V + fU + gH\frac{\partial \zeta}{\partial y} + \kappa V - \mathbf{f}_V \right] e^{-i\omega t} dt = 0 \tag{15}$$

$$\zeta e^{-i\omega t}\Big|_0^T + \int_T \left[\left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} \right) + i\omega\zeta - \mathbf{f}_\zeta \right] e^{-i\omega t} dt = 0. \tag{16}$$

The first terms of the above equations are evaluated at the boundaries of the time domain, and the state variables are cyclic for each tidal frequency. Assuming that the time domain is such that the values are the same at the initial and final conditions, these terms will be zero. With the first terms removed everything in the brackets must equal zero. This removes all dependency to the model’s time period.

$$i\omega U - fV + gH\frac{\partial \zeta}{\partial x} + \kappa U = \mathbf{f}_U \tag{17}$$

$$i\omega V + fU + gH\frac{\partial \zeta}{\partial y} + \kappa V = \mathbf{f}_V \tag{18}$$

$$\left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} \right) + i\omega\zeta = \mathbf{f}_\varsigma. \tag{19}$$

The above equations can be grouped together to form the following:

$$\Omega\mathbf{U} + gH\nabla\zeta = \mathbf{f}_\mathbf{U} \tag{20}$$

$$\nabla \bullet \mathbf{U} + i\omega\zeta = \mathbf{f}_\zeta \tag{21}$$

where $\Omega = \begin{bmatrix} i\omega + \kappa & -f \\ f & i\omega + \kappa \end{bmatrix}$ and $\mathbf{U} = \begin{bmatrix} U \\ V \end{bmatrix}$.

The state vector for the new frequency domain model includes the amplitude and phase of U,V (volume transports) and ζ (SSH) for each constituent frequency (ω) at each grid point. TOPS combines the frequency domain momentum and continuity equations into a single equation for SSH by first solving for $\mathbf{U}$ in Eq. (20):

$$\mathbf{U} = -gH\Omega^{-1}\nabla\zeta + \Omega^{-1}\mathbf{f}_{\mathbf{U}} \tag{22}$$

and substituting into Eq. (21),

$$\nabla \bullet gH\Omega^{-1}\nabla\zeta - i\omega\zeta = \nabla \bullet \Omega^{-1}\mathbf{f}_{\mathbf{U}} - \mathbf{f}_{\zeta}. \tag{23}$$

Therefore, when solving the forward shallow water equations, OTIS uses this combined equation (Eq. (23)) to calculate the phase and amplitude of SSH. It then backs out the volume transports from the momentum equations (Eq. (22)).

5.2.5 Inverse Solution

By defining the following discrete operators: [$\mathrm{G} \equiv gH\nabla$], [$\mathrm{D} \equiv \nabla\bullet$], [$\mathrm{C} \equiv \Omega^{-1}$] and [$\mathrm{A} \equiv \mathrm{DCG} - i\omega$], the solutions to Eqs. (22) & (23) can be expressed as follows:

$$\begin{bmatrix} \zeta \\ \mathbf{U} \end{bmatrix} = \begin{bmatrix} -\mathrm{A}^{-1} & \mathrm{A}^{-1}\mathrm{DC} \\ \mathrm{CGA}^{-1} & \mathrm{C} - \mathrm{CGA}^{-1}\mathrm{DC} \end{bmatrix} \begin{bmatrix} \mathbf{f}_{\zeta} \\ \mathbf{f}_{\mathbf{U}} \end{bmatrix} \tag{24}$$

and the adjoint of equation (24) can be expressed as:

$$\begin{bmatrix} \lambda_{\zeta} \\ \lambda_{\mathbf{U}} \end{bmatrix} = \begin{bmatrix} -\mathrm{A}^{-1*} & \mathrm{A}^{-1*}\mathrm{G}^{*}\mathrm{C}^{*} \\ \mathrm{C}^{*}\mathrm{D}^{*}\mathrm{A}^{-1*} & \mathrm{C}^{*} - \mathrm{C}^{*}\mathrm{D}^{*}\mathrm{A}^{-1}\mathrm{G}^{*}\mathrm{C}^{*} \end{bmatrix} \begin{bmatrix} L_{\zeta} \\ L_{\mathbf{U}} \end{bmatrix}, \tag{25}$$

where the superscript asterisk represents the conjugate transpose. The first matrix on the RHS of equation (24) is the model inverse ($\mathbf{S}^{-1}$) in Eq. (3); and similarly, the first matrix on the RHS of equation (25) is the model inverse transpose ($\mathbf{S}^{-T}$) in Eq. (7). The most computationally expensive part of this model is the factorization of the coefficient matrix (A). Since (A) is the same for all representers, OTIS accelerates the calculation of representers by formulating and storing A^{-1} and A^{-1*} in RAM.

By combining Eqs. (8), (24), and (25), the representers for both SSH and volume transport can be solved with the following equation:

$$\begin{bmatrix} r_{\zeta} \\ r_{\mathbf{U}} \end{bmatrix} = \mathbf{S}^{-1}\delta\mathbf{f} = \begin{bmatrix} -\mathrm{A}^{-1} & \mathrm{A}^{-1}\mathrm{DC} \\ \mathrm{CGA}^{-1} & \mathrm{C} - \mathrm{CGA}^{-1}\mathrm{DC} \end{bmatrix} \delta\mathbf{f}, \tag{26}$$

where $$\delta\mathbf{f} = \textstyle\sum_f \lambda_k = \begin{bmatrix} 0 & 0 \\ 0 & \sum_f \end{bmatrix} \begin{bmatrix} -\mathrm{A}^{-1*} & \mathrm{A}^{-1*}\mathrm{G}^{*}\mathrm{C}^{*} \\ \mathrm{C}^{*}\mathrm{D}^{*}\mathrm{A}^{-1*} & \mathrm{C}^{*} - \mathrm{C}^{*}\mathrm{D}^{*}\mathrm{A}^{-1}\mathrm{G}^{*}\mathrm{C}^{*} \end{bmatrix} \begin{bmatrix} L_{\zeta} \\ L_{\mathbf{U}} \end{bmatrix}. \tag{27}$$

The first term on the RHS of the above equation is the model error covariance. Plainly, there is no error to the continuity equation (OTIS assumes continuity to be exact) and no cross-covariances. For Navy purposes, only SSH measurements will be assimilated and only SSH representers (r_{ζ}) will be computed. Therefore the measurement functionals for volume transport will be zero ($L_{\mathbf{U}} = 0$). With this simplification, Eq. (27) will reduce to:

$$\delta\mathbf{f} = \begin{bmatrix} 0 & 0 \\ 0 & \Sigma_f \end{bmatrix} \begin{bmatrix} -\mathrm{A}^{-1*} & \mathrm{A}^{-1*}\mathrm{G}^*\mathrm{C}^* \\ \mathrm{C}^*\mathrm{D}^*\mathrm{A}^{-1*} & \mathrm{C}^* - \mathrm{C}^*\mathrm{D}^*\mathrm{A}^{-1}\mathrm{G}^*\mathrm{C}^* \end{bmatrix} \begin{bmatrix} L_\zeta \\ 0 \end{bmatrix} \tag{28}$$

$$= \begin{bmatrix} 0 \\ \Sigma_f\ \mathrm{C}^*\mathrm{D}^*\mathrm{A}^{-1*}\Delta_\zeta \end{bmatrix},$$

and the representers for SSH (Eq. (26)) can be solved as follows:

$$r_\zeta = \begin{bmatrix} -\mathrm{A}^{-1} & \mathrm{A}^{-1}\mathrm{DC} \end{bmatrix} \begin{bmatrix} 0 \\ \Sigma_f\ \mathrm{C}^*\mathrm{D}^*\mathrm{A}^{-1*}L_\zeta \end{bmatrix}. \tag{29}$$

$$= \mathrm{A}^{-1}\mathrm{DC}\,\Sigma_f\ \mathrm{C}^*\mathrm{D}^*\mathrm{A}^{-1*}L_\zeta$$

When all representers are calculated (Eq. (29)) for all SSH measurements, OTIS then assembles the Representer Matrix ($R_{jk} = L_j\mathbf{r}_k$) and solves for the representer coefficients (β_k) using Eq. (10). Rather than solving the inverse solution by summing all of the representers, as in (Eq. (9)), OTIS instead accumulates all of the measurement functionals into a single array, weighted by their corresponding representer coefficients ($\sum_k \beta_k L_k$), and uses this to force a final representer calculation for SSH and volume transport. By replacing L_ζ with $\sum_k \beta_k L_k$ in Eq. (28) and inserting this result into Eq. (26), the results will not be representers but rather the correction that will bring the initial solution ($\mathbf{u}_0$) into agreement with the model dynamics and data (this correction replaces the last term in Eq. (9)):

$$\begin{bmatrix} \delta\zeta \\ \delta\mathbf{U} \end{bmatrix} = \begin{bmatrix} -\mathrm{A}^{-1} & \mathrm{A}^{-1}\mathrm{DC} \\ \mathrm{CGA}^{-1} & \mathrm{C} - \mathrm{CGA}^{-1}\mathrm{DC} \end{bmatrix} \begin{bmatrix} 0 \\ \Sigma_f\ \mathrm{C}^*\mathrm{D}^*\mathrm{A}^{-1*}\left(\sum_k \beta_k L_k\right) \end{bmatrix} \tag{30}$$

$$= \begin{bmatrix} \mathrm{A}^{-1}\mathrm{DC}\,\Sigma_f\ \mathrm{C}^*\mathrm{D}^*\mathrm{A}^{-1*}\left(\sum_k \beta_k L_k\right) \\ \mathrm{C}\,\Sigma_f\ \mathrm{C}^*\mathrm{D}^*\mathrm{A}^{-1*}\left(\sum_k \beta_k L_k\right) - \mathrm{CG}\delta\zeta \end{bmatrix}.$$

Finally, the inverse solution is calculated using Eq. (9):

$$\begin{bmatrix} \hat{\zeta} \\ \hat{\mathbf{U}} \end{bmatrix} = \begin{bmatrix} \zeta_0 \\ \mathbf{U}_0 \end{bmatrix} + \begin{bmatrix} \delta\zeta \\ \delta\mathbf{U} \end{bmatrix}. \tag{31}$$

5.2.6 Reduced Basis Approach

The inverse method described in the previous section assumes that representers are calculated for all measurements. However, since TOPEX measurements are relatively dense and there can easily be tens of thousands of measurements for a reasonably sized domain, it is prudent to find an approximation that will reduce the number of representer calculations. OTIS uses the Reduced Basis Approach to accomplish this. This approach is underlined by the assumption that representers are going to be well correlated at nearby measurement locations. When all representers are calculated for all measurements, then Eq. (9) provides an exact inverse solution. However, if there are K measurements and they need to be assimilated using a subset of N representers, then Eq. (9) only produces an approximate inverse solution ($\mathbf{u}$):

$$\mathbf{u} = \mathbf{u}_0 + \sum_{k=1}^{N} \beta_k \mathbf{r}_k. \tag{32}$$

To improve the approximation of this solution, an alternative method must be used to calculate the representer coefficient (β) (instead of using Eq. (10)) that takes into account all of the data. This alternative method begins with reformulating the cost function (Eq. (6)) to be in terms of β.

As described in the previous section, the approach begins by calculating the N representers using Eq. (29) and formulating the ($N \times N$) representer matrix ($R_{jk} = L_j \mathbf{r}_k$). The representer matrix is such that each column of this matrix consists of each representer evaluated at all representer locations. For the Reduced Basis Approach, an additional matrix ($\mathbf{P}$) is required; $P_{jk} = L_j \mathbf{r}_k$ is a ($K \times N$) matrix with each column consisting of each of the N representers evaluated at all data locations. Note that the measurement functionals L_j used to compile $\mathbf{R}$ are only those measurement functionals that apply to the representers, whereas the L_j used to compile $\mathbf{P}$ includes all measurement functionals. The multiplication of this new matrix $\mathbf{P}$ and β can be defined as

$$\mathbf{P}\beta = \sum_{k=1}^{N} \beta_k \mathbf{L}\mathbf{r}_k, \tag{33}$$

where $\mathbf{L}$ is a matrix containing all K measurement functionals (each of the K rows of $\mathbf{L}$ is the measurement functional corresponding to each of the K data locations). By multiplying Eq. (32) by $\mathbf{L}$ and substituting in Eq. (33), the following definition can be made:

$$\mathbf{Lu} = \mathbf{Lu}_0 + \mathbf{P}\beta. \tag{34}$$

One more definition is needed in order to write the cost function in terms of β:

$$J_{Model}\left[\mathbf{u}\right] = \left(\mathbf{Su} - \mathbf{f}_0\right)^T \sum_f{}^{-1} \left(\mathbf{Su} - \mathbf{f}_0\right) = \beta^T \mathbf{R} \beta. \tag{35}$$

This definition comes from Bennett, 2002. If Eqs. (34) and (35) are inserted into the original cost function (Eq. (6)), then the cost function can be expressed in terms of β:

$$J\left[\beta\right] = \left(\mathbf{Lu}_0 + \mathbf{P}\beta - \mathbf{d}\right)^T \sum_e{}^{-1} \left(\mathbf{Lu}_0 + \mathbf{P}\beta - \mathbf{d}\right) + \beta^T \mathbf{R}\beta. \tag{36}$$

In order to find the β that minimizes this cost function, its first variation needs to be determined.

$$\begin{aligned} J\left[\beta + \delta\beta\right] - J\left[\beta\right] = {} & \left(\mathbf{Lu}_0 + \mathbf{P}\left(\beta + \delta\beta\right) - \mathbf{d}\right)^T \sum_e{}^{-1} \left(\mathbf{Lu}_0 + \mathbf{P}\left(\beta + \delta\beta\right) - \mathbf{d}\right) \\ & - \left(\mathbf{Lu}_0 + \mathbf{P}\beta - \mathbf{d}\right)^T \sum_e{}^{-1} \left(\mathbf{Lu}_0 + \mathbf{P}\beta - \mathbf{d}\right) \\ & + \left(\beta + \delta\beta\right)^T \mathbf{R}\left(\beta + \delta\beta\right) - \beta^T \mathbf{R}\beta. \end{aligned} \tag{37}$$

With a little reordering, the following is produced:

$$J\left[\beta+\delta\beta\right]-J\left[\beta\right]=\left(\mathbf{P}\delta\beta\right)^T\sum_e{}^{-1}\left(\mathbf{L}\mathbf{u}_0+\mathbf{P}\beta-\mathbf{d}\right) \tag{38}$$
$$+\left(\mathbf{u}_0{}^T\mathbf{L}^T+\beta^T\mathbf{P}^T-\mathbf{d}^T+\delta\beta^T\mathbf{P}^T\right)\sum_e{}^{-1}\mathbf{P}\delta\beta$$
$$+\delta\beta^T\mathbf{R}\beta+\left(\beta^T+\delta\beta^T\right)\mathbf{R}\delta\beta.$$

There are two terms in the above equation that contain a square of the deviation of β ($\delta\beta^T\mathbf{P}^T\sum_e{}^{-1}\mathbf{P}\delta\beta$ and $\delta\beta^T\mathbf{R}\delta\beta$). These terms are relatively small and can be removed. By removing these terms and rearranging, the following is produced:

$$J\left[\beta+\delta\beta\right]-J\left[\beta\right]=\left(\mathbf{P}\delta\beta\right)^T\sum_e{}^{-1}\left(\mathbf{L}\mathbf{u}_0+\mathbf{P}\beta-\mathbf{d}\right)+\left[\left(\mathbf{P}\delta\beta\right)^T\sum_e{}^{-1}\left(\mathbf{L}\mathbf{u}_0+\mathbf{P}\beta-\mathbf{d}\right)\right]^T \tag{39}$$
$$+\delta\beta^T\mathbf{R}\beta+\left[\delta\beta^T\mathbf{R}\beta\right]^T.$$

Clearly the four terms in the above equation are symmetric and therefore the transpose can be removed. The above equation reduces to the following:

$$J\left[\beta+\delta\beta\right]-J\left[\beta\right]=2\delta\beta^T\left(\mathbf{P}^T\sum_e{}^{-1}\left(\mathbf{L}\mathbf{u}_0+\mathbf{P}\beta-\mathbf{d}\right)+\mathbf{R}\beta\right)=0. \tag{40}$$

The cost function minimum is reached when its first variation is zero. As this is approached, then $\delta\beta$ must also approach zero. With this being the case the quantity in the parenthesis in Eq. (40) must also equal zero:

$$\mathbf{P}^T\sum_e{}^{-1}\left(\mathbf{L}\mathbf{u}_0+\mathbf{P}\beta-\mathbf{d}\right)+\mathbf{R}\beta=0. \tag{41}$$

Solving for β produces

$$\beta=\mathbf{P}^T\sum_e{}^{-1}\left(\mathbf{d}-\mathbf{L}\mathbf{u}_0\right)\left(\mathbf{P}^T\sum_e{}^{-1}\mathbf{P}+\mathbf{R}\right)^{-1}. \tag{42}$$

The updated list of representer coefficients takes into account all of the data. This equation, however, is very costly to compute. **P** and **R** are full matrices and can be quite large, so taking their inverse would be impractical.

OTIS uses a more efficient calculation to determine β:

$$\beta=\mathbf{EQS}\left(\mathbf{S}^2+\nu\mathbf{I}\right)^{-1}\mathbf{W}^T\left(\mathbf{d}-\mathbf{L}\mathbf{u}_o\right), \tag{43}$$

where **W**, **Q**, and **S** are the singular value decomposition of

$$\sum_e{}^{-1}\mathbf{PE}=\mathbf{WSQ}^T \tag{44}$$

and

$$\mathbf{E}=\mathbf{R}^{-1/2}. \tag{45}$$

In order to calculate **E, R** is decomposed into $\mathbf{R}=\mathbf{V\Psi V}^T$, where **V** are orthonormal eigenvectors and $\mathbf{\Psi}$ is a diagonal matrix composed of eigenvalues. Therefore, it is clear that **E** can be calculated as $\mathbf{E}=\mathbf{R}^{-1/2}=\mathbf{V\Psi}^{-1/2}$.

5.2.7 The Damping Parameter

In equation (43), ν is the damping parameter that controls the relative fit between the dynamics and data within the cost function. If the data and dynamic error covariances are exact, then the value of this coefficient should be ‘1’. However, the error covariances are not exact and ν is scaled to account for this error difference. OTIS was initially set up with a GUI to manually adjust ν via trial and error. Operationally, this method is unsuitable. Therefore, code was added in TOPS to automatically determine the optimal value of ν by computing the generalized cross-validation function for an array of ν values (Zaron and Egbert, 2006). The calculated generalized cross-validation function can be defined as a prediction error and will be an inverted bell-curve function of the damping parameter. From this curve the value of ν that produces the minimum prediction error is determined and used in equation (43).

5.2.8 Sampling Strategy for Representers

Figure 5.2.8-1 displays an example of how the improved **lat_lon.f** (OTIS) program would determine the representer locations for the Yellow Sea. This plot was created using 250 representers. The calculation of representers is the most time consuming part of OTIS. Therefore, the selection of the total number of representers is important and should be dependent on computer speed and grid size. The sampling strategy automatically uses all IHO tide gauge locations from the data file **IHO_data.dat** and sets them as representer locations in the file **lat_lon.rep**. These are the red diamonds in Figure 5.2.8-1. The program then checks that the current number of representers is less than the maximum value. If the maximum value (250) is reached or exceeded, the program will stop, and TOPS will only assimilate IHO data. Since there are about 80 IHO stations in the Yellow Sea, the program will continue. The **lat_lon.f** then appends all TOPEX crossover locations (the seven blue diamonds) to the **lat_lon.rep** file. After IHO and TOPEX crossovers are selected, the remaining representers (about 160 blue dots in this plot) are distributed along the TOPEX tracks within the domain (minus the crossover points) based upon the inverse of bathymetry. Hence, shallower regions will have a finer resolution of representers, because correlation length scales of data and dynamics typically decrease as the water depth decreases. For example, if the TOPS user wishes to add additional representers to Fig. 5.2.8-1. If a representer is added in deep water at a TOPEX data point next to an existing representer, then the amount of information added to the system will be minimal. This is because the two nearby locations will most likely be well correlated. The correlation between two nearby locations will be smaller in shallower waters, therefore increasing the amount of added information to the system. This particular method of distributing representers was proven to be beneficial in Egbert and Erofeeva (2002).

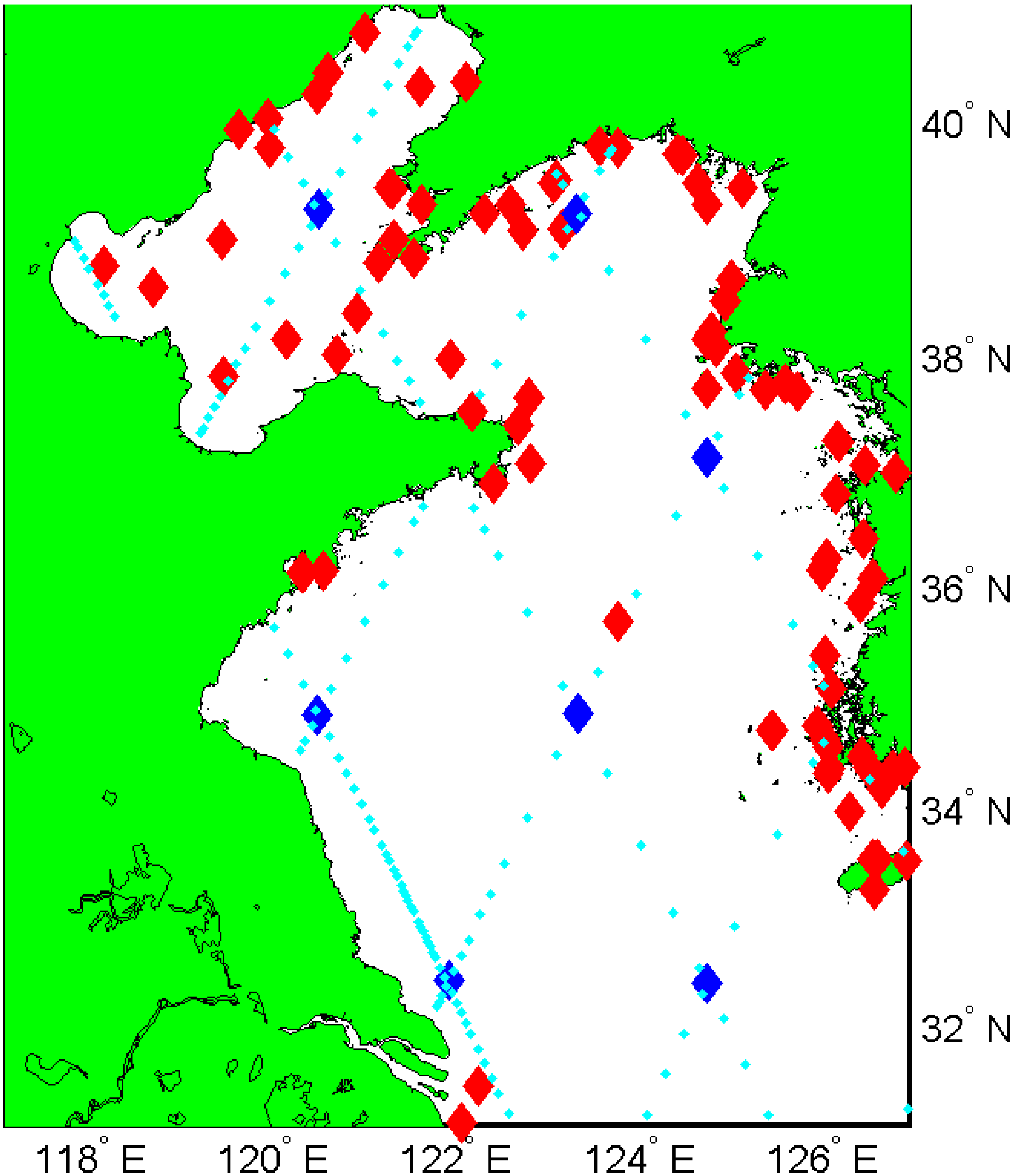

Figure 5.2.8-1: Example plot generated from the **lat_lon.f** (OTIS) program. Red diamonds indicate IHO tide gauge locations, dark blue diamonds are TOPEX crossover locations, and the small blue squares are all other representer locations.

6.0 Primary TOPS Fortran Routines

6.1 Initialization, Setup, and General Subroutines (OTIS/bin)

6.1.1 (OTIS/bin/blockgen.f)

Subroutine	Description
blockgen	BLOCKGEN initializes units and general constants used for all missions. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** n/a **Common Blocks:** common/cunits, /constsr8, /constsi4, /constsi2

6.1.2 (OTIS/bin/crossdat.f)

Subroutine	Description
crossdat	Subroutine CROSSDAT makes time series records in a file opened on nunit for one cross-over point. **Calling Sequence:** crossdat(nunit,irec,lat1,lon1,lat2,lon2,t,h, maxlen, iflag, date1, date2, ndates) **Data Declaration:** Integer nunit, irec, icycle, iflag, ndates, i,k, maxlen, Real lat1, lon1, lat2, lon2, t, h, date1, date2 **I/O:** stdout, write nunit **Common Blocks:** n/a

6.1.3 (OTIS/bin/iflag16.f)

Subroutine	Description
iflag16	This subroutine decomposes two-byte integer flag words into logicals. **Calling Sequence:** iflag16(iflag,lbit16) **Data Declaration:** Integer iflag Logical lbit16 **I/O:** n/a **Common Blocks:** n/a

6.1.4 (OTIS/bin/j_days.f)

Subroutine	Description
caldat	This routine converts Julian day to month, day, and year. From Press et al., 1986. **Calling Sequence:** caldat (Julian, mm, id, iyyy) **Data Declaration:** Integer Julian, mm, id, iyyy **I/O:** n/a **Common Blocks:** n/a
date_mjd	DATE_MJD converts date to MJD. **Calling Sequence:** date_mjd(mm, id, iyyy, mjd) **Data Declaration:** Integer mm, id, iyyy,mjd

Subroutine	Description
	I/O: n/a **Common Blocks:** n/a
j_days	Converts MJD to Julian day. JD= MJD +2400001. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** read, write stdout **Common Blocks:** n/a

6.1.5 (OTIS/bin/lat_lon.f)

Subroutine	Description
lat_lon	The program reads data from the pathfinder database in ../topex for some chosen rectangular area *(lat1,lon1)-(lat2,lon2)*. The limits have to be entered from ../prm/grid (standard grid file). The output is an ASCII list of lats, lons, and quality flags. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** stdout; open, read, close units 1, 2, 4, 8, 7; write 2, 4, 7; read indir, inflag **Common Blocks:** common/cflag, /cmission, /constsi2, /cunits
rd_com_line	**Calling Sequence:** rd_com_line(grid,fname,rcro,ralt,n1,n2,n3, dcro,dalt,m1,m2,m3, rep_max,qmode) **Data Declaration:** Logical rcro, ralt,dcro, dalt,rcop,qmode Character grid, fname,arg Integer n1,n2,n3,m1,m2,m3,k,i,rep_max **I/O:** stdout; read arg **Common Blocks:** n/a
usage	**Calling Sequence:** usage(qmode) **Data Declaration:** Logical qmode **I/O:** read, write stdout **Common Blocks:** n/a

6.1.6 (OTIS/bin/makedat.f)

Subroutine	Description
getarg	Subroutine GETARG is a special fix for the HP machine. It does not invoke HP9000_800 directives. **Calling Sequence:** getarg(n, s) **Data Declaration:** Integer n Character s **I/O:** read, write stdout **Common Blocks:** n/a
makedat	MAKEDAT is a program that reads the data sites' lats and lons from the files, reads TOPEX data for the area and saves them for the above lats/lons, and writes data files in **tpxbin.dat** format. The output data file is used by MAKEB to calculate **B.dat** bad flags.

Subroutine	Description
	Calling Sequence: n/a **Data Declaration:** n/a **I/O:** stdout; open, read, close units 1, 2, 7; read inmss, indir, inssh, inflag, stdout; write units 2, 8, 9 **Common Blocks:** common/cflag, /cmission, /constsi2, /cunits
prime_check	PRIME_CHECK checks if *n1* is a prime number: iflag=1/other, if yes/no. If yes, *n2*<*n1*, which is the closest number divisible by 6. If no, then *n2*=*n1*. **Calling Sequence:** prime_check(n1,iflag,n2) **Data Declaration:** Integer n1,n2,iflag **I/O:** n/a **Common Blocks:** common/cflag, /cmission, /constsi2, /cunits
rd_com_line	**Calling Sequence:** rd_com_line(fin,fout,qmode,altav) **Data Declaration:** Logical altav,qmode Character fout,fin **I/O:** stdout **Common Blocks:** n/a
usage	**Calling Sequence:** usage(qmode) **Data Declaration:** Logical qmode **I/O:** read, write stdout **Common Blocks:** common/cflag

6.1.7 (OTIS/bin/mix_ave.f)

Subroutine	Description
mix_ave	This subroutine is for mixing and averaging two time steps. **Calling Sequence:** mix_ave(t1,h1,ncyc,t2,h2,count,flag) **Data Declaration:** Integer ncyc, flag, count Real t1, h1, t2,h2 **I/O:** read, write stdout **Common Blocks:** n/a
mix_ers	**Calling Sequence:** mix_ers(t1,h1,ncyc,t2,h2) **Data Declaration:** Integer ncyc, flag, count Real t1,h1,t2,h2 **I/O:** read, write stdout **Common Blocks:** n/a
reorder	**Calling Sequence:** reorder(t,h,n) **Data Declaration:** Integer n Real t,h **I/O:** n/a **Common Blocks:** n/a

6.1.8 (OTIS/bin/obstime.f)

Subroutine	Description
obstime	OBSTIME calculates exact time in modified Julian date and fraction of the day for a specific sea surface residual height. **Calling Sequence:** obstime(time,icycle,ntrack,idx,isdata) **Data Declaration:** Integer icycle, ntrack, idx Double Precision time Logical isdata **I/O:** read intime **Common Blocks:** common/cmission, /constsr8, /constsi4, /cunits

6.1.9 (OTIS/bin/openfiles.f)

Subroutine	Description
openfiles	OPENFILES opens all files required to read and interpret the ocean pathfinder ERS1 collinear database. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** write iout6, stdout; open inephm, inmss, inflag, intime, inssh, indir **Common Blocks:** common/cmission, /cfnamersr, /cunits

6.1.10 (OTIS/bin/otis_check_files.f)

Subroutine	Description
otis_check_files	This program reads and inspects OTIS input files and output from OTIS forward and final solution models. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** write 6, 21, 22, 23, 32, 33, 42, 43, 52, 53, stdout; open, read, close 199, write 20, read 10 **Common Blocks:** common/cmission, /cfnamersr, /cunits

6.1.11 (OTIS/bin/rtloadtopex.f)

Subroutine	Description
rtlooadtopex	Subroutine RTLOADtopex loads ephemeredes for a specific track of a repeat cycle for TOPEX. **Calling Sequence:** rtloadtopex(ntrack,nperiod,ilat,ilon) **Data Declaration:** Integer ilat, ilon,nperiod,ntrack **I/O:** read inephm **Common Blocks:** common/cunits

6.1.12 (OTIS/bin/topexinit.f)

Subroutine	Description
topexinit	This subroutine initializes TOPEX specific constants for reading collinear databases. It initializes names and record sizes of all files used.

Subroutine	Description
	Calling Sequence: n/a **Data Declaration:** n/a **I/O:** n/a **Common Blocks:** common/cmission, /cflag, /cfnamersr

6.2 Primary TOPS Subroutines (OTIS/src)

6.2.1 Forward Time Step Subroutines

6.2.1.1 (Otis/src/fwd_ts/interepSAL.f)

Subroutine	Description
interpSAL	Interpolates complex *n x m* SAL array sal (from TPXO) onto the model grid. **Calling Sequence:** interpSAL(sal0,n0,m0,th_lim0, phi_lim0, theta_lim, phi_lim, mz, ierr, sal) **Data Declaration:** Integer mz, ierr, n0, m0 Complex sal0, sal Real th_lim0, phi_lim0, theta_lim, phi_lim **I/O:** n/a **Common Blocks:** n/a

6.2.2 MKB Subroutines

6.2.2.1 (OTIS/src/mkb/create_A.f)

Subroutine	Description
create_A	This subroutine was tuned to the tide gauge data as they were in old, Mediterranean **tpxoMED.dat**, that is, already harmonically analyzed for eight constituents, constituent number: *ic=int((k+1)/2)*, where "k" is the row number (starting from 0). lat lon 0 h(ic).Re 0 0.0 lat lon 0 h(ic).Im 1 0.0 lat lon 0 h(ic+1).Re 2 0.0 lat lon 0 h(ic+1).Im 3 0.0 **Calling Sequence:** create_A(k,A) **Data Declaration:** Integer k Complex A **I/O:** n/a **Common Blocks:** n/a

6.2.2.2 (OTIS/src/mkb/def_cid.f)

Subroutine	Description
def_cid	**Calling Sequence:** def_cid(nc0,cid,ind) **Data Declaration:** Integer nc0,ind Character cid

Subroutine	Description
	I/O: stdout **Common Blocks:** n/a

6.2.2.3 (OTIS/src/mkb/def_form.f)

Subroutine	Description
def_form	**Calling Sequence:** def_form(fname,ifmt) **Data Declaration:** Integer ifmt Character fname **I/O:** open, read, rewind, close unit 1; stdout **Common Blocks:** n/a

6.2.2.4 (OTIS/src/mkb/filter_outliers.f)

Subroutine	Description
filter_outliers	**Calling Sequence:** filter_outliers(t1,t2,h1,h2,L) **Data Declaration:** Integer L Real t1,t2,h1,h2 **I/O:** stdout **Common Blocks:** n/a

6.2.2.5 (OTIS/src/mkb/height.f)

Subroutine	Description
height	This real function returns height from a model array of complex constituents. **Calling Sequence:** filter_outliers(t1,t2,h1,h2,L) **Data Declaration:** Integer nc Complex A,P **I/O:** n/a **Common Blocks:** n/a

6.2.2.6 (OTIS/src/mkb/loadModel.f)

Subroutine	Description
cut	Integer function. **Calling Sequence:** cut(name) **Data Declaration:** Character name **I/O:** n/a **Common Blocks:** n/a
loadModel	This subroutine reads binary model files with the following Fortran unformatted binary format: • Rec 1 (header): n, m, nc, theta_min, theta_max, phi_min, phi_max, const_1, const_2, ...const_nc, where const_j - constituent ID char*4. • Rec 2: 1st constituent elevations (*n* x *m* complex). • Rec 3: 2nd constituent elevations. • Rec nc+1: constituent nc elevations.

Subroutine	Description
	Calling Sequence: loadModel(Pr,n,m,nc0,cid,th_mod,ph_mod,mask,fname) **Data Declaration:** Integer n,m,nc0, mask Complex Pr Character cid, fname Real th_mod, ph_mod **I/O:** open, read, close unit 1; stdout **Common Blocks:** n/a
rd_constituents	**Calling Sequence:** rd_constituents(cid) **Data Declaration:** Character cid **I/O:** stdout, open, read, close unit 17 **Common Blocks:** n/a

6.2.2.7 (OTIS/src/mkb/loadModel_uv.f)

Subroutine	Description
loadModel_uv	This subroutine reads binary model transport files with the following Fortran unformatted binary format: • Rec 1 (header): n, m, nc, theta_min, theta_max, phi_min, phi_max, const_1, const_2, ...const_nc, where const_j - constituent ID char*4. • Rec 2: 1st constituent transports (2 x *n* x *m* complex). • Rec 3: 2nd constituent transports. • Rec nc+1: constituent nc transports (m^2/s). **Calling Sequence:** loadModel_uv(Pru,Prv,n,m,nc0,cid,th_mod,ph_mod,mu,mv,fname) **Data Declaration:** Integer n,m,nc0, mu,mv Complex Pru,Prv Character cid, fname Real th_mod, ph_mod **I/O:** open, read, close unit 1; stdout **Common Blocks:** n/a

6.2.2.8 (OTIS/src/mkb/lp_tide.f)

Subroutine	Description
lp_tide	This real function is a long period tide height correction. It assumes that entire nodal correction arrays *pu*(20), *pf*(20) are passed ==> offset=17. **Calling Sequence:** lp_tide(time,lat,pu,pf) **Data Declaration:** Real time, lat, pu, pf **I/O:** n/a **Common Blocks:** n/a

6.2.2.9 (OTIS/src/mkb/make_a.f)

Subroutine	Description
make_a	This subroutine computes *A* matrix elements for one data point if *t2*==0. It computes *A* for absolute height at *t1* if *t2*>0, and computes *A* for cross-over difference (*h1*-*h2*) at

Subroutine	Description
	(*t1*,*t2*). **Calling Sequence:** make_a(interp,ind,nc,t1,t2,pu,pf,w,A,l_sal) **Data Declaration:** Real t1,t2,w,pu,pf Logical interp, l_sal Integer ind, nc Complex A **I/O:** stdout **Common Blocks:** n/a
mkw	**Calling Sequence:** mkw(interp,ind,nc,wr) **Data Declaration:** Real wr Logical interp Integer ind, nc **I/O:** n/a **Common Blocks:** n/a

6.2.2.10 (OTIS/src/mkb/makeB.f)

Subroutine	Description
makeB	Program MAKEB creates a dense *A* matrix for a time series of data points at one location, stores it in real format, performs singular value decomposition (SVD), and finally assembles the results into a *B* matrix. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** read, write stdout; open, write, close units 10, 13, 21; open, read, close units 1, 10, 13 **Common Blocks:** n/a
wrBha	**Calling Sequence:** wrBha(irec,iounit,m_type,lat,lon,hhat,theta,dummy) **Data Declaration:** Real hhat, theta, lat, lon, dummy Integer irec, iounit, m_type **I/O:** write iounit **Common Blocks:** n/a

6.2.2.11 Astronomical Tide Subroutines (OTIS/src/mkb/nodal.f)

Subroutine	Description
arguments	This is a kernel routine for HAT53 subroutine. It calculates tidal arguments. **Calling Sequence:** arguments(time1, arg, f, u) **Data Declaration:** Double Precision time1, arg, f, u **I/O:** write iounit **Common Blocks:** n/a
astrol	This subroutine computes the basic astronomical mean longitudes s, h, p and N. N is not N’, as N is decreasing with time. These formulae are for the period 1990-2010 and were derived from David Cartwright (pers. comm. 11/90). Time is UTC in decimal MJD. All longitudes are returned in degrees. **Calling Sequence:** astrol(time, shpn)

Subroutine	Description
	Data Declaration: Double Precision time, shpn **I/O:** n/a **Common Blocks:** n/a
nodal	**Calling Sequence:** nodal(dtime,latitude,pu,pf) **Data Declaration:** Real dtime, latitude, pu, pf **I/O:** stdout **Common Blocks:** n/a

6.2.2.12 (OTIS/src/mkb/rd_com_line.f)

Subroutine	Description
rd_com_line	**Calling Sequence:** rd_com_line(prior,data,long_p,ha_only,sub_only,interp,cor_mod, con8, ave,tg, append, l_sal,umod,err_cm,Bha, one_point, ifmt, mod_fname,data_fname, out_fname,cor_fname) **Data Declaration:** Logical prior, data, long_p, ha_only, sub_only, interp, cor_mod, con8, ave, tg, append, l_sal, umod, Bha, one_point Character mod_fname, data_fname, out_fname, cor_fname Integer ifmt Real err_cm **I/O:** stdout; open, read, close unit 1 **Common Blocks:** n/a
usage	**Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** stdout **Common Blocks:** n/a

6.2.2.13 (OTIS/src/mkb/read_adcp.f)

Subroutine	Description
read_adcp	**Calling Sequence:** read_adcp(i_unit,irec,lat,lon,t1,h1,iuv,ief,ifmt) **Data Declaration:** Logical ief Integer i_unit, irec, iuv, ifmt Real t1,h1, lat, lon **I/O:** read, write stdout, read i_unit **Common Blocks:** n/a
write_adcp	**Calling Sequence:** write_adcp(i_unit,lat,lon,t1,h1,d1) **Data Declaration:** Integer i_unit Real t1,h1,d1, lat, lon **I/O:** write i_unit **Common Blocks:** n/a

6.2.2.14 (OTIS/src/mkb/read_cm.f)

Subroutine	Description

Subroutine	Description
read_cm	**Calling Sequence:** read_cm(iounit,irec,cid,lat,lon,hu,iuv,ief,uerr,ifmt) **Data Declaration:** Logical ief Integer iounit, irec, iuv, ifmt Real hu,uerr, lat, lon Character cid **I/O:** read, write stdout; open, read, write, close iounit **Common Blocks:** n/a

6.2.2.15 (OTIS/src/mkb/read_rad.f)

Subroutine	Description
defps	**Calling Sequence:** defps(fname,ctmp,k1,k2) **Data Declaration:** Integer k1,k2 Character fname, ctmp **I/O:** open, read, close unit 2; stdout **Common Blocks:** n/a
read_rad	**Calling Sequence:** read_rad(fname,irec,lat,lon,hu,the,phi,ief,cid,uerr,ifmt) **Data Declaration:** Logical ief Integer irec, ifmt Real hu,the,phi,uerr, lat, lon Character cid, fname **I/O:** open, read, close unit 15; stdout **Common Blocks:** n/a

6.2.2.16 (OTIS/src/mkb/read_tg.f)

Subroutine	Description
read_tg	**Calling Sequence:** read_tg(iounit,irec,cid,lat,lon,h1,ief,ifmt,ertg) **Data Declaration:** Logical ief Integer iounit, irec, ifmt Real h1,ertg, lat, lon Character cid **I/O:** read, write stdout; read, rewind, write, close iounit **Common Blocks:** n/a

6.2.2.17 (OTIS/src/mkb/readTpxo.f)

Subroutine	Description
readTxpo	**Calling Sequence:** readTpxo(i_unit,irec,L,lat,lon,t1,h1,t2,h2,ief,dif,ifmt) **Data Declaration:** Logical ief,dif Integer i_unit, L, irec, ifmt Real t1,h1,t2,h2, lat, lon **I/O:** read, close i_unit; write unit 19; stdout **Common Blocks:** n/a

6.2.2.18 (OTIS/src/mkb/ts_syn.f)

The code *ts_syn* gives a simple, generalized program for synthesizing elevation, transport and currents time series at a chosen location/time using any tidal solution in the standard format. The code is compiled in **OTIS/local/“MyArea”/exe/** as other OTIS codes, but is not tuned to a certain grid. Therefore, the user may generate time series from the same directory for tidal models on any grids covering a chosen location.

Subroutine	Description
caldat	This subroutine converts Julian day to month, day, & year. The code is from Press et al., 1986. The only modification is that real arithmetic is done in r*8. To convert modified Julian day, call this routine with Julian = MJD + 2400001. **Calling Sequence:** caldat (Julian,mm,id,iyyy) **Data Declaration:** Integer Julian, mm,id,iyyy **I/O:** n/a **Common Blocks:** n/a
date_mjd	This subroutine converts date to MJD. **Calling Sequence:** date_mjd(mm,id,iyyy,mjd) **Data Declaration:** Integer mm,id,iyyy,mjd **I/O:** n/a **Common Blocks:** n/a
ts_syn	Time series synthesis program using a tidal solution in interpreting a standard format. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** read, write stdout; write sdate, unit 3; open, read, close units 1,3 **Common Blocks:** n/a

6.2.2.19 (OTIS/src/mkb/write_cm.f)

Subroutine	Description
write_cm	**Calling Sequence:** write_cm(iounit,cid,lat,lon,hu,uerr,du,iuv) **Data Declaration:** Integer iounit, iuv Real hu,du, lat, lon,uerr Character cid **I/O:** write iounit **Common Blocks:** n/a

6.2.2.20 (OTIS/src/mkb/write_rad.f)

Subroutine	Description
write_rad	**Calling Sequence:** write_rad(iounit,cid,lat,lon,hu,du,the,phi) **Data Declaration:** Integer iounit Real hu,du, the,phi,lat, lon,uerr Character cid **I/O:** write iounit **Common Blocks:** n/a

6.2.2.21 (OTIS/src/mkb/write_tg.f)

Subroutine	Description
write_tg	**Calling Sequence:** write_tg(iounit,cid,lat,lon,h,d,damp,dph,dReIm) **Data Declaration:** Integer iounit Real h,d, damp,dReIm, dph, lat,lon Character cid **I/O:** write iounit **Common Blocks:** n/a

6.2.2.22 (OTIS/src/mkb/writeTpxo.f)

Subroutine	Description
writeTpxo	**Calling Sequence:** writeTpxo(i_unit,irec,L,lat,lon,t1,h1,t2,h2,ifmt) **Data Declaration:** Integer i_unit,irec, ifmt Real h1,h2,t1,t2, lat,lon,L **I/O:** stdout; write i_unit **Common Blocks:** n/a

6.2.3 RP_DP Subroutines (OTIS/src/rp_dp)

6.2.3.1 (OTIS/src/rp_dp/atgf.f)

Subroutine	Description
atgf	**Calling Sequence:** atgf(ic,c_id,cobc) **Data Declaration:** Integer ic Character c_id, cobc **I/O:** n/a **Common Blocks:** n/a
force_in	**Calling Sequence:** force_in(cforce) **Data Declaration:** Character cforce **I/O:** open, read unit 1 **Common Blocks:** n/a
rd_obc	**Calling Sequence:** rd_obc(nob,cobc,hobc) **Data Declaration:** Integer nob Character cobc Complex hobc **I/O:** stdout; open, read, close unit 1 **Common Blocks:** n/a
rd_obc_uv	**Calling Sequence:** rd_obc_uv(nob_u,nob_v,cobc,u_obc,v_obc) **Data Declaration:** Integer nob_u,nob_v Character cobc Complex u_obc, v_obc **I/O:** open, read unit 1; stdout **Common Blocks:** n/a

6.2.3.2 (OTIS/src/rp_dp/BSI_weights.f)

Subroutine	Description
BSI_weights	This is a bilinear spline interpolation (BSI) weight subroutine for delta forcing. **Calling Sequence:** BSI_weights(node,theta,phi,theta_lim,phi_lim, dx,dy,mask,n,m, ww,iw,jw) **Data Declaration:** Character node Real theta, phi, theta_lim, phi_lim, dx, dy,ww Integer mask, n,m,iw,jw **I/O:** stdout **Common Blocks:** n/a
ipshift	Function IPSHIFT creates periodic shift maps *i* to *i+ish*, mod *n*; (always between 1 and *n*, never 0). **Calling Sequence:** ipshft(i,ish,n) **Data Declaration:** Integer i, ish, n **I/O:** n/a

6.2.3.3 (OTIS/src/rp_dp/CDG.f)

Subroutine	Description
op_C	**Calling Sequence:** op_C(u1,v1,u_v,v_u,t) **Data Declaration:** Complex u1,v1,u_v, v_u Character t **I/O:** n/a **Common Blocks:** n/a
op_C1	**Calling Sequence:** op_C1(u1,v1,u_v,v_u) **Data Declaration:** Complex u1, v1, u_v, v_u **I/O:** n/a **Common Blocks:** n/a
op_C2	**Calling Sequence:** op_C2(u1,v1,u_v,v_u,t) **Data Declaration:** Complex u1, v1, u_v, v_u Character t **I/O:** n/a **Common Blocks:** n/a
op_D	**Calling Sequence:** op_D(u1,v1,z1) **Data Declaration:** Complex u1, v1, z1 **I/O:** n/a **Common Blocks:** n/a
op_G	**Calling Sequence:** op_G(z1,u1,v1,t) **Data Declaration:** Complex u1, v1, z1 Character t **I/O:** n/a **Common Blocks:** n/a
op_IB	**Calling Sequence:** op_IB(u1,v1,ic) **Data Declaration:** Complex u1, v1 Integer ic

Subroutine	Description
	I/O: n/a **Common Blocks:** n/a

6.2.3.4 (OTIS/src/rp_dp/checklim.f)

Subroutine	Description
checklim	**Calling Sequence:** checklim(t_lim,p_lim,n0,m0,nc0) **Data Declaration:** Real t_lim, p_lim Integer nc0, m0, n0 **I/O:** write unit 6, 0 **Common Blocks:** n/a

6.2.3.5 (Otis/src/rp_dp/constit.f)

Subroutine	Description
constit_all	**Calling Sequence:** constit_all(nc,c_id,omega,alpha,ispec,ph,amp) **Data Declaration:** Real alpha, ph, amp, omega Character c_id Integer nc,ispec **I/O:** write unit 0 **Common Blocks:** n/a
constit_in	This subroutine gets the constituent information from the constituents file. **Calling Sequence:** constit_in(cconstit,c_id) **Data Declaration:** Character c_id, cconstit **I/O:** open, read, close unit 1; write unit 0 **Common Blocks:** n/a
constit_omega	**Calling Sequence:** constit_omega(nc, c_id,omega) **Data Declaration:** Real omega Character c_id Integer nc **I/O:** write unit 0, 6 **Common Blocks:** n/a

6.2.3.6 (Otis/src/rp_dp/covsc_in.f)

Subroutine	Description
cov_white	This subroutine is using scales for each constituent. They are determined by averaging over the grid, reset for "white noise" covariance. **Calling Sequence:** covsc_white() **Data Declaration:** n/a **I/O:** n/a **Common Blocks:** n/a
covsc_in	**Calling Sequence:** covsc_in(ccov,ob_var_sc,rb_var_sc,int_var_sc) **Data Declaration:** Real ob_var_sc, rb_var_sc, int_var_sc Character ccov

Subroutine	Description
	I/O: open, read, close unit 1; stdout; write unit 6 **Common Blocks:** n/a

6.2.3.7 (Otis/src/rp_dp/dcomb.f)

Subroutine	Description
blspwt_set	**Calling Sequence:** blspwt_set() **Data Declaration:** n/a **I/O:** write unit 28; stdout **Common Blocks:** common/datablk
blspwt_set	**Calling Sequence:** blspwt_set(ndat) **Data Declaration:** Integer ndat **I/O:** stdout **Common Blocks:** common/datablk
dcomb	**Calling Sequence:** dcomb(ic,bname,nrep1,nrep2) **Data Declaration:** Character bname Integer ic, nrep1, nrep2 **I/O:** open, read, close unit 10; stdout **Common Blocks:** common/datablk
dcomb_cg	**Calling Sequence:** dcomb_cg(x_in, nx, ik) **Data Declaration:** Real x_in Integer ik, nx **I/O:** n/a **Common Blocks:** common/datablk, /rmultblk
read_sites	**Calling Sequence:** read_sites(vel_rep) **Data Declaration:** Logical vel_rep **I/O:** stdout; open, read, rewind, close unit 1 **Common Blocks:** common/datablk

6.2.3.8 (Otis/src/rp_dp/delta.f)

Subroutine	Description
delta	This subroutine takes the location of a point elevation measurement and information about whether it is to be treated as a boundary elevation, and outputs the forcing and BCs for the backward problem. **Calling Sequence:** delta(itype,theta,phi,th,ph,ierr) **Data Declaration:** Integer itype, ierr Real theta, phi, th, ph **I/O:** stdout **Common Blocks:** n/a

6.2.3.9 (Otis/src/rp_dp/diffuse.f)

Subroutine	Description
diffuse	**Calling Sequence:** n/a

Subroutine	Description
	Data Declaration: n/a **I/O:** open, read, close unit 99; stdout; write unit 11 **Common Blocks:** n/a

6.2.3.10 (Otis/src/rp_dp/ds_subs.f)

Subroutine	Description
comp_S	**Calling Sequence:** comp_S(ic,ip,iq) **Data Declaration:** Integer ip, iq, ic **I/O:** n/a **Common Blocks:** n/a
f16	**Calling Sequence:** f16(ip,iq) **Data Declaration:** Integer ip, iq **I/O:** n/a **Common Blocks:** n/a
f17	**Calling Sequence:** f17(ip,iq) **Data Declaration:** Integer ip, iq **I/O:** n/a **Common Blocks:** n/a
f20	**Calling Sequence:** f20(ip,iq) **Data Declaration:** Integer ip, iq **I/O:** n/a **Common Blocks:** n/a
f21	**Calling Sequence:** f21(ip,iq) **Data Declaration:** Integer ip, iq **I/O:** n/a **Common Blocks:** n/a
f24	**Calling Sequence:** f24(ip,iq) **Data Declaration:** Integer ip, iq **I/O:** n/a **Common Blocks:** n/a
f25	**Calling Sequence:** f25(ip,iq) **Data Declaration:** Integer ip, iq **I/O:** n/a **Common Blocks:** n/a
f28	**Calling Sequence:** f28(ip,iq) **Data Declaration:** Integer ip, iq **I/O:** n/a **Common Blocks:** n/a
f29	**Calling Sequence:** f29(ip,iq) **Data Declaration:** Integer ip, iq **I/O:** n/a **Common Blocks:** n/a

6.2.3.11 (Otis/src/rp_dp/fwd_fac.f)

Subroutine	Description
caoutb	**Calling Sequence:** caoutb(z_unit,uv_unit) **Data Declaration:** Integer z_unit, uv_unit **I/O:** write z_unit, uv_unit **Common Blocks:** n/a
fwd_fac	This is a CM-FORTRAN direct solver for LTEs. The program must be compiled with the correct grid size (set in 'include/**gridsize.h**'). If the grid size specified in the grid file is not the same, the program will terminate with a warning. By default all input files are expected to be in the default input directory (set in CPATHIN ... see below), and to have standard names. The default input directory can be changed with the -i option, and the full path name of any input file can be specified with the -g, -c, and -b options. INPUT files: (1) **grid** (change with -g<file>).METRY: This is a bathymetry grid file that contains a list of open boundary nodes, a header specifying gridsize latitude and longitude limits, and time step in sec. Its format is FORTRAN sequential binary; 3 records. The first is the number of OB nodes, the second is a list of OB nodes (i,j) and the third is bathymetry (=depth), with pos. real numbers = 0 on land. (2) **constituents** (change with –c<file>). It is specified with character*2 strings (e.g., 'm2') and also contains information on how many time steps to take. The format ASCII. (3) **obc** This is the open boundary condition file. It is made using the first two files to specify the open boundary locations and constituents list, then sampling the current global file (**TPXO.3.ot**) to estimate the open boundary elevations. The format is FORTRAN sequential binary; one record = a complex*16 array *n_obc(nc,nob)*, where *nc* is the number of constituents specified in file (2), and *nob* is the number of open boundary nodes specified in the header record for file (1). **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** stdout; read arg; open, close units 1, 3; open, write, close units 10,11 **Common Blocks:** n/a
glob_case	This subroutine is an addition to the FWD_FAC program for the global case. **Calling Sequence:** glob_case(BS,SB,nm,m3,x2,ic,ipiv) **Data Declaration:** Integer nm, m3, ic, ipiv Complex x2, BS, SB **I/O:** stdout; open, read, write,close unit 15 **Common Blocks:** n/a
glob_case_c	This is a conjugate to the GLOB_CASE subroutine above for the global case. **Calling Sequence:** glob_case_c(BS,SB,nm,m3,x2,ic,ipiv) **Data Declaration:** Integer nm, m3, ic, ipiv Complex x2, BS, SB **I/O:** stdout; open, read, write, close unit 15 **Common Blocks:** n/a

Subroutine	Description
usage	**Calling Sequence:** usage() **Data Declaration:** n/a **I/O:** stdout **Common Blocks:** n/a

6.2.3.12 (Otis/src/rp_dp/glob_case.f)

Subroutine	Description
glob_case	This subroutine is an addition to the fwd_fac for the global case. **Calling Sequence:** glob_case(BS,SB,nm,m3,x2,ic,ipiv) **Data Declaration:** Complex BS, SB, x2 Integer nm, m3, ic, ipiv **I/O:** stdout, open, read, close unit 15 **Common Blocks:** n/a

6.2.3.13 (Otis/src/rp_dp/glob_case_c.f)

Subroutine	Description
glob_case_c	This subroutine is a conjugate to the **glob_case.f** for the global case. **Calling Sequence:** glob_case_c(BS,SB,nm,m3,x2,ic,ipiv) **Data Declaration:** Complex BS, SB, x2 Integer nm, m3, ic, ipiv **I/O:** stdout, open, read, close unit 15 **Common Blocks:** n/a

6.2.3.14 (Otis/src/rp_dp/gsmooth.f)

Subroutine	Description
gsmooth	This subroutine will only work if LTECO has previously been called and the common blocks are intact. **Calling Sequence:** gsmooth(atmp,gm,iuvflag,niter) **Data Declaration:** Integer iuvflag, niter Real atmp, gm **I/O:** stdout **Common Blocks:** n/a

6.2.3.15 (Otis/src/rp_dp/h_uv.f)

Subroutine	Description
h_uv	This subroutine will only work if LTECO has previously been called and the common blocks are intact. **Calling Sequence:** h_uv(ic,ll) **Data Declaration:** Integer ic Logical ll **I/O:** stdout **Common Blocks:** n/a

6.2.3.16 (Otis/src/rp_dp/inner.f)

This file contains all of the various inner product functions. All files contain '../include/**fwd_common.h**'.

Subroutine	Description
eval_crms	**Calling Sequence:** eval_crms(atmp,ma) **Data Declaration:** Integer ma Real atmp
eval_rms	**Calling Sequence:** eval_rms(atmp,ma) **Data Declaration:** Integer ma Real atmp
function eval_mean	**Calling Sequence:** eval_mean(atmp,ma) **Data Declaration:** Integer ma Real atmp
Weighted Inner Products	
eval_ipu	**Calling Sequence:** eval_ipu(atmp,btmp) **Data Declaration:** Real atmp,btmp
eval_ipv	**Calling Sequence:** eval_ipv(atmp,btmp) **Data Declaration:** Real atmp,btmp
eval_ipz	**Calling Sequence:** eval_ipz(atmp,btmp) **Data Declaration:** Real atmp,btmp
Unweighted Inner Products	
eval_ipu0	**Calling Sequence:** eval_ipu0(atmp,btmp) **Data Declaration:** Real atmp,btmp
eval_ipv0	**Calling Sequence:** eval_ipv0(atmp,btmp) **Data Declaration:** Real atmp,btmp
Boundary Intervals	Note that z is written as an area integral. The user must remember to divide out by the appropriate quantity.
eval_ibu	**Calling Sequence:** eval_ibu(atmp,btmp) **Data Declaration:** Real atmp,btmp
eval_ibv	**Calling Sequence:** eval_ibv(atmp,btmp) **Data Declaration:** Real atmp,btmp
eval_ibz	**Calling Sequence:** eval_ibz(atmp,btmp) **Data Declaration:** Real atmp,btmp
Complex Inner Products	
eval_icu	**Calling Sequence:** eval_icu(tmpc) **Data Declaration:** Complex tmpc
eval_icub	**Calling Sequence:** eval_icub(tmpc) **Data Declaration:** Complex tmpc
eval_icv	**Calling Sequence:** eval_icv(tmpc) **Data Declaration:** Complex tmpc

Subroutine	Description
eval_icvb	**Calling Sequence:** eval_icvb(tmpc) **Data Declaration:** Complex tmpc
eval_icz	**Calling Sequence:** eval_icz(tmpc) **Data Declaration:** Complex tmpc
eval_iczb	**Calling Sequence:** eval_iczb(tmpc) **Data Declaration:** Complex tmpc

6.2.3.17 (Otis/src/rp_dp/interp_rpx.f)

Subroutine	Description
interp_rpx	This subroutine interpolates a complex *nt* x *n* x *m* array *uv* at point *xlat*, *xlon*. **Calling Sequence:** interp_rpx(uv,nt,n,m,mz,th_lim,ph_lim,xlat,xlon, uv1,ierr,mtype) **Data Declaration:** Integer ierr, n,m, mtype, mz, nt Real th_lim, ph_lim, xlon, xlat Complex uv, uv1 **I/O:** stdout **Common Blocks:** n/a

6.2.3.18 (Otis/src/rp_dp/ipshift.f)

Subroutine	Description
ipshift	Function IPSHIFT creates periodic shift maps *i* to *i+ish*, mod *n*; (always between 1 and *n*, never 0). **Calling Sequence:** ipshft(i,ish,n) **Data Declaration:** Integer i, ish, n **I/O:** n/a

6.2.3.19 (Otis/src/rp_dp/lteco.f)

Subroutine	Description
lteco	This subroutine opens and reads a grid file. It constructs Finite Difference Coefficients and masks and depths for *u* and *v* nodes. *All* coefficient arrays are full. **Calling Sequence:** lteco(cgrid,b,ah,h0) **Data Declaration:** Real b, ah, h0 Character cgrid **I/O:** open, read unit 1; stdout; open, write unit 12 **Common Blocks:** n/a
read_ob	**Calling Sequence:** read_ob(nob,iob) **Data Declaration:** Integer nob, iob **I/O:** read unit 1 **Common Blocks:** n/a

6.2.3.20 (Otis/src/rp_dp/makeE, makeE_fwd.f)

Subroutine	Description

Subroutine	Description
fr_vel	With this subroutine, when *ik*=1, *u* scales are read. When *ik*=2, *v* scales are read. **Calling Sequence:** fr_vel(ik) **Data Declaration:** Integer ik **I/O:** open, read unit 1; stdout **Common Blocks:** n/a
makeE	This subroutine computes the inverse matrix E for a single constituent. It is the REPX version of MakeE. **Calling Sequence:** makeE(ic) **Data Declaration:** Integer ic **I/O:** open, close unit 1; stdout **Common Blocks:** n/a
makeE_d	This subroutine computes direct matrix E, using forcing *gu*, *gv*, *gz* with no inverting. It is the horizontal gradient of tide generating potential and boundary conditions for a forward problem. **Calling Sequence:** makeE_d(ic) **Data Declaration:** Integer ic **I/O:** open, close unit 1; stdout **Common Blocks:** n/a
makeE_fwd	This subroutine computes the inverse matrix E for a single constituent. It is the FWD_FAC version of MakeE with special drag treatment. **Calling Sequence:** makeE(ic) **Data Declaration:** Integer ic **I/O:** stdout; open, read, close unit 1 **Common Blocks:** n/a

6.2.3.21 (Otis/src/rp_dp/mkwts.f)

Subroutine	Description
def_seg	**Calling Sequence:** def_seg(iseg,mask,nn,L,nseg) **Data Declaration:** Integer iseg, mask, nn, L, nseg **I/O:** read, write stdout **Common Blocks:** n/a
def_seg_ob	**Calling Sequence:** def_seg_ob(iob,nob,seg,nseg) **Data Declaration:** Integer iob, nob, seg, nseg **I/O:** n/a **Common Blocks:** n/a
mkwts	This program makes latitude-dependant weights for discrete approximation of integrals on the C-grid and boundary weights are output in arrays in common block common/misc. It also makes coefficients for HV smoother (spatial correlation part of covariance) **Calling Sequence:** mkwts() **Data Declaration:** n/a **I/O:** open, read, close unit 99; read, write stdout; open, write, close unit 20 **Common Blocks:** n/a

6.2.3.22 Model Covariance Smoother Subroutine (Otis/src/rp_dp/modelcov.f)

Subroutine	Description
ahv	This subroutine applies Ah/Av to all horizontal/vertical segments. **Calling Sequence:** ahv(alpha_VH,nm,nseg,iseg,guv,w,ik) **Data Declaration:** Integer nm, nseg, iseg, ik Complex guv Real w, alpha_VH **I/O:** stdout **Common Blocks:** n/a
modelcov	This routine applies the diffusion model covariance smoother to adjoint system solution vectors. More precisely, in the notation of EBF this computes `C_f*W-1*u,` where *u* is the input vector, stored in arrays *gu*, *gv* (including unsmoothed/unscaled forcing and coastal boundary conditions), and *gz* (open boundary conditions). The smoothed and scaled fields are returned in the same arrays. Of note: 1) Input arrays *gu*, *gv*, *gz* are produced from array *z* (solution to conjugate transposed wave equations) by routine WAVEFRCT. 2) Only one constituent/one representer is done at a time, denoted by *ic*. Arrays for covariance scaling currently allow for interconstituent correlations. This is reflected in the two indices for *usc* and *vsc*. This version assumes NLP = 1, and NL = NC 3) This routine first multiplies *u* by W^{-1}, where *W* is a diagonal matrix of integration weights (necessary for solving a conjugate transpose system, not adjoint, as with time stepping). 4) After calling this routine, call WAVEFRC to convert smoothed forcing and boundary conditions into the RHS of the factored wave equation solution. **Calling Sequence:** modelcov(ic,gu1,gv1,gz1) **Data Declaration:** Integer ic Complex gu1, gv1, gz1 **I/O:** stdout **Common Blocks:** n/a
smth_1d	**Calling Sequence:** smth_1d(obseg,d,nseg,alpha) **Data Declaration:** Integer nseg Complex obseg Real alpha, d **I/O:** n/a **Common Blocks:** n/a
smth_ob	**Calling Sequence:** smth_ob(nob,lob,iob,gz1,ob,nseg_ob,alpha,bseg,zvar) **Data Declaration:** Integer nob, lob, iob, nseg_ob, bseg Complex gz1 Real ob, alpha, zvar **I/O:** stdout; open, read, close unit 20 **Common Blocks:** n/a

6.2.3.23 (Otis/src/rp_dp/out_file_init.f)

Subroutine	Description
out_file_init	**Calling Sequence:** out_file_init(nrept,irep,cfout,cpathout,npathout) **Data Declaration:** Integer nrept, irep, npathout Character cfout, cpathout **I/O:** write ctemp **Common Blocks:** n/a
out_file_uv	**Calling Sequence:** out_file_uv(nrept,irep,cfout,cpathout,npathout) **Data Declaration:** Integer nrept, irep, npathout Character cfout, cpathout **I/O:** ctemp **Common Blocks:** n/a

6.2.3.24 Run Parameter Subroutines (Otis/src/rp_dp/param_subs.f)

Subroutine	Description
fwd_params	**Calling Sequence:** fwd_params(pname,p,qmode,uv_obc,rad_obc,mk_drag,dfname) **Data Declaration:** Real p, mk_drag Logical qmode, uv_obc, rad_obc Character pname, dfname **I/O:** n/a **Common Blocks:** n/a
mkb_params	**Calling Sequence:** mkb_params(pname,mod_fname,data_fname,cor_fname, prior, data,cor_mod) **Data Declaration:** Logical prior, cor_mod, data Character pname, mod_fname, data_fname, cor_fname **I/O:** n/a **Common Blocks:** n/a
mkSpeed_params	**Calling Sequence:** mkSpeed_params(pname,p,qmode,tfile) **Data Declaration** Real p Character pname, tfile Logical qmode **I/O:** stdout **Common Blocks:** n/a
q_params	**Calling Sequence:** q_params(pname,p,qmode,ramx) **Data Declaration:** Real p, ramx Character pname Logical qmode **I/O:** n/a **Common Blocks:** n/a
rd_run_param	**Calling Sequence:** rd_run_param(pname,p) **Data Declaration:** Real p Character pname **I/O:** open, read, close unit 1 **Common Blocks:** n/a

Subroutine	Description
reduce_params	**Calling Sequence:** reduce_params(nreps,p,qmode,sige,trunc,ramx,n_blk,i1,i2) **Data Declaration:** Real p, sige, ramx Logical qmode, uv_obc Integer n_reps, n_blk, i1, i2, trunc **I/O:** stdout **Common Blocks:** n/a
repx_params	**Calling Sequence:** repx_params(pname,p,qmode,uv_obc,mk_drag, int_var_sc, rb_var_sc, ob_var_sc,dfname,ccov) **Data Declaration:** Real p, mk_drag, int_var_sc, rb_var_sc, ob_var_sc Character pname, dfname,ccov Logical qmode, uv_obc **I/O:** n/a **Common Blocks:** n/a
rlc_params	**Calling Sequence:** rlc_params(pname,p,qmode,uv_obc,mk_drag, int_var_sc, rb_var_sc, ob_var_sc,dfname,k_rlz,z_prior,uv_prior,ccov) **Data Declaration:** Real p, mk_drag, int_var_sc, rb_var_sc, ob_var_sc Character pname, dfname, z_prior, uv_prior, ccov Logical qmode, uv_obc Integer k_rlz **I/O:** n/a **Common Blocks:** n/a
sml_params	**Calling Sequence:** sml_params(pname,p,qmode,uv_obc,mk_drag, int_var_sc,rb_var_sc,ob_var_sc,int_var_sc_s,rb_var_sc_s,ob_var_sc_s,dfname,ccov) **Data Declaration:** Logical uv_obc, qmode Character pname, dfname, ccov Real p, mk_drag, int_var_sc, rb_var_sc, ob_var_sc, int_var_sc_s, rb_var_sc_s, ob_var_sc_s **I/O:** n/a **Common Blocks:** n/a
sr_params	**Calling Sequence:** sr_params(pname,p,qmode,prior) **Data Declaration:** Real p Character pname, prior Logical qmode **I/O:** n/a **Common Blocks:** n/a
varest_params	**Calling Sequence:** varest_params(pname,p,qmode,no_diff, uv_obc,mk_drag, z_prior, uv_prior,dfname) **Data Declaration:** Real p, mk_drag Character pname, z_prior, uv_prior, dfname Logical qmode, no_diff, uv_obc **I/O:** n/a **Common Blocks:** n/a

6.2.3.25 Posterior Error Calculation Subroutines (Otis/src/rp_dp/pe_subs.f)

These are subroutines used for posterior error calculations and matrix reduction (including blocking version).

Subroutine	Description
b_slv	This subroutine solves for b. Calculate c = U S (S*S + sig I) ^-1 W' D (and then the representer coefficients bhat = E*c_bar). **Calling Sequence:** b_slv(W,s,U,E,dp,sige,nreps,trunc,bhat,nlp,nl) **Data Declaration:** Real W, s, U,dp, sige Character pname Complex E, bhat Integer nreps, trunc, nlp, nl **I/O:** open, write, close unit 2 **Common Blocks:** n/a
cut	This is an integer function. **Calling Sequence:** cut(name) **Data Declaration:** Character name **I/O:** n/a **Common Blocks:** n/a
mprod	This is a complex matrix product subroutine. **Calling Sequence:** mprod(A,na,ma,B,nb,mb,C) **Data Declaration:** Complex A, B, C Integer na, ma, nb, mb **I/O:** stdout **Common Blocks:** n/a
qr_reduce_b	**Calling Sequence:** qr_reduce_b(G,nr,ncol,rio,d,dp) **Data Declaration:** Real G, rio, d, dp Integer nr, ncol **I/O:** stdout **Common Blocks:** n/a
rd_b	This subroutine reads out **B.dat** and arranges the lats and lons of evaluation sites, as well as the type and index of evaluation site. **Calling Sequence:** rd_b(fname,nrows,rlat,rlon,mtype,mrow, B,sigma,dp) **Data Declaration:** Real rlat, rlon, sigma, dp Character fname Integer mtype, mrow, nrows Complex B **I/O:** open, read, close, unit 21; stdout **Common Blocks:** n/a
rd_b1	This version of RD_B does not return matrix B, but does return *nrec* - to read small B(*1, nc*) for a data site. It reads out **B.dat** and arranges the lats and lons, type and index of evaluation sites. **Calling Sequence:** rd_b1(fname,nrows,rlat,rlon,mtype,mrow,sigma,dp,nrec) **Data Declaration:** Integer nrows, mtype, mrow, nrec Real rlat, rlon, sigma, dp

Subroutine	Description
	Character fname **I/O:** open, read, close unit 21; stdout **Common Blocks:** n/a
rd_b1b	This version of RD_B does not return matrix B, but does return *nrec* - to read small B(*1*, *nc*) for a data site. It reads out **B.dat** and arranges the lats and lons, type and index of evaluation sites. **Calling Sequence:** rd_b1b(fname,nrows,rlat,rlon,mtype,mrow,B,sigma,dp,nrec) **Data Declaration:** Integer nrows, mtype, mrow, nrec Real rlat, rlon, sigma, dp Character fname Complex B **I/O:** open, read, close unit 21; stdout **Common Blocks:** n/a
rd_bb	This is a blocking version of RD_B. The blocking version does not return matrix B, but does return *nrec* - to read small B(*2*nc*,*nc*) for a data site. It reads out **B.dat** and arranges the lats and lons, type and index of evaluation sites. **Calling Sequence:** rd_bb(fname,nrows,rlat,rlon,mtype,mrow,sigma,dp,nrec) **Data Declaration:** Real rlat, rlon, sigma, dp Character fname Integer mtype, nrows, mrow, nrec **I/O:** open, read, close unit 21; stdout **Common Blocks:** n/a
rd_rp	This subroutine loads R or P matrices. **Calling Sequence:** rd_rp(fname,P,ndat,nreps,k0,l0,np,mp) **Data Declaration:** Character fname Integer ndat, nreps, k0, l0, np, mp Complex P **I/O:** open, read, close unit 3; stdout **Common Blocks:** n/a
scale	This subroutine makes an *sc* array for scaling data. **Calling Sequence:** scale(sigma,sigtg,ndat,sc) **Data Declaration:** Real sigma, sigtg, sc Integer ndat **I/O:** n/a **Common Blocks:** n/a
sigscl	This routine scales data and B matrices using site dependent scales provided in *sc* (the same scale for all data at one site). **Calling Sequence:** sigscl(B,d,nsite,sc) **Data Declaration:** Real sc,d Complex B Integer nsite **I/O:** n/a **Common Blocks:** n/a

Subroutine	Description
sigscl_b	This routine scales data and B matrices using site-dependent scales provided in *sc* (the same scale for all data at one site). **Calling Sequence:** sigscl_b(d,nsite,sc) **Data Declaration:** Real sc, d Integer nsite **I/O:** n/a **Common Blocks:** n/a
tinv	This subroutine is an inverse of an upper triangular matrix using Basic Linear Algebra Subprograms (BLAS). **Calling Sequence:** tinv(r,ncol) **Data Declaration:** Integer ncol Real r **I/O:** n/a **Common Blocks:** n/a

6.2.3.26 (Otis/src/rp_dp/r_sites.f)

Subroutine	Description
read_sites	**Calling Sequence:** read_sites(fname,ntotal,rlats,rlons,rid, rtype,the,phi) **Data Declaration:** Integer rid, rtype, ntotal Real rlats, rlons, the, phi Character fname **I/O:** open, read, close unit 1; stdout **Common Blocks:** n/a

6.2.3.27 (Otis/src/rp_dp/rd_c_alpha.f)

Subroutine	Description
interp	This subroutine interpolates real *n* x *m* array onto point *xlat*, *xlon*. **Calling Sequence:** interp(r,n,m,th_lim,ph_lim,xlat,xlon, r1,ierr) **Data Declaration:** Real r1, r, th_lim, ph_lim, xlat, xlon Integer ierr, n,m **I/O:** read, write stdout; write unit 0 **Common Blocks:** n/a
ipshft	This is a function that performs periodic shift maps *i* to *i*+*ish*, mod *n*; always between 1 and *n*, never 0. **Calling Sequence:** ipshft(i,ish,n) **Data Declaration:** Integer i, ipshift, n, ish **I/O:** n/a **Common Blocks:** n/a
rd_c_alpha	This subroutine is for debugging. It may be adjusted to suit the environment. **Calling Sequence:** rd_c_alpha(iuv,con,var) **Data Declaration:** Real var Character con Integer iuv

Subroutine	Description
	I/O: stdout; open, read, close unit 7 **Common Blocks:** n/a

6.2.3.28 (Otis/src/rp_dp/rd_num.f)

Subroutine	Description
rd_num	**Calling Sequence:** rd_num(arg,nrep1,nrep2) **Data Declaration:** Integer nrep1, nrep2 Character arg **I/O:** read arg **Common Blocks:** n/a

6.2.3.29 (Otis/src/rp_dp/read_b.f)

Subroutine	Description
read_b	READ_B reads out **B.dat** and arranges lats and lons, type and index of evaluation site. **Calling Sequence:** read_b(nrows,rlat,rlon,the,phi,mtype,mrow,lat2,lon2) **Data Declaration:** Integer nrows, mtype, mrow Real rlat, rlon, the, phi, lat2, lon2 **I/O:** open, read, close unit 21; stdout; write unit 6 **Common Blocks:** n/a

6.2.3.30 (Otis/src/rp_dp/reduce_b.f)

Subroutine	Description
reduce_b	The routine REDUCE_B calculates the representer coefficients that are used to form the final inverse solution. It calculates the representer coefficients. If the maximum available RAM value is properly set in run_param, REDUCE_B will automatically generate a warning if blocking needs to be used or if a greater number of blocks should be used to fit the matrix calculations into available memory. By default, no blocking is set (that is the number of blocks is equal to one), but this can be changed in **run_param** or in command line using -n<number_of_blocks> option. To compile type `make reduce_b` in **OTIS/local/MyArea/exe/run_param**. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** open, read, close units 3, 4, 21, 25, 26; read, write stdout; write units 13, 3, 25, 26; read arg **Common Blocks:** n/a
rlc_cg_cor	**Calling Sequence:** rlc_cg_cor(nl,trunc) **Data Declaration:** Integer nl, trunc **I/O:** stdout; open, read, write, close unit 17 **Common Blocks:** n/a
usage	**Calling Sequence:** usage() **Data Declaration:** n/a **I/O:** stdout;

Subroutine	Description
	Common Blocks: n/a

6.2.3.31 Representer Calculation Program (Otis/src/rp_dp/repx.f)

Subroutine	Description
mk_rlc_cg_cor_inc	**Calling Sequence:** mk_rlc_cg_inc(m3,nm,nl,rlc_cg_only) **Data Declaration:** Integer m3, nm, nl, Logical rlc_cg_only **I/O:** stdout; open, read, write, close unit 21; open, write, close 17 **Common Blocks:** n/a
repx	This is a test version of the representer calculation program using a direct solver of the wave equation in elevation. It is modified from an old time stepping version. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** read, write stdout; read arg; open, close unit 1; open, write, close unit 3 **Common Blocks:** n/a
usage	**Calling Sequence:** usage() **Data Declaration:** n/a **I/O:** stdout **Common Blocks:** n/a

6.2.3.32 Representer Calculation Program (Otis/src/rp_dp/rlc.f)

Subroutine	Description
cut	Integer function. **Calling Sequence:** cut(name) **Data Declaration:** Character name **I/O:** n/a **Common Blocks:** n/a
rlc h_uv	This program used to be called DIRECTSLV. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** read, write stdout; read arg; open, read, close units 1, 3; write units 3, 6, bnum **Common Blocks:** common/datablk
usage	**Calling Sequence:** usage() **Data Declaration:** n/a **I/O:** stdout **Common Blocks:** n/a

6.2.3.33 Representer Calculation Program (Otis/src/rp_dp/rpx_to_p.f)

The program RPX_TO_P creates the Hermitian representer matrix **R** corresponding to harmonically analyzed data at the representer sites (i.e., the elements of **R** are elevation or velocity representers, evaluated at each representer site), and the matrix **P** (representers for harmonically analyzed data evaluated at all data locations). The calculation is controlled by the representer list in ../prm/**lat_lon.rep**. The matrices **P** and **R**, together with **B** (from the previous step) are used to do the matrix computations needed for finding the representer coefficients. Note that you have to run rpx_to_p twice in different modes to get both **P** and **R**. Also note that representers must be calculated (by REPX) and placed in **OTIS/local/“MyArea”/repx** before this program can be run.

Subroutine	Description
mklist	**Calling Sequence:** mklist(nrep,cfrep,cfruv,ireps) **Data Declaration:** Character cfrep, cfruv Integer nrep, ireps **I/O:** write ctemp **Common Blocks:** n/a
rd_num	**Calling Sequence:** rd_num(arg,nrep1,nrep2) **Data Declaration:** Character arg Integer nrep1, nrep2 **I/O:** read arg **Common Blocks:** n/a
rpx_to_p	This program reads from direct access representer files (one representer per file) to construct the generalized, possibly rectangular, representer matrix for multi-constituent representers, which may be block correlated. **Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** read, write stdout; open, read, write, close unit 1; open, write, close new_unit, units 13, 18; read arg **Common Blocks:** n/a
usage	**Calling Sequence:** usage() **Data Declaration:** n/a **I/O:** stdout **Common Blocks:** n/a
wrt_blk	**Calling Sequence:** wrt_blk(rm,n_blk,ntot,nrep,i_blk,iounit) **Data Declaration:** Complex rm Integer n_blk, ntot, nrep, i_blk, iounit **I/O:** read, write stdout; write iounit **Common Blocks:** n/a

6.2.3.34 Representer Calculation Program (Otis/src/rp_dp/SALset.f)

Subroutine	Description
SALset	This is a direct solver version (one constituent) of the SALset found in Section 6.2.1.10. SALset reads in a tidal loading-ocean self attraction file in standard model output format, interpolates it onto the current grid, if necessary, and computes gradients. It adds the result to forcing arrays *gu* and *gv*. This routine is called after calling the LTECO and ATGF subroutines. SALset uses FD coefficient weights computed in LTECO to calculate gradients of TLOSA "equilibrium height". **Calling Sequence:** SALset(ic,c_id,c_sal)

Subroutine	Description
	Data Declaration: Integer ic Character c_sal, c_id **I/O:** open, read, close unit fid; stdout; open, write, close units 0, 33 **Common Blocks:** n/a

6.2.3.35 (Otis/src/rp_dp/ Sfac.f)

Subroutine	Description
Sfac	This subroutine generates and factors a matrix for the wave equation in elevation. It is derived from shallow water equations on the C-grid for a single constituent. The number *ic* makes $m3 = 3*m+4$ and *nm* are array dimensions for SB. **Calling Sequence:** Sfac(ic,m3,nm,SB,II,JJ,KK,ipiv) **Data Declaration:** Integer ic, m3, nm, II, JJ, KK, ipiv Complex SB **I/O:** stdout **Common Blocks:** n/a

6.2.3.36 (Otis/src/rp_dp/varest.f)

Subroutine	Description
varest	**Calling Sequence:** n/a **Data Declaration:** n/a **I/O:** read, write stdout; read arg; open, read, close units 1, 3, 99; open, write, close unit 15 **Common Blocks:** n/a

6.2.3.37 (Otis/src/rp_dp/wrt_uvsc.f)

Subroutine	Description
wrt_uvsc	**Calling Sequence:** wrt_uvsc(usc,vsc,n1,m1,ncu,gm,niter,l, gm_ob,niter_ob,l_ob,zvar) **Data Declaration:** Integer n1, m1, ncu, niter, niter_ob Real usc, vsc, zvar, gm, l, gm_ob, l_ob **I/O:** open, write, close unit 1; stdout **Common Blocks:** n/a

7.0 FORTRAN Common Blocks

7.1 COMMON Blocks (OTIS/bin)

COMMON/ CUNITS	**Type**	**Description**
in5	Integer	
indir	Integer	
inephm	Integer	
inflag	Integer	
inmss	Integer	
inssh	Integer	
intime	Integer	
iout6	Integer	
COMMON/ CONSTRSR8	**Type**	**Description**
secday	Real	
COMMON/ CONSTSI4	**Type**	**Description**
iundf4	Integer	
COMMON/ CONSTSI2	**Type**	**Description**
inundf2	Integer	
COMMON/ CMISSION	**Type**	**Description**
ncycles	Integer	
nperiod	Integer	
nrecldir	Integer	
nreclephm	Integer	
nreclmss	Integer	
nreclsshf1	Integer	
nrecltime	Integer	
numrevs	Integer	
COMMON/ CFLAG	**Type**	**Description**
errflag	Character	
COMMON/ CFNAMERSR	**Type**	**Description**

7.2 COMMON Blocks (OTIS/rp_dp)

COMMON/	**Type**	**Description**

DATABLK		
depth		
i0		
i1		
j0		
j1		
nrep		
phi		
rid		
rph		
rth		
rtype		
spwt		
the		
COMMON/ RMULTBLK	**Type**	**Description**
Bm		
ic		
ipiv_cb		
sig_e		

8.0 TOPS Main Argument Variables

8.1 Primary TOPS Variables

Variable	Description
alat	Latitude of grid in °N.
alat2	Latitude of pt to be located in ° N.
amsk	2D land-sea mask array: =0/1 at pts to be not_plotted/plotted.
b_slv	Solve for b.
cau(nc,n,m)	Real and imaginary parts of steady state complex amplitudes for *u*.
cav(nc,n,m)	Real and imaginary parts of steady state complex amplitudes for *v*.
caz(nc,n,m)	Real and imaginary parts of steady state complex amplitudes for *z*.
cint	Contour interval. If *cint*<0, contour lines are selected so that zero contour is between two contour lines.
cmin, cmax	Min and max contours. If *cmin*=*cmax*=0, min and max contours are calculated from *f*.
cobc	File name for open BC file.
con	Constituent ID (char*4).
count	Averaged values counter.

Variable	Description
cu, cv0, cvp	Continuity.
cut	Finds index of last, non-empty symbol in a string.
dt	Time step.
du,dv	Dissipation: Applied half at the forward and half at the backward time step.
e	Inverse sqrt of square (calculated) representer matrix.
elon	Longitude of grid in °E.
elon2	Longitude of pt to be located in °E.
f	Field to be contoured.
fu,fv	Coriolis.
gu(nc,n,m)	Array of complex forcing amplitudes for variable *u*, constituent l.
gv(nc,n,m)	Array of complex forcing amplitudes for variable *v*, constituent l.
gz(nc,n,m)	Array of complex forcing amplitudes for variable *z*, constituent l.
hu	Interpolated depth for *u* nodes.
hv	Interpolated depth for *v* nodes.
i_blk	Corresponds to block number.
icycle	Repeat cycle number.
id	Day.
id(nc)	Constituent ID's (e.g., m2, s2 etc.).
idx	Index within the revolution.
iflag	I*2 - flag word with individual bits set.
ilat i*4	Latitude array in microdegrees.
ilon i*4	East longitude array in microdegrees.
intx, inty	Number of intervals to be labeled on *x* and *y* axes.
iob,job	Indices of boundary pts at elev pts.
iobi, jobi	Indices of interior pts next to open bndy pts.
isdata	Is .false. if there is no data available for this track.
istat	Returned stats flag: =0 pt lies outside grid. =1 pt lies within grid, location found.
istep1, istep2	First and last time step number.
iyyy	Year.
kob	Index to denote direction of associated interior pt: (1 = +*x*, 2 = -*x*, 3 = +*y*, 4 = -*y*, 0 = corner pt).
lbit16	Logical array with MSB corresponding to index at 1 of array: -true means corresponding bit in *lflag* is set to 1. -false means corresponding bit in *iflag* is set to 0.
lendplt	Logical flag to end plot (if true).
lintit	Number of lines in title.
m	Number of latitude subdivisions.
mjd>0	Modified Julian days.

Variable	Description
mm	Month.
mn	Integer m3.
mprod	C=A*B (complex).
mu,mv,mz	Masking arrays for *u*,*v*, and *z*, respectively.
mz	Array mask.
n	Number of longitude subdivisions.
n,m	Dimensions of grid.
n,m	Number of rows, columns of h-nodes.
n,m	Dimensions of field *f* to be contoured.
n_blk	Records - switch for the header.
nc	Number of frequency components (tidal constituents).
ncmax	Max number of tidal constituents.
ncprmx	Maximum number of constituents.
ncsmx	Max number of tidal constituents.
ndat	Integer, number of data sites.
ndatmx	Maximum number of points.
nft	Starting step for harmonic analysis.
ni	Leading dimension of arrays *amsk* and *f*.
nindrfi*4	Number of geo-referenced indices in revolution *ntref*.
nl	Integer, number of constituent groups.
nlp	Integer, number of constituents per group.
nmax, mmax	Maximum grid dimensions (see **nobmx.h**).
nob	Total number of open boundary pts.
nobmx	Maximum allowable number of open bndy pts.
nreps	Integer, number of representers.
nsamp	Sampling frequency for harmonic analysis.
nsmax	Max number of IHO stations in domain.
nt	Number of evaluation times for temporal average.
ntrack	Track number within the repeat cycle.
ntref i*4	Track number (1 - 501).
omega(nc)	Forcing frequencies (angular frequency).
ph_lim	Give latitude and longitude limits of grid.
pu, pv	Pressure.
qr_reduce_b	Blocking version of *qr_reduce*.
rd_b1	Read matrix ../dat/**b1.dat** (no *b* returned).
rd_b1b	Read matrix ../dat/**b1.dat** (*b* returned).
rd_bb	Blocking version of RD_B.
rd_rp	Read ../dat/**p.dat** or ../dat/**r.dat**.
scale	Find scales.
sige	Error variance. If all other variances (including dynamical error variances) are

Variable	Description
	correct, this should be = 1.
sigscl	Scales *B* and *d* with *sc*.
sigscl_b	Blocking version of *sigscl* (no *B* scaling).
th_lim	Give latitude and longitude limits of grid.
time	Modified Julian date of sea surface height (returned as decimal MJD).
title	Title for plot.
u0	Coefficients for interior *u* nodes (horizontal smoothing).
ui	Coefficients for interior *u* nodes (vertical smoothing).
ujm	Coefficients for smoothing OB *u* nodes.
umax	Maximum velocity scale.
umin	Minimum velocity scale used in the drag coefficient.
uv	Assumed given on "h-nodes" of C-grid.
v0	Coefficients for interior *v* nodes (horizontal smoothing).
var(n1,m1)	Fractional error due to discretization of elevation gradient.
vi	Coefficients for interior *v* nodes (vertical smoothing).
vjm	Coefficients for smoothing OB *v* nodes.
w,s,u	SVD of *G* matrix.
wbu	Gives weights for boundary *u* nodes.
wbv	Gives weights for boundary *v* nodes.
wbz	Gives weights for boundary *z* nodes.
wcu	Cos(theta) for *u*-rows.
wcv	Cos(theta) for *v*-rows.
wiu	Gives weights for interior *u* nodes.
wiv	Gives weights for interior *v* nodes.
x,y	Returned grid pt location.
xmin,xmax	Min and max values of *x* (lon) to be labeled on plot.
xtit,ytit	Titles for *x* and *y* axes.
ymin,ymax	Min and max values of *y* (lat) to be labeled on plot.

9.0 NOTES

9.1 Acronyms and Abbreviations

Acronym	Description
ADCP	Acoustic Doppler Current Profiler
AMD	Advanced Micro Devices
ASCII	American Standard Code for Information Interchange
BC	Boundary conditions
BLAS	Basic Linear Algebra Subprograms
bndy	boundary
BSI	Bilinear Spline Interpolation
CM	Connection Machine
CODAR	Coastal Ocean Dynamics Application Radar
d	Day
DBDB2	Digital Bathymetric Database, resolution 2 km
DF	Derivative Function
EAS	East Asian Seas
ERS1/2	European Remote Sensing Satellites 1 and 2
FD	Finite Difference
GUI	Graphical User Interface
GVC	General Vertical Coordinate
I/O	Input/Output
IHO	International Hydrographic Office
LHS	Left Hand Side
LTEs	Laplacian Tide Equations
m	Meter
MB	Megabytes
MJD	Modified Julian Date
mm	Month
NCOM	Navy Coastal Ocean Model
NOAA	National Oceanographic and Atmospheric Administration
NRL	Naval Research Laboratory
OBC	Open Boundary Conditions
OSU	Oregon State University
OTIS	OSU Tidal Inversion Software
PC	Personal Computer
PSI	Planning Systems, Incorporated
pt	point
RAM	Random Access Memory
RELO NCOM	Relocatable Navy Coastal Ocean Model
RHS	Right Hand Side
RMS	Root Mean Square

Acronym	Description
SAL	Self-attraction/Loading
SDD	Software Design Description
SSC	Stennis Space Center
SSH	Sea Surface Height
SVD	Singular Value Decomposition
SWE	Shallow Water Equations
T	Time
TBC	Tidal Boundary Condition
TDB	Tidal Database
TG	Tide Gauge
TLOSA	Tidal Loading-Ocean Self Attraction File
TOPEX	TOPography EXPeriment
TOPS	Tidal Open-boundary Prediction System
UTC	Coordinated Universal Time
VTR	Validation Test Report
WVS	World Vector Shoreline