AFIT/GE/ENG/92D-36

An Object-Oriented Computer Aided Design Program for Modern Control Systems Analysis

THESIS

Frederick L. Trevino, Capt, USAF
AFIT/GE/ENG/92D-36

Approved for public release; distribution unlimited

AFIT/GE/ENG/92D-36

An Object-Oriented Computer Aided Design Program for Modern Control Systems Analysis

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Frederick L. Trevino, B.S.
Capt, USAF

December 1992

Approved for public release; distribution unlimited

Preface

There are two prevailing, and often conflicting, work ethics that drive any product. The first is that quantity has its own quality and the second is that quality has its own quantity. This author is an ardent subscriber to the second ethic. Thus, throughout this work, I've striven to produce a perfect product. While I fully recognize that perfection can never be achieved, it remains, nonetheless, the goal, the bulls-eye towards which I aim. But what is perfection in software and precisely what is the goal of this investigation? Very simply, it is to produce a computer aided design program for control systems engineers that is at once intuitively easy to use, completely bug free, and precisely accurate in all numerical computations. Any such attempt involves much more that just control systems theory but also encompasses numerical methodology, software engineering, and human factors engineering. Obviously, the scope of such an effort is well beyond any single masters thesis effort and much work remains at the conclusion of this one. Hopefully, we've laid a good foundation for future development. Much of this research is just that, a foundation upon which to build. We've changed the very format of ICECAP-PC from a functionally structured form to a highly modular object-oriented format. Furthermore, we've looked at every single basic numerical algorithm, changed most of them, and brought the numerical methodology of ICECAP-PC into the state of the art. Finally, we've provided links for future toolbox design in areas of H_∞, LQR, LQG, etc. I look forward to seeing future developments of this program.

I owe a debt of gratitude to many people who've supported this effort foremost among whom is my lovely wife Patricia. I'm constantly amazed at her patience having put up with me for 12 years while I got a BS and now an MS in electrical engineering. We've been through both good times and tough times and always we did it together. I'm also indebted to my three sons whom I've ignored far too much and whom I look forward to spoiling absolutely rotten. Thanks are also due to my very capable thesis buddy-dude, Wayne Bell, and my advisor Dr. Gary Lamont. Its been great.

These are dynamic times. No one in my class envisioned a military draw-down that would put AFIT students out of the Air Force and into the street upon graduation. I've known many of them for years being enlisted together, commissioned together, and brought to AFIT together. Together, we've learned camaraderie, friendship and even brotherhood. We've served our country well and it is to them that I dedicate this thesis, my final Air Force product.

Table Of Contents

Table of Contents

Table of Figures

Table of Listings

Abstract

This investigation is a continuation of the ICECAP-PC research project conducted under Prof. Gary B. Lamont at the Air Force Institute of Technology. It is an ongoing development of a public domain Computer Aided Design (CAD) package for Control Engineering students and faculty with a special emphasis on education. This investigation focuses on three areas. First, an object-oriented environment is specified, designed and implemented. The functional structure of ICECAP-PC is then converted into an object-oriented structure because object-orientation provides an advanced logic and implementation structure and effectively addresses common software engineering issues. Additionally, a new interface featuring pull-down menus, mouse support, and single stroke access provides optimal user friendliness. Second, the numerical methods used for modern control capabilities are updated providing state of the art numerical capabilities. Third, using a program extension known as a toolbox, a basic MIMO (Multiple Input Multiple Output) faculty is created to augment the MISO capabilities. The MIMO QFT toolbox allows the entry and manipulation of up to a 3x3 MIMO plant matrix of transfer functions for the QFT solution to MIMO control systems problems.

1 Introduction

CACSD (Computer Aided Control System Design) software development is an inherently complex process because of (1) the multitude of mathematical operations and capabilities required and (2) the variety of requirements posed by the end users. Most programs today are modular in structure and tasks are compartmentalized into functional procedures (subroutines). User commands are processed through a hierarchical tree of procedures until the command is fully executed. Relatively new, and growing rapidly, is the use of object-oriented programming (OOP) techniques. While the software engineering community embraces this technique with open arms, other engineering disciplines, control systems engineering inclusive, have been slow to adopt this technology. This is unfortunate because object-orientation provides an ideal structure for CACSD program design because structural complexities are elegantly resolved.

1.1 History of CACSD (Kheir, 1988)

The evolution of CACSD (Computer Aided Control System Design) directly follows the evolution of control system theory itself. The defense industry has been the major driving force in the development of automatic control systems since World War II as increasingly sophisticated weaponry was needed. In the period between 1930 to 1950, the largest category of problems was the SISO (Single Input Single Output) systems and design was predominantly performed in the frequency domain. Prominent figures were W.R. Evans and H. Nyquist. Early control systems problems were solved without the aid of computer and only simple cases (by today's standards) could be considered.

As systems become more complex and MIMO (Multiple Input Multiple Output) problems were presented, the frequency domain graphical techniques of the past decades failed to produce solution. Among others, R. Kalman lead engineers into the time domain where systems were represented in state space by a set of first order differential equations in place of nth order ordinary differential equations. Concurrent, with these developments, was the advent of the computer age. Engineers began producing code to solve very specific problems. However, there was no single collection of routines unified under a single package until the late

seventies. In the early days of computing, the largest obstacle was the lack of interactive systems. Programs were place on punch cards and run in batch mode; a process that was extremely time consuming.

In the early seventies, researchers began to look again at the frequency domain for control systems solutions and diversification of theory began to rapidly take place. Different approaches were presented for different classes of problems. Concurrent with these developments, was the development of collections of routines previously scattered. In 1977, Fredrick L. O'Brian in his masters thesis a the Air Force Institute of Technology developed the *Consolidated Computer Program for Control System Design* which was a collection of these routines (O'Brian, 1977). In 1978, follow on work at AFIT produced TOTAL, recognized as the first interactive CACSD program. In 1979, another such product, INTOPS was produced. In the fall of 1989, K.J. Åström and G. Golub held the first Conference On Numerical Techniques in Control. At this conference, Cleve Moler, demonstrated the newly released MATLAB software package which has since become the most widely used program for control systems design.

MATLAB was not designed for CACSD use at all and had many shortcomings (Kheir, pg 644). Several other software programs have since come to the surface to address these shortcomings including $Matrix_x$ by Integrated Systems and Control-C from Systems Control Technology. Many of these programs, now much more mature than at their inception, offer considerable power but come at considerable price. There still exists a need for a quality public domain CACSD program directed at the educational community. ICECAP-PC fulfills this requirement.

1.2 History of ICECAP-PC

Since 1977, graduate students at the Air Force Institute of Technology (AFIT), under the direction of Prof. Gary B. Lamont, have contributed to the ICECAP (Interactive Control Engineering Computer Analysis Package) program. ICECAP traces its origin to a masters thesis entitled *Consolidated Computer Program for Control System Design* by Fredrick L. O'Brian. As a follow on effort, Stanley Larimer, in his thesis effort entitled *An Interactive Computer-Aided Design Program for Discrete and Continuous Control System Analysis and Synthesis* created the program known as TOTAL (Larimer, 1978). TOTAL incorporated the ability to analyze systems in both the discrete and the continuous time domains and was developed in FORTRAN for the

CDC Cyber. In 1981, Glen Logan rehosted TOTAL for the DEC VAX-11/780 and renamed the program ICECAP. In 1982, Charles Gembarowski added the menu driven interface implemented in Pascal. Work continued on the VAX version of ICECAP through many thesis cycles and in 1985, Susan Mashiko and Gary Tarjyruski developed the program for the personal computer renaming it ICECAP-PC. This version of ICECAP-PC went through nine design revisions, the last being 9.0A. The current task and subject of this thesis is the complete and comprehensive re-hosting of ICECAP-PC into an object-oriented format using the interface definition of Turbo Vision in Turbo Pascal 6.0. Object-orientation provides an advanced logic and implementation structure and effectively addresses the software engineering issues of extendibility, maintainability, reusability, etc. Additionally, ICECAP-PC provides for the addition of toolboxes to the basic program. Toolboxes provide additional capability to ICECAP by implementing new control system theories without modification of the core ICECAP-PC code. In this latest edition of ICECAP-PC, both MISO and MIMO QFT toolboxes are included.

ICECAP-PC is a public domain CACSD tool targeted for educational use. Every effort is made to ensure that ICECAP-PC Release 10 is mathematically correct, rich in capability, and both easy and quick to use. The purpose is simply to challenge the state of the art in CACSD software design. Thus, the new ICECAP-PC is easier to use, more accurate, faster, leaner, more capable, and robust than any prior version.

1.3 General Objectives

This project encompasses three basic objectives: The development of an object-oriented, user friendly CACSD environment, the refinement of numerical methods used by ICECAP-PC in the solution of modern and classical control problems, and the development and inclusion of QFT MISO and QFT MIMO toolboxes.

The Object-Oriented CACSD Environment

The first general objective is to provide an object-oriented CACSD environment to perform basic control system analysis functions such as polynomial and matrix manipulations, and time and frequency domain analysis with a user-friendly interface including graphical presentations, help, and macro facilities. While these already exist in the current modular version of ICECAP-PC, they are not provided with the sophistication available with modern software design techniques. In this case, we mean both structural sophistication and interface sophistication where structural sophistication is the use of modern software constructs (i.e. object-orientation) to enforce software engineering principles and interface sophistication is the use of a window based, event driven interface based on well developed human factors engineering principles (Bell, 1992). Very few engineering programs are written with human factors concepts included in the design process. A good interface is not merely a luxury but frees the engineer to concentrate on the problem at hand not having to struggle with the computer itself.

The Refinement Of Numerical Methods

The second objective, the refinement of numerical methods used in ICECAP-PC is important for two reasons. First, mathematical procedures of previous ICECAP versions are dated having been developed in the late seventies and early eighties. Much progress has been made since then and even the standard usage of Linpack (Dongarra, 1979) and Eispack (Smith, 1976) routines no longer provides optimal speed and accuracy. Second, the mathematical engine of previous ICECAP-PC versions was developed by a series of control systems engineering students with little knowledge of computer engineering and numerical methodology. As a result, the algorithms in the previous ICECAP-PC mirrored solution techniques taught in standard math courses taken by engineering students. However, some of these same methods can be unstable, inaccurate or both when codified into a computer program. A very good example of this is the quadratic equation itself. Directly

programming the quadratic equation can yield very large roundoff error in the vicinity of $b^2 \approx 4ac$ because the subtraction of small similar numbers leads to loss of significant digits on a computer. However, the optimal numerical solution of the quadratic equation is easily obtained from a number of numerical methods texts and is now included in ICECAP-PC.

Toolboxes

In order to project ICECAP-PC into the future, it is necessary to provide extendibility of the basic ICECAP-PC program. While the basic ICECAP-PC provides many useful functions, it doesn't include many of the more powerful and recent control theories such as H∞, LQG and QFT. An early design decision (discussed in chapter 2) was to provide hooks for their future implementation by way of toolboxes. In this way, future control engineering students can add to ICECAP-PC by devising a toolbox to implement a specific theory. In fact, our final design goal is the development of QFT toolboxes for the solution of MISO and MIMO control systems problems. This is special interest to the Air Force and to flight control problems because of its ability to incorporate plant variation. While some work has been done recently in the area of MISO QFT program design (Yaniv, 1992), we seek a program specifically tailored to educational purposes. Therefore, the inclusion of interactive boundary specification and loop shaping methodology is appropriate. With these tools, a student can manually generate bounds on a graphics terminal as well as modify the L-Zero curve. After learning the theory behind these techniques, the student can do subsequent designs in the automatic generation mode.

The MIMO QFT development effort is a natural extension to the MISO effort (Bell, 1992). The MIMO technique centers around the description and definition of the MIMO plant (a matrix of transfer functions) and its decomposition into equivalent MISO loops. The method is affirmed to work provided certain conditions, diagonal dominance etc., are met. The most difficult part of developing a MIMO QFT CAD program is not the method itself, but rather developing a numerical engine with sufficient robustness to handle the intricate mathematics accurately. Specifically, the inversion of matrix P, a matrix of transfer functions, poses a difficult numerical problem; one that is beyond the scope of floating point arithmetic for large order systems. Inverting such a matrix requires root finding accuracy to the number of decimal places used for all other floating point calculations.

1.4 Report Organization

This report presents the results of this investigation and design effort by developing all three subject areas from general concept to specific implementation. Chapter 2 presents the research conducted on object-oriented design, numerical methodology and Quantitative Feedback Theory. It does so in a general manner exploring numerous alternatives and discussing basic concepts. Chapter 3 gives a set of specific requirements for ICECAP-PC based on the research presented in Chapter 2. Chapter 4 discusses the actual design and implementation of ICECAP-PC. The discussion of chapter 4 presents each of the major code families with specific attention paid to program structure, choice of numerical methodology, and decision process involved in critical implementation areas. Chapter 5 presents both an in depth look at the toolbox concept and specifically the MIMO QFT toolbox. Chapter 6 provides a discussion on the testing plan used and events experienced in the development of ICECAP-PC code in general and of the numerical methods developed. Chapter 7 contains the concluding remarks and recommendations for future ICECAP-PC development.

1.5 Summary

In summary, this investigation covers three major engineering disciplines. Object-oriented software engineering, the ability to design and write computer programs, is central to this effort. Equally important is the development of advanced CACSD software and problem solving techniques. Specifically, the development of an object-oriented implementation of the MIMO QFT problem is unprecedented. Finally, mathematical rigor is fundamental to any CACSD program and much attention is paid to this discipline in this investigation.

2 General Technical Review

2.1 Introduction

This chapter presents the literature review and investigation in three key subject areas: (1) Object Oriented Modeling and Design, (2) Numerical Methodology and (3) Quantitative Feedback Theory. Thus it forms a general basis of knowledge from which CACSD program specifications are made, a design defined, and program implemented. While this is not an all inclusive investigation of all disciplines required in CACSD design, it does touch on those areas crucial to this investigation.

Section 2.2, *Object Oriented Design Discussion and Terminology,* gives a general discussion on the emerging and still somewhat nebulous field of object oriented analysis, modeling, and design. Object-orientation is not a mere program structure, but a unique developmental methodology that views a problem space in a different context than any classical approach. The structure of an object-oriented model parallels the real world, which is a collection of things or objects, more closely than a functionally decomposed program. As mentioned above, it is still a developing field, and there are as many modeling systems as there are authors all of which differ in syntax but agree in basic logical decomposition. That is, the software engineer models the problem space using OOA (object-oriented analysis) techniques, designs a program specification using OOD (object-oriented design) techniques and develops the program code using OOP (object-oriented programming) techniques; each following the unique logic axioms of object-orientation. This project adheres to the modeling system presented by Rumbaugh (Rumbaugh, 1991).

Section 2.3, *Literature Review of Numerical Methodology,* covers several algorithms that solve mathematical problems of special interest in ICECAP-PC. It addresses questions about the mathematical engine that drives the final CACSD design. What is the best way to invert a general complex matrix? What are the alternatives? How do we design a numerical procedure to yield maximum accuracy? What are the pitfalls? All of these questions, and others, are explored therein.

Section 2.4, *Literature Review of the QFT Technique,* covers the mathematical and theoretical foundations of the Quantitative Feedback Theory developed by Prof Isaac Horowitz. This section is included

herein because it forms the general basis of knowledge for the development of the MISO QFT and MIMO QFT toolboxes as mentioned in chapter 1.

Another area of research of special interest to this project is the design of *human interfaces*. This research is presented by another student who was also involved in the ICECAP-PC project (Bell, 1992). The reader is referred to this study for the general conceptual development of the human interface of ICECAP-PC.

2.2 *Object Oriented Design Discussion and Terminology*

Introduction

In the software design process, the objective is to develop a high-level design and then to decompose this design into lower level modules until reaching the primitive level (implementation/coding). Two software design approaches to decomposition are functional and object-oriented design.

Most packages today are modular and are written using functional programming techniques. Tasks are modularized into functional procedures (subroutines). Commands given by the user are processed through a hierarchical tree of procedures until the commanded function is fully carried out. Thus a functional program is a collection of sequential statements which the program sequentially executes, operating on its information in the way each program line instructs.

An object-oriented (OO) program is also modular except the modules are objects. Objects are executable records that contain both data and procedures (methods) that operate on that data. Further, OO programs are not sequential in nature but are event-driven. An event is, for example, a user selecting a command from a menu object. The menu object contains an event-handler that sends appropriate messages to the other objects in the program telling each of them what function the user has asked to be performed. Each object can operate on its data in order to achieve that function. Thus the big difference between OO programs and functional programs can be viewed as nouns doing verbs (objects responding to messages) instead of verbs acting on nouns (sequential statements doing something to stored data). This section explains what is involved in using OOD, and what advantages and disadvantages OOD has over functional design.

Terminology

In order to understand OOD, the reader needs a comprehensive understanding of the terms used in the object-oriented field. The following section is an exhaustive dictionary of the terms used in this section.

Abstract Data Types: Examples of data types can include integers, characters, and boolean variables. However, the state of the data can be viewed after associated operations. A more formal method of defining these data types is the concept of abstract data types (ADT). By formal definition, an ADT is a three-tuple, (D,F,A), where D represents the domains of the data type, F the functions, methods, or operations on the data

type, and A is a set of axioms (first-order predicate calculus) that encode the desired semantics of the operations. Generally, an ADT is an encapsulated data structure and associated operations without an explicit enumeration of the axioms in order to provide ease of development. The invisibility of an ADT's state and the separation of its interface component from its implementation are the distinguishing features which separate an ADT from a simple data type such as an integer.

Object: In order to understand the concept of an object, several preliminary concepts dealing with computers and computer programs must be understood. All computer programs operate on structured data sets. Typical data structures are stacks, queues, arrays and records. A reasonably new and important data structure is that of an object. An object is an instance of an *object class ADT* and contains both data and methods (executable procedures) that operate on that data. Thus an object is unlike other typical computer data structures because it can modify its information and can send instructions to other objects to change their information. An object is implemented in a computer program by three main parts: its *data*, an *event handler*, and *methods*. Two of these are, in fact, the (D, F) of the ADT data type mentioned previously. The domain *data* is the useful information the object maintains and is stored in the computer's memory under the object's name. The *event handler* is a listing of messages to which the object can respond and the functional *methods* the object should enact if the corresponding message is received. The *methods* are the typical computer program instructions which operate on the object's information and which send messages to other objects.

Methods: As was mentioned previously in the definition of Object, methods (also called operations or functions) are executable procedures that operate on the object's data. Their counterpart in functional programming would be subroutines. The set of object methods can be further classified by their functional type.

Constructors: These methods generate or construct memory locations for an object's instantiated (see below) variables.

Destructors: These methods remove an instantiated object from the computer's memory and release that memory for later use.

Selectors: These methods perform a selection of the current state generally to output some data or make a decision as to the next process (a series of tasks or methods).

Iterators: These operations perform an iteration over the object data structure.

Exceptions: These methods, after determining that an error has occurred, perform a desired set of tasks. Although these terms are useful for understanding, most applications do not classify specific object operations.

Class: A class is a higher level of abstraction than an object, that is, a set of objects can share a common structure and common behavior. A class is defined as a collection of operations (methods) and the data types the methods operate on. When the data and methods can be accessed by other objects, they are visible by definition. If they can not be accessed by another object, then the data and methods are defined as invisible; i.e., information hiding. The class interface consists of public (visible) elements, private (not visible) elements, and protected (visible only to subclasses) elements. In selecting a class, the criteria includes reusability, complexity (time and space), and applicability. This definition represents a general ADT concept as previously presented.

Instantiation: An object is by definition an instantiation or instance of a class. An instance can be thought of as the variable name where the class is the type the variable name is defined as being. So while a class defines the methods to operate on data and describes the data types to describe the data structure, classes do not contain any real data until they are instantiated into existence as an object. Objects of the same class, therefore, share the same methods but not the same instantiated variables (or domains)! Two objects of the same class may each own a variable named DATA_ITEM, but if Object1 changes its instance of the variable, the DATA_ITEM in Object2 is unchanged. However, the method they each run to modify that DATA_ITEM is indeed the same exact method inherited from their common parent class.

Inheritance: An important property of an object is *inheritance*. Class inheritance is a relationship among classes where by one class shares the structure and behavior defined in one or more other classes. An object by definition inherits the methods and data types of its associated class when it is instantiated. Thus, under instantiation, unique data variables are created for the object, but the class methods are used for the object methods. In addition the instantiated object can have new code defining its own unique methods. This is called ***polymorphism*** and is discussed later. An inheritance hierarchy of objects is a tree structure that permits any object in the tree to inherit and operate using any method or data type in an object class higher in the tree. Thus

the object has inherited the methods and data structures higher in the tree. The utility of this inheritance is that once an object type has been fully written and tested, the programmer never needs to modify it again. These same tested capabilities can be used by future objects by simply declaring them to be children of the first object's class. This vastly simplifies the process of software development and maintenance providing far better reusability than simple modular software design.

Inheritance and instantiation differ in the way they are implemented in different programming languages. Classes in Turbo Pascal V 6.0 are defined with the *type* statement and brought into existence upon being instantiated in the *var* section of the program. For instance, an object called NewPoly, an instantiation of class PolyObjectType which is a child of the parent class DataIOType, could be declared with the following statements:

```
PROGRAM SomeProgram;
TYPE

Polynomial = Array[1..Max_Coefficients] of real;

[Parent Object Class Declaration]
DataIOType = object
     {Data Structures}
     Poly1      : Polynomial;
     Poly2      : Polynomial;
     Poly3      : Polynomial;
     {Methods}
     PROCEDURE  : RetrievePoly(PolyName: String; var OutPoly: Polynomial);
     PROCEDURE  : StorePoly(PolyName: String; var OutPoly: Polynomial);
end;

{Child Object Class Declaration}
PolyObjectType = object(DataIOType)
     {No new data structures}
     {Some new methods}
     CONSTRUCTOR: Init;
     PROCEDURE  : HandleEvent(EventType); {EventType is a reserved word}
     PROCEDURE  : AddPoly(InPoly1,InPoly2: Polynomial: var OutPoly: Polynomial);
     PROCEDURE  : MultPoly(InPoly1,InPoly2: Polynomial: var OutPoly: Polynomial);
     DESTRUCTOR : Done;
end;

VAR
        NewPoly   : PolyObjectType;

BEGIN
        NewPoly.Init;
        ...
        ...
        NewPoly.Done;
END.
```

Listing 2-1: Basic Inheritance

In the above example, a parent object class called DataIOType is defined to have three polynomials (two operands and a result) and basic file I/O procedures to retrieve and store the polynomials to file. The object class PolyObjectType is declared as a child of DataIOType. Thus, it *inherits* all three polynomials and the two I/O procedures (methods) from DataIOType. They are present in PolyObjectType just as though they had been explicitly declared. Note further that PolyObjectType has an event handler, a constructor and a destructor. The HandleEvent method is necessary for external communication (message reception) with other objects. The constructor initializes a specific instance of the PolyObjectType class when a variable of the type PolyObjectType is instantiated. In this case, NewPoly becomes a specific instantiation of the class PolyObjectType. Each such variable declaration constitutes an instantiation of PolyObjectType and this can be accomplished as many times as the programmer desires.

The above example clearly demonstrates the principles of inheritance and instantiation. Inheritance defines the structure and capabilities of an object class while instantiation defines lines of ownership and control of actual data. The two concepts are quite different and understanding this difference is key to understanding OOP.

Polymorphism: If a child class redefines a method of its parent class, it is said to be *polymorphing* that method. For example, if the previous example class PolyObjectType defined a method called RetrievePoly just like in its parent, and then changed the program statements RetreivePoly executes, then one would say PolyObjectType polymorphed the method RetreivePoly. (Pressman)

Object-Oriented Analysis (OOA): OOA is an approach to problem definition and partitioning. The product of OOA is a high level design of objects depicted in a set of diagrams (object diagram, instance diagram, state diagram, data flow diagram, etc.). Thus, the nature of the objects involved in a problem space, the relationships the object have to each other, and a high level view of data movement between objects is fully specified by the OOA process. (Rumbaugh, 1991)

OOA Notation: Notation of OOA can be expressed in a set of hierarchical graphs: class diagrams, object diagrams, module diagrams and process diagrams, state transition diagrams and timing diagrams. Although not enumerated here, specific icons are associated with the characteristics of each diagram (Grady).

A class diagram presents each class and its relationship with other classes. the dynamic behavior of the class is represented by a state transition diagram which portrays the transition from state to state as caused by an event as well as the actions resulting from a state change. The object diagram presents each object and its relationship to others. Since objects are created and destroyed during program execution, the object diagram represents the dynamics of the object. Object diagrams are prototypical classifications. By construction, class and object diagram development document the logical design of the system. Module diagrams present the encapsulation of classes and objects. All the diagrams should be evaluated in terms of the formerly mentioned metrics.

Object-Oriented Design (OOD): OOD is a medium level process, the product of which is a fully specified and decomposed yet uncoded software design. While OOA strictly views the problem space of a design, OOD views the OOA in relation to the impending implementaion issues that must be addresses. Algorithm design, numeric formulas, parameter passing, etc. are all specified during the OOD phase. (Rumbaugh, 1991)

Object-Oriented Programming (OOP): OOP is a programming technique in which the data structures are represented as cooperating collections of objects, each object being an instance of a class within a hierarchy of classes that permit inheritance. OOP evaluation of objects (or classes) can be done using standard programming discipline metrics such as object coupling, cohesion, sufficiency, completeness and primitiveness. Coupling refers to the relationship between objects, cohesion refers to the relationship between internal object constructs, sufficiency and completeness refer to the object having enough of all possible behaviors so as to be useful, and primitiveness is when a desired program behavior can only be implemented by accessing invisible structures of an object. (Pressman)

The Object-Oriented Process

The object-oriented process consists of identifying the classes and objects at some level of abstraction, identifying the data and operations of each class and object, identifying the relationships between classes and objects, and implementing the classes and objects into modules. Object-orientation allows the programmer to take advantage of three important software design concepts: abstraction, information hiding, and modularity. The

process begins with object-oriented analysis (OOA). Pressman offers an easily understood approach for OOA. (Pressman, pp 334-346)

First, a paragraph is written in plain English language that describes the function to be performed by the computer program. Objects are extracted from the paragraph by underlining the nouns in the sentences. Attributes of objects are extracted by underlining the adjectives of the sentence and grouping them with their associated objects (nouns they modify). Methods are identified by underlining all the verbs, verb phrases, and predicates in the sentences. Attributes of the methods are found by underlining all the adverbs and grouping them with their associated methods (verbs they modify). Now each grouping of objects and methods is identified as either part of the solution or part of the problem. Now we are ready to enter the object-oriented design (OOD) phase.

Again a paraphrased and modified methodology for the object-oriented process from the work of Pressman is borrowed. The steps are:

1 Define the problem to be solved

2 Decompose the problem into objects

3 Determine each object's required data

4 Determine each object's required methods

5 Determine interfaces between objects and methods

6 Determine a parent-child hierarchy related to the data and methods

7 Determine inheritance/polymorphism relationships related to the data and methods

8 Create a user-interface object

9 Create each object

The first four steps are already arrived at in the OOA phase of the design. Obviously, there exist many techniques for iteratively applying these four steps further down levels of abstraction until finally arriving at the primitive level. At this lowest level the objects required to solve the problem are obvious, as are the methods they need to perform, and the data they need to own.

Determining interfaces between objects and methods is done by determining how each object depends on the others. From this it can be determined what messages each object needs to send to the other objects and when. Thus event-handling routines are designed for each object, so they can perform the desired methods when the message is received.

At the implementation level, we see an overwhelming advantage of object-orientation. The objects that have methods and data types in common are grouped into classes. These classes can be completely separate from one another, or they may have common data items or common methods. Whenever possible, blocks of code should not be repeated, so the common data and methods are grouped into a class of their own and other classes are made children of that new class. This class may not make sense as an object in itself and there may never be an object who is a direct instantiation of it, but its children have tighter code since they have this "library" of ready-made methods to use.

Now the step of creating an object who serves as the menuing system interface between the human user and the internal objects as well as the output screen interface between the internal objects and the human user is added. This object contains the overall event-handling method as well as most of the file I/O and screen I/O.

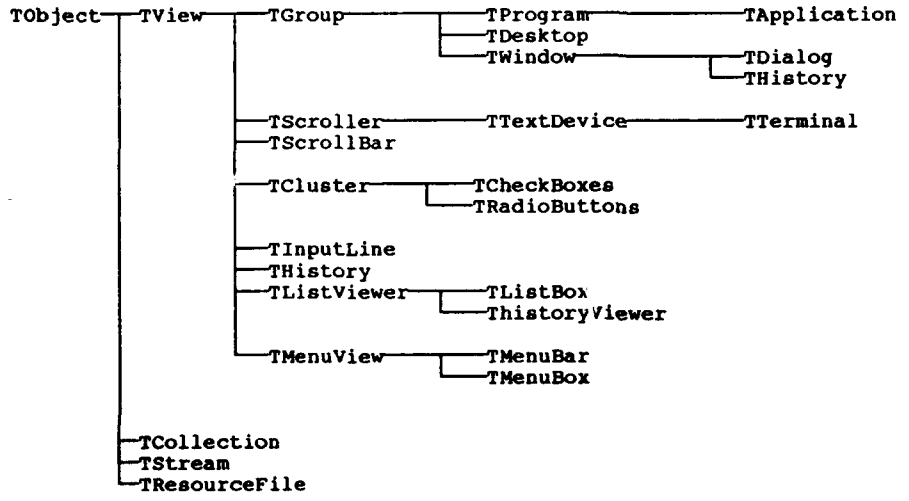
The final step is to pick a particular language and implement the design in code. The next section describes just such a language.

Borland Turbo Vision

Borland$_{TM}$ provides support for object-orientation and an excellent pre-defined object library in its Turbo Vision package available with Turbo Pascal V 6.0 and Turbo C++. The Turbo Vision object library provides a predefined framework to develop object-oriented windowing applications including:

1. Multiple, resizeable, overlapping windows (view the same function on multiple planes such as s, z and w)
2. Mouse support
3. Drop-down menus and dialogue boxes for user input
4. Buttons, scroll bars, check boxes and radio buttons
5. Standardized event handling for keyboard and mouse events

Experience with Turbo Vision has shown that while it presents an extremely steep learning curve, time spent learning it is worthwhile. One word of caution: if the decision is made to build an application with Turbo Vision, the entire project should be built using Turbo Vision objects and standards. Attempting to mix standard functional code with Turbo Vision objects will only create memory conflicts. Another point to note is that Turbo Vision programs are not portable between platforms. Turbo Vision programs are limited to MS-DOS computers.



Listing 2-2: Abbreviated Turbo Vision Family Tree

Turbo Vision consists of a family tree of predefined object types that provide a basic user interface. This tree is shown in Listing 2. The term "family tree" is used to indicate the inheritance lines of each object. This is a different concept than a simple hierarchical tree.

From Listing 2, it is seen that the root object is TObject. TObject has no ancestors and is extremely limited functionally. It has a constructor (Constructor INIT), a destructor (Destructor DONE), and a method (Procedure FREE) that disposes of the object and frees its memory. TObject has six children, among whom are TView, TCollection, TStream, and TResourceFile. TView is of primary importance because its children provide the user interface.

TView is the parent to all objects that can write to the screen. The converse is also true. All objects that can write to the screen are children of TView. The Turbo Vision standard is that all screen writes be accomplished via the TView.Draw method. While it is possible to use the standard Pascal *write* and *writeln* statements, it violates the Turbo Vision standard and their use is strongly discouraged. The *writeln* and *read* statements employed for user input are replaced by dialogue boxes which are descendants of TView. TView has another important property worth noting; it is the lowest object on the tree that is capable of message transmission and reception. Thus, any object that needs to communicate with other objects should be a descendant of TView whether or not they are visible. All the workhorse type objects in Turbo Vision are made descendants of TView. For example, a matrix object, a polynomial object, or a transfer function object should all be descendants of TView. TView has several descendants, among whom are TApplication, TDesktop, TMenuBar, TWindow, TDialog, and TScroller. The following discussion is limited to these objects as they are of primary importance. For further information the Borland Turbo Vision Guide that comes with Turbo Pascal V 6.0 is highly recommended. (Turbo Vision Guide, 1991)

TApplication is a child of TProgram which is a child of TGroup which is a child of TView. Thus TApplication is a descendant, several generations removed, of TView. ***The focal point of any Turbo Vision program is always a child of TApplication which the programmer must define.*** Furthermore, there should only be one TApplication object for any given program. This object owns via instantiation all other objects, handles all message dispatching, communicates directly with the main menu, manages idle times, and processes computer errors.

To write a hypothetical program, say a DSP program for example, the main program would be a child of TApplication. It would be declared in under Turbo Vision as shown in listing 2-3.

```
PDSP = ^TDSP
TDSP = Object(TApplication){Note TDSP is a child of TApplication}
      CONSTRUCTOR: Init;
      PROCEDURE  : HandleEvent(var Event: TEvent); virtual;
      PROCEDURE  : Idle; virtual;
      PROCEDURE  : InitMenuBar; virtual;
      PROCEDURE  : InitStatusLine; virtual;
      PROCEDURE  : OutOfMemory; virtual;
      DESTRUCTOR : Done; virtual;
end;
```

Listing 2-3: Hypothetical DSP Object Class

From this code it is seen that TDSP is a child of TApplication. PDSP is a pointer type to the TDSP type. Turbo Vision makes heavy use of dynamically allocated variables and uses pointers abundantly. Each virtual method listed above is a polymorphed version of identically named methods in ancestor objects. In other words, it is the programmers responsibility to overwrite the ancestor methods in order to fully define the desired interface and program operation--any inherited method not redefined uses its parent's method as is. The InitMenuBar and InitStatusLine methods instantiate the menu bar and status line objects that define the user interface and menu structure. The HandleEvent method processes all events (menu events, mouse events, keyboard events, message broadcast events, etc) and sends messages to the proper objects in response to these events. The OutOfMemory method guards against memory overflow errors and the Idle method does background maintenance during idle periods when the user has not requested any commands.

TDialog is a child of TWindow which is a child of TGroup which is a child of TView. Descendants of TDialog provide pop-up dialogue boxes for user input. Dialogue boxes contain radio buttons, check boxes, list boxes, and input lines. Radio buttons are input devices that allows the user to choose only one item among a palette of options. Check boxes are input devices that allow the user to choose any combination of items among a palette of options. List boxes provide a list of items to choose from such as files on disk or directories. Input lines provide text entry of a string variables. Each of these (radio buttons, check boxes, list boxes and input lines) are themselves object descendants of TView. Specifically TRadioButtons is the radio button object, TCheckBoxes is the check box object, TListBox is the list box object and TInputLine is the input line object. Each are polymorphed and instantiated into a descendant of a TDialog object by the programmer. Examples are abundant in the Turbo Vision Guide.

Other objects worth brief mention are TDesktop, TMenuBar, TStatusLine, and TWindow. TDesktop is a child of TGroup which is a child of TView. It is simply the background view upon which all other visible views appear. TMenuBar is the menu bar object that displays and controls drop down menus. Note that in the above application object definition, TDSP. InitMenuBar instantiates a child of TMenuBar into the application object. TStatusLine provides a bottom frame to display and control shortcut keystrokes and other useful information such as remaining heap size. TWindow is merely a frame that borders views with a frame.

Event Handling is always a big design concern in OOP. In Turbo Vision, all event handling is processed via a TEvent type record. TEvent is a record that identifies the type of event that has occurred and the specific command should one be directed. All events are not commands, however, all commands are events. For example, the movement of a mouse pointer is not a command, however it is an event. All Turbo Vision object have event handlers to process TEvent records, however the polymorphed descendant of TApplication is the focal point for all event handling. Assume the following basic event handler for the TDSP descendant of TApplication described previously.

```
PROCEDURE TDSP.HandleEvent(var Event: TEvent);
{Note the Event variable is a record of TEvent type}

procedure DosShell;
begin
   ...
end;

begin
   TApplication.HandleEvent(Event);
   if Event.What = evKeyDown then begin
        {Desktop Hotkeys}
        'A', 'a': About;
        'C', 'c': Calculator;
        'X', 'x': DosShell;
   end;

   if Event.What = evCommand then begin
      case event.command of
         cmTFCopy    : Message(TransFunction, evBroadcast, brTFCopy   , nil);
         cmTFDefine  : Message(TransFunction, evBroadcast, brTFDefine , nil);
         cmTFDisplay : Message(TransFunction, evBroadcast, brTFDisplay, nil);
     end;
   end;
end;
```

Listing 2-4: Event Handler For Hypothetical DSP Object

This example shows the basic operation of a hypothetical TDSP event handler. Note that the first action taken is a call to the parents event handler (TApplication.HandleEvent). This is to process non-command events

such as mouse movement and cursor key presses. Turbo Vision does a nice job of this and relieves the software engineer from a great burden! If the Event.What field is equal to the predefined integer constant named evKeyDown and the key pressed is an "a", "c", or "x", the appropriate subroutines are called. For proper OOP, these subroutines should be local to the TDSP.HandleEvent method. If the Event.What field is equal to one of the predefined integer constants named cmTFCopy, cmTFDefine, or cmTFDisplay, TDSP.HandleEvent sends a message to an object instantiated as TransFunction. Note that 1) the message is directed to a specific instantiated object and that 2) message transmission is a predefined function of Turbo Vision. Thus the software engineer is again relieved of a great burden! The message is transmitted as an event of evBroadCast type and sends the command brTF... which is a predefined integer constant (the software engineer must predefine these constants into a global unit). The TransFunction object must then contain its own event handler to receive this message and process it accordingly.

As demonstrated in the previous paragraph, Turbo Vision provides several tools to relieve the software engineer of many mundane chores of interface design while allowing all the benefits of programming in a standard high-level language. Execution speed, numerical precision, and mathematical algorithms are all designed with far greater control and efficiency in such a language than could ever be attained by commercial control system packages with their own language interpreters. Once the initial steep obstacle of learning OOP and the Turbo Vision toolbox are mastered, building applications becomes a quick and rewarding task.

Advantages of OOP Over Functional Programming

The following opinions were formed from specific experiences with first having tried to modify and debug the functional version of ICECAP-PC (a CACSD package) and then having translated it into OO code. While the experiences discussed here are from a specific package, they can certainly--based on current literature of similar design projects--be generalized.

A typical danger spot in functional programming is opening a data file in one section of the code and then closing it (or forgetting to close it, or hitting some conditional branching statement that bypasses the close command) in some later section of code. In OOP, a database-type object is used that is the only object in the

program that can get and save data from files. Therefore, only one OPEN command and one CLOSE command are used for the entire program.

An advantage of the user interface object is that it abstracts the programmer from having to worry about any user I/O while writing the mathematical code, etc. One object deals with user requests and translates requests into event messages to be sent to the "workhorse" objects. Likewise, the same object returns the workhorse answers to the user in some screen format.

The same nature of abstraction allows the software engineer to abstract a problem to a higher level for debugging or original design. For example, if the math object wants to tell the user I/O object to print its answer to screen, it doesn't need to know how the I/O object does it, it just sends the I/O object a message telling it what information to print, and the I/O object can take care of it. From the software engineer's point of view, during debugging or design they can design at the highest level of abstraction listing the upper level tasks that need to be done to solve the problem and assume that some object can do each task. Then the programmer moves down one level of abstraction and takes each task and breaks it down into sub-tasks assuming some object (or method) can do each sub-task. This is done down to the primitive/coding level. Debugging is broken down the same way. The programmer looks at the input and output of the highest level object. If it is wrong, he looks at the input and output of each of the objects in the next level down. He then only has to break down the object that had incorrect output. Because each object is self-contained, it makes maintenance very easy.

After the main objects in the program have been fully defined in terms of what data they would need and what methods they would need, inheritance is used to decrease the size of the code. Parent objects are defined for all the main categories of workhorse objects. The parent objects contains all the methods that the workhorse objects hold in common. This means that each of the separate objects could be smaller because they could globally access the methods they inherited from their parent. The parent contains methods like ones to decipher user textual input, ones to work with data files, and other general purpose type methods.

Because the object library from Borland Turbo Vision is available for use, productivity is much higher than would be typically expected in software development. The functional version of ICECAP-PC always suffered in the contemporary human factors engineering area because only so much time could be devoted to

menuing systems and output screen formatting and context sensitive help screens. Using the professionally packaged object library of Turbo Vision, the software engineer is able to focus almost completely on mathematical algorithms and let the commercial package take care of user I/O. Because of this, the authors have been able to work on expanding ICECAP-PC's CACSD toolboxes beyond that which could otherwise have been accomplished. Furthermore, future thesis students will be able to go even farther since the overhead of porting the ICECAP-PC subroutines into an OO environment has already been accomplished.

OOP disciplines produce more reliable code due to modular debugging and using existing objects that have been debugged through years of use. In the case of ICECAP-PC, the benefits of two worlds have been inherited . At the lowest level, the program has its I/O based on a commercially produced and tested package (Turbo Vision). At mid-level, the object methods are based on the basic control system algorithms from ICECAP-PC (developed over several thesis projects and used by a large student body). After the OO program had been tested at all levels of abstraction and it was apparent that each object performed its functions properly and that all the objects communicated among themselves properly, new objects could be added to the existing reliable code with a high degree of confidence in the reliability of the CACSD package as a whole.

The same OOP disciplines produce more maintainable code due to the self-sufficiency of objects. Proper OOP techniques avoid the use of global variables and low functional independence which often plagues functional program modules. If each object is compiled separately and it responds with expected output responses to test inputs, then it does not display the undesirable dependence qualities of low cohesion or coupling with some other object. This research finds that following proper OOP disciplines results in highly cohesive code, because each object is functionally bound to operate on its data alone. Of course, while objects higher on the parent-child tree own more data, they still perform only one higher level function: it receives messages to change its data in some way, and it can do it. Then that higher level object is made up of smaller objects who are each functionally bound to operate on their more specific piece of data. This recurses down through the object tree until the primitive level is reached. At this lowest level very cohesive methods (subroutines) are written. In the same way, following proper OOP disciplines results in low coupling between objects, because each object is again functionally bound to operate on its data alone.

Disadvantages of OOP Over Functional Programming

Only two disadvantages with using OOP have been experienced, neither of which are directly related to OOP itself. The first can be attributed to learning a new programming language and learning a new way of thinking about algorithms to solve problems. The second can be attributed to the decision to operate within an MS-DOS environment.

Any time a new programming syntax must be adopted, there is a learning curve that must be overcome. With OOP this is doubly true, because not only must the syntax of Turbo Vision, or some other OO language package, be learned, but the software engineer must also change their logical concept of problem solution. Humans typically think in functional terms. For example, if a person wants to sign his name on a piece of paper, the algorithm he might imagine to solve this problem might be:

(a) I hold inkpen.

(b) I lay paper flat on table.

(c) I move inkpen to trace out my name.

The person who thinks in terms of objects might imagine this algorithm:

MAN: Name, sign yourself.

NAME: Paper, display me.

PAPER: Hand, lay me flat on the table.

INKPEN: Hand, use me to trace out the NAME pattern on the PAPER

In this example, consider the HAND as being the primitive level or the coding level of the methods. Humans tend to think more in terms of the first example scenario; therefore, the transition into OOP is not as intuitively easy as using functional programming techniques.

Any time code is developed within the MS-DOS environment, limitations are placed on how much memory room is available for use by the program. A stack cannot be larger than 64K, variable declarations cannot be larger than 64K, and the compiled program and heap space (dynamic variable space) cannot exceed 640K. The 640K barrier can be overcome in Turbo Pascal by breaking the compiled code into overlay units, but even then each unit cannot be larger than 64K and must be able to be compiled to some extent separately from

the other units. These memory restrictions place some limit on how closely a programmer can follow the generally accepted rules of OOP.

Concluding Remarks on Object-Oriented Programming

Object-orient design and programming has grown to a standard practice because of benefits over functional design and programming. Such advantages include the reuse of existing software components, more maintainable systems, reduction of developmental risk, and use of OOP language constructs. Disadvantages include the higher cost of development and possible performance degradation due to message passing, the multi-layer abstraction, hierarchy of classes, and associated memory and execution overhead.

The object-oriented approach generally results in smaller systems because of reusable subsystems and thus are more amenable, providing a economic framework for evolution. The original ICECAP-PC was developed using the functional design approach as were its predecessors. The new OO version of ICECAP-PC provides for better maintenance and user interface.

2.3 Literature Review of Numerical Methodology For Modern Control

Introduction

Computer Aided Design (CAD) programs play a role of increasing importance in the art and science of engineering. The modern engineer, in any discipline, must be must be fluent in the use of such programs as it forms their fundamental tool kit. Systems are designed, models are analyzed, and simulations are performed long before project implementation. Mistakes discovered after production begins are prohibitively expensive. The engineer tends to explicitly trust the mathematical answers generated by a CAD program; therefore, it is crucial that such programs consistently provide the best possible numerical accuracy and, if desired, an explicit measurement of error. Unfortunately, while the state of the art in numerical methodology is quite high, the state of the market is not. Quite often, very expensive engineering programs are replete with inaccuracies, inconsistencies or both because insufficient attention was paid to the numerical algorithms.

Numerical accuracy is an abstract concept. Sufficient accuracy is another abstraction that is often difficult to define. Furthermore, it is application dependant. How much accuracy is enough? For financial calculations, two digits of accuracy is sufficient. However, if an engineer wants to use the number π in an N^2 algorithm, they want the impossible: infinite accuracy. Since π can only be approximated in a computer, the error between the approximation and the true value of π, some constant, is propagated N^2 times and the error of the algorithm can grow exponentially. Minimizing, or at least defining this error requires considerable effort.

The engineering community must make the required effort. Designs can fail, bridges can collapse, planes can crash and people can die because of design errors caused by numerical inaccuracy. For the above case, possible improvements include increasing the computer word length (the number of bits used to represent a number) and working symbolically with π. Discovering and inventing ways to minimize and bind inaccuracy should be part of any CAD research.

Numerical Accuracy Issues

Current literature generally defines two types of numerical error: hardware induced error, commonly known as roundoff error; and algorithmic error, commonly known as truncation error. Roundoff error is a term that describes the error between a real number and a computer's floating-point approximation. It is hardware dependant and beyond the control of the software engineer. While there are some techniques available to minimize this error, it's impossible to eliminate it on a digital computer.

Roundoff error occurs because all floating point representations of real numbers are, in fact, approximations of those numbers. Modern computers generally use floating point approximations defined by IEEE Standard 754 (IEEE, 1985). This standard defines three floating point formats: a single precision 32 bit format; a double precision 64 bit format; and an extended precision 80 bit format. Each format consists a single sign bit, a small number of exponent bits and a large number of fraction bits. Real numbers are represented in exponential form using excess 127 or excess 1023 code. As you might expect, increasing the number of bits increases both the resolution by which numbers are represented and the range of numbers represented. The single precision format provides a range from 1.5×10^{-45} to 3.4×10^{38} with 7 to 8 significant digits. The double precision format provides a range from 5.0×10^{-324} to 1.7×10^{308} with 15 to 16 significant digits. The extended precision format provides a range from 3.4×10^{-4932} to 1.1×10^{4932} with 19 to 20 significant digits. ***It is important to realize that any real number with a decimal component smaller than the resolution of the format in use cannot be precisely represented.*** The difference between the actual value of the number and its floating-point representation is the roundoff error. ICECAP-PC uses the extended precision format exclusively.

The second error, truncation error, is solely a function of algorithm design. Herein lies the trap that catches many unsuspecting engineers. ***It is often a mistake to program a mathematical function the way it is taught in school.*** Consider, for example, the quadratic equation. Programming the terms under the radical (b^2-4ac) in a straightforward manner will yield considerable error for $b^2 << ac$. This is caused by the subtraction of two numbers with grossly different exponents and a corresponding loss of significant digits. Other forms of the quadratic equation (not discussed here) yield significantly less truncation error. Many other common functions display similar characteristics.

Algorithms that inhibit the growth of error are referred to as *stable* algorithms. *Unstable* algorithms are those whose structure permit the unbounded growth of error with successive iteration. Such algorithms are of limited utility. They are not; however, altogether useless.

A final note on the numerical accuracy issue: It is commonly and mistakenly held that wider data paths yield greater numerical accuracy. A 32 bit computer is often perceived more accurate than a 16 bit computer and an "engineering workstation" is often perceived more accurate than a PC. *In fact, any computer conforming to IEEE Standard 754 will, in theory, provide identical accuracy.* Differences in observed computed values are invariably the result of algorithm differences and operating system differences rather than the width of a particular data path. Thus, a PC program using an IEEE-754 80 bit extended precision number is more accurate than a mainframe using an IEEE-754 32 bit real number given identical program design! With this in mind, we now focus on specific algorithms used in ICECAP-PC.

The Matrix Condition Number

Systems of equations expressed as variants of the familiar form **AX=B**, including state space representation, may or may not lend themselves to effective numerical computation. If the A matrix is nearly singular, i.e. the Hilbert matrix, it is said to be ill conditioned. Numerical computation with such a matrix produce large error! Gerald makes the following worst case example: (Gerald, 1978)

$$\begin{vmatrix} -0.002 & 4.000 & 4.000 \\ -2.000 & 2.906 & -5.387 \\ 3.000 & -4.031 & -3.112 \end{vmatrix} \times X = \begin{vmatrix} 7.998 \\ -4.481 \\ -4.143 \end{vmatrix} \qquad \text{Eq. 2-1}$$

Using Gaussian elimination without pivoting for inversion (a very unstable algorithm) and four places of accuracy he arrives at a computed solution of $X = (-1496, 2.000, 0.000)^T$. The actual solution is $X = (1.000, 1.000, 1.000)^T$. While this is clearly, in his own words, an "exaggeration" of reality, it does demonstrate the potential for numerical error in ill conditioned problems.

It is desirable to 1) be able to detect in advance when a coefficient (plant) matrix is ill conditioned and 2) to be able to predict and nullify subsequent error. Such capability is provided by the ***condition number*** of the matrix given by the following equation: (Chapra, 1988)

$$Cond[A] = \|A\| \cdot \|A^{-1}\| \qquad \text{Eq. 2-2}$$

This equation says that the condition number of matrix A is equal to the norm (size) of the matrix times the norm of its inverse. High condition numbers indicate ill conditioning of a matrix. The lowest and best condition number is 1. A singular matrix has a condition number of ∞.

Use of this equation requires both a method to compute the norm of a matrix and the inverse of the matrix itself. Obviously, the computed inverse of a poorly conditioned matrix is in error as is the computed condition number; however, since the principle application of the condition number is qualitative in nature, this does not usually cause a problem. *As the reciprocal of the condition number approaches the floating point resolution of computer ($1x10^{-43}$ for ICECAP), numerical error becomes increasingly dominant!*

Several norm definitions exist for matrices, all of which may be used to compute an approximate condition number. The most accurate is the 2 norm $\|A\|_2$. This norm is related directly to the eigenvalues of the matrix and is the minimum norm. (Chapra, 1988) Furthermore, its calculation does not require the inverse of the matrix and is therefore least error prone. It is, however, the most difficult to compute and requires singular value decomposition (SVD). Other norms are the euclidian norm (sometimes called the Froebinius Norm), the maximum-magnitude norm, the column-sum norm and the row-sum norm. Chapra demonstrates the use of the row-sum norm to calculate a condition number and predicting error bounds for a 3x3 Hilbert matrix using the basic premise that computational error is directly proportional to the condition number (Rabinowitz, 1978).

If d is the number of significant digits allowed by a given floating point format and n is the condition number then the maximum predicted computational error is given by: (Chapra, 1988)

$$\epsilon = 10^{(Log(n)-d)} \qquad \text{Eq. 2-3}$$

Condition number calculation was never a feature in any version of ICECAP-PC to date. It is to be included as an integral feature of ICECAP-PC Release 10, the object oriented version.

Matrix Inversion

Matrix inversion is a fundamental process in linear algebra and all engineering disciplines. The solution of **AX=B** depends on the inversion of matrix A. There are several inversion algorithms to choose from including direct formula application, Gauss-Jordan Elimination in several variants, Singular Value Decomposition, and perhaps the most popular, LU Decomposition. We now consider some of these.

Previous versions of ICECAP-PC used the direct application of the following formula:

$$A^{-1} = \frac{Adj(A)}{|A|} \qquad \text{Eq. 2-4}$$

While this yields satisfactory results for small order systems, it is inaccurate for poorly conditioned systems and quite slow for larger matrices.

Gauss-Jordan Elimination

Gauss-Jordan Elimination is a good, basic, and stable method of matrix inversion. Most textbooks dismiss Gauss-Jordan in favor of LU decomposition; however, this may soon change as Gauss-Jordan is ideal for vector operations in parallel processing. Furthermore, variants of Gauss-Jordan processes allow the computation of pseudo-inverses for matrices of less than full rank. (Jones, 1991) If this isn't enough, other extensions provide polynomial curve fitting solutions for over-determined data sets much more quickly than the common least-squares methods!

Gauss-Jordan uses elementary row and column operations to convert the matrix into upper triangular form. The result of Gauss-Jordan Elimination is the following form:

$$\begin{vmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{32} & A_{43} & A_{44} \end{vmatrix} \Rightarrow \begin{vmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ 0 & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ 0 & 0 & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & \alpha_{44} \end{vmatrix} \qquad \text{Eq. 2-5}$$

As a review, the elementary operations are:

- Any two rows/columns may be interchanged.
- Any row/column may be added or subtracted to/from another row/column.
- Any row/column may be multiplied by a non-zero real number.

Once in this form, several matrix properties are easily established including the rank, the eigenvalues (if full rank), the determinant, and the inverse. There are several variants of Gauss/Gauss-Jordan Elimination exhibiting differing degrees of sophistication.

"Elementary Gaussian Elimination" uses only the second operation. While simple, this method is also unstable as small values on the main diagonal allow the growth of error in following operations. Having large values on the upper tiers of the main diagonal and smaller values on the lower tiers is a desirable trait not implemented in Elementary Gaussian Elimination.

A process commonly known as "pivoting", implemented in Gauss-Jordan Elimination, removes the above instability by using the first operation, row/column swapping, to increase upper diagonal element values whose computations occur early in the process. There are two pivoting schemes for Gauss-Jordan Elimination. Partial Pivoting allows the interchange of rows. Full Pivoting allows the interchange of both rows and columns. Full Pivoting is generally regarded as only slightly better than partial pivoting and most authors consider the extra book keeping more trouble than it's worth. Scaling is another option often implemented under Gauss-Jordan Elimination. Scaling uses the third operation, multiplication of a row or column by a real number, to normalize rows of the operand matrix.

Matrix inverses can be calculated one of two ways using Gauss-Jordan Elimination. First, if the original square non-singular matrix is augmented on the right with an identity matrix of equal dimension, it is possible to perform row operations until the original matrix is an identity and the right augment contains the inverse. Second, if the original matrix A (square and non-singular) is converted to upper triangular form and matrix B is defined as an identity matrix of equal dimension as A, the equation AX=B is solved using backwards substitution. With this scheme, the lowest row, which contains only a single element and one unknown, is

solved first. This solution is used to solve the second lowest row which contains two elements and two unknowns. The process continues until all rows are solved and the inverse calculated.

An Interesting Variant

An interesting variant of Gauss-Jordan elimination is one that places identity augment matrices to the right and below the operand matrix prior to row and column operations. The form of such operation is:

$$A \rightarrow \begin{vmatrix} A & I \\ I & 0 \end{vmatrix} \rightarrow \begin{vmatrix} I & T \\ S & 0 \end{vmatrix} \qquad \text{Eq. 2-6}$$

$$A^{-1} = ST$$

Row operations on matrix A and its right identity augment result in matrix T. Column operations on matrix A and its lower augment result in matrix S. The multiplications of matrices S and T produces A^{-1}. The advantage of this method is its ability to find the pseudo-inverse of A to solve AX=B for A less than full rank. This capability is not present in any current CAD package and contradicts the classical notion that a solution for AX=B only exists for a non-singular A. In the case of a singular A, the equations above take the following form:

$$A \rightarrow \begin{vmatrix} A & I \\ I & 0 \end{vmatrix} \rightarrow \begin{vmatrix} I_r & 0 & T \\ 0 & 0 & M \\ - & - & - \\ S & N & 0 \end{vmatrix} \qquad \text{Eq. 2-7}$$

$$A_p^{-1} = S \cdot T \qquad N \perp S \text{ and } M \perp T$$

The above equations indicate that if M is orthogonal to T and N is orthogonal to S, then a solution to AX=B exists. If orthogonallity is not present at the outset, it can often be realized by the use of the Graham-Schmidt orthogonalization process. As an added benefit, the rank of the original matrix A is equal to the dimension of the resulting I_r matrix.

LU Decomposition

LU Decomposition is a very good alternative to the above process because it is at once fast, accurate, and stable. LU Decomposition is a simple factorization scheme that factors a matrix into the products of a lower and upper triangular matrices. It is the method of choice of most commercial PC computer math programs because it is three times faster than Gauss-Jordan Elimination. The result of LU decomposition has the following form:

$$\begin{vmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ \alpha_{21} & 1 & 0 & 0 \\ \alpha_{31} & \alpha_{32} & 1 & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & 1 \end{vmatrix} \times \begin{vmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{21} & \beta_{31} & \beta_{41} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{vmatrix} \qquad \text{Eq. 2-8}$$

A straight forward and refined LU decomposition routine appears in *Numerical Recipes* by Cambridge University Press. (Press, 1989) This routine uses *Crout's algorithm* to solve for the individual α's and β's. Crout's algorithm solves for these elements in the following manner:

For $j = 2..N$
  For $i = 1..j$

$$\beta_{ij} = \alpha_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj} \qquad \text{Eq. 2-9}$$

  For $i = J+1..N$

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left(\alpha_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj}\right)$$

Finding the inverse of the A matrix, as with Gauss-Jordan, involves back substitution.

Singular Value Decomposition

Singular Value Decomposition (SVD) is another factoring technique that transforms a matrix into the product of three matrices of the following form:

$$A_{mxn} = U_{(mxn)} \cdot W_{d(nxn)} \cdot V_{o(nxn)}^{T} \qquad \text{Eq. 2-10}$$

U is a nxn matrix, W_d is a diagonal matrix, and V is a nxn orthogonal matrix.

SVD is ideally suited for nearly singular matrices. (Press, 1989) A by product of SVD is the 2-norm condition number which is easily derived from the W_d matrix. It is simply the ratio of the largest number divided by the smallest number along the main W_d diagonal. A divide by zero, indicating an infinite condition number, occurs if the matrix is singular. The inverse of a matrix using SVD is calculated as follows:

$$A^{-1} = V \cdot (Diagonal(1/W_j)) \cdot U^T$$

The Determinant

Determinants are easily calculated using both Gauss-Jordan Elimination and LU Decomposition. In the case of Gauss-Jordan Elimination, the determinant is simply the product of all elements along the main diagonal of the upper triangular result. For LU Decomposition, it is the product of the determinants of both matrices; however, since the diagonal elements of the lower triangular matrix are all one, the determinant of the lower triangular matrix is equal to one. The overall determinant is thus the product of main diagonal elements of the upper triangular matrix.

Another approach to solve for the determinant is to use Cramer's rule and solve for the determinants of all of the minor matrices. This; however, is a slow and inefficient process, especially for large order matrices.

Eigenvalues

The eigenvalue problem is a difficult one, especially in the study of numerical math. Consequently, most texts on the subject spend considerable time discussing very simple special cases such as real symmetric matrices. Unfortunately, such cases are of little value to the control systems engineer whose task is control real world, unsymmetrical devices. A control systems CAD package, like ICECAP-PC, requires a general solution for the eigenvalues of an arbitrary non-symmetric complex matrix. Two techniques seem to offer the most promise.

The first technique is an iterative solution. We simply find the coefficients of the characteristic equation for an arbitrary square matrix. We then find the roots of this equation using standard root finding techniques. This is what previous versions of ICECAP-PC did.

The authors of *Numerical Recipes*, Cambridge University Press, state *"Root searching in the characteristic equation...is usually a very poor computational method for finding eigenvalues"*. (Press, 1989) However, no other referenced source makes any verification, explicit or implicit, of this statement. Furthermore, root finding packages for low order polynomials (less than 20) are quite good. With closed form solutions for the characteristic equation, it is reasonable to expect accurate solutions using this technique. Therefore it remains a viable option.

The second technique is to transform the operand matrix into Hessenberg Form and then compute the eigenvalues using the QR Shift algorithm. This method is defined in several sources for arbitrary *real* non-symmetric matrices; however, precious little is said about complex matrices.

An Upper Hessenberg matrix has the following form:

$$A_h = \begin{vmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\ 0 & A_{32} & A_{33} & A_{34} & A_{35} \\ 0 & 0 & A_{43} & A_{44} & A_{45} \\ 0 & 0 & 0 & A_{54} & A_{55} \end{vmatrix} \qquad \text{Eq. 2-12}$$

It is upper triangular with the addition of a single sub-diagonal element set.

An arbitrary non-symmetric matrix **A** may be have eigenvalues that are extremely sensitive to changes in element values. In such cases, small roundoff errors can cause large changes in the calculated eigenvalues. Therefore, the first task in dealing with a general matrix is to prepare it for Hessenberg conversion by de-sensitizing or *balancing* it. Balancing makes use of similarity transformations to make corresponding rows and columns have similar norms while maintaining the original eigenvalues. *Numerical Recipes* lists *subroutine Banalc* that uses diagonal matrices whose elements are multiples of 2 to balance an input matrix (Recipes, 366).

While balancing is not a mandatory task in the process, it yields greater accuracy and stability and is therefore considered an integral part of the eigenvalue process.

The second task in the process is the transformation of the balanced matrix A_b into an Upper Hessenberg form A_h as shown previously. There are two methods of doing this. The first method, Gaussian Elimination, is the quicker of the two and, as mentioned earlier, it is quite stable. The second method, Householder Reduction, uses a sequence of orthogonal similarity transformations and is even more stable than Gaussian Elimination. Furthermore, it is of special interest because it forms the basis of the third step in the numerical eigenvalue problem: The QR Shift Algorithm.

The Householder Matrix is a symmetric orthogonal matrix of the following form:

$$
\begin{aligned}
H &= I - 2\frac{VV^T}{|V|^2} \\
V &= X + \sigma Z \\
X &= |\, A_{21} \;\; A_{31} \;\; \cdots \;\; A_{n1} \,|^T \\
\sigma &= |\, X \,| \\
Z &= |\, 1 \;\; 0 \;\; 0 \;\; 0 \;\; \ldots \,|^T
\end{aligned}
\qquad \text{Eq. 2-13}
$$

The Householder Matrix H is completely determined by the first column of the operand matrix A_b and is of order one less than A_b. The X vector is the second through nth elements of the first column of A. Z is a vector of the same dimension as X with a 1 in the first position and zeros otherwise.

Householder transformation requires n operations for a matrix A of order n. For any k such that $1 \leq k \leq n$; $A_{k+1} = U_k^{-1} A_k U_k$. The end result of this transformation is an Upper Hessenberg Matrix if the original matrix A is non-symmetric and a tridiagonal matrix if the original matrix A is symmetric. Each successive transformation matrix U_k is formed as follows:

$$U_1 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & H_1 & \\ 0 & & & \end{vmatrix} \quad U_2 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & H_2 & \\ 0 & 0 & & \end{vmatrix} \quad U_3 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & H_3 \end{vmatrix} \qquad \text{Eq. 2-14}$$

The third and final task is to calculate the eigenvalues using a QR Shift algorithm. QR Shift is defined by the following:

$$\begin{aligned} Q &= H^{-1} \\ R &= H \cdot A \\ A_{s+1} &= R \cdot Q = H \cdot A \cdot H^{-1} \end{aligned} \qquad \text{Eq. 2-15}$$

QR Shift is an iterative technique that places real eigenvalues along the main diagonal of the resultant matrix. These eigenvalues are easily identified when the element just below the main diagonal approaches zero as the diagonal element approaches the true eigenvalue. To test the validity of the approach, we developed a Matlab macro file to perform QR Shift and display the result. An input matrix and resultant are shown below:

```
InMat =

     1     1    -2
    -1     2     1
     0     1    -1

% This matrix has real eigenvalues at -1, 1, 2
% After just 10 Iterations the resulting matrix is

Result =

    2.0517    1.2821    0.5691
   -0.0442   -1.1936    2.2495
   -0.0251   -0.1770    1.1420

% Note that the subdiagonal elements are approaching zero while the diagonal elements
are approaching the true eigenvalues.
```

Listing 2-5: QR Shift Algorithm Test

The element that is closest to an eigenvalue is element[1,1]. Its subdiagonal element[2,1] is also converging to zero the fastest. Depending on the structure of the program, either the [1,1] element or the [n,n] element will converge first. When its sub-diagonal element shrinks below the floating point resolution of the machine, the eigenvalue should be extracted and the matrix deflated to dimension n-1. Iteration continues on the deflated matrix.

Little is said in literature about the application of QR Shift to the general complex valued matrix. After a great deal of experimentation, we found that complex conjugate eigenvalues appear in 2x2 blocks matrices along the main diagonal rather than along the main diagonal itself! With this discovery, we were able to successfully apply the QR Shift algorithm to the general complex-valued matrix.

Concluding Remarks on Numerical Methods

The meat of any CACSD program is its mathematical foundation. Without a solid math engine, one cannot hope to design a good engineering program. The mistake made by many engineers is programming numerical routines to match formulas taught in school. This often leads to excess error. This review covers some alternative algorithms used in modern control systems analysis. The condition number calculation yields a measure of expected error in future calculations and should be used as a stop light by the engineer. Several good algorithms exist for the inversion of matrices including Gauss-Jordan Elimination, LU Decomposition, Singular Value Decomposition, etc. We found LU decomposition to be a very useful and flexible algorithm providing answer to the determinant problem as well as matrix inversion and the solution to the equation AX=B. The eigenvalue problem, also basic to modern control systems, can be solved by either finding the roots of the characteristic equation of a matrix or by the use of QR Shift. Through this investigation, we found a way to find eigenvalues for the general complex valued matrix that is both accurate and stable.

2.4 *Literature Review of the Quantitative Feedback Theory*

Consider designing a practical linear time invariant feedback controller for a plant model with uncertainty in parameter and disturbance. Due to Plant Uncertainty, there is a set {P} of plants. Consider, for example, a second-order plant model with the following variations:

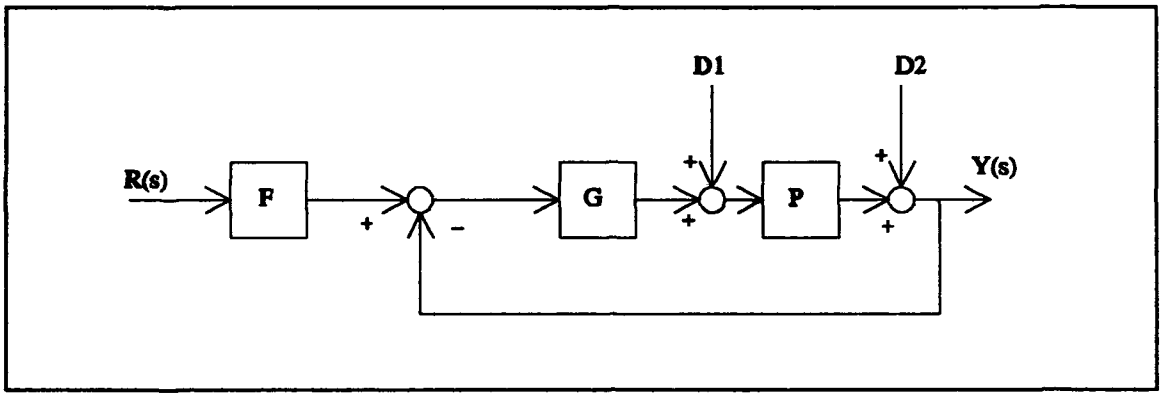


Fig. 2-1: MISO QFT System Model

$$P = \frac{k}{As^2 + Bs + C}$$

$$A = [1\text{-}4]$$
$$B = [2\text{-}6]$$
$$C = [10\text{-}20]$$
$$k = [50\text{-}84]$$

Eq. 2-16

The set {P} of plants consist of all possible combinations of plant variations which could be a very large number of specific plants. The QFT method, developed by Dr Isaac Horowitz, quantitatively defines the problem in the form of (1) sets $\{T_R\}$ of acceptable command or tracking input/output relations and (2) sets $\{T_D\}$ of acceptable disturbance input/output relations and (3) a set of {P} of possible plants. The design objective is to guarantee that the control ratio, $T_R = Y/R$, is a member of $\{T_R\}$ for all P in {P}. Although this technique can be used for a variety of system structures, this package only emphasizes structured uncertainty including non-minimum phase models.

The QFT design (QFD) approach is a frequency domain technique that provides robust performance despite plant uncertainties and disturbances. The general model has three inputs: a tracking input R(s), a plant disturbance D_1 and a measurement disturbance D_2 as shown in Fig. 1-31. ICECAP-PC currently handles only

the case of plant disturbance D1. P is the symbol for the plant, G is the compensator to be designed and F is a pre-filter that also requires design. L = GP is defined as the loop transmission (open-loop transfer function). If only G as a design parameter is available, it called a single degree of freedom control loop. If F is added, two degrees of freedom are available. Thus, the QFT method is a two degree of freedom design.

This approach has sufficient generality for modelling a variety of systems. The closed-loop transfer function for a tracking input is by definition

$$T_R(s) = \frac{FGP}{1+GP} = \frac{FL}{1+L} \qquad \text{Eq. 2-17}$$

$$L = PG \text{ is Loop Transmission}$$

Specifications

To design a controller, various closed-loop performance bounds are specified which must be satisfied for all plant variations. The various specifications are generally in terms of the frequency response of the following tracking and disturbance response transfer functions.

Tracking Specification:

The closed loop tracking transfer function for plant P_j is given by

$$T_{R_j}(\omega) = \frac{FGP_j}{1+GP_j} = \frac{FL_j}{1+L_j} \quad \forall P_j \in \{P\} \qquad \text{Eq. 2-18}$$

$$B_L(\omega) < |T_{R_j}(\omega)| < B_U(\omega)$$

where all T_{Rj} responses (one for each plant) must lie between the B_U and B_L specifications and where the maximum tracking error is defined as (D'Azzo, pg 429, 692)

$$\delta_r(\omega) = B_U(\omega) - B_L(\omega) \qquad \text{Eq. 2-19}$$

There are two methods of deriving δ_r. In the first method, the time domain tracking specifications (M_p-max, M_p-min, T_s-max, T_s-min, etc) lead to two transfer functions, T_m and T_n, which describe the upper and lower

bounds permitted in both time and frequency domains. δ_r is then defined as the difference between the frequency responses $\delta_r = B_U(w) - B_L(w) = Tru(w) - Trl(w)$. In the second method, $B_L(w)$ and $B_U(w)$ are input directly in the frequency domain as a set of data points.

Successful QFT design is enhanced if $\delta_R(w)$ monotonically increases with frequency. Using the augmented model (D'Azzo, 693), this is accomplished for high frequencies by adding a zero to Tru and a pole to Trl at w_h.

If the MISO tracking relationships are part of a larger MIMO problem, the tracking bounds B_u and B_l must be constricted to account for the tracking responses of other plant element modeled as disturbance inputs to the MISO loop. Currently, this is done manually off line and is not addressed in ICECAP-PC.

Stability Specifications (Phase Margin, Gain Margin)

The constraint on the distance from the -1+j0 point is given by (Yaniv, 2; D'Azzo, 309)

$$|1 + GP_j| \geq x \qquad \text{Eq. 2-20}$$

where $x \leq 1$ is a chosen parameter. A larger x for a given frequency results in a smaller steady state sinusoidal error. Another parameter, chosen to reduce the ocsillatory nature of the design, is given by (D'Azzo, 312).

$$\left|\frac{GP_j}{1 + GP_j}\right| \leq y \qquad \text{Eq. 2-21}$$

For an equivalent second order system, larger values of y result in smaller values of zeta.

These equations relate to the desired gain margin, phase margin, peak overshoot and M_l contour (nichols plot) as given by the following.

$$gm = \frac{1}{1 - x}$$
$$\gamma = 180° - 2\cos^{-1}\left(\frac{x}{2}\right) \qquad \text{Eq. 2-22}$$

Given a set {P} of all possible plants, the stability specifications describe the region in the frequency domain that none of the plants P in {P} should violate. In the QFT design technique, the set of {P} plants form a frequency domain template, or a region of possible responses over the parameter variation range. All plants P

in {P} must remain outside the stability margins in the QFT design. Phase margin (τ), gain margin (gm) and peak overshoot (Mp) are all mathematically related and each can be derived from the other for a second order or equivalent second order system as follows:

$$gm = -20 Log_{10} \frac{M_p}{1 - M_p} \qquad \text{Eq. 2-23}$$

$$M_t = 20 Log_{10} \frac{10^{-\frac{gm}{20}}}{1 + 10^{-\frac{gm}{20}}} \qquad \text{Eq. 2-24}$$

$$Mp = e^{\frac{Ln10 M_t}{20}} = \frac{10^{-\frac{gm}{20}}}{1 - \frac{gm}{20}} \qquad \text{Eq. 2-25}$$

$$\gamma = 180° - \cos^{-1}\sqrt{1 - 10^{-\frac{M_t}{10}}} \qquad \text{Eq. 2-26}$$

Disturbance Specification

The disturbance transfer functions for the two disturbance inputs of figure 1 are given by the relations of Eq. 2-27 and Eq. 2-28 (D'Azzo, 446). $M_D(w)$ is the upper disturbance bound defined in the specifications. Recent QFT designers have begun to define $M_D(w)$ as a constant (i.e -20db) rather than a transfer function. Using a constant allows for both easier and more conservative design.

Plant Disturbance D_1

$$T_{D_{1_j}} = \frac{P_j}{1 + L_j} \quad \forall P_j \in \{P\} \qquad \text{Eq. 2-27}$$

$$\left| T_{D_{1_j}}(\omega) \right| < \left| M_D(\omega) \right|$$

Measurement Disturbance D_2

$$T_{D_{2_j}} = \frac{P_j}{1 + L_j} \quad \forall P_j \in \{P\} \qquad \text{Eq. 2-28}$$

$$\left| T_{D_{2_j}}(\omega) \right| < \left| M_D(\omega) \right|$$

The Design Process

As mentioned above, the objective of the QFT technique is to find a G & F to guarantee that the closed-loop response is within prescribed limits of the various bounds despite plant uncertainty and disturbance inputs as represented in the set notation. The ICECAP-PC QFT package closely follows the design procedure specified by Dr Horowitz and Dr Houpis (D'Azzo, pg728). This procedure is summarized as follows:

1. Synthesize upper and lower tracking response transfer functions T_{ru} and T_{rl} to meet minimum and maximum specifications.
2. Synthesize T_D, the upper disturbance response transfer function.
3. Define a set of plants $\{P_j\}$ from all possible $\{P\}$ such that the frequency response of the set $\{P_j\}$ defines the perimeter of all possible frequency responses of $\{P\}$.
4. Select a representative nominal plant P_o from the set $\{P_j\}$. Normally, the best selection is the lower-left plant as seen on a nichols chart template.
5. Determine the disturbance bounds $B_D(w)$ on the loop transmission $L_0(w)$.
6. Determine the tracking bound $B_R(w)$.
7. Define the composite bounds as the most restrictive combination of the bounds determined in steps 6 and 7.
8. Design the loop transmission $L_0(w)$ for the nominal plant $P_0(w)$ to meet, as closely as possible, the composite bounds determined in step 7.
9. Synthesize the prefilter F(w).
10. Simulate system behavior to verify correct design.

The MIMO QFT Theory

The QFT MIMO synthesis problem requires conversion into a number of MISO single-loop feedback problems in which parameter uncertainly, external disturbances, and performance tolerances are derived from the original MIMO problem. The combined solutions to these MISO single-loop problems achieve the desired performance for the MIMO plant. The basic approach is a point-wise frequency domain MISO synthesis technique. A 3x3 MIMO feedback structure is shown in Fig. 2-2.

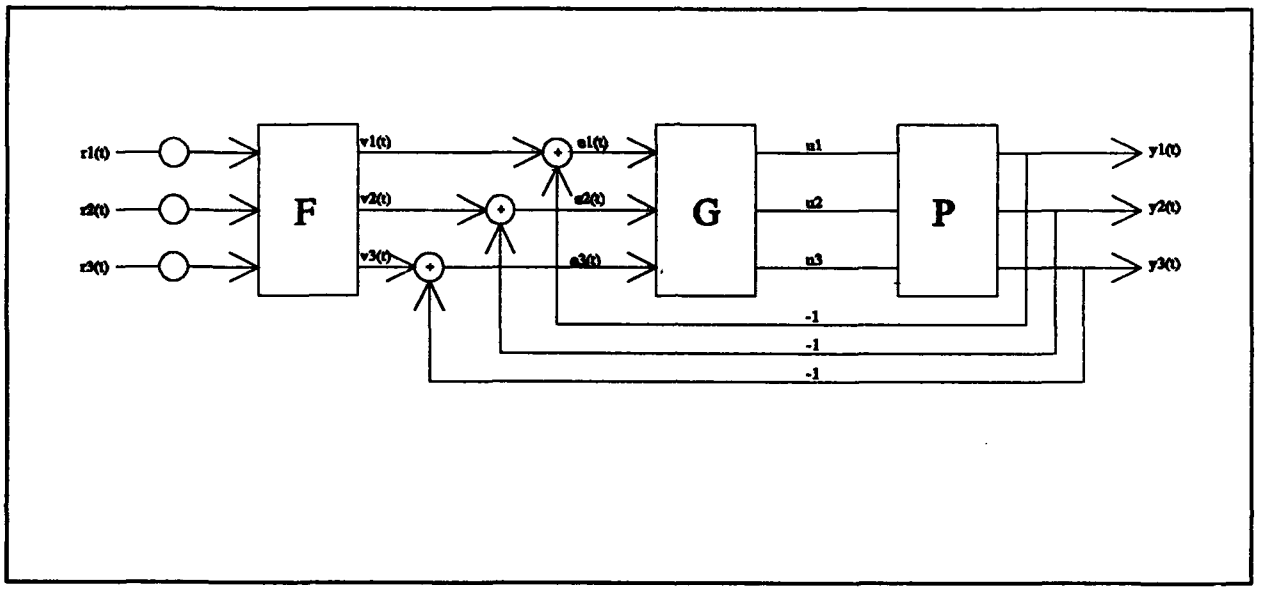


Fig. 2-2: Three by three MIMO QFT Model

Plant models for figure 2 are developed in one of two formats: differential equation form and state space form. The general state-space model is manipulated as follows:

The state-space model representation for a LTI MIMO system is given by

$$\dot{X}(t) = AX(t) + BU(t)$$
$$Y(t) = CX(t) \qquad \text{Eq. 2-29}$$

where X is an m vector, Y is an n vector and U is an r vector. A,B, and C are constant matrices of the proper dimension.

The plant transfer-function matrix P(s) is defined in state space form as

$$P(s) = C[sI - A] - B \qquad \text{Eq. 2-30}$$

When the system is defined in differential equation form, we start with the following relations:

$$\begin{aligned} a_1(s)y_1(s) + b_1(s)y_2(s) + c_1(s)y_3(s) &= d_1u_1(s) + e_1u_2(s) + f_1u_3(s) \\ a_2(s)y_1(s) + b_2(s)y_2(s) + c_2(s)y_3(s) &= d_2u_1(s) + e_2u_2(s) + f_2u_3(s) \\ a_3(s)y_1(s) + b_3(s)y_2(s) + c_3(s)y_3(s) &= d_3u_1(s) + e_3u_2(s) + f_3u_3(s) \end{aligned} \qquad \text{Eq. 2-31}$$

This set of differential equations can be represented in matrix notation as

$$\begin{vmatrix} a_1(s) & b_1(s) & c_1(s) \\ a_2(s) & b_2(s) & c_2(s) \\ a_3(s) & b_3(s) & c_3(s) \end{vmatrix} Y(s) = \begin{vmatrix} d_1 & e_1 & f_1 \\ d_2 & e_2 & f_2 \\ d_3 & e_3 & f_3 \end{vmatrix} U(s) \qquad \text{Eq. 2-32}$$

which yields the following:

$$\begin{aligned} M(s)Y(s) &= NU(s) \\ Y(s) &= M^{-1}NU(s) \\ Y(s) &= P(s)U(s) \\ P(s) &= M^{-1}N \end{aligned} \qquad \text{Eq. 2-33}$$

This plant transfer function matrix $P(s) = [p_{ij}(s)]$ is a member of the set $P = \{P(s)\}$ of possible plant matrices which are functions of the uncertain plant parameters. If the equivalent plant matrix P resulting from the three matrices is not square, a weighing matrix W can be used to form an effective square plant.

In CACSD practice, one of three explicit methods can be used to define the region of plant uncertainty. The first is based upon the physical modeling of various plants representing the variety of possible plants. The second includes the selection of only a finite set of P matrices, representing the extreme boundaries of plant pole/zero uncertainty. The third considers the variations in plant coefficients by considering a preselected number of plants to represent the maximum variations. A convex hull is then closed around these plants to derive the minimum number of plant models to represent the variation.

An mxm MIMO closed-loop system can be represented by three mxm transfer function matrices, F, G, and P. There are m^2 closed-loop system transfer functions $t_{ij}(s)$ (transmissions) contained within its system

transmission matrix or system tracking matrix. $T_R(s) = \{t_{ij}(s)\}$, relates the outputs $y_i(s)$ to the inputs $r_j(s)$, that is, $Y_i(s) = t_{ij}(s)r_j(s)$. In a quantitative problem statement there are tolerance bounds on each $t_{ij}(s)$, giving a set of m^2 acceptable regions $T_{ij}(s)$ which are to be specified in the design, thus $t_{ij}(s) \varepsilon T_{ij}(s)$ and $T(s) = \{T_{ij}(s)\}$. These regions may also be directly given in the frequency domain.

The following system equations define the input/output relation of Fig. 2-2:

$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{Pu} \\ \boldsymbol{u} &= \boldsymbol{GE} \\ \boldsymbol{E} &= \boldsymbol{FR} - \boldsymbol{Y} \end{aligned} \qquad \text{Eq. 2-34}$$

In these equations G(s) is the matrix of compensator transfer functions and is often simplified so that it is diagonal. F(s) is the matrix of refilter transfer functions which may also be a diagonal matrix. The combination of these equations yields a 2 degree-of-freedom feedback structure

$$\boldsymbol{Y} = [\boldsymbol{I} + \boldsymbol{PG}]^{-1}\,\boldsymbol{PGFr} \qquad \text{Eq. 2-35}$$

where the system tracking control ratio relating r to y is

$$\boldsymbol{T_R} = [\boldsymbol{I} + \boldsymbol{PG}]^{-1}\boldsymbol{PGF} \qquad \text{Eq. 2-36}$$

The disturbance model is given as

$$\boldsymbol{T_D} = [\boldsymbol{I} + \boldsymbol{PG}]^{-1}\boldsymbol{P} = \{\boldsymbol{d_{ij}}\} \qquad \text{Eq. 2-37}$$

The MIMO design objective is to determine a F and G for all plants in {P} such that

(1) the closed-loop control ratio is stable and

(2) the norm of $t_{ij}(w)$ is bounded: $a_{ij}(w) \le t_{ij}(w) \le b_{ij}(w)$ for $w \le w_o$

(3) the disturbance, Eq. 2-37, is bounded (disturbance rejection).

A linear mapping from a MIMO system structure results in m^2 MISO equivalent systems, each with two inputs and one output. One input is designated as a "desired" tracking input and the other as a disturbance input. To develop this mapping consider the inverse of the plant matrix represented by

$$P^{-1} = \begin{vmatrix} p^*_{11} & p^*_{12} & \cdots & p^*_{1m} \\ p^*_{21} & p^*_{22} & \cdots & p^*_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ p^*_{m1} & p^*_{m2} & \cdots & p^*_{mm} \end{vmatrix} \qquad \text{Eq. 2-38}$$

The m^2 effective plant transfer functions are formed by defining

$$q_{ij} \equiv \frac{1}{p^*_{ij}} \qquad \text{Eq. 2-39}$$

A Q matrix is defined as:

$$P^{-1} = \begin{vmatrix} q_{11} & q_{21} & \cdots & q_{1m} \\ q_{21} & q_{22} & \cdots & q_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ q_{m1} & q_{m2} & q_{m3} & q_{mm} \end{vmatrix} = \begin{vmatrix} p^*_{11} & p^*_{12} & \cdots & p^*_{1m} \\ p^*_{21} & p^*_{22} & \cdots & p^*_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ p^*_{m1} & p^*_{m2} & \cdots & p^*_{mm} \end{vmatrix} \qquad \text{Eq. 2-40}$$

where $\mathbf{P} = [p_{ij}]$, $\mathbf{P}^{-1} = [p^*_{ij}] = [1/q_{ij}]$, and $\mathbf{Q} = [q_{ij}] = [1/p^*_{ij}]$.

the matrix $\mathbf{P}^{-1}$ is partitioned to form

$$P^{-1} = [p^*_{ij}] = [\frac{1}{q_{ij}}] = \Lambda + B \qquad \text{Eq. 2-41}$$

where $\Lambda = \{\lambda_{ii}\}$ is the diagonal part and $\mathbf{B} = \{b_{ij}\}$ is the off-diagonal component of $\mathbf{P}^{-1}$. Thus, $\lambda_{ii} = 1/q_{ii} - p_{ii}$, $b_{ij} = 1/q_{ij} = p_{ij}$ for $i \neq j$. Pre-multiplying Eq. 2-36 by $\mathbf{P}^{-1}[\mathbf{I}+\mathbf{PG}]$ yields

$$\begin{aligned} P^{-1}[I + PG]T_R &= P^{-1}[I + PG][I + PG]^{-1}PGF \\ [P^{-1} + G]T_R &= GF \end{aligned} \qquad \text{Eq. 2-42}$$

If we let $\mathbf{P}^{-1} = \Lambda + \mathbf{B}$, where Λ contains the diagonal terms and $\mathbf{B}$ contains the off diagonal terms, Eq. 2-42 becomes

$$\begin{aligned} [\Lambda + B + G]T_R &= GF \\ [\Lambda + G]T_R &= GF - BT \\ T_R &= [\Lambda + G]^{-1}[GF - BT] \end{aligned} \qquad \text{Eq. 2-43}$$

Each of the m^2 matrix elements on the right side of Eq. 2-43 can be interpreted as a MISO problem. A fixed point mapping based on Schauder's fixed point theorem (D'Azzo, pg 699) is defined by $\mathbf{Y}(\mathbf{T_R})$ as:

$$\boldsymbol{Y(T_R) = [\Lambda + G]^{-1}[GF - BT]} \qquad \text{Eq. 2-44}$$

where $\mathbf{G} = \{g_{ij}\}$ is assumed to be diagonal and each member of $\mathbf{T_R}$ is from the acceptable set $\{\mathbf{T_R}\}$. If this mapping has a fixed point, i.e., $\mathbf{T_R} \in \{\mathbf{T_R}\}$ such that $\mathbf{Y}(\mathbf{T_R}) = \{\mathbf{T_R}\}$, then this $\mathbf{T_R}$ is a solution of Eq. 2-43.

The control ratios for the desired tracking of the inputs by the corresponding outputs for each feedback loop of Eq. 2-44 have the form

$$y_{ii} = w_{ii}(v_{ij} + d_{ij}) = y_{r_i} + y_{d_{ij}} \qquad \text{Eq. 2-45}$$

where

$$w_{ii} = \frac{q_{ii}}{1 + g_{ii}q_{ii}} \qquad \text{Eq. 2-46}$$
$$v_{ij} = g_{ii}f_{ii}$$

The interaction between the loops has the form

$$d_{ij} = -\sum_{k=i} \frac{t_{kj}}{q_{ik}} \qquad k = 1,2,\cdots,m \qquad \text{Eq. 2-47}$$

and appears as a "disturbance" input in each of the feedback loops.

There are many other aspects and intricate implementation details not covered here because they are beyond the scope of this discussion. The interested reader is referred to other excellent sources for these discussions (Houpis, 1987).

Concluding Remarks on Quantitative Feedback Theory

This review discussed the mathematical foundation of the Quantitative Feedback Theory by first discussing the MISO problem and then briefly developing the MIMO problem. The MISO technique yields a controller that is capable of compensating a single output system with bounded parameter variation. The MIMO

technique is then a natural extension of the MISO technique in that a system is decomposed into a set of equivalent MISO loops and design proceeds element by element in the MISO fashion.

2.5 Summary

In this chapter, we developed a general basis of knowledge in three critical areas. First, we discussed the general concepts of object-oriented programming including the analysis, design, and implementation of object-oriented program structures. We discussed object-oriented terminology, talked about Borland's Turbo Vision object oriented fourth generation language, and discussed the advantages and disadvantages of object-orientation. We found that from the standpoint of software engineering goals, reusability, extendibility, etc, object-orientation is far superior to any other known construct. However, it does present a steep learning curve as it requires both a different logic structure and a different implementation structure presenting the new programmer unknowns in both the problem space and implementation space of a computer program.

The second general area of discussion was numerical methodology. We discovered the importance of careful numerical algorithm design and how carelessness can easily lead to numerical inaccuracy. Additionally, we researched several alternatives to problems common to CACSD programs such as determinants, matrix inverses, and the eigenvalue problem. In each case, we seek the best algorithm for the most general problem.

Finally, we developed the general mathematical background for the Quantitative Feedback Theory. This development provides a roadmap for future implementation of QFT toolboxes (a complete discussion on the toolbox concept is given in chapter 4).

The general knowledge attained from the research outlined in this chapter forms the basis for the specifications given in chapter 4 and for the actual implementation of ICECAP-PC as given in chapter 5. Our next task is to develop the specifications for ICECAP-PC.

3 Requirements and Specifications

3.1 Introduction

This chapter defines specific requirements of the ICECAP-PC program. First, we consider general traits such as accessibility, portability, etc. While many of these traits seem obvious, they do bear on our final platform selection and program structure. Second, we describe the human interface. Third, we develop a list of desired mathematical capabilities that define the basic math engine of ICECAP-PC.

We do not develop an OOA model as described in chapter 2. While a complete OOA model would certainly be expected in large government software contracts, experience tells us that future ICECAP-PC programmers are unlikely to use it. We found little use for previous functional models. It would, therefore, be an unneeded expense of valuable time. Instead, a basic object model of ICECAP-PC is given within chapter 4, *Design and Implementation*, to aid in the discussion of the actual ICECAP-PC structure. In this way, we can highlight the critical areas of the program design in an effective manner.

3.2 General Traits

Accessibility:

ICECAP-PC should be readily available to the widest audience possible. Ideally, ICECAP should be available to every controls engineering student across the country. The best way of achieving this is to write the program for the MS-DOS compatible platforms as this is by far the most popular computer platform available. Additionally, the ICECAP-PC program should be a ".exe" file, rather than a macro of some other mathematical program. In order to make the program accessible, we also need to provide it with as minimal cost. As ICECAP-PC is a public domain program, we provide it free of charge to any interested party.

Accuracy

As discussed in chapter 2, Numerical accuracy is fundamental in any CACSD program. We want to provide the best possible numerical support for control engineering calculations. This has several implications. First, we must insure that we are using state of the art numerical techniques. Every major algorithm must undergo through review to ensure this. Second, as a program that specifically addresses control systems

engineering we can make certain assumptions. For instance, we can assume that a realizable transfer function will have real coefficients in both numerator and denominator. This directly bears on our root finder as we can implement a faster and more accurate algorithm by assuming exact conjugates for complex pairs. Third, this impacts the range of expected values. Very few control systems, and certainly no educational systems, have transfer functions with coefficients less than 1×10^{-6} or greater than 1×10^{8}. Thus the range of numbers where we desire best accuracy is quite achievable and reasonably defined. Fourth, the accuracy requirement encourages the use of the highest precision floating point number available, the IEEE Std 754 80 bit extended number for all calculations as discussed in chapter 2.

Graphics Presentation

Graphic representation of system response is basic to the study of control systems. ICECAP-PC has always provided a rich set of graphic tools that are easy to use and understand. However, we desire improvements in the graphics engine by (1) providing direct plots of graphic screens to printers, (2) providing interactive graphics for educational purposes, and (3) providing multiple graphs. Interactive graphics allow the user to modify a transfer function and watch the effect on a frequency or time response. Thus a user can see directly the effect of added poles and zeros on a transfer function. This is especially important for the QFT problem. Multiple graphics would allow the display of several graphs at once. For instance, the user should be able to view both a time and frequency domain plot of a transfer function simultaneously. All of these would greatly improve ICECAP-PC. However, we must emphasize that this is a relatively low priority in this project being superseded by the structural redesign of the program itself.

Reusability

ICECAP-PC is a two-way educational program. First it provides educational support for the control systems student, the user. Second, it provides an educational experience for contributing authors who over the years have developed the program. ICECAP-PC code should be structured in such a way as to provide support to other engineering students in programming projects. For instance, the mathematical engine should be easily ported to other programs in other disciplines. This is a matter of code design and placement.

Speed

Finally, we desire ICECAP-PC to be very fast in interface and calculation. First, we seek a nimble responsive interface that provides rapid command execution. This requires (1) concise interface code, (2) the intelligent use of memory (a scarce resource in MS-DOS) and (3) an interface based on the fewest possible keystrokes for command execution. Command line interfaces do not provide good execution speed and are often quite frustrating. Menu based interfaces are often sluggish and present too much overhead. We need to combine the best of both worlds providing rapid menus and single line data (matrix, polynomial, transfer function) definitions. Second we seek rapid numerical solution wherever possible without risking accuracy.

Portability

ICECAP-PC need not be portable across different platforms because of the prevailing accessibility of MS-DOS machines on which ICECAP was meant to run. However, ICECAP-PC should run independently and not require the purchase of additional software or hardware other than a 80286 or better personal computer. Because of the numerical sophistication required by ICECAP-PC algorithms, we made the arbitrary decision not to support micro-processors lower than the 80286. Programs in an MS-DOS platform run faster if they are compiled with 80286 code rather than 8088 code.

MS-DOS compatible computers come in a variety of configurations, and where possible we should support advanced features such as expanded memory, math co-processors, high resolution graphics, etc. Many high level languages specifically address these features as compiler options and our choice of platform includes these considerations.

Intuitive Interface

In chapter 1, we stated that a major design goal was the development of a truly intuitive user interface. We now state the general characteristics of this interface based on research conducted by Wayne Bell (Bell, 1992).

1. The interface should be based on a pull-down menu structure accessible via keyboard or mouse for ease of use.
2. Keyboard commands should be executed with minimal keystrokes. The ideal number of keystrokes for a given command execution is 1.

3. The interface should be event-driven. An event driven interface is one that provides the user maximum control rather than constraining user actions through a hierarchal menu structure.

4. The interface should be very fast and responsive. Command execution speed should be limited only by the user.

5. The interface should display efficient use of memory and leave as much as possible for data manipulation. This is of primary importance in a PC environment with limited memory.

6. Data entry for matrices, polynomials and transfer functions should be in a format common to several commercial engineering programs such as MatLab and $Matrix_X$ thus providing a smooth learning curve for new users. Direct entry of complex numbers should be allowed.

7. The interface should provide a log file capability so that work performed is saved to a formatted ascii file that can be turned in as homework or examined by a practicing engineer.

8. Mathematical structures should appear in commonly accepted forms. Matrices should look like matrices, polynomials should look like polynomials, etc.

9. Both scientific notation and fixed decimal numerical display should be available to provide maximum control of numerical display to the user.

10. A pop-up context sensitive help facility should be provided and easily accessible.

11. All graphics data should be output to an ascii text file so that it can be imported to other programs.

12. A user programmable macro language should be provided for educational purposes and black box testing.

3.3 Human Interface Specification

The previous section lists a set of general goals for an intuitive interface. A set of specific specifications are now given to meet these goals. Again the interface specification is the result of the study conducted by Wayne Bell (Bell, 1992). They are defined as follows:

Menu System.

The menu system should allow access to any command by two ways: (1) a hierarchical menu activated by a mouse or by keyboard, (2) a toolbar icon activated by a mouse or by pressing a hot key on the keyboard. The menu system should maintain the look and feel of Microsoft Windows. The menu should be a list of command words displayed across the top line of the screen. The user selects the desired command by pointing at the desired command word with the mouse pointer and pressing the mouse button. This produces a "pop-down" menu that extends a new list of command words below the command just selected. If selected, these commands should either produce the desired output or display a dialogue box prompting for further details before producing the desired output. Under no circumstances will the menu levels exceed below the "pop-down" menus (no layers of menus are allowed).

Commands that require a certain sequence in activation will not be available until the user completes the prior commands. This should be conveyed to the user by displaying unavailable commands in a different color in the menu listings and by making them insensitive to activation. Also these commands will appear in their chronological order in the menu listing to assist the user in remembering.

The input of large collections of data should be represented in pictorial form whenever possible. One example of this is a system build function where the user connects icons with input/output lines and then defines what functions the icons represents. This system build function should be able to drive graphic displays allowing the user easy interaction with the iterative design process.

On-Line Help.

Context sensitive help should be available down to the command and error message level. It should allow activation either by a hot key or icon activation or by menu selection. Help will appear in a separate window and always allow access to a list of keywords and topics. By selecting from these lists, the user should be able to read all available help on every topic.

Data Display.

Multiple graphics windows should be allowed to be viewed and updated simultaneously. All command executions should update all active windows and create new windows if appropriate. Menus and icons should be accessible during graphics displays allowing a user to modify model parameters in one window and watch the

effects on a display in another window. Different windows should allow their data to be displayed in different domains (time, frequency, etc.).

3.4 Mathematical Specification

The mathematical engine of ICECAP-PC will be based on complex arithmetic to provide maximum numerical flexability. We decided on this format based on the requirements of state space analysis. While realizable control systems are always represented by real valued plant matrices, a designer will often have the system representation based on an a complex diagonal plant matrix readily available. This infers two things. First, this means that the entire data structure of the previous versions of ICECAP-PC must be redefined as they are based on real numbers rather than complex numbers. Second, this means that we must allow for the direct entry of complex numbers for polynomial and transfer function factors (but not their coefficients) and for matrices. While this seems like a lot of labor, the effort produces a highly adaptive mathematical structure.

ICECAP-PC will perform the following operations:

Matrix Operations
- Addition
- Adjoint
- Controllability Matrix
- Determinant
- Eigenvalues
- Eigenvectors (Modal Matrix)
- Hermite Normal Form
- Inverse
- Linear Quadratic Regulator
- Multiplication
- Observability Matrix
- Rank
- Resolvent Matrix
- Scaler Multiplication
- State Space to Transfer Function
- Subtraction
- Transpose

Polynomial Operations
- Addition
- Polynomial Curve Fitting
- Multiplication
- Subtraction

Transfer Function Operations
- Addition
- Domain Conversion (s,z,w')
- L^Hospitals Rule
- Multiplication
- Partial Fraction Expansion
- Subtraction
- Time Domain Response
- Time Domain Equation
- Transfer Function to State Space

Time Domain Analysis
- Figures of Merit
- Impulse Response
- Ramp Response
- Root Locus
- Step Response

Frequency Domain Analysis
- Bode Plots, Interactive
- Margins (gain and phase)
- Nichols Charts, Interactive
- Polar Plots

Listing 3-1: Desired Mathematical Capabilities for ICECAP-PC

3.5 Programming Standards

Programming standards are extremely important in any project involving groups. However, no previous thesis developed a set of guidelines for program structure. Rather, the standards developed related to decomposition and modeling methodology. We now present the following set of basic syntax rules to follow in the development of ICECAP-PC. These rules are arbitrary and stated for the sake of continuity.

Program Code

- Make indents with three spaces. Don't use tab characters in the source code.

- Make all main procedures visible in the interface section of their prospective units, listed in alphabetical order. Don't declare any hidden procedures.

- List all units in the uses clause of the main program file whether or not the main program file calls the unit directly. Experience shows that some compilers (Borland Turbo Pascal) cause intermittent mathematical errors in units not listed in the uses clause.

- Compile the program with full boolean evaluation turned off. Some numerical procedures require this.

- Denote internal procedures (procedures of procedures) as follows:

```
PROCEDURE ImAMainRoutine          {Note--Large Caps}

procedure ImAnInternalRoutine     {Note--Small letters}
var
   variables: variable
begin
   Internal Routine Code
end;

begin
   Main Routine Code
end;
```

Listing 3-2: Internal Procedure Format

Comments

- Denote comments with { and } to facilitate proper software maintenance.

- To temporarily comment out source code, place brackets placed on the left hand margin as shown in listing 3-3.

```
{
   This is garbage code
}
```

Listing 3-3: Temporarily Commenting Out Code

- Comment out embedded debugging code as shown in listing 3-4.

```
   Program Code Here

{Algorithm Test Code}
{->
   Debugging Code Here;
<-}

   Program Code Here;
```

Listing 3-4: Embedded Debugging Code

From this listing, note that the debugging code is commented out with {-> and <-} placed at the left hand margin. Also note that the debugging code is indented properly with the normal program code. Finally, the debugging code is noted with a {Algorithm Test Code} header placed on the left hand margin. The advantage to using this system is that debugging code is readily activated by completing the {->} and {<-} symbols.

3.6 Summary

This chapter lays the foundation for the implementation of ICECAP-PC by giving a set of specifications for the program. The specifications are first given in general form in section 3.2. Here we see a set of desired characteristics for the ICECAP-PC program that are easily satisfied in an object-oriented format. Section 3.3 specifies the desired program interface. The interface itself is not specified because it is dependant on the choice of platform given in chapter 4. However, the look, feel and behavior of the interface is clearly specified. Section 3.4 lists a set of mathematical capabilities that must be included in ICECAP-PC. Section 3.5 gives a set of guidelines for the structure of the code. These specifications now form the basis for the development of the program itself.

4 Design and Implementation

4.1 Introduction

This chapter describes the design (both high and low level), structure and implementation specifics of ICECAP-PC Version 10. Sections 4-1 through 4-4 cover the high level design aspects in which selection of platform, selection of data structure, development of the object model and the basic user interface are developed. The rest of the chapter is organized in object-oriented logic. We discuss the overall design of each critical object class; class by class. This allows us to thoroughly review the crucial aspects of important object classes such as algorithm selection, and individual object model. We do not include the complete object model description either in discussion or appendix. Such an inclusion would require extensive documentation that would likely sit unused by future developers of ICECAP-PC. Rather, we rely on highly self documenting code and use critical object models to highlight the important design decisions and characteristics of each important class.

As in any large programming project, we went through several design iterations, some minor and some quite extensive such as the change from simple objects to object families and actors. As we progressed, our concept of the finished program evolved from a very simple model to a more complex modular model required to meet the speed and memory limitations of a PC. The end product is a fast, efficient and accurate program with inherent expansion capability.

Consideration of Platforms:

Various languages offer considerable capability and were seriously considered. Since Ada is the only legally recognized language by the DOD (Department of Defense), it was given serious consideration. However, as this is an educational project for the authors we were not constrained to the use of Ada. Furthermore, Ada does not support object orientation because of its lack of inheritance and is extremely memory consumptive. It is impossible to implement a full featured engineering program on the PC using Ada. The second language considered was Borland's C++. This is an excellent language that supports object-orientation with its own version of Turbo Vision. Furthermore, C code tends to be memory conservative and C compilers support a large number of memory models (small, medium, large, huge). However, the existing version of ICECAP is written in

Pascal and the use of C++ would entail the complete re-writing of all procedures, a task to be avoided in this initial object oriented rehosting.

Selection of Platform:

After careful consideration of available platforms, we decided to use Borland Turbo Pascal V 6.0 and its fourth generation extension, Turbo Vision, as the development environment for the object-oriented version of ICECAP-PC. This allows maximum compatibility with previous ICECAP versions. Additionally, it enforces sound software practices and supports IEEE Std 754 extended precision numbers.

4.2 The ICECAP-PC Object Model

The development of the object model, discussed in chpater 2, is given with Rumbaugh's modeling syntax (Rumbaugh, 1991), as shown in Figures 4-1 through 4-3. Using this definition, hollow triangles indicate generalization (inheritance), triangles with ellipses underneath indicate generalization of objects not shown on the macro-model, and direct lines indicate association. The connection between the desktop and the object controller shows the relationship between the two (the desktop owns the object controller). The line between the Transfer Function {Abstract} and the TFBinary Ops classes shows inheritance (TF Binary Ops is a child of Transfer Function). Abstract objects, those denoted by the {abstract} designator are superclass objects that are never instantiated directly. Objects shown with an oval instead of a box are objects rather than object classes and are shown as such because there is never multiple instances of these.

The First Model

Our first object model is shown in Figure 4-1. From this figure, we see that ICECAP-PC is decomposed into a set of simple objects all of which are owned by the desktop directly. The matrix object is responsible for all matrix operations (see listing 4-1), the polynomial object responsible for all polynomial operations and so forth. Conceptually, this is a very nice model to work from. Ideally, each object could reside in memory at all times and simply respond to the events generated by the desktop. Note the inheritance structure for the matrix, polynomial and transfer function objects. They are all children of a common *Library* parent sharing a common data structure as discussed later in section 4.3.

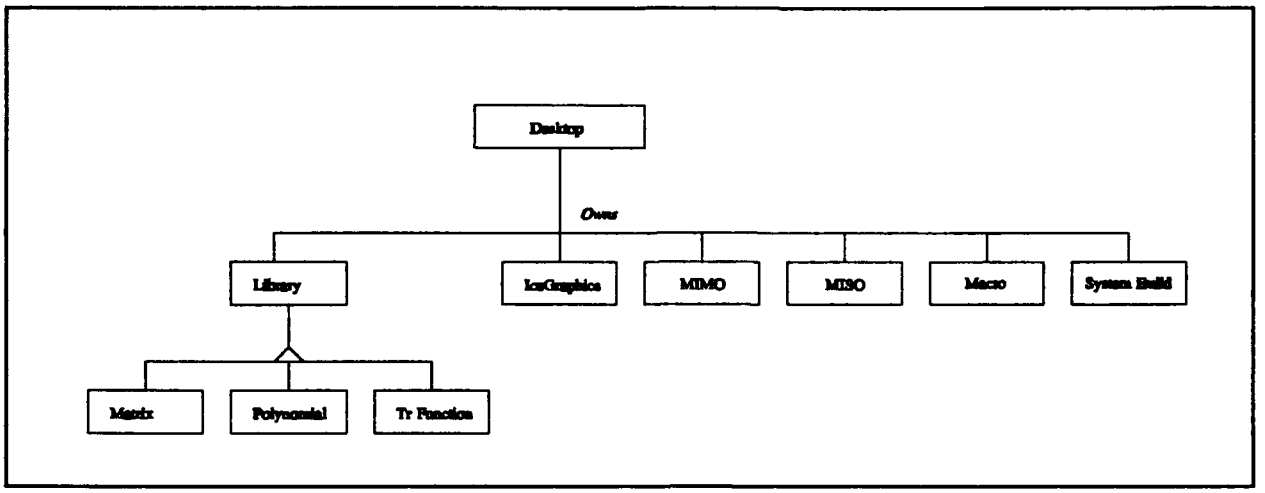


Fig. 4-1: First Object Model of ICECAP-PC

The problem with this model is its memory requirements. Each major object is actually quite large. Remember, the matrix object contains all data and methods to perform all operations we want could ever want to do to a matrix. With several of these large object residing in memory concurrently, we simply ran out of memory on a PC. Therefore, we had to institute some sort of instantiation control as shown in Figure 4-2.

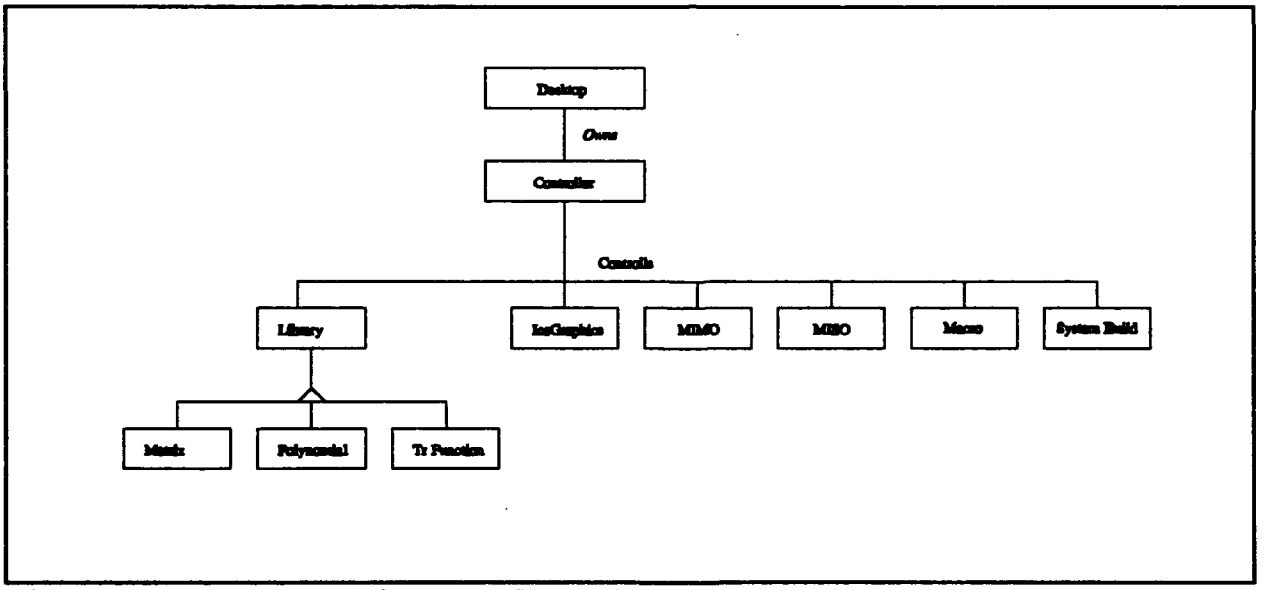


Fig. 4-2: Second Object Model of ICECAP-PC

The Second Model

Under this structure, an object controller handles the instantiation and disposal of each object. Thus in order to calculate the inverse of a matrix, the desktop would pass a message to the object controller telling it to

take the inverse, the object controller would instantiate a matrix object into memory and send it a message telling it to take the inverse, the matrix would take the inverse and display the result, and finally the object controller would dispose of the matrix object. We found this structure to fit quite well within the memory constraints of a PC. However, we also found it to be *VERY VERY SLOW*. The instantiation of a complete matrix object took up to 20 seconds on a 12MHz 80286 based personal computer. Calculation and display time were added to the instantiation time of the matrix object. We therefore had to break the matrix, polynomial, and transfer function objects into smaller groups, each of which could be instantiated and disposed of quickly. This, in addition to problems with the data structure discussed later in section 4-3 led to the development of the final object model shown in Figure 4-3.

The Final Model

As seen in this model, the desktop owns the status bottom, the toolbar, the menu bar, the view window, and the object controller. The object controller is an invisible object that controls all operational objects. *Operational objects* are those that perform specific calculations or CACSD capability for ICECAP-PC rather than simple interfacing. As shown previously, the purpose of the object controller is to preserve maximum memory by instantiating and disposing of all operational objects at the specific time they are required. With few exceptions, only one operational object resides in memory at any given time and the one(s) that do are very small.

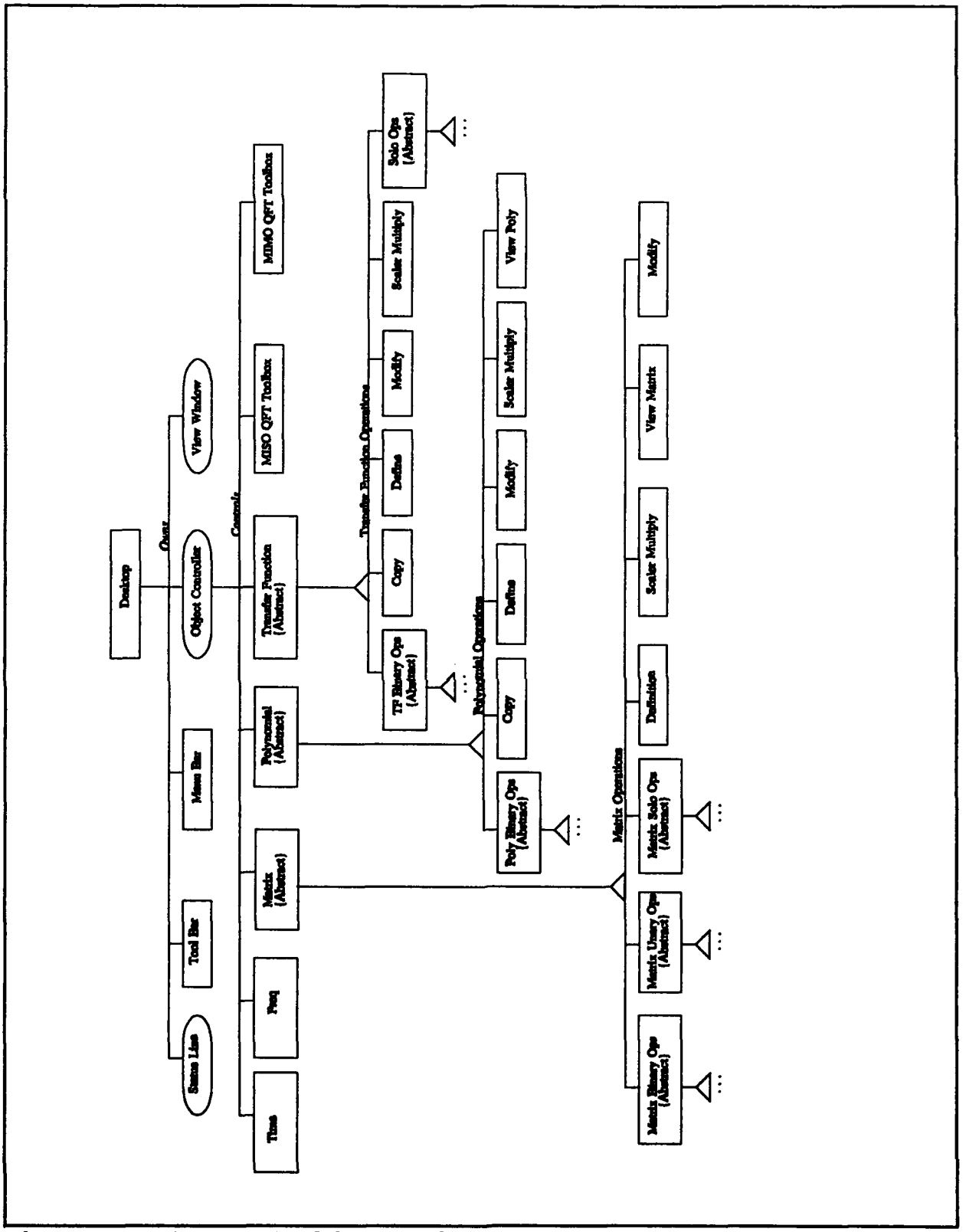


Fig. 4-3: ICECAP-PC Object Model

There are three classes of operational objects, two classes of graphics objects, and two classes of toolbox objects in ICECAP-PC. The operational object classes consist of entire families and their family tree is partially displayed in Figure 4-3. The matrix family consists of all matrix operational object classes. The polynomial family consists of all polynomial operational object classes. The transfer function family consists of all transfer function operational object classes. Currently, there are two toolbox object classes that implement MISO QFT and MIMO QFT. Future ICECAP-PC research will, no doubt, spawn more toolbox classes. The differences between toolbox object classes and standard ICECAP-PC operational object classes is discussed later.

From the macro-object model of Figure 4-3, we see immediate demonstration of the logical difference between object-oriented decomposition and functional decomposition. Note that each of the three main operational families have a "Define" object. These are three distinct code modules complete with data and method to perform the definition operation for their respective families. They each reside in their own Pascal file and are completely self contained. External action is limited to their simple instantiation; a simple process requiring two lines of code. They do the rest. Under functional decomposition this would not be the methodology used. Rather, some case statement would parse the "Define" command to a hierarchical tree of subroutines. A subsequent tree would parse the "Matrix" option and so forth to produce the required interface. Under previous versions of ICECAP, this tree was quite deep. One immediate benefit from object-oriented decomposition is the savings of stack space. Previous versions of ICECAP-PC would often fail because of stack overflow with a 64KB stack (the maximum allowed for a PC). The object-oriented version of ICECAP-PC runs reliably with a 16KB stack because the decomposition is more direct and few subroutine calls are placed on it.

4.3 The ICECAP-PC Data Structure

Basic Data Types

At the outset of this thesis effort, we attempted to unify all data under a single data structure as shown in Figures 4-1 and 4-2. Thus the data used by the matrix, polynomial and transfer function objects is represented as shown if listing 4-1. As seen in this listing, a complex number is defined as a record of extended and the basic data structure for matrices, polynomials, and transfer functions are thereupon built.

```
Extended_complex  = record
     realpart : extended;
     imagpart : extended;
end;

Matrix = record
     name    : string;        {Holds Matrix/Poly/TF Name}
     complex : boolean;       {Used For Poly, TF and Matrices}
     num_rows: integer;       {Used For Matrices and TF Degree}
     num_cols: integer;       {Used For Matrix/Poly/TF Degree}
     element : array[ 1..max_rows, 0..max_cols] of extended_complex;
end;
```

Listing 4-1: First Data Structure

The advantage to this approach is immediately apparent when using object orientations. All operations common to matrices, polynomials and transfer functions can be defined for a parent object type and simply inherited in its children. For example, the data input line for each of these can be declared as a method in a parent. The matrix, polynomial and transfer function objects would inherit this method as children of a common parent.

As development progressed, however, we found that several subtleties made continued use of a unified data structure undesirable. First, the maximum dimension for a matrix is 12x12; an arbitrary choice based on educational needs. However, we wanted the capability to operate on polynomials of 20th or larger order (also an arbitrary choice). A 20th order polynomial takes 12.8KB of memory when represented as a 20x20 square matrix internally. This is an undesirable waste of memory. Second, we desire to restrict polynomial definitions to real numbers when given in coefficient form and complex conjugates when given in factored form. However such restriction is unwanted for matrix definitions. Third, the polynomial record needs some field for the normalized polynomial gain whereas no such field is needed for matrices. Because of these conflicts we decided to use three different record types for matrices, polynomials and transfer functions as shown in listing 4-2. Under this data structure, we retain a pascal record as the basic number element and define matricies, polynomials and transfer functions with their own unique record set. Note that a transfer function is merely a record of two polynomial records.

```
Extended_Complex  = record
  realpart : extended;
  imagpart : extended;
end;

Root_Poly_Type  = array[ 1..max_Degree] of extended_complex;
Coeff_Poly_Type = array[ 1..max_Degree] of extended;

Matrix = record
   name      : string;
   complex   : boolean;
   domain    : char;
   samp_per  : extended;
   num_rows  : integer;
   num_cols  : integer;
   element   : array[ 1..max_rows,0..max_cols] of extended_complex;
end;

Polynomial = record
   name      : string;
   gain      : extended;
   degree    : integer;
   factored  : Root_Poly_Type;
   polyform  : Coeff_Poly_Type;
end;

TransFunc = record
   name      : string;
   domain    : char;
   samp_per  : extended;
   num       : Polynomial;
   den       : Polynomial;
end;
```

Listing 4-2: Final Data Structure

ICECAP-PC has, by arbitrary decision, ten predefined matrices, ten predefined polynomials, and ten predefined transfer functions. This doubles the amount of data from the previous versions and should more than adequately address educational needs. Matrices are names "Matrix A, Matrix B, and so on. Polynomials are named "Polynomial A"... and transfer functions are named "Transfer Function A"... and so forth. ICECAP-PC does not allow for user named matrices polynomials and transfer functions. Such capability would complicate the data retrieval and storage problem in a way that is beyond the scope of this work.

Since Turbo Pascal does not provide a predefined complex number data type, it was necessary to define a complex number as shown in the first record. Currently, max_rows and max_cols is set at 12. Max_Degree is set at 20. Therefore, the maximum matrix dimension is 12 and the maximum polynomial order of 19. These are not hard, fast values and can easily be increased by changing a single global variable defining their dimension.

The complex boolean element in the matrix record is set true if any of the matrix elements contain an imaginary value. This is done for two reasons. First, if a matrix is known to contain only real numbers, the entire imaginary side of the matrix can be set equal to 0.0 to avoid round off error. Second, the display of real

matrices, polynomials and transfer functions uses a sufficiently different algorithm to warrant the use of such a flag.

We decided against linked lists in the matrix record for three reasons. First, it was desirable to maintain enough compatibility with the current ICECAP-PC data structure to minimize the modifications to the numerical algorithm library. Second, since the data structures are never permanently stored in memory, heap space never runs critically short. Data variables (matrices, polynomials, and transfer functions) are only stored in memory during execution of manipulative routines. They are stored on disk at all other times. Thus, for example, to add two polynomials, ICECAP retrieves two polynomials from disk placing them in the heap, allocates heap space for the result, performs the operation, displays the result, stores the result to disk, and disposes of all the variables. Finally, operations on arrays are faster than operations on linked lists. Bearing in mind the limitations of a PC, it was desired to process numerical operations as quickly as possible.

Global Variables

In an object oriented program structure, there should be few if any global variables. Thus we intentionally minimized the number of such variables. Global variables in ICECAP-PC are located in the *globals.pas* file. This file contains global TYPE declarations, such as those shown in Figure 2, a few global graphics variables required by Turbo Pascal, and a set of message definitions as required by Turbo Vision. Message definitions are integer constants used for message passing between objects. As they must be available to all objects, they are inherently global. Such constants are named with either a br- or cm- prefix such as brInsertViewerData which is a message passed to the viewer object telling it to accept data into the screen buffer. Command constants, those with a cm- prefix, are generated from either the main menu or from push button objects such as those displayed on the toolbar at the right of the screen. No other object generates command constants and all command constants are directed to the desktop object. Upon receipt of a command constant, such as cmPolyAdd, the desktop generates a broadcast message to the appropriate object. In this case the message brPolyAdd is passed to the Polynomial object family. In addition to the above variable types, the global variable file also contains some commons sense variables such as ComplexZero and ComplexOne which represent 0+j0 and 1+j0 respectively. Their inclusion as global variables prevents their repeated declaration in

every math routine that may need them. In comparison to the size of the ICECAP-PC program, the global variable file is indeed small.

4.4 The ICECAP-PC Interface

The ICECAP-PC interface, designed to meet the specifications of chapter 2 and based on the user interface research conducted by Wayne·Bell (Bell, 1992), is shown in Figure 4-4.

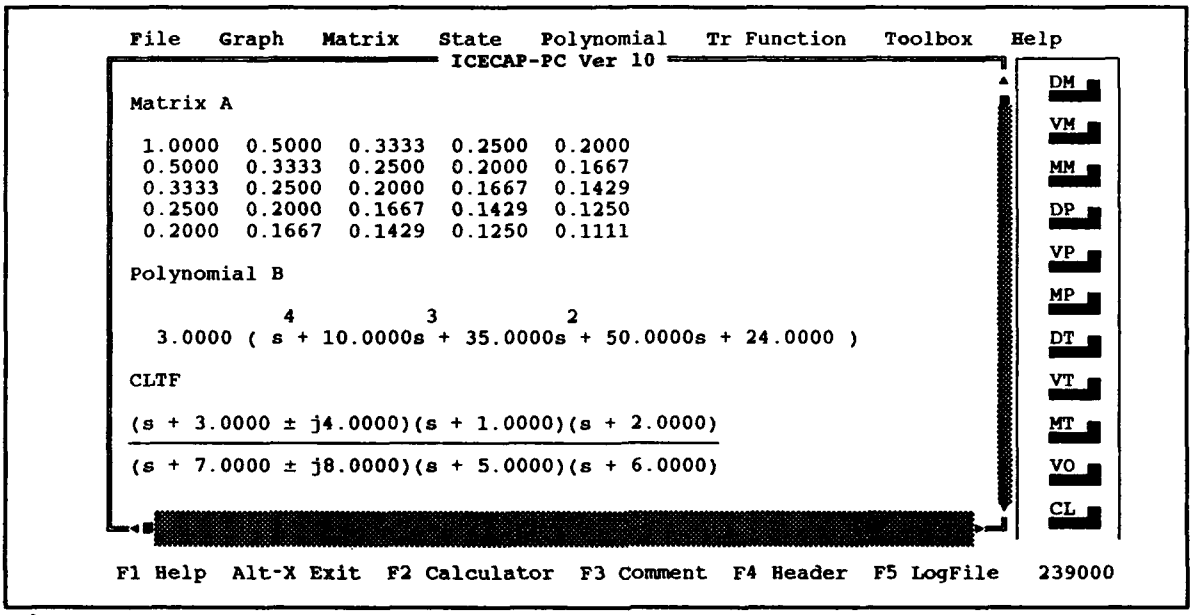


Fig. 4-4: ICECAP-PC Interface

The Interface, per the specification of chapter 2, is event-driven, mouse supportive, and based on the top row of drop down menus. The top menu, known as the *Menu Bar*, consists of nouns which produce menus of verbs. There is no second, level structure, that is, all selections are available from the main menu. On-line context sensitive help is available by either the F1 key or the right mouse button. All menus are accessed with one keystroke that corresponds to the highlighted letter. Matrix operations are given in the *Matrix* menu, state space operations in the *State* menu, polynomial operations in the *Polynomial* menu and transfer function operations in the *Tr Function* menu. Additionally, file operations, i.e. shelling to dos, exiting, and starting the log file are accessed in the *File* menu. Each menu selection is accessible from either a single key stroke or via mouse selection. Pressing "S" or selecting "State" with the mouse produces the state space drop down menu.

Quick command access is provided by the *Toolbar*. This is the vertical column of buttons located on the right hand side of the screen. These are available with mouse selection only. If the user does not have a mouse, they may disable the toolbar with the *File-View Options* selection. Each button on the toolbar generates a command to ICECAP. Buttons have the following definitions:

- DM - Define Matrix
- VM - View Matrix
- MM - Modify Matrix
- DP - Define Polynomial
- VP - View Polynomial
- MM - Modify Polynomial
- DT - Define Transfer Function
- VT - View Transfer Function
- MT - Modify Transfer Function
- VO - View Options
- CL - Clear Screen

The *heap viewer*, located directly underneath the toolbar, displays the amount of available memory. Typically, 200-300 KB of memory is available for data during an ICECAP-PC session. This is about four to five times as much as was available with ICECAP 9A, the last functionally written version!.

The bottom row is the *Status Line*. The status line provides reference and quick access to several interface functions. Pressing F1 provides context sensitive help. F2 provides a pop-up calculator. F3 provides a comment line dialog box for users to inter comment lines into the screen directly. F4 provides a dialog box to define the header that can be placed at the top of the screen. F5 toggles the log file on or off.

The center of the screen is the *View Window*. The window provides a horizontally and vertically bi-directional scrolling screen 230 lines long and 256 characters wide. Text data and comments are displayed in the View Window. In Figure 4-4, the following items are displayed: A 5x5 hilbert matrix in 4 point fixed decimal notation; A polynomial is in coefficient form and the same numerical format; and a transfer function in factored form. Several display formats are available, but not shown. Numbers may be displayed in fixed decimal notation with 1-17 decimal places for in scientific notation with 1-17 decimal places. Polynomials and transfer functions may be displayed in polynomial (coefficient) form, factored form or in root list form.

While the interface itself is quite interesting, the important concept here is that everything displayed on the screen is itself an object with a specific place in the overall structure of ICECAP-PC. The desktop is the object that contains and owns all other objects displayed. The desktop itself is invisible because it is completely covered up. The menu bar is an object that generates commands to the desktop. The menu selection *Matrix-View* generates the cmMatrixView command (an internal command) to the desktop. The toolbar is a aggregate collection of button objects, each of which also generate commands to the desktop. Pressing the VM button also generates the cmMatrixView command to the desktop. The view window is itself another object that provides several intricate functions discussed later.

We have fully discussed the high level design of ICECAP-PC and are ready to discuss specifics. The rest of this chapter will now deal with design specifics on a class by class basis. First, we will review the lines of inheritence and the relationship between ICECAP-PC objects and Turbo Vision objects. We will then discuss the three main operation object families, each consisting of a lineage of object classes.

4.5 The Application Family

Another view of the object model is provided in listing Figure 4-5. This listing is similar to that found in the Borland Turbo Vision Guide (pg 66). This specifically shows the inheritance relationship between ICECAP-PC object classes and the Borland Turbo Vision object class set. The Turbo Vision object classes shown in Figure 4-5 are not depicted in Figure 4-2. Also note that class names correlate, but do not match identically, between the object model of Figures 4-2 and 4-5. This is because Figure 4-2 is a macro level concept model while listing 4-5 is an inheritance chart. Object names in the actual pascal code match those found in Figure 4-5.

Interface Object Classes

ICECAP-PC object types are all children of types in Turbo Vision. Figure 4-5 shows the object class hierarchy of ICECAP-PC. All ICECAP specific object types are denoted with an asterisk.

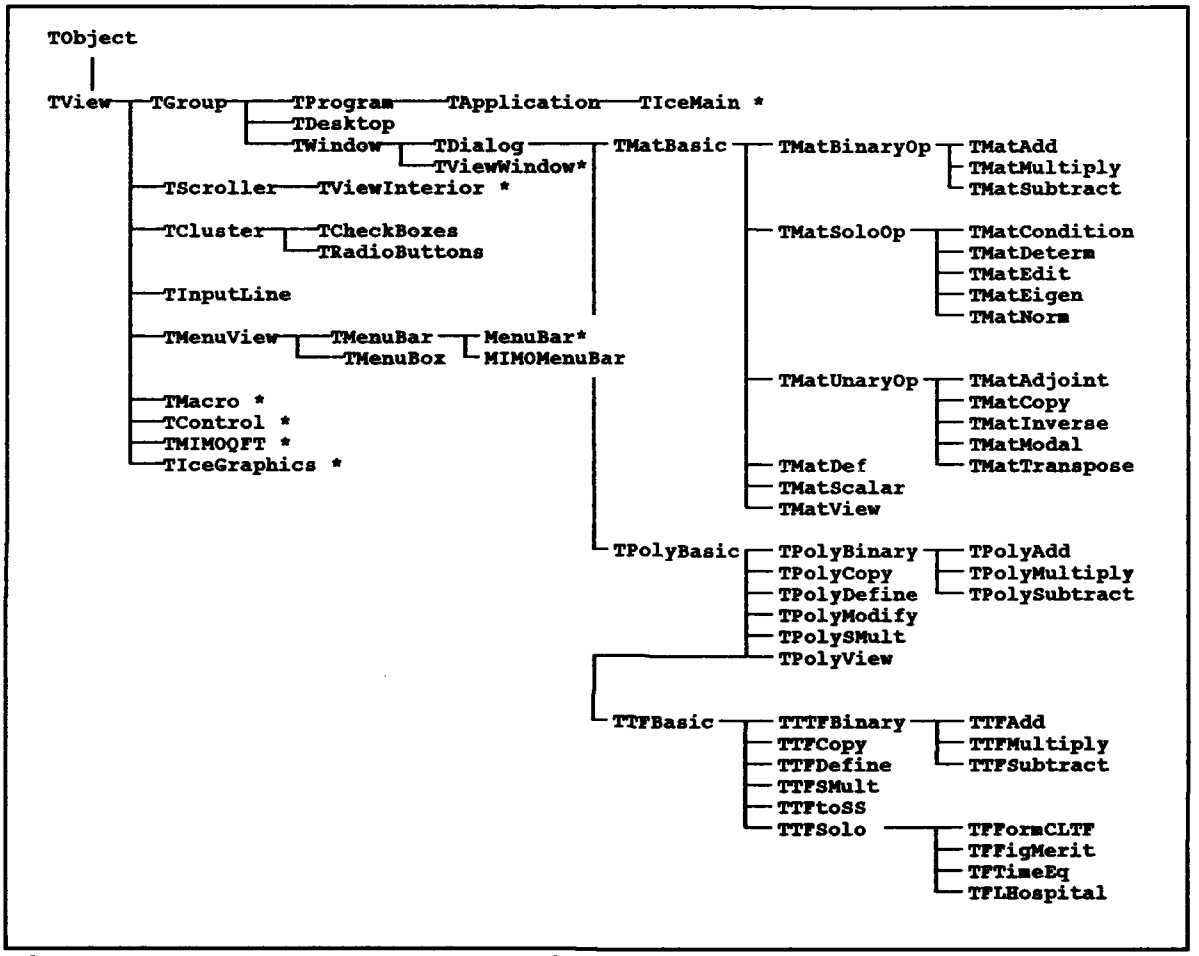


Fig. 4-5: ICECAP-PC Class Hierarchy

TIceMain is the main program object class. *TIceMain* is a child of *TApplication* which is predefined under Turbo Vision. Under the Turbo Vision standard, every program has one TApplication descendant which acts as a task scheduler. As such, TApplication descendants automatically own a *TDesktop* object that performs these tasks. This desktop object is depicted at the top level of Figure 4-5. The main job of *TIceMain* is to process events such as user keystrokes, mouse movement or messages from other objects. Additionally, it processes background tasks updating the screen buffer and the heap viewer, during idle periods. By defining a child of TApplication, we inherit the basic program structure of Turbo Vision and add event handling for

ICECAP-PC objects, single stroke menus (not defined in Turbo Vision), and help access using the right mouse button. Furthermore, we define the specific appearance of the screen and user interface shown in Figure 4-4. *IceMain* is the instance of *TIceMain* and owns all other objects through instantiation.

Here, as done in chapter 2, we must carefully distinguish between inheritance and instantiation. Inheritance defines the structure and capabilities of an object class while instantiation defines lines of ownership and control of actual data. The two concepts are quite different. *MatDiag* is an object, a specific instance of *TMatDiag* type (class) that is instantiated into IceMain. *IceMain* owns all other objects by similar instantiation. Specifically, it owns the objects *MenuBar* (an instance of TMenuBar), *MainWindow* (an instance of TViewWindow) along with several others.

MenuBar is a child object type (class) of TMenuBar which is defined under Turbo Vision. The Turbo Vision TMenuBar object type defines an empty generic drop down menu with capability of command generation and context sensitive help. By defining the MenuBar child of TMenuBar, we fill in the empty menu, specify a command set and develop an ICECAP-PC specific help system. Extra menu bars are added with each toolbox. For instance, the MIMO QFT Toolbox object contains a MIMOMenuBar definition for use in MIMO problems.

TViewInterior is the object class responsible for data display. The object model for this class is shown in Figure 4-6. This important object type (class) owns three I/O devices. First, it owns the view screen and is the sole object in ICECAP-PC that is allowed to write to it. The view screen is the information display area of Figure 4-4. The second I/O device owned by *TViewInterior* is the log file which is a text file named Logfile.txt. When the user elects to have a log file copy of screen writes, TViewInterior sends all subsequent screen writes to the log file. The third I/O device owned by *TViewInterior* is a header file. The header file is a record of the student's name, the course and the professor's name stored on disk. When the user elects to turn the log file on, they may place the header at the beginning of the log. Thus, the student is provided a completely formatted report including a standardized header containing student name, course number, professor, etc.

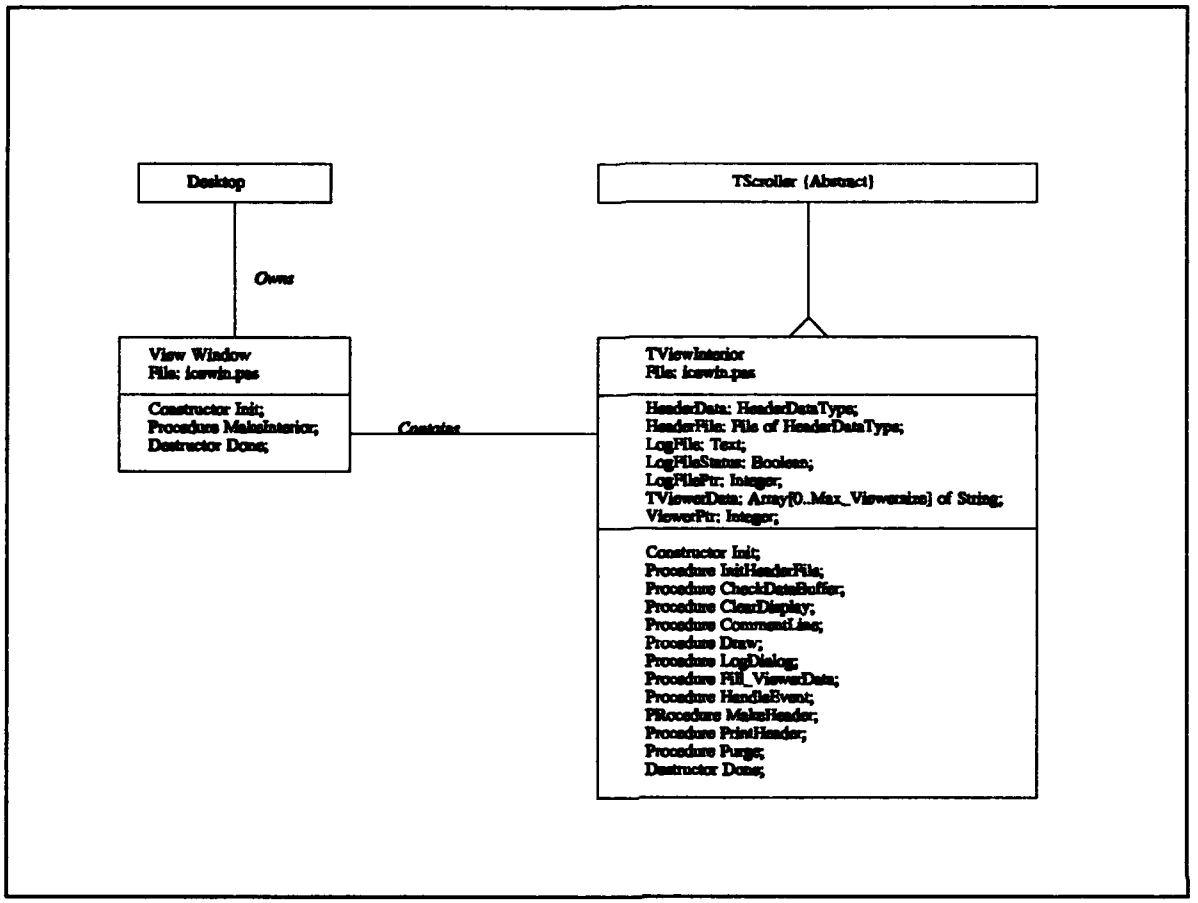


Fig. 4-6: The TViewInterior Object Model

All information displayed on screen is stored in a string array variable named *ViewerData.* ViewerData is an array of 230 lines each 256 characters long. Thus the screen display reflects only a small portion of the entire ViewerData array. ICECAP-PC supports both horizontal and vertical (forwards and backwards) scrolling over the entire ViewerData array. Thus up to 10 previous screens worth of information can be viewed simply by scrolling it. Vertical scrolling takes place on a line by line basis and horizontal scrolling on a character by character basis.

Information is stored to *ViewerData* and subsequently written to screen via a single global string variable named *ViewString*. When another object needs to send information to the screen it does so one line at a time by first formatting *ViewString* and then requesting a screen write from the *ViewInterior* object. Strings stored in *ViewString* are copied to the *ViewerData* array upon receipt of an *brInsertViewerData* message.

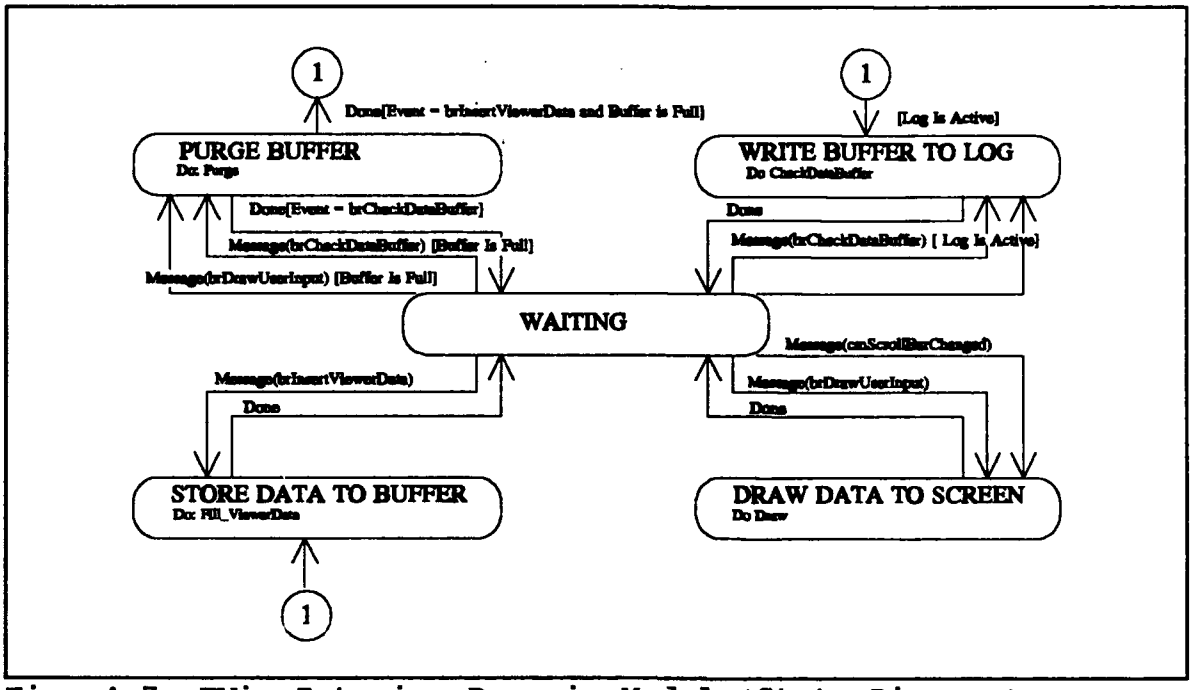


Fig. 4-7: TViewInterior Dynamic Model (State Diagram)

Figure 4-7 shows this as a change of state from WAITING to STORE DATA TO BUFFER. ViewString is then written to screen upon receipt of a *brDrawUserInput* message. It is a two step process. Defining screen writes as a two step process greatly increases the speed of the operation. An object can send several formatted text strings to *ViewerData* very quickly because the process doesn't involve the relatively slow screen write process. Afterwards, it can write all the information to the screen in a single write operation by sending the *brDrawUserInput* message. The latter operation is shown in Figure 4-5 as a change of state from WAITING to DRAW DATA TO SCREEN.

When the ViewerData array is nearly full, a call to the TViewInterior.Purge service deletes the top 1/4 of the array and shifts the rest of the array upwards. Normally, this takes place during idle periods and is invisible to the user. *TIceMain* has an idle time processor that performs background tasks and sends a *brCheckDataBuffer* message to *TViewInterior*. The br suffix indicates a broadcast message and is actually an integer constant. The purge operation, however, is not restricted to idle times. If a large amount of text is sent to *ViewerData* and it gets within one line of being full, a purge operation is automatically triggered to prevent overflow. The purge operation is depicted in Figure 4-7 as a transition from the WAITING state to the PURGE

BUFFER state. Note that both the idle time event and the overflow prevention event are shown. In the case of overflow prevention, the next states after a purge operation are the STORE DATA TO BUFFER state and the WRITE BUFFER TO LOG state if the log is active. When all purge and store actions are complete, *TViewInterior* returns to the waiting state.

ViewInterior is the only instance of *TViewInterior* as prescribed. However, it is a simple task to include multiple instances of TViewInterior. This may prove desirable in future developments of ICECAP-PC as different planes (s,z,w and w') could be represented in unique windows and operations performed in a pseudo-multi tasking environment.

The Object Controller Object Class

TControl is a child of *TView* whose purpose is to instantiate operational objects and insure their proper disposal when their function is complete. We were able to develop a uniform method of instantiation and disposal with the discovery that ***under Turbo Vision, any child object can be instantiated as an instance of a superclass object type.*** Thus in *TControl*, a temporary variable of type *TView (TempView := PView)* , is instantiated as instances of children of *TDialog* (the reader may need to refer to listing 4-4). For example, to instantiate a transfer function multiplication object, we declare "TempView := New(PTFAdd, Init('Transfer Function Addition, '+'));". TempView, being a child of *TView*, is a great-grandparent class of *TTFAdd*, yet the specific instance is of type *TTFAdd*. The parameters passed in the Init procedure appear in the dialog box as text and are necessary because *TTFAdd* is a child of *TTFBinary* which is parent to all transfer function binary math operational objects. With this capability, all instantiations can take place inside a single case statement located in the *TControl.HandleEvent* method. Inclusion of the controller object greatly simplifies the chain of command for future programmers of ICECAP-PC.

4.6 Operational Object Classes

ICECAP operational objects include the *TMatBasic* family, the *TPolyBasic family*,the *TTFBasic* (transfer function) family, the *TFreq* family, and the *TTime* family. MIMO QFT and MISO QFT are classified as toolbox objects and are discussed later. Operational objects are those used under the main menu and perform basic calculations needed by the control systems engineer. These operations are matrix operations, transfer function

operations, polynomial operations, and time and frequency domain simulations. However, in general terms, toolbox objects are those that define their own menu system replacing the default IceCap menu bar.

The Matrix Family

TMatBasic is a child of *TDialog*. *TDialog* is a pre-defined object type (class) under Turbo Vision that supports pop-up window dialog boxes for user I/O. Under the Turbo Vision standard, all user I/O is done with dialog boxes. Thus to add two matrices, the command is issued via pull-down menus and the matrix operands and resultant chosen on a pop-up dialog box. Dialog boxes provide several kinds of I/O including check-boxes, radio-buttons, inputlines, and pushbuttons. Check-boxes provide multiple item selection. For instance several polynomials can be displayed at once by picking several check-boxes. Radio-buttons provide unique item selection. For instance, the sum of two transfer functions can only be equal to a single transfer function. Inputlines provide text entry. Matrix, transfer function and polynomial definitions are accomplished on an inputline. Push-buttons issue commands. The toolbar on the ICECAP-PC screen is simply a dialog box with several pushbuttons.

TMatBasic is the parent object class for all matrix objects. It adds the ability to read matrix data from disk, load matrix data to disk, display matrices to screen, and perform numerical conditioning on matrices. Children of *TMatBasic* include the TMatDef, the *TMatBinaryOp* family, the *TMatUnaryOp* family, *TMatScaler*, the *TMatSoloOp* family and *TMatView*. *TMatBasic* is defined in Listing 4-4.

```
PMatBasic = ^TMatBasic;
TMatBasic = Object(TDialog)
   PROCEDURE  ConditionMatrix (var SomeMatrix: Matrix; InitMat : Boolean);
   PROCEDURE  DisplayMatrix (var Mat: Matrix);
   PROCEDURE  RetrieveData (var OldMatrix: Matrix; Stor_Loc: Integer);
   PROCEDURE  StoreData (var NewMatrix: Matrix; Stor_Loc: Integer);
end;
```

Listing 4-3: TMatBasic Object Class

The ConditionMatrix method acts as a filter for mathematical operations to prevent illegal floating point operations. Specifically, if a matrix is real valued, the entire complex plane is set to identically zero. Elements of the matrix array beyond the Num_Rows and Num_Cols fields are also set to identically zero. Numbers less than a predefined ZeroVal threshold (set at 1×10^{-100}) are also redefined as zero. All matrix math operations in

ICECAP-PC call ConditionMatrix to condition the operands before the operation takes place and then the resultant after the operation. The input filter action is especially important because Turbo Pascal will occasionally redefine a zero value passed in a parameter string as a number close to $1x10^{-3500}$ or smaller. Operations on such numbers can cause program failure.

TMatDef provides an input line and the ParseLine method to read user matrix definition input. Matrices are entered in much the same manner as with $Matlab_{tm}$ and $Matrix_{XTM}$. The chief differences are 1) *TMatDef* allows a 256 character input, 2) direct entry of complex numbers is specifically allowed, and 3) no brackets surround the data. Elements are separated with either a space or comma and rows are separated with semi-colons. Complex numbers are entered directly with either an i or j preceding the complex portion and no spaces between real and complex parts. For instance 1j2 or 1i2 defines 1 + 2j. The entry of 1 j2 (with a space) results in two elements 1 and 0 + 2j. The entry of complex numbers does not produce complex conjugates as happens during transfer function definition. *TMatDef* is defined as shown in listing 4-4.

```
PMatDef = ^TMatDef;
TMatDef = Object(TMatBasic)
   AbortIt    : Boolean;
   NewMatrix  : MatPtr;
   InputLine  : PInputLine;
   TheHistory: PHistory;
   Operand    : PRadioButtons;
   Format     : PRadioButtons;
   CONSTRUCTOR Init;
   PROCEDURE   HandleEvent(var Event: TEvent); virtual;
   PROCEDURE   MakeDiagonal (var Mat: Matrix; var Abort: Boolean);
   PROCEDURE   MakeHilbert (var Mat: matrix; var Abort: boolean);
   PROCEDURE   MakeIdentity (var Mat: Matrix; var Abort: boolean);
   PROCEDURE   ParseLine (TheInput: String;  VAR OutMatrix: Matrix; Abort: Boolean);
   PROCEDURE   InverseParseLine(var InputLine : String);
   DESTRUCTOR  Done; Virtual;
end;
```

Listing 4-4: TMatDef Object Class (Matrix Definition)

One of the most difficult tasks of this project was the development of *TMatDef.Parseline*. The Parseline method reads the user input matrix definition string and translates it into a numerical matrix. It has several interesting features. Either a space or comma may be used as a delimiter between elements on the same row. Furthermore any number of spaces or commas are allowed. Thus "1 2 3 4" and " 1 2 3,, 4" are treated identically and the result is the row vector [1 2 3 4]. Semicolons delimit the rows and my be surrounded by any number of spaces. Thus "1 2 3;4 5 6", "1 2 3 ; 4 5 6", "1 2 3 ; 4 5 6;" , and "1,2 3;4 5,6" are all

treated identically resulting in a 2x3 matrix. Complex numbers are indicated with either a j or i and must conform to the basic standard of no spacing between real and imaginary components. Thus "1j2", "1-j2", and "1j-2" all define the same complex number 1 + 2j. However the entry of "1 j 2" creates three numbers 1+0j, 0+1j, and 2+0j and the result is a row vector. Exponential notation is specifically allowed and numbers may be defined with either a small e or capital E. Thus "1e2j100" is a perfectly valid number. The difficulty in coding the parseline method was the inclusion of all the possible varieties of acceptable input. The result of our effort is a very robust and stable method.

As seen previously, the *TMatDef* object also provides methods supporting the quick definition of identity matrices, hilbert matrices and diagonal matrices. Hilbert and identity matrices are defined by simply entering a single number to describe the dimension of the matrix. Define diagonal matrices by entering the diagonal elements separated with either spaces or semicolons.

The *TMatBinaryOp* family performs all binary mathematical operations on matrices (addition, subtraction, and multiplication). Since *TMatBinayOp* is a child of *TMatBasic*, the entire family tree spawned by *TMatBinaryOp* inherits all methods and variables contained in *TMatBasic*. The object model of *TMatBinaryOp* is shown in Figure 4-8. Note the addition of a diamond in Figure 4-8. Using Rumbaugh's syntax (Rumbaugh, 1991), the diamond shows aggregation. That is the binary operations class is an aggregate composition of three radio button sets that specify the operands and resultant.

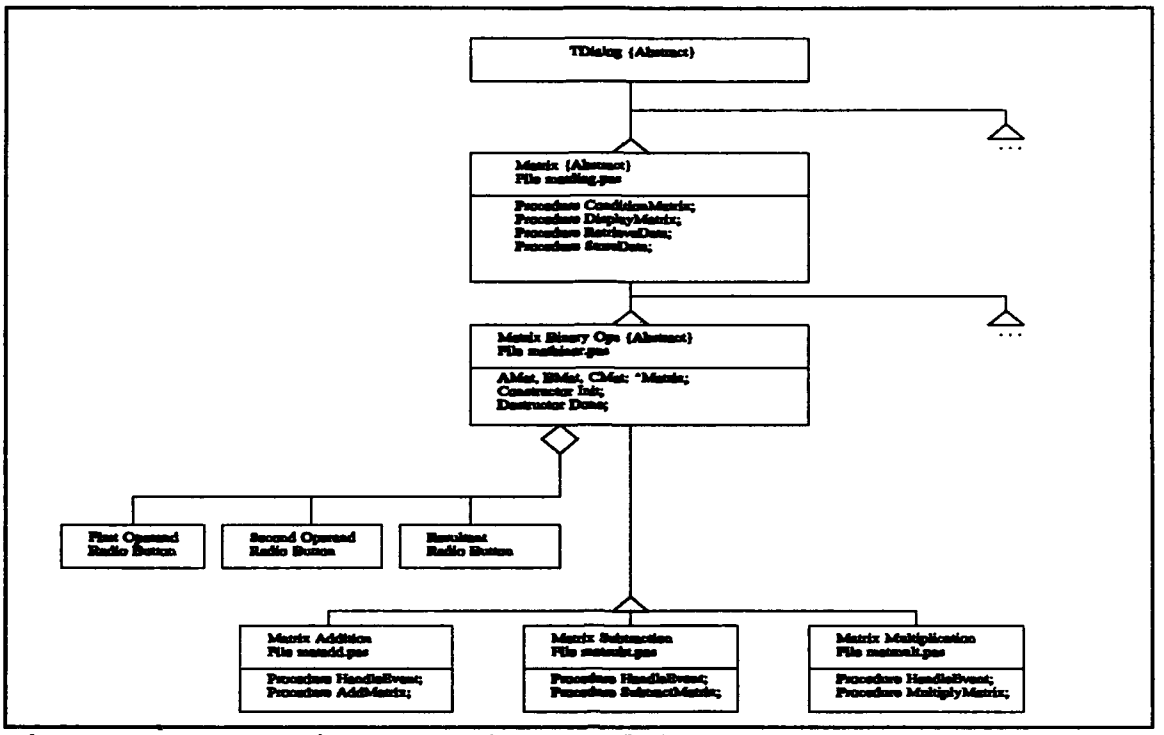


Fig. 4-8: TMatBinaryOp Class Model

The data flow diagram for *TMatBinaryOp* is shown in Figure 4-9.

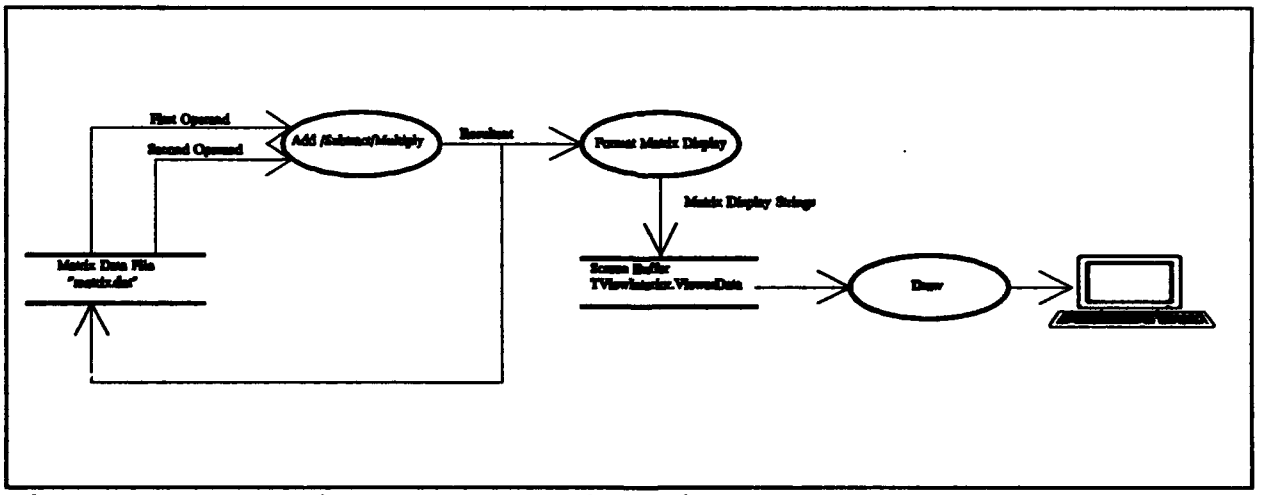


Fig. 4-9: TMatBinaryOp Data Flow Diagram

In this Figure, the two sets of horizontal lines indicate data stores (Rumbaugh, 1991). Note that the matrix data is read off disk, the resultant calculated and stored back on disk while also being displayed on screen. The answer is not held in computer memory.

The *TMatUnaryOp* family performs all unary mathematical operations (adjoint, copy, inverse, modal and transpose). Since *TMatUnaryOp* is a child of *TMatBasic*, the entire family tree spawned by *TMatBinaryOp* inherits all methods and variables contained in *TMatBasic*. Matrix inversion is performed by the *TMatInverse* object, a child of *TMatUnaryOp*. Inverses are calculated for the general complex matrix via LU Decomposition and Forward/Back Substitution as described in part II of this thesis. This algorithm is new to ICECAP-PC as previous versions used the Adjoint/Determinant formula, an inaccurate if not downright unstable algorithm. The object model of *TMatUnaryOp* is shown in Figure 4-10.

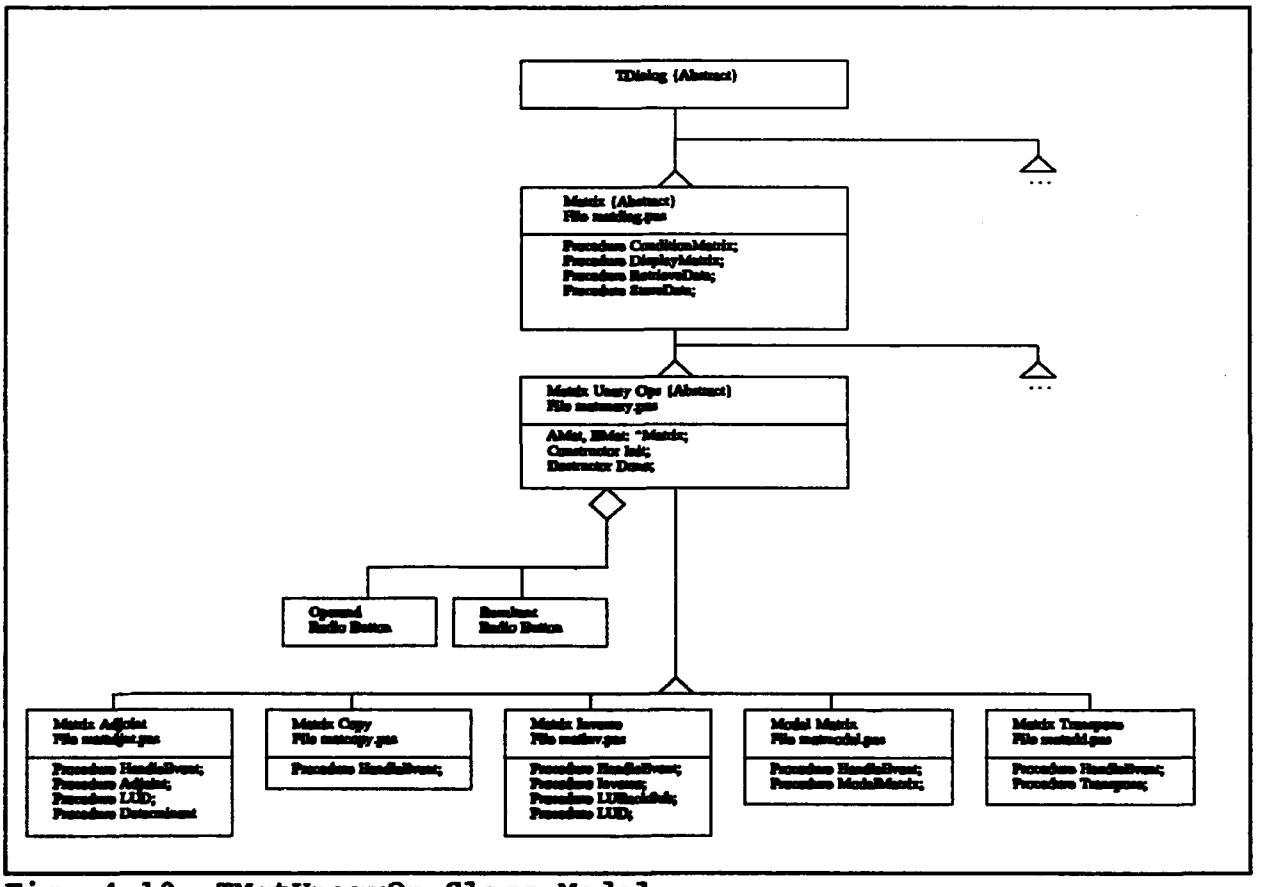


Fig. 4-10: TMatUnaryOp Class Model

The *TMatSolo* family performs solo operations or those operations which do not have a matrix resultant. These operations are the condition number calculation, the determinant calculation, eigenvalue calculation and matrix norm calculation. Members if the *TMatSolo* family include *TMatConditionNumber*, *TMatDet*, *TMatEigenvalue*, *TMatInverse*, and *TMatNorm*. The object model of *TMatSolo* is shown in Figure 4-11.

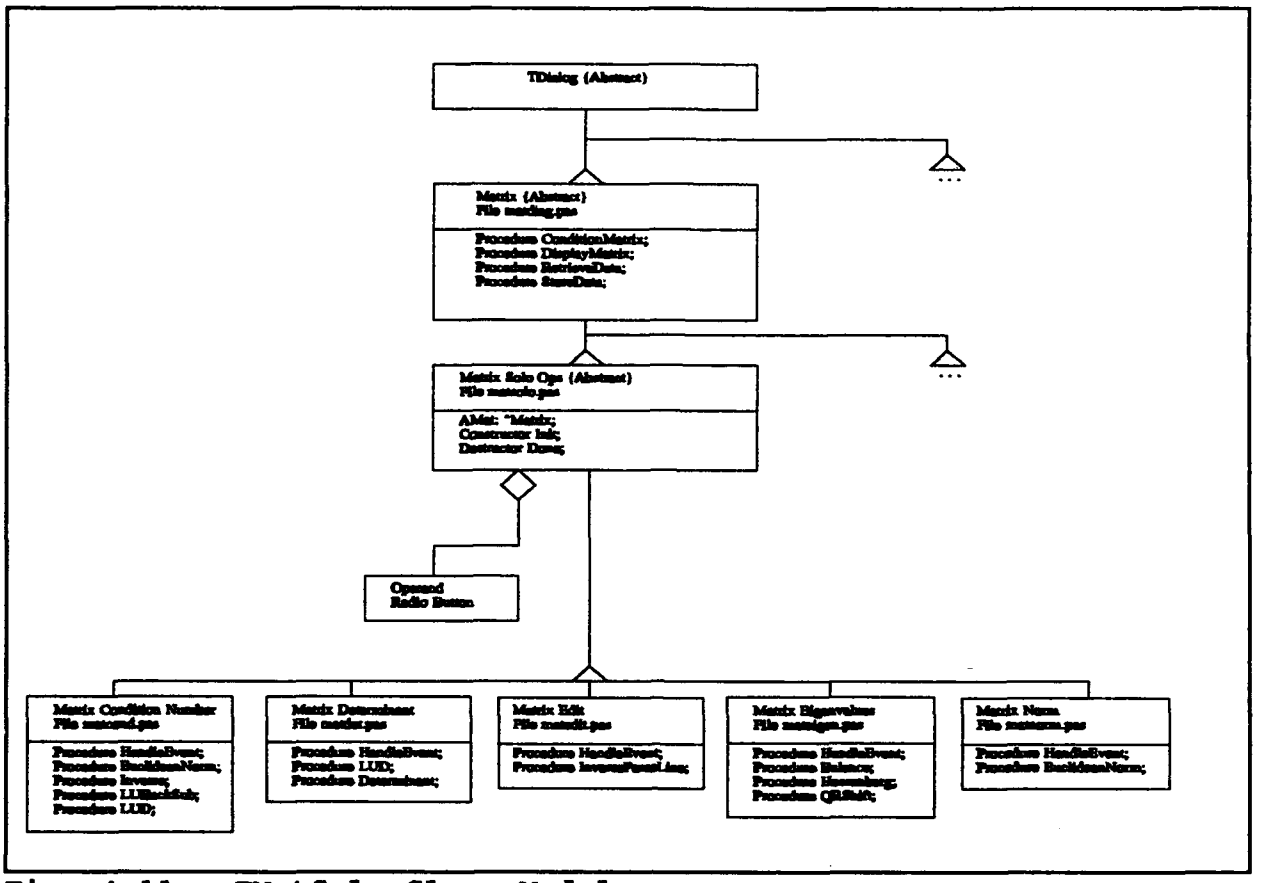


Fig. 4-11: TMatSolo Class Model

There are several ways of calculating the norm of a matrix including the euclidean norm, the row sum norm, the column sum norm, and the 2-norm. ICECAP-PC calculates the Euclidean norm which is simply the square root of the sum of all elements squared. Calculation of the norm is new to this version of ICECAP-PC.

The condition number is found by the simple multiplication of the norm of a matrix times the norm if the inverse of the matrix. For ill conditioned matrices, this value will be quite high and possibly inaccurate. However, the inaccuracy is of little consequence for a control systems engineer because the very existence of an ill conditioned system alerts the engineer that special design consideration is required. The exact amount of ill conditioning isn't needed. Calculation of the condition number is also new to this version of ICECAP-PC.

The matrix determinant is calculated via LU Decomposition. This method is discussed extensively in part II of this thesis. Under LU Decomposition, a matrix is decomposed into the product of two matrices, one upper triangular and one lower triangular. ICECAP-PC uses Crout's algorithm which places 1.0 in each diagonal

element of the lower triangular matrix. The matrix determinant is then simply the product of diagonal elements of the upper triangular matrix.

Eigenvalues are calculated in a three part process involving balancing, conversion to upper Hessenberg form and QR Shift application. The object class performing eigenvalue calculation is *TMatEigen*. *TMatEigen* is shown if Figure 4-11. *TMatEigen* is a child of *TMatSolo*. The balance method performs an orthogonal transformation on the operand matrix to reduce the condition number while maintaining the same eigenvalue set. The Hessenberg method performs another orthogonal transformation to put the balanced matrix into Upper Hessenberg form. The third method, QRShift performs the final transformation in which eigenvalues appear on the main diagonal if real or as block diagonal 2x2 sub-matrices if complex.

As a final note on the matrix object, recall that in section 4-2 we discussed why we deliberately chose not to define a single matrix object with all possible operations contained therein. Under the object oriented paradigm, this would indeed be a logical structure. Such a structure could be described in Listing 4-5 with ellipses indicating all possible matrix operations:

```
PMatrix = ^TMatrix;
TMatrix = object(TDialog)
     Constructor Init;
     Procedure HandleEvent(var Event: TEvent); virtual;
     Procedure Add;
     Procedure Define;
     Procedure Edit;
     Procedure Eigenvalues;
     Procedure Multiply;
     Procedure Subtract;
     Procedure Transpose;
     ...
     ...
     Destructor Done; virtual;
end;
```

Listing 4-5: Original Matrix Object Class Definition

Our original design called for a single matrix object class, a single transfer function object class and a single polynomial object class. However, we simply could not fit all these objects into the limited memory provided by the PC at the same time. Our next option was to instantiate the object only when an operation in that class was needed and immediately dispose of it. However, we were instantiating and disposing of very large objects (12-20 KB in size) and suffering a terrible price in speed. Our final, and most effective option was to

make objects out of verbs, namely the mathematical operations themselves. Thus the matrix object became a family of objects each contained in a separate pascal file and compiled in individual units. The average size of the instantiated object for a given operation is now about 800 bytes of memory vs 12KB with the single matrix object. Furthermore, we've experienced about a tenfold increase in interface speed making ICECAP quite nimble.

The Polynomial Family

The polynomial family consists of a parent, *TPoly*, which like *TMatrix* is a child of *TDialog*. *TPoly* is defined as follows:

```
PPolBasic = ^TPolBasic;
TPolBasic = object(TDialog)
   PROCEDURE  ConditionPoly (var SomePoly : Polynomial);
   PROCEDURE  DisplayPoly (var Poly: Polynomial; Format: Integer;
                           var TopStr, PolyStr: String);
   PROCEDURE  FormatPoly (var NewPoly: Polynomial; PolyorFact: Char;
                          var AbortCode: Integer);
   PROCEDURE  RetrievePoly (var OldPoly : Polynomial; Stor_Loc: Integer);
   PROCEDURE  StorePoly (var NewPoly: Polynomial; Stor_Loc: Integer);
end;
```

Listing 4-6: TPoly Object Class Definition (Polynomials)

The DisplayPoly method of the *TPolyBasic* class returns two string variables for any input polynomial, one for the coefficient string and one for the exponent string. Polynomials may be displayed in one of three ways: polynomial coefficient form, factored form, and root form. Additionally, each form can be displayed in either fixed decimal notation or scientific notation. Polynomial coefficient form and factored form displays are self explanatory. Root form display is a simple vertical listing of the roots of a given polynomial. Since both coefficient form and roots are stored internally inside the polynomial record, there is no loss of accuracy when the same polynomial is alternatively displayed in different forms.

The FormatPoly method builds the factored portion of the polynomial record entered when entry is in coefficient form and builds the coefficient form of a polynomial entered in factored form. Thus for a polynomial defined in coefficient form, roots are calculated only once during the definition process. If a polynomial is defined in factored form with complex roots, the FormatPoly method ensures the complex conjugate is also entered. Thus polynomial definition is restricted to real coefficient values.

Operations allowed on polynomials are addition, subtraction, multiplication and multiplication by a scaler. Each operation is controlled by independent objects.

The Transfer Function Family

The transfer function family consists of a parent *(TTFBasic)* which like *TMatBasic* is a child of *TPolyBasic*. The *TTFBasic* class is defined in listing 4-7.

```
PTFBasic = ^TTFBasic;
TTFBasic = object(TPolBasic)
   PROCEDURE   ConditionTF (var SomeTF : TransFunc);
   PROCEDURE   DisplayTF (var TF: TransFunc; Form: Integer);
   PROCEDURE   FormatTF (var NewTF: TransFunc; PolyorFact: Char;
                         var AbortCode: Integer);
   PROCEDURE   RootCancel (var N1, D1, N1_Simplified, D1_Simplified: Polynomial;
                           var AbortCode : Integer);
   PROCEDURE   RetrieveTF (var OldTF : TransFunc; Stor_Loc: Integer);
   PROCEDURE   StoreTF (var NewTF : TransFunc; Stor_Loc: Integer);
end;
```

Listing 4-7: TTF Object Class Definition (Transfer Functions)

The *TTFBasic* object class, like the *TMatBasic* class, features methods to read, condition and store data. These methods rely and call upon their inherited polynomial methods. For instance, FormatTF simply calls FormatPoly twice, once for the numerator polynomial and once for the denominator polynomial. Additionally, a method called RootCancel provides root cancellation between numerator and denominator polynomial elements. The cancellation threshold is set at 1×10^{-8}.

Children of *TTFBasic* include the *TTFBinary* family, *TFCopy*, *TFDefine*, *TFModify*, *TTFSMult*, *TTFtoSS*, the *TTFUnary* family, and *TTFView*. The *TTFBinary* family contains object classes to perform transfer function addition, multiplication and subtraction. Additionally, *TFLHospital*, a child of *TTFBinary* provides application of L'Hospitals rule by symbolically reducing a transfer function to the form K/S^n or S^n/K where K is some constant and S^n is some polynomial of degree n. While all practical control system transfer functions reduce to the for K/S^n, ICECAP-PC nevertheless handles the other case as well. Transfer function addition, subtraction and the application of L'Hospitals rule are new to this version, ver 10, of ICECAP.

Operations performed by children of *TTFBasic* are transfer function addition, subtraction and multiplication; multiplication of a transfer function by a scaler, transformation to state space (matrix) format,

calculation of figures of merit, formation of CLTF from conventional control elements GTF and HTF (the forward and feedback transmittances), and symbolic reduction via L'Hospitals rule.

4.7 Summary

This section discussed both high level design and the object-oriented implementation of ICECAP-PC, the problems that confronted us and the decisions that we made during the development. Decision making is an important part of any design project and we had far more to make than could be enumerated herein. During this project, we progressed through several structural changes, interface changes and numerical algorithm changes. At every juncture, we sought the best solution based on the stated criterea of chapter 2; namely numerical accuracy and utility for the most general problem case. One of the most challenging aspects of this project was the problems posed by MS-DOS itself and its extremely limited memory capacity. This, in fact, drove most of our fundamental structural development and forced us into many revisions. In the end, we solved the memory riddle by making very small objects that performed a single task, such as transposing a matrix. The existence of an object representing a verb rather than a noun is a violation of object-oriented logic; however, it became a survival tool that both saved ICECAP-PC and even made it quite nimble and fast even on a 286.

5 The MIMO QFT Toolbox

5.1 Introduction

This section describes the MIMO QFT Toolbox of ICECAP-PC. Discussion covers the toolbox concept, the interface, and the toolbox program structure including data structures and algorithms.

5.2 ICECAP-PC Toolboxes

The most difficult step in OOD is deciding what are the objects. When the viewpoint of the lowest level of abstraction is taken, it would seem logical to the strictest OO designer that each polynomial and matrix and transfer function should be an object. Then when the user asks for a frequency response of the open loop transfer function (OLTF), the OLTF object would draw a frequency response of itself on the screen. However, when the viewpoint of the OO software engineer (the person who must ***implement*** the design) is taken, it becomes more logical to think of some nebulous transfer function object that owns each of the individual transfer function data records. When the user asks for two transfer functions to be added together, the transfer function object services carry out the task. How could two objects add themselves to each other? There would have to be some owning object who could simultaneously access both their individual data records. The ICECAP-PC data structure has been designed using the latter implementation viewpoint.

Toolboxes differ from standard ICECAP-PC objects discussed previously in two important ways. First, toolboxes are children of *TView* rather than *TDialog*. Since *TView* is the parent of *TDialog*, it contains less functionality and is leaner. However, it still retains message passing capability which is key to successful implementation of object oriented programming under Turbo Vision. Second, toolboxes are packaged with their own menu bar and tool bar. The menu bar appears at the top of the screen and presents drop down menu selection. The tool bar appears at the right of the screen and provides rapid command selection. When ICECAP-PC loads a toolbox, it must first dispose of its own menu bar and toolbar and then instantiate the menu bar and tool bar of the desired toolbox. By convention, all tool box menu bars and tool bars are located in same Pascal file as the toolbox itself.

Two main toolboxes are implemented in this thesis cycle, one of which is part of this research. The MISO QFT toolbox provides the tools necessary to perform Quantitative Feedback Analysis on Multiple Input, Single Output (MISO) systems. The second toolbox, written for this thesis, is the MIMO QFT toolbox. This toolbox provides the tools necessary for transformation of a Multiple Input Multiple Output (MIMO) system description into an equivalent set of MISO problems. Upon decomposition and validation, the MISO toolbox is then used to perform QFT design.

As indicated on the ICECAP-PC menu itself, other toolboxes are planned for future thesis projects. These include a non-linear systems toolbox, an LQG toolbox, and an H-Infinity toolbox. Additionally, it may prove desirable to implement the Multi-Porter method as well as a variety of others.

5.3 MISO QFT Toolbox

The QFT toolbox within ICECAP-PC provides the basic set of mathematical algorithms necessary to design control systems using QFT control techniques. The toolbox, by design, is composed of one class called TQFT. This object class contains all the services required to define tracking and disturbance specifications, enter plant variations, enter disturbance models, generate plant templates, generate bounds, design a nominal loop transmission, generate a controller, design a filter, simulate responses, and generate a comprehensive report. It includes interactive graphics capability for the development of δ_R, the plant templates, the composite bounds, the nominal loop transmission, and the prefilter. The MISO QFT Toolbox is discussed in detail in a thesis presented by Wayne Bell (Bell, 1992), and in the QFT Users Manual included in the appendix.

5.4 MIMO QFT Toolbox

Introduction

The MIMO QFT (Multiple Input Multiple Output Quantitative Feedback Theory) is a natural extension of the MISO technique of the previous section. The heart of the MIMO technique is the decomposition of a MIMO problem into MISO equivalent loops as described in chapter 2. The strength of QFT is its recognition that no two things are identical. Two motors off the same assembly line spin at some rpm ± some δ. An airplane with ice on its wings flies differently than the same plane without ice. QFT defines these plants as a region bounded by some predictable uncertainty. Because of its overwhelming success in MISO problems,

further development brought its adaptation to MIMO design problems. MIMO QFT is a relative newcomer among MIMO design methods and much of the control systems engineering community is still unaware of its potential. However, some AFIT students have successfully used it to develop aircraft control systems. (Wheaton, 1990; Kobylarz, 1988)

The effective utilization of MIMO QFT requires a robust CACSD (Computer Aided Control Systems Design) program capable of solving problems defined as matrices of transfer functions. One program was recently developed by an AFIT student (Sating, 1992). Sating's MIMO QFT program (which is quite good) runs on a Sun workstation and requires Mathematica and $Matrix_X$. Mathematica's rich programming language provides the basis for this program. The deficiency of this program is its lack of portability. While AFIT and Wright Laboratory at Wright Patterson Air Force Base are replete with Sun workstations, this is not the general case in industry. Over the course of this effort, I visited Armco Industries in Middletown OH, Western Power Administration in Denver CO, Lawrence Livermore National Labs in Livermore CA, and the Federal Aviation Administration in Seattle WA. None of these agencies had Sun Workstations available and fewer yet had Mathematica and $Matrix_X$ installed. All of them had 80486 based PCs. Thus, there still exists a need for a good MIMO QFT CACSD program based on a PC, especially one tailored to academic institutions.

The ICECAP-PC MIMO QFT toolbox addresses this problem by providing an object oriented CACSD program for the solution of MIMO control systems problems. As far as we know, this is the first object oriented MIMO QFT CACSD package. The objective under this thesis cycle is simply to allow the efficient description of the uncertain plant set, validate its usability and realizability, perform matrix inversion to form the Q matrix (the inverse of P), verify the inversion of Q, and test for diagonal dominance. The MIMO QFT toolbox does not currently provide tools to perform loop shaping, bound generation, or prefilter design. These are provided with the MISO QFT toolbox. Additionally, the MIMO QFT toolbox does not contain any graphics capability. Again, this is done with the MISO toolbox. However; future thesis students may wish to add these analysis capabilities to the MIMO QFT toolbox. It would be a worthwhile project.

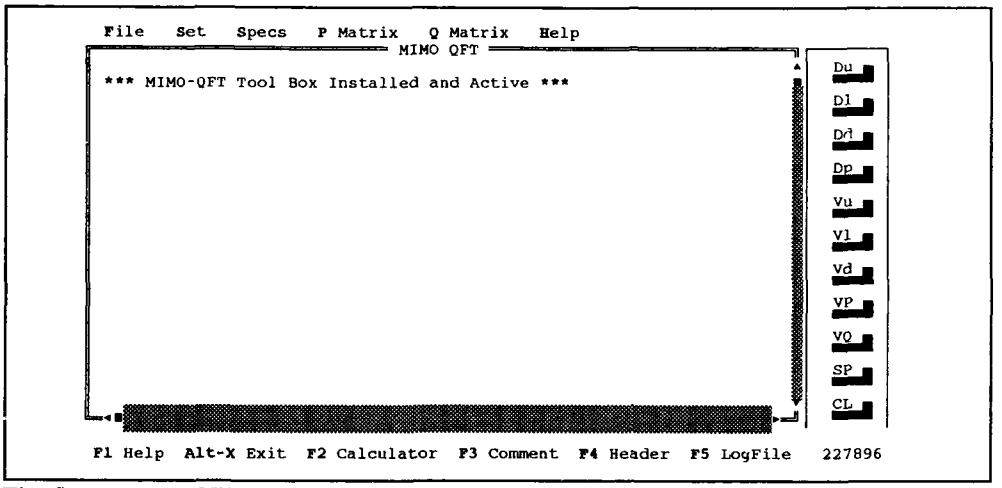


Fig. 5-1: MIMO QFT Toolbox Interface

The interface to the MIMO QFT toolbox is shown if figure 5-12. This interface resulted from several that were developed using fast prototyping, most of which were modifications of the MISO QFT menu structure. It is similar in function to the ICECAP-PC interface shown in figure 4-2. Operations involving the P matrix are under the P Matrix drop-down menu and operations involving the Q matrix are under the Q matrix drop-down menu. Stability specification operations are under the Specs menu. The buttons on the right of the screen perform the following actions:

Du - Define Tru

Dl - Define Trl

Dp - Define Plant p

Vu - View Tru

Vl - View Trl

Vd - View Td

Vp - View Plant p

VP - View Plant Matrix P

VO - View Options

SP - Select Current Plant

CL - Clear Screen

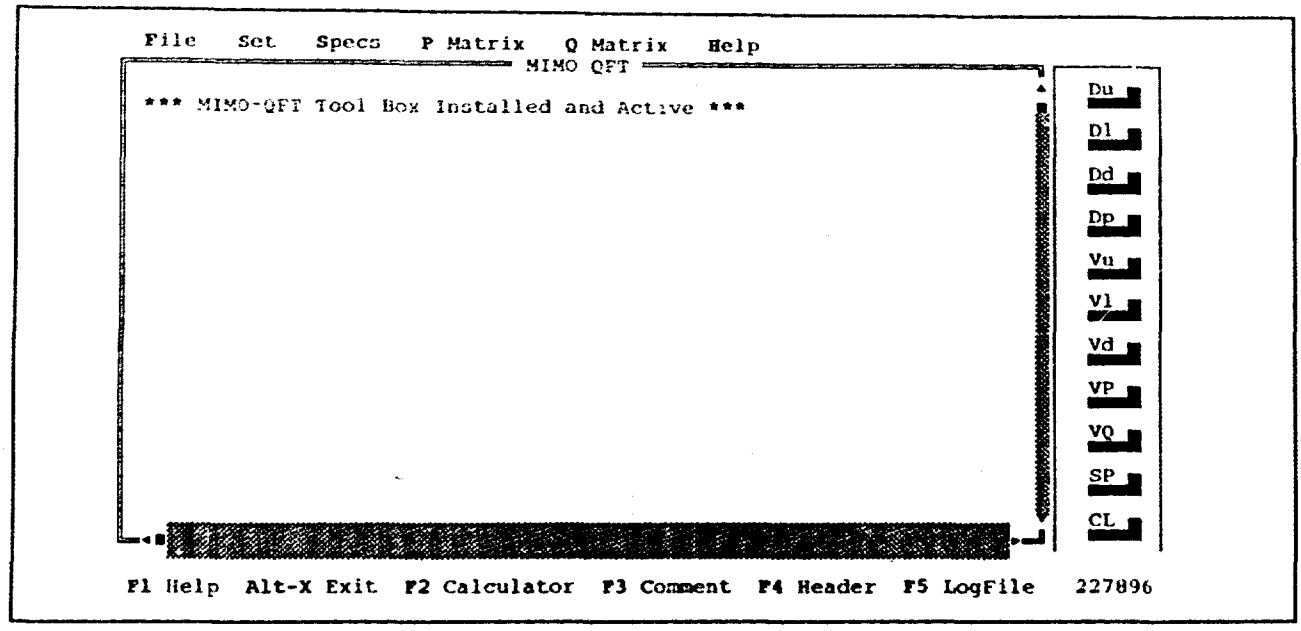


Fig. 5-1: MIMO QFT Toolbox Interface

The interface to the MIMO QFT toolbox is shown if figure 5-12. This interface resulted from several that were developed using fast prototyping, most of which were modifications of the MISO QFT menu structure. It is similar in function to the ICECAP-PC interface shown in figure 4-2. Operations involving the P matrix are under the P Matrix drop-down menu and operations involving the Q matrix are under the Q matrix drop-down menu. Stability specification operations are under the Specs menu. The buttons on the right of the screen perform the following actions:

Du - Define Tru

Dl - Define Trl

Dp - Define Plant p

Vu - View Tru

Vl - View Trl

Vd - View Td

Vp - View Plant p

VP - View Plant Matrix P

VO - View Options

SP - Select Current Plant

CL - Clear Screen

Use of the menus closely tracks the normal procedure for a MIMO QFT design process. However, being an event driven program, the user is not constrained to a pre-determined sequential set of menu choices. A short description of how the interface performs each task follows.

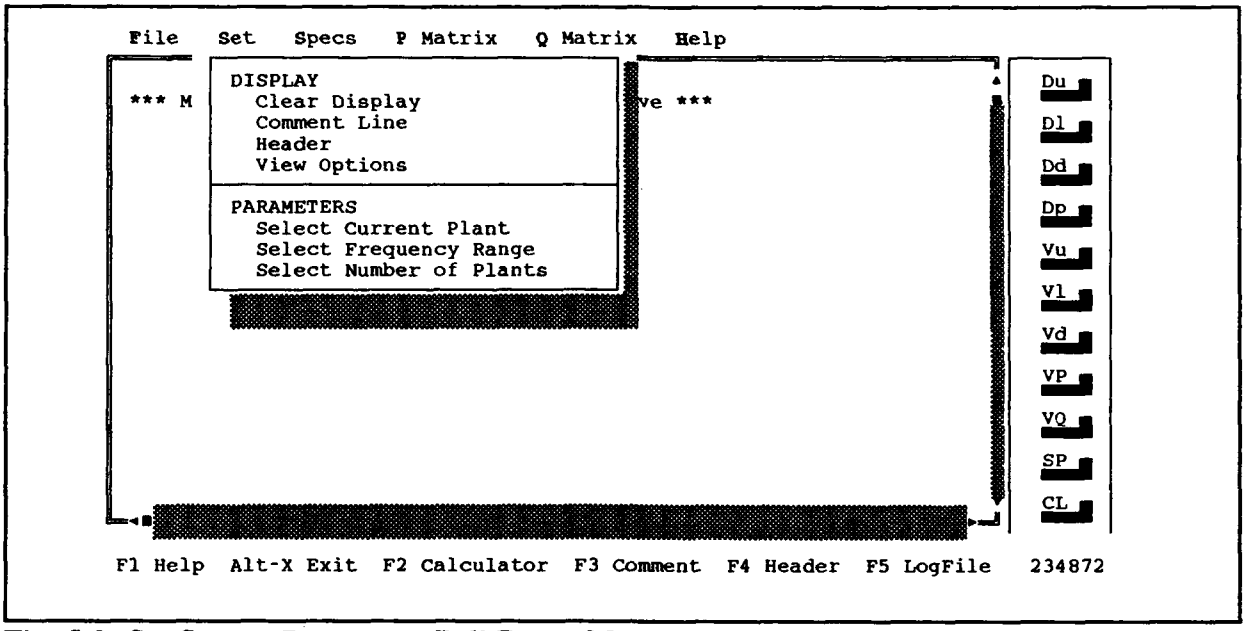


Fig. 5-2: Set System Parameters Pull Down Menu

Set System Parameters

The *Set* (Set System Parameters) menu provides the user a way to (1) set the display characteristics such as the view options and (2) set the system parameters. Three system parameters should be defined before the design process begins.

First the user can set the current plant from among 20 cases. The MIMO QFT Toolbox provides up to 20 three by three plant matrices that conceptually form a three dimensional set of plants 20 levels deep. The user is not constrained to three by three problems and can just as easily work two by two problems. Future work may expand the dimension to higher levels.

Second, the user should set the number of plant cases defined. This determines the number of templates to be developed. The user is not constrained to this choice after defining it. Analyzing additional plant merely entails the resetting of this value at any time in the design process.

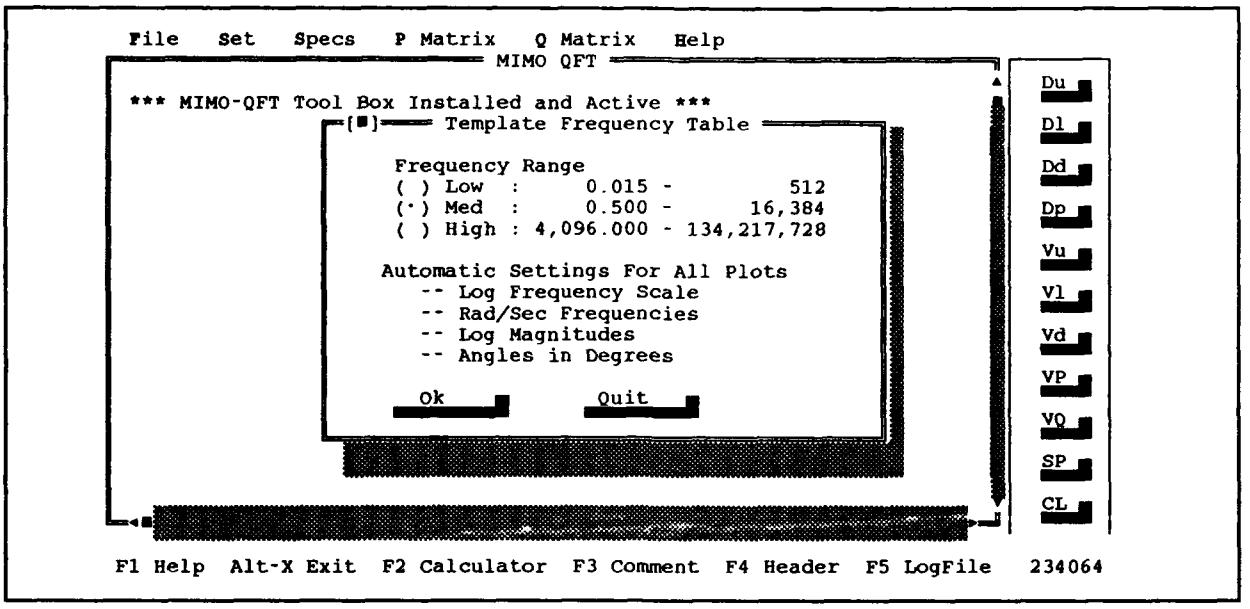


Fig. 5-3: Set Frequency Range Menu

Third the user can choose one of three frequency ranges for the development of plant templates and all frequency response calculations. In the low frequency range, 16 frequencies 1 octave apart from .015 - 512 Rad/Sec are defined. In the medium frequency range, 16 frequencies 1 octave apart from .5 - 16384 Rad/Sec are defined. In the high frequency range, 16 frequencies 1 octave apart from 4,096 - 134,217,728 Rad/Sec are defined. This frees the user from having to specify a set of frequencies for each plot and calculation performed during the design process. Selection of the frequency range is shown in figure 5-3.

Define Stability Specifications

The definition of the stability specs includes the entry of the transfer functions T_{ru}, T_{rl}, T_d and the stability margins (phase margin, gain margin, and peak overshoot). The menu to perform these actions is shown in figure 5-4. From this menu, the user can define, display or edit T_{ru}, T_{rl} or T_d. Stability margins are also specified from this menu.

The dialog box defining T_{ru} is shown in figure 5-5. The dialog boxes for T_{rl} and T_d are identical, however definition of Trl for off diagonal elements is not allowed. There are several things to note in figure 5-5. First, transfer function can be defined in one of two forms. Figure 5-5 shows a fictitious polynomial (coefficient) entry of 1 2 3; 4 5 6 7 to define $s^2+2s+3/4s^3+4s^2+6s+7$. Factored and root transfer function entry is

identical, however they are displayed differently. Also note that we have defined every single off diagonal Tru by the same transfer function. This is one of the great strengths of the ICECAP-PC MIMO QFT Toolbox. We can define any number of transfer function elements in the plant set with a single entry.

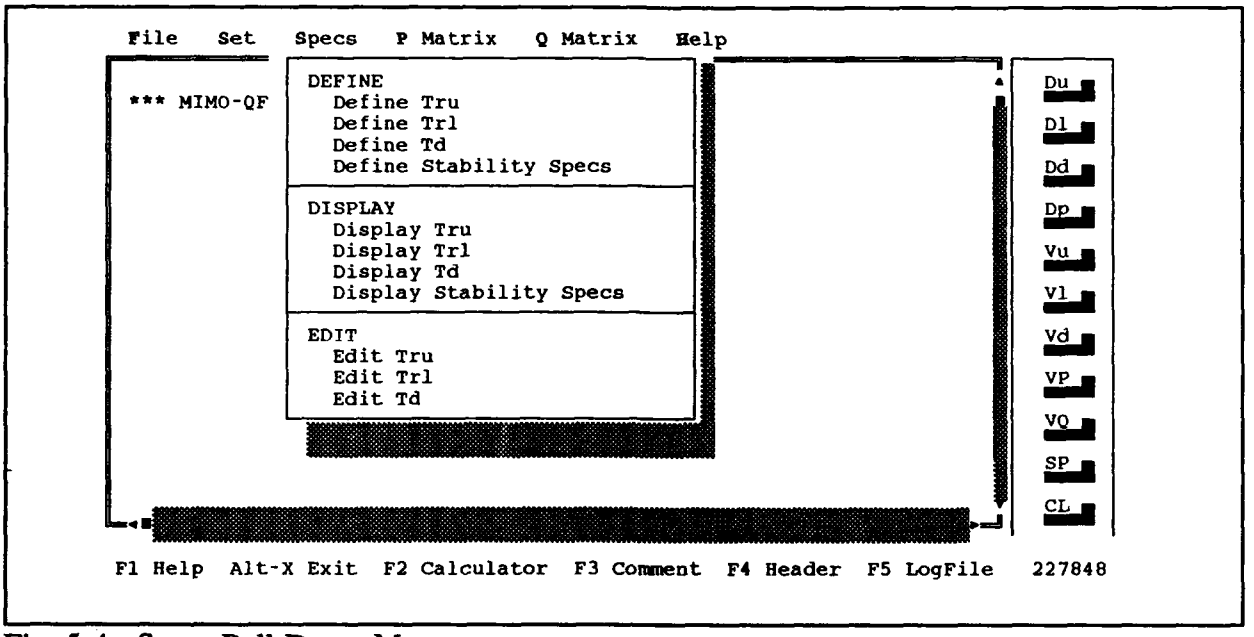


Fig. 5-4: Specs Pull-Down Menu

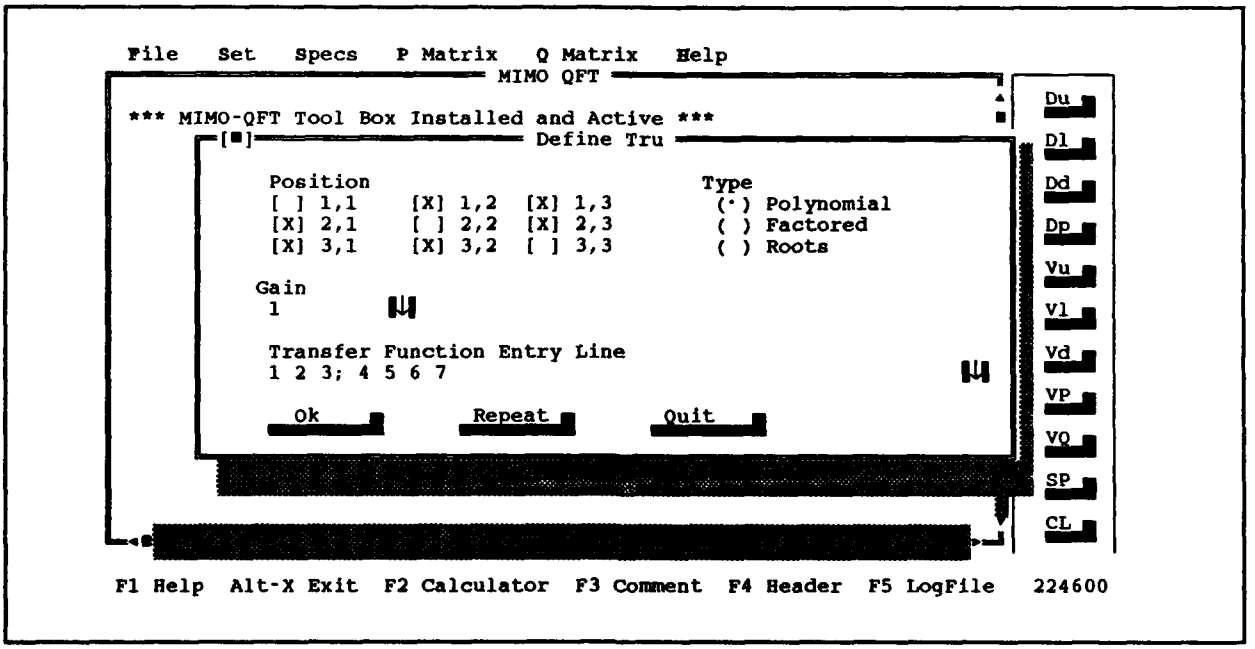


Fig. 5-5: Tru Definition Dialog Box

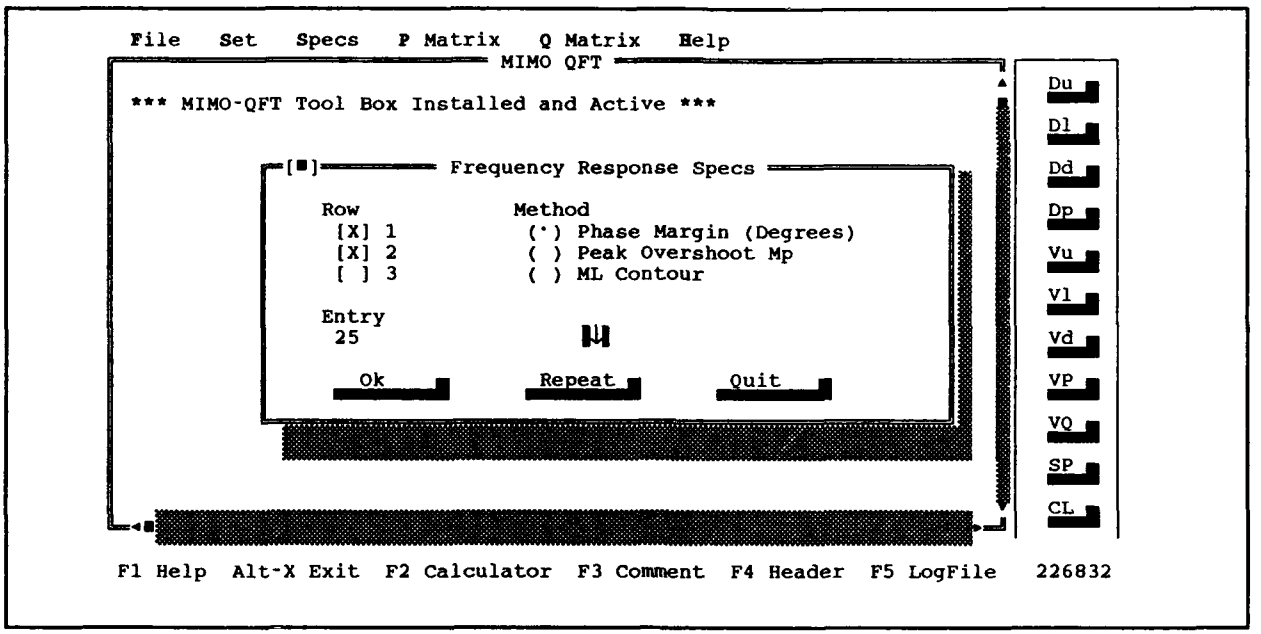


Fig. 5-6: Stability Margins Dialog Box

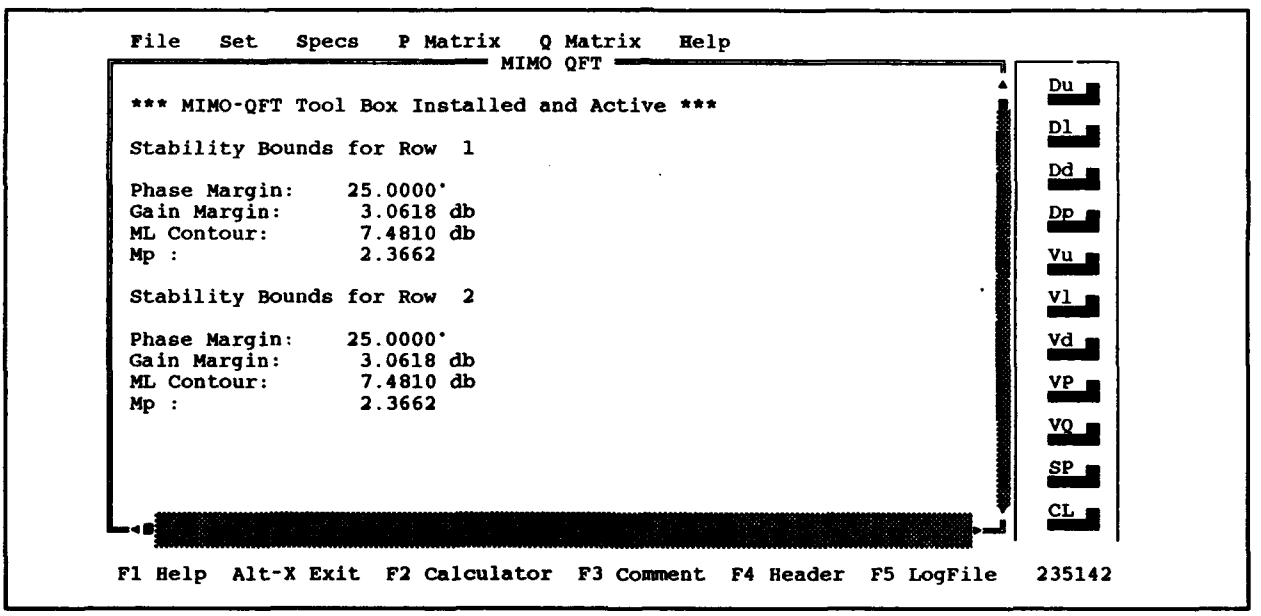


Fig. 5-7: Display of Stability Margins

Figures 5-6 and 5-7 show the definition and display of stability margins using eq. 2-23 through 2-26. Note from figure 5-6 that again we can define the stability margins for as many rows as we desire at a single time. The user can define the margins as either a phase margin, peak overshoot or Ml contour. The direct definition of gain margins is not allowed for two reasons. First, the calculation of the other margins given the

gain margin is quite awkward and prone to numerical error (see eq 2-23 - 2.26). Second, the sensitivity of the calculated error for the other three margins for a small error in gain margin is quite high. This is easily seen on any nichols chart. Note how any change in gain margin can produce a drastic change in the corresponding Ml contour and the accompanying phase margin. Figures 5-6 and 5-7 show the definition of a 25° phase margin for rows 1 and 2 of the plant set. Note that all data in the MIMO-QFT Toolbox is held from session to session and is static in nature. Therefore, exiting the toolbox doesn't erase the set of data built from previous sessions.

Define Plants

The definition of plant elements is identical to the process for defining Tru. Therefore, we shall not loiter over this subject. The only difference is that there are 20 matrices of plant instead of just one matrix of Tru, Trl, and Td. Plants are selected with the *Set-Current Plant* menu selection or the *SP* pushbutton.

Plant Checks

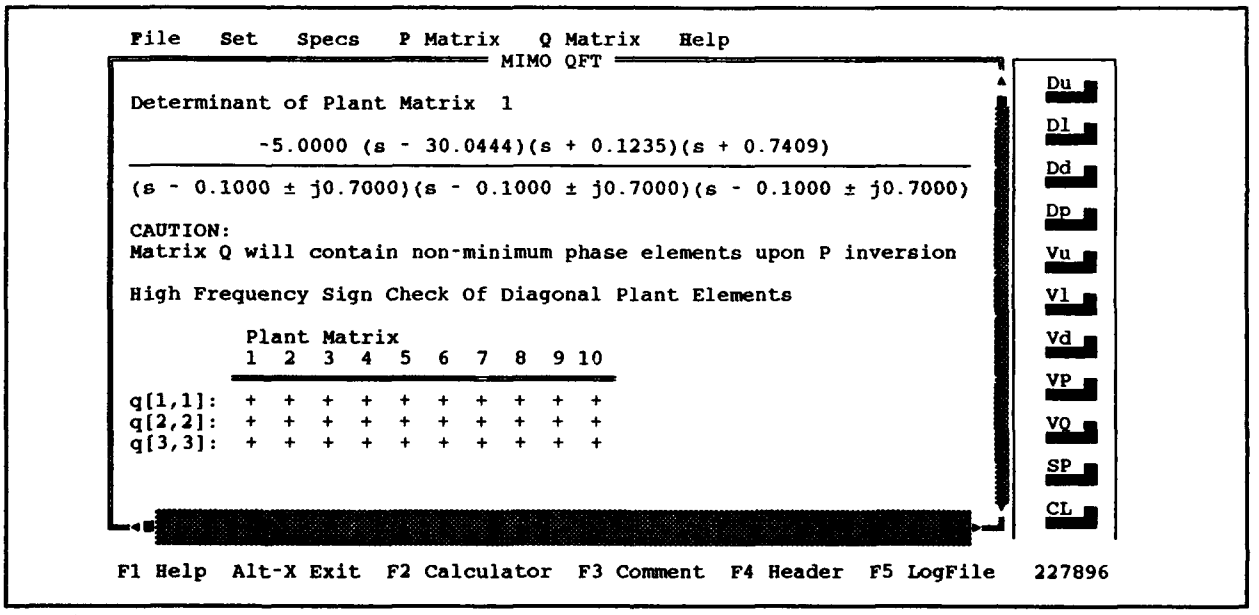


Fig. 5-8: Plant Determinant and HF Sign Check

After all plants have been defined, the designer normally performs two checks to ensure (1) the inverses of the plants exist (2) the Q matrix will contain non-minimum phase elements and (3) the sign of the determinant does not change over the entire set of plant cases. The MIMO QFT Toolbox allows these checks as displayed in figure 3-17. Note in figure 3-17 that (1) the P matrix is non-singular because of the existence of the determinant

(2) the Q matrix will contain non-minimum phase terms because the numerator of the determinant has a right half plane pole and (3) the sign of all diagonal elements (over the entire plant set) as w -> ∞ is, per application of L'Hospitals rule, all positive. The reader should understand that while this figure only shows the determinant of one plant in the entire set, it must in fact be taken for each of the 1-20 plants defined.

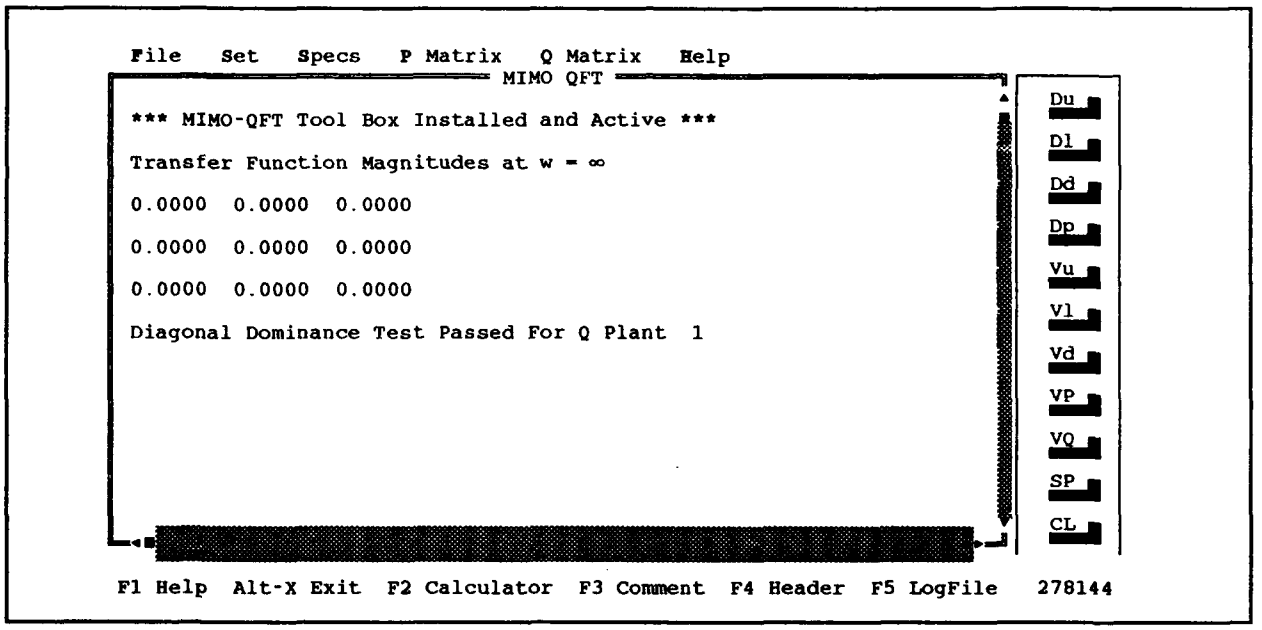


Fig. 5-9: Diagonal Dominance Check of Q Matrix Plant 1

Q Matrix Functions

All Q matrix functions are accessible under the Q Matrix menu. From this menu, the user is able to define, edit or view q matrix elements or an entire plant matrix at one time. In some design cases, the user may wish to define the Q matrix directly rather than inverting the P matrix. If the Q matrix is found by the standard procedure of inverting the P matrix, then the user should verify the inverse as poorly conditioned plants can always create numerical error. The Q Matrix menu provides an inversion check which multiplies the original P matrix by its inverse and displays an identity transfer function matrix if the inversion is validated. Additionally, the user may perform a diagonal dominance check as shown in figure 3-19. Again, the reader should understand that the diagonal dominance check is performed for all plant cases and is a pre-requisite of successful QFT design.

Once the plant set is defined, inverted and validated for QFT design, the user then exits the QFT toolbox and performs QFT design on each of the individual MISO loops. Currently this capability is provided in the ICECAP-PC MISO QFT toolbox.

Program Structure of the MIMO QFT Toolbox

As with any ICECAP-PC toolbox, the MIMO QFT is build on the object-oriented paradigm under Borland Turbo Vision. The toolbox object, *TMIMO*, is a child of *TView*, a predefined object type of Turbo Vision containing message transmission capability. *TMIMO* is defined in Listing 5-1.

The first thing likely noted about the *TMIMO* object is its relatively large size. This size causes some performance degradation on a PC, but unlike the original *TMatrix* and *TTransFunc* objects, it is not enough to warrant its restructuring into actors.

The pascal text file that contains this object definition, MIMO.PAS, also contains a menu definition and toolbar definition that replace the standard ICECAP-PC menu bar and toolbar as shown if figure 5-1. The menu bar and toolbars are also object classes themselves that are instantiated by the TMIMO.InitMenuBar and TMIMO.InitToolBar methods.

The *TMIMO* Data Structure

One of the fundamental questions confronting us at the outset of the project was whether or not it is even possible to do MIMO QFT design on a PC. The MIMO problem is inherently memory intensive because a system is described mathematically as a set of transfer function matrices, $\{P_{ij}\}$, as described previously. Could a personal computer, with its very limited memory configuration, handle several matrices whose elements themselves were transfer function? Could you perform matrix inversion, critical to the MIMO process, on such a matrix using a PC? Is a PC as accurate as other computers, say a Sun workstation?

The answer to the accuracy problem is based on the very definition of floating point numbers as discussed in Chapter 2. Yes, a PC is just as accurate as other computers because they are all using the same numbering system. However, the memory issue takes some careful design. The MIMO QFT Toolbox resolves the problem by using the disk drive as an extension of memory and placing all transfer functions not under operation in a set of files. All operations occur one transfer function at a time and memory usage at any given

```
PMIMO = ^TMIMO;
TMIMO = object(TView)
   {Variables}
   Freq_Array      : FreqArrayType;
   Freq_Range      : Integer;
   MIMO_FileNames  : MIMONamesType;
   MIMO_QFiles     : MIMONamesType;
   NumPlants       : integer;
   PlantDimension  : Integer;      {Dimension of square plants P & Q 2-3}
   StabilityBounds: StabilityBoundsType;
   CurrentPlant    : Word;         {Current plant P.  Used for plant entry  }
   ph_mar          : extended;     {phase margin spec of current MISO loop}
   mlm             : extended;     {Mp on U-Contour of current MISO loop }
   nom_plant       : integer;      {Nominal Plant Matrix P of MIMO system}

   {Administrative Routines}
   CONSTRUCTOR Init;
   PROCEDURE   Init_MIMOQFT_DataFile;
   PROCEDURE   Init_MIMOMenuBar;
   PROCEDURE   Init_MIMOToolBar;
   PROCEDURE   HandleEvent(var Event: TEvent); virtual;
   PROCEDURE   DisplayStabilityBounds (Row: Integer);
   PROCEDURE   DisplayUContour;

   {QFT Math Routines}
   FUNCTION    IsDominant(Plant, Dimension: integer): Boolean;
   PROCEDURE   TFdeterminant (PlantMatrix: word; Dimension: integer;
                              var Det: Transfunc; var AbortCode: Integer);
   PROCEDURE   InvertP (Plant: Integer; Dimension: Integer; DispFormat: Integer);
   PROCEDURE   PlantMultiply (Stor_Loc: integer; Dimension: integer;
                              Format: integer; var AbortCode: integer);

   {MIMO QFT User Interface Routines}
   PROCEDURE   ChooseCurrentPlant;
   PROCEDURE   ChooseFreqRange;
   PROCEDURE   ChooseNumOfPlants;
   PROCEDURE   CopyPlants;
   PROCEDURE   Determinant_Dialog;
   PROCEDURE   EditAPlant(Choice: integer; Stor_Loc: integer; Header: String);
   PROCEDURE   InvertP_Dialog;
   PROCEDURE   DefineStabilityBounds;
   PROCEDURE   DefineTF (Choice: Integer; Stor_Loc: Integer; Header: String);
   PROCEDURE   DefineTFGuts(NewTF:TFPtr; MakeData: MakeTFDataType;
                            Choice: Integer; TFNum: Integer; Header: String);
   PROCEDURE   DisplayTFMatrix (Choice: Integer; Header: String);
   PROCEDURE   DominanceDialog;
   PROCEDURE   HFSignCheck;
   PROCEDURE   VerifyInverse;
   PROCEDURE   ViewStabilityBounds;
   PROCEDURE   ViewTFGuts (Choice: Integer; Stor_Loc: Integer; Header: String);
end.
```

Listing 5-1: The MIMO QFT Toolbox Class Definition

instant is quite conservative. In the case of matrix inversion for the transfer function matrix, the computer only needs to load and use n^2 transfer functions, about 50K worth of data. This is still well within the capabilities of a PC.

The transfer function file set consists of 9 files for the P matrix and 9 files for the Q matrix; a total of 18 files. Each file contains the transfer function set for a given element in the plant matrix set. The P matrix files are named MIMO_XX.DAT where the XX is the element number. The transfer function set for element

[2,2] is located in MIMO_22.DAT. Currently, each file can hold a set of 20 transfer functions with future growth limited only by disk space. When the plant matrix P is inverted and reciprocated, the elements of Q are stored in the file set MIMOQ_XX.DAT. Additionally, the MIMOQ_XX.DAT file set contains the transfer functions describing the upper tracking response (Tru), lower tracking response(Trl), disturbance response(Td), and controller design transfer functions (FCTF, FTF, etc). The MIMOQ_XX.DAT files are compatible with the MISO data files and are used in the decomposed MISO solution of the MIMO system.

Conceptually, the user is presented with a 3x3 matrix of transfer function 20 levels deep in which to describe a set of plant instances spanning the region of plant parameter uncertainty. The objectives of the MIMO object is simply to allow the efficient description of the uncertain plant set, validate its usability and realizability, perform matrix inversion to form the Q matrix (the inverse of P), verify the inversion of Q, and test for diagonal dominance. The MIMO QFT object does not directly provide tools to perform loop shaping, bound generation, or prefilter design. These are provided with the MISO QFT toolbox for each transfer function element. Additionally, the MIMO QFT toolbox does not directly contain any graphics capability. Again, this is done through the MISO toolbox.

TMIMO Methods

The *TMIMO* object contains several unique and interesting routines including IsDominant, TFDeterminant, InvertP, HFSignCheck, and VerifyInverse. We now turn our attention to discussing each of these methods.

After the user fully defines a plant set spanning the region of plant parameter uncertainty to their satisfaction, they must check each plant case (a single transfer function matrix) to see 1) if the plant is singular having no inverse and 2) if the inverse will contain minimum phase elements (transfer function elements with numerator factors in the right half plane of the s domain). This is done by taking the determinant of the plant. The existence of a determinant proves non-singularity and the lack of positive zeros in the numerator of the determinant proves the inverse will be non-minimum phase. Therefore, prior to inverting the P matrix, the designer must calculate the determinant of the plant. The *TMIMO*.TFDeterminat method provides the mathematical engine to find this determinant using LU Decomposition. ***The ICECAP-PC Toolbox is unique***

among emerging QFT programs in its use of LU Decomposition for the calculation of the determinant and the inversion of the transfer function matrix P. Typically, MIMO QFT students are taught to find the determinant by using the cofactor method with the admission that such determinant is of very high order and root finding is quite difficult. Using the cofactor method, root cancellation cannot take place until the entire determinant is formed. LU Decomposition has the inherent advantage that root cancellation is performed on an element by element basis and the growth of an artificially large order polynomial with common numerator and denominator factors is eliminated. The reader is encouraged to work an example by hand to see this process take place. As a consequence, it is much faster, more accurate, and more stable than cofactor calculation. Furthermore, in contrast to the cofactor method, there is no need to find a common denominator among transfer function elements and then divide out the common denominator before finding the inverse (Houpis, Pg E-1). Again, common denominator elements cancel out on an element by element basis. Upon completion of determinant calculation, the determinant is displayed with a message declaring the passage or failure of the non-minimum phase test in the format shown in figure 3-17. Note that transfer function display can be in coefficient, factored or roots only form.

After inspection of determinants for the 1 to 20 plant cases, a check is normally made of the sign of each diagonal transfer function element at high frequencies ($w \rightarrow \infty$). The check ensures that for a given diagonal transfer function element set $\{p_{ii}\}$, the sign of the element does not change as $w \rightarrow \infty$. ICECAP-PC implements this by using a symbolic manipulation of L'Hospitals rule. Transfer functions are reduced to the form K/S^n or S^n/K where K is a constant and S^n is a polynomial or order n. Physically realizable transfer functions will reduce to the form K/S^n. The reduced transfer function are not displayed in this case. Rather, only their sign is given as shown in figure 3-17. Figure 3-17 shows the calculation of only 10 plant cases; however, up to 20 plants can be analyzed at one time this way.

After the above validations are made, the user normally inverts the plant matrices (P matrices) to form the Q matrices. In some cases, the Q matrices will be known directly. The MIMO QFT Toolbox allows for the direct entry of Q in such cases. Plant matrix inversion is the function of the InvertP method. This function drives the LU Decomposition and Forward/Back Substitution processes to find the plant inverse. Standard LU

Decomposition and re-substitution involves two processes known as pivoting a scaling discussed in section II. Very simply, in order to preserve stability during the inversion of a real valued matrix, the LU Decomposition algorithm searches downward from the main diagonal to find the row with the largest value in the column under operation. It then swaps the current row with the found row. This process is known as pivoting and prevents numerical error by placing larger values in high locations. Scaling involves the row-wise search for the smallest value and the division of all elements in a row by this value. The pivots are stored as a series of permutations in a column vector and deciphered by the re-substitution algorithm. The question we encountered is how do we implement pivoting and scaling when the matrix contains transfer function elements? Do we look for the largest gain or the largest gain at a desired frequency? In ICECAP-PC, we simply took our best shot and implemented scaling based on the value of the normalized gains of the transfer function elements. It seems to work well and future students may wish to look into this further.

One of the last steps performed in the MIMO QFT Toolbox is the diagonal dominance check on the Q matrix. Diagonal dominance is a prerequisite for successful QFT design. For a three by three matrix, diagonal dominance is satisfied if as $w \to \infty$

$$\left|p^*_{123}\right| \geq \left|p^*_{12}\right|\left(\left|p^*_{21}p^*_{33}\right| + \left|p^*_{23}p^*_{31}\right|\right) + \left|p^*_{13}\right|\left(\left|p^*_{22}p^*_{31}\right| + \left|p^*_{21}p^*_{32}\right|\right) + \left|p^*_{11}p^*_{23}p^*_{32}\right| \qquad \text{Eq. 5-1}$$

In order to implement this check, ICECAP-PC uses the symbolic manipulation of L^Hospitals rule for each transfer function element in a given Q matrix. It then checks the sign of the gain of the manipulated functions and builds the chart shown in figure 3-18. In building this table, it applies eq 3-17 to check the passage or failure of the diagonal dominance condition.

As a final note, we pause to compare the ICECAP-PC MIMO-QFT Toolbox with two other programs currently available. The first was developed by Prof. Oded Yaniv of Tel Aviv University, the second by Mr Richard Sating of the Air Force Institute of Technology. Prof. Yaniv's program is designed to develop MISO QFT systems only and is written in Clipper with links to C subroutines. It is a very functional program with excellent graphics and is written for MS-DOS compatible computers. Mr Sating's MIMO QFT program is the very first MIMO QFT program ever written and is also well written. Sating's MIMO QFT program is written in

Mathematica's rich programming language and is specific to Sun Sparc Stations. Thus it remains unavailable to a large engineering population. Program requirements are a Sun Workstation, Mathematica, and $Matrix_X$, the latter providing the graphics engine, the former providing the mathematical engine. The ICECAP-PC toolbox cannot boast the functionality of the previous two programs because the purpose of its development was not a full scale MIMO QFT design program but rather a feasibility proof that MIMO QFT could be accomplished in an MS-DOS compatible platform. The ICECAP-PC MIMO QFT Toolbox allows for the definition of stability specifications, the definition of plant matrices, the inversion of matrix P to form matrix Q and the performance of all validity checks. It provides files compatible with the MISO QFT Toolbox. The ICECAP-PC MIMO QFT Toolbox is the first object-oriented MIMO QFT program ever written and the first to use LU Decomposition for the inversion of matrix P. Further development of the toolbox will provide full functionality.

5.5 Summary

In our discussion of the MIMO QFT Toolbox, we discussed the nature of ICECAP-PC toolboxes in general, discussed the interface of the MIMO QFT Toolbox and finally discussed the mathematical algorithms used. The toolbox concept lends easy integration of other control system design theories into the ICECAP-PC program and future students should use the MIMO QFT Toolbox as a model. The many interface examples are given with the hope that the reader takes interest in the new ICECAP format and object-oriented programming for engineering solutions in general. Finally, we discussed the numerical algorithms used for the MIMO QFT engine. Particularly important is our use of LU Decomposition for the inversion of the transfer function plant matrix P. This, to our knowledge, has never been done successfully before, at least not in the halls of QFT proponents.

Work performed during this research on this toolbox to date does two things. First, it proves that MIMO QFT can indeed be implemented on a PC and proposes that there is a tremendous need for such a package. Second, it lays the foundation, framework, plumbing and exterior cover to a promising development for future AFIT students. Remaining is the finishing work of adding graphics capabilities and interactive design in the MIMO toolbox. Many of the tools are already available in the MISO Toolbox and need only to be ported to the MIMO Toolbox.

6 Testing and Validation

6.1 Introduction

This section covers the testing and validation of numerical algorithms used in ICECAP-PC as well as the correct operation of the user interface. It does not contain code certification tests because the Turbo-Pascal integrated environment contains several excellent debugging tools including the ability to allocate and de-allocate break points on the fly, the ability to monitor and change variable values during program execution, and a *watch window* where the software engineer can watch a complete range of variables while single stepping through the code. Virtually all code used in this project was validated using this environment and very little was left in the way of script files, text reports, etc.

We did perform extensive testing on algorithms to ensure the best possible numerical accuracy. The results of several tests on numerical algorithms are contained in Appendix B. It was impossible to include all these tests; therefore, this section contains a representative sample.

6.2 Black Box Testing

Black box testing treats a program as a black box that accepts an input, operates on it, and generates an output. Using a large test set of previously solved exemplars, a program can be verified to work over a large range of problems. Text books proved to be a source for examples, but not the best source for examples. Text books tend to have problems worked on dated software packages that may not have the numerical precision of ICECAP-PC. A better source tends to be MATRIXx and MATLAB. The best source, of course, are problems worked by hand.

An important group of black box problems is to use zero transfer functions or zero matrices, etc. These examples often uncover divide by zero errors and floating point overflows.

Another important group of exemplars were those for the transfer function, polynomial, or matrix definition lines. Different users input the item definition in different ways. Spaces are put in different places, commas are used instead of spaces, complex conjugate pairs may both be entered or not, the complex letter could be put either before or after the sign of the complex value (i.e., 2 -j2 or 2 j-2), etc. The effort was made

to keep as few rules as possible placed on the user. To do this, there was required a large number of exemplars to try to cover every forseeable type of user input.

6.3 Macro File Object

The purpose of the macro files is to store all of the black box example pages for easy rerun after any significant change to the ICECAP-PC package. The same examples can be used for tutorial purposes by new users to watch the ICECAP-PC commands in execution. The macro file object operates on a simple principle: it replicates the keystrokes the user would use. The macro files consist of a single keystroke per line. The macro object simply reads the line and enters the keystroke directly into ICECAP-PC's GetEvent method simulating the human keyboard input. There are also special commands for inserting comments and pauses and end of file.

Later work could easily add a macro recording function to allow users to record keystroke sequences into a macro file for later rerun. This would be easy to implement by just reversing the code in the macro object.

The difficulty of creating the macro object was in determining what event codes were generated when the user pressed a key. The Event.What field is equal to evKeyDown. The Event.KeyCode field is equal to a HEX value unique for each character. The Event.CharCode is equal to the ASCII character for the key. The Event.ScanCode is equal to a HEX value unique for each key. ALT and CTRL key combinations follow the same rules. Some special keys like the ENTER or ALT key combinations also have unique settings for the Event.Buttons field and the Event.Command field.

If not for the IDE, it would have been impractical to trap these events during runtime and examine their content. A single keystroke can generate several separate events (updating screen colors, updating what is displayed, internal flags, etc), so several conditions must be checked on the trap you set. However, once every possible keystroke was trapped and the event record examined, it was easy to recreate the keystroke from within a macro file.

6.4 White Box Testing and the Integrated Development Environment

White box testing is like black box testing, but it does not ignore the internal code implementation of the algorithm. To do white box testing, an example must be constructed to test every decision point in the code.

If it were not for the IDE, this task would have been insurmountable to test a package as large as ICECAP-PC. However, with the IDE each routine could be single stepped through while watching a window showing all the variables. This is the single greatest contributor to the reliability of the new ICECAP-PC code. "Playing computer" by watching the state transitions of the data allowed us to not only completely debug our code but also to create better algorithms more suitable to a computer implementation. By watching all the variables, you become aware of which very large numbers are being added to very small ones, which variables are being divided by variables very close to zero, etc.

The highest contributor to the unreliability of the old ICECAP-PC code was that it used too much memory to run within the IDE. Its only white box testing was accomplished using print statements inside the code to dump variable values now and then. The IDE is infinitely more powerful than this antiquated method of white box testing. Under this environment, the software engineer can develop a code module, specify a watch window to monitor control variables, and then single step through the code while watching the specified variables. Further capabilities provided by the IDE are the ability to set a break point and execute the code up to that point, the ability to change control variable values on the fly during an interrupt in program execution, and the ability to directly monitor the stack and available memory. The result is a much more efficient and effective debugging process than can be achieved using classical methods. ***We've caught several types of program errors that could not be detected using standard methods. Many of these errors were not the result of the program but of the compiler itself!*** We must stress that virtually all compilers have internal bugs and a big lesson learned is that even perfect code can fail due to compiler error. A synopsis of a few of these follows:

A Little Something is Always Wrong

Many times, after we developed the most beautiful and obviously perfect code module and even tested it with several test cases, we would catch little errors in control variables when we simply single stepped through the procedure. Yes it worked right but for the wrong reason!. The problem with these kinds of errors is that sooner or later the procedure is doomed to fail because somebody somewhere came up with a case you didn't plan on! We cannot over stress the importance of single stepping through every single procedure to validate that it works for the right reason and that ALL control variables operate properly.

Incorrect Evaluation Of Boolean Operators

Another problem that caused us no end of grief was the occasional incorrect evaluation of conditional tests. When this occurred, it was a constant problem and easy to find under the IDE. Nevertheless it was at times frustrating to evaluate something as simple as "if 1 < 2 then..." and find the evaluation to be false. The remedy was often to place dummy code such as i := 1; j := 2; if (i < j) or (j > i) then...". Again, this type of error would take days or even weeks to catch using classical debugging techniques because the software engineer would rightfully pull all the hair out of his/her head while insisting that there was nothing wrong with their code!

Parameters Passed Incorrectly

One of the most grievous compiler errors we found was the occasional, unpredictable incorrect passing of a extended parameter. Most often this occurred when the parameter passed was input as 0.0 and taken back out as 1×10^{-4932}. The very next mathematical operation on this number is sure to produce an invalid floating point operation (error 207). The remedy was often the rearrangement of the parameter set into a different order. However, we found that often when a value of 0.0 is retrieved from a disk file, it becomes 1×10^{-4932} in the process! Obviously we needed a way to check all values for a very small number and set them to identically 0.0 if they were below a specified threshold. This was just the beginning of tribulation.

Intel Microprocessors, 0.0, and Excedrin Headaches

The most difficult number to work with in Turbo Pascal is 0.0. The problem is that Turbo Pascal incorrectly evaluates some functions at 0.0 AND that different Intel microprocessors handle 0.0 differently. While an 80286 microprocessor with an 8087 co-processor passes 0.0 as 0.0, an 80386 and 80486 passes 0.0 as 1×10^{-4932}. The result is often unpredictable, untraceable, spontaneous, unrepeatable chaos!

Naturally, one expects that if we compare a very small number, say 1×10^{-4932} with 0.0 we will cause an invalid floating point operation (error 207). Therefore, we defined a global variable named *Zero* and set it equal to 1×10^{-100}. The objective is that if a number was less than *Zero*, we would set it exactly equal to 0.0 and prevent invalid floating point operations (error 207). It was a good idea. However, if the parameter is passed incorrectly as discussed above, a 0.0 became 1×10^{-4932} and when this is compared to 1×10^{-100} the computer fails anyway. Yes, if fails with an error 207.

We decided to get smart about this. If we compared the exponents rather than the numbers themselves we could detect a small number before ever comparing two very small numbers--even if the parameter was passed incorrectly. Based on this, we developed a special function called IsZero as shown in listing 7-1.

```
FUNCTION IsZero(A: extended): boolean;
begin
   IsZero := True;
      If trunc(log10(abs(A))) <= trunc(log10(abs(Zero))) then
         IsZero := true
      else
         IsZero := false;
   end;
end;
```

Listing 6-1: First Futile Effort at IsZero

In this procedure (*we took great pride in trunc(log10(abs()))*) the values of the exponents are compared. If the exponent of A was less than or equal to the exponent of *Zero* then the function returned as true. The problem here was the ABS call. The ABS function compares the original value of A to guess what: 0.0! The result is again computer crashage with an invalid floating point operation (error 207). We hope the reader hates seeing error 207 as much as we did!

```
FUNCTION IsZero(A: extended): boolean;
var
   T : extended;
begin
   IsZero := True;
   If (A <> 0.0) then begin
      T := trunc(log10(abs(A)));
      If T < trunc(log10(Zero)) then
         IsZero := true
      else
         IsZero := false;
   end;
end;
```

Listing 6-2: The Final Version of IsZero

Once again, we huddled and attempted to replace the ABS call with an ABSOL call of our making. However, this effort was fruitless. Finally, we came up with a modified solution shown in listing 7-2. The modifications first tested A to see if it is exactly 0.0. Then we declare a dummy variable T and set it equal to the trunc(log10(abs(A))). Then we compare T with the trunc(log10(abs(Zero))). Note this comparison cannot

take place properly without the dummy variable because of the problem of occasional incorrect boolean evaluation discussed above. This seems to be the solution to fix our problems! We hope so because we are out of ideas!

The 19th Decimal

Another problem that can often compound other problems is that if the 19th decimal position of an extended number is 6, any mathematical operation on that number will fail with an invalid floating point operation (error 207). Now consider the problem of developing a Log_{10} routine needed to fix the 0.0 problem listed above. Turbo Pascal does not have a build in $Log_{10}()$ function so we had to make our own. Turbo Pascal does have a built in Ln() function so we proceed as shown in eq 7-1 (Kreyzsig, pg A-52).

$$Log_{10}(x) = .43429\ 44819\ 03251\ 92765\ 11289\ 18917 * Ln(x) \qquad \text{Eq. 6-1}$$

Now look at the 19th decimal number. Lo and behold, it's a 6! We found numerous errors caused by our initial Log 10 function because of this. In fact, all we had to do was define a number as this constant and the program would fail with an invalid floating point operation (error 207).

```
FUNCTION Log10(x:extended): extended;
{Original Icecap code replaced using Log(X) = Log(e) * Ln(X)                }
{Log(e) = 43429 44819 03251 82765 11289 18917                               }
{Kreysig, Advanced Engineering Mathematics, 5th Ed, Pg A-52                 }
begin
   If X <= 0 then
      begin
         MessageBox(^C'Error: Log(Negative or Zero) Not Defined', nil, mfError+mfOKButton);
         Log10 := 0;
      end
   else
      Log10 := 0.4342944819032518277 * Ln(X);
end;
```

Listing 6-3: The Log_{10} Function

Note that the Log_{10} function is also used in the IsZero() function call. We remedied this by rounding the constant as shown in listing 7-3 so that we didn't have a 6 in the 19th decimal position. However, engineer beware, any other extended number with a 6 in the 19th bit position that occurs anywhere in ICECAP-PC may well cause an unexplained failure!

Incorrect Long Integer Division

On one occasion we found an incorrect long integer division. It only affected one isolated procedure and did not cause the kind of system wide problem noted above. However, we found that if you have a long integer x and perform (x * 65536)/2 you will get an invalid floating point operation (error 207).

This set of compiler errors is given for a simple reason. We want to demonstrate that certain classes of errors occur even in perfectly coded programs. These kinds of errors can be impossible to trace using classical debugging techniques that assume any mistakes to be the software engineers. Use of the IDE allowed us to find these mistakes, develop some corrective action, and proceed with the coding process. Undoubtedly, previous versions of ICECAP-PC suffered from these kinds of errors that were left undetected.

6.5 User Requests

Feedback from a large student user group has provided continued insight into how to improve the user interface and required functions. The complaints and frustrations along with the compliments have been used to adjust the user interface and scope of functions. The student group includes all the controls students who use ICECAP-PC to do their homework.

6.6 Other Validation Tools

While the core of software validation consisted of the simple single stepping through each module, there were some instances, i.e. numerical algorithms, that demanded different testing schemes. Here we were testing not the operation of the code but the viability of a specified numerical algorithm. This was often done with inserted debugging code to allow the observation of numerical variables rather than control variables. Appendix B lists a representative sample of these tests.

For each given algorithm, we developed several test cases that we felt reasonably tested it. In the case of the Parseline procedure discussed in chapter 4 (sec 4-6, pg 17), we developed approx. 50 different ways to

input a matrix to exercise the algorithm. For the eigenvalue algorithm, we developed a Matlab test bench program to aid in the development of QR Shift for the general complex matrix. In the case of matrix operations, we pulled several matrices with known solutions from textbooks to test the algorithm. Each algorithm was tested against a known solution set and against other programs. We compared ICECAP-PC's root finder to Matlab's to find that ICECAP-PC consistently is 3-4 decimal places more accurate than Matlab! Furthermore, we handle the case of repeated real roots much more accurately. We placed it in the ICECAP-PC code only after we had tested each algorithm to our satisfaction.

The Root Finder

Appendix B shows listings of our tests comparing ICECAP-PC's root finder with that of PC-Matlab. In all cases we exceed the accuracy attained in Matlab, often by several decimal places. However, in the process we noticed something that forced us to modify our algorithm. The ICECAP-PC root finder first calls the Laguerre method [ref #] to make the initial cut at the roots. The Laguerre method returns a set of values whose error with the actual roots is symmetrical with the actual roots in the case of root multiplicity. Hence coefficient reconstruction is extremely accurate after root finding with the Laguerre algorithm. The second step used by the ICECAP-PC root finder is the polishing of the rough roots with either the Bairstow method [ref #] in the case of real pairs or with Brents method [ref #] in the case of single real roots or high root multiplicity. We were able to achieve extremely accurate roots with these two polishing methods. Improvements of 4-5 decimal places over Laguerre's method was common. However this added accuracy came at a loss of error symmetry about the actual values. ***Polynomial coefficient reconstruction is often degraded with the better roots found by the polishing methods!*** We finally modified our algorithm as follows.

We modified the root finder to take the first cut using Laguerre, store the Laguerre roots, take the second cut using Bairstow and Brent, and store the polished roots, reconstruct the polynomial coefficients with the Laguerre roots and find a reconstruction error, reconstruct the polynomial coefficients with the polished roots and find the polished reconstruction error, and compare the two errors taking the root set yielding the least reconstruction error. Very often ICECAP-PC settles on the less accurate set of roots because the coefficient reconstruction error is minimized.

6.7 Summary

The predominant software validation tool for this project was the IDE itself. The IDE provides all necessary tools to perform in-depth debugging and catch both logical error in the ICECAP-PC code and errors generated by the compiler itself. The testing of numerical algorithms, however, still proceeded along more conventional lines, especially during the algorithm development phase. A set of these tests are included as appendix B.

7 Conclusions and Recommendations

7.1 Conclusions

The purpose of this research project was the development of a CACSD (computer aided control system design) program to meet the needs of an ever more sophisticated educational system. In a day when computer analysis and simulation plays an increasingly important role, control systems engineers need a solid grasp of the computer sciences. This project exercises the disciplines required by todays control systems engineer: mathematically rigorous control systems engineering, numerical analysis proficiency, advanced object-oriented programming skills and human interface engineering.

Fundamental control systems engineering is, of course, the heart of the ICECAP-PC program. While engineers of other disciplines will undoubtedly find utility in the basic ICECAP-PC program, they are in fact directed at the control systems engineer. Both classical and modern control capabilities are provided in the basic ICECAP program. Furthermore, the advanced Quantitative Feedback Theory is fully implemented in both its MISO (Multiple Input Single Output) and MISO (Multiple Input Multiple Output) incarnations. The addition of interactive bounds generation and interactive L_0 development provides an ideal education platform for QFT. Of particular importance for this research was the development of plant matrix inversion using LU Decomposition, a method that minimizes numerical error and provides root cancellation at low polynomial orders where it is most effective.

Numerical analysis, an art often overlooked by line engineers, was fundamental to the success of this project. Virtually every numerical algorithm in ICECAP-PC underwent some form of modification or outright replacement. In the case of simple binomial operations on matrices, polynomials and transfer functions, we added the concept of numerical conditioning to prevent numerical errors at the fringes of the floating point definition. In the case of core algorithms, we put in a new engine. We replace cofactor manipulation with LU Decomposition for finding the determinant of a matrix. We replaced the Adjoint/Determinant calculation with LU Decomposition and re-substitution for matrix inversion. We added matrix condition number calculation which was absent in prior versions. We replaced root finding on the characteristic polynomial with QR Shift for eigenvalue computation. We replaced an inefficient root finder with a greatly improved version that utilizes the

Laguerre method, the Bairstow method, and Brent's method to find the roots to the floating point accuracy of the machine. Additionally, we have specific recommendations for further improvement on the root finder. Again, the high point of this project was the inversion of a matrix of transfer functions using LU Decomposition.

Object-oriented development and programming, which involves not only a new code structure but an entirely new logic and modeling process, is the software development basis of the future. Object oriented decomposition provides software reusability, extendibility, and maintenance in a way functional decomposition never could. This is why virtually all new commercial operating system development is taking place using object orientation. The engineering disciplines are late comers to object-orientation and none of the more respected programs are built on this paradigm as of yet. In this regard, ICECAP-PC is now a pioneer in the area of placing a CACSD package in an object oriented format. Certainly, this should draw much attention.

At this point, we must emphasize that excellence in all of the above disciplines mean nothing if the human interface is ignored. In todays business climate, intuitiveness is key rather than superfluous to software design. An intuitive program is quickly learned saving expensive engineering time for the actual design process. The adage that "the utility of a program is inversely proportional to its interface" is absolute nonsense. There are many excellent object-oriented interface development tools that remove much of the burden of interface development such as mouse support, drop down menus, etc. ICECAP-PC uses one such tool, the Borland Turbo Vision language extension.

The result of exercising the above disciplines is a CACSD program that challenges the state of the art in CACSD program design in form, in use, and in utility. Furthermore, ICECAP-PC specifically addresses the educational needs of engineering students by providing several features not available in other packages. First, ICECAP-PC has a context sensitive help system not only for program operation, but for control theory as well. The design guidelines and insights of AFIT professors could be easily incorporated into the ICECAP-PC help system and indeed should be. This could be of more value than the program itself!. Second, the interface of ICECAP-PC is quickly learned and students, who are notoriously short of time, can quickly grasp the concepts of program operation and immediately proceed to control system design. Third, ICECAP-PC provides an

educationally formatted log file producing well formatted, standardized reports for homework turn in. With these qualities, ICECAP-PC makes an ideal platform to "spread the gospel" of the Quantitative Feedback Theory.

There is still much work to be done. We didn't provide conversion between the s, z and w' domains. There is still work to be done restructuring the MISO QFT toolbox into actors and adding graphical capability to the MIMO QFT toolbox. There is room for further toolbox development in the ares of LQR, LQG and H∞. However, we laid complete, cohesive foundation paving the way for future development of ICECAP-PC.

7.2 Recommendations

A Scientific Calculator

ICECAP-PC includes a basic calculator courtesy of Borland Turbo Vision. This is a basic calculator capable of addition, subtraction, multiplication, and division. The control systems engineer could use an extended scientific calculator tailor made to control systems. This should include trigonometric functions, exponential functions, and control calculations such as zeta for a given root location, etc. Improving the calculator makes an excellent first step into object-orientation and Turbo Vision for future ICECAP-PC programmers and would be a worthwhile project for a student in the first half of their AFIT studies.

Multiple Windows

From the outset of this project, we considered having multiple display windows, one for each domain (s, z, w, and w'). Each window, being a different color and displaying data specific to their own domain would provide an interesting and intuitive interface. We didn't implement this because we didn't implement the different domains. Using object-orientation, this becomes an easy task. One merely creates multiple instances of the same window. No modification of the window object is required.

Use of Actors

The last major breakthrough for this project was the use of actors, very small objects that performed one specific mathematical function. This was a break from classical object-oriented logic that views a data item such an a polynomial as an object that can do all of its own manipulations. Large objects are impractical and cumbersome on a PC. However, there is virtually no limit to the number of very small specialized objects one can declare. Using actors, we were able to make ICECAP-PC very fast and nimble. All future ICECAP-PC

development should follow this pattern. In fact, there still is much work to be done converting the current object-oriented code into an actor format. Currently, only the matrix, polynomial, and transfer function families are in this structure. One word of caution: The order of units in the uses clause of the main program effects the success of this structure. This contradicts the Borland programming manuals. If the order of the units clause is wrong, the actors will cause a constant hard disk access that is both annoying and very slow.

The Root Finder

The current root finder, crucial to MIMO QFT, reaches performs at the level of the floating point resolution of the machine. This isn't good enough. ICECAP should guarantee root finding to 20 significant decimal digits. In order to do this, calculations must be performed with far greater resolution, say 100-200 decimal digits. None of the IEEE 754 floating point number definitions provide for this. Therefore, the root finder must declare its own data structure and operate this way. Defining a new data structure for a number entails defining all the base mathematical routines such as addition, subtraction, etc. Several options are available and should be studies. Consider the following definitions

```
Number = Record
   Integerpart: extended;
   Decimalpart: extended;
end;

Number = Record
   IntegerPart: Array[1..1000] of integer;
   DecimalPart: Array[1..200] of integer;
end;

Number = Record
   UpperIntegerPart: String;
   LowerIntegerPart: String;
   DecimalPart:      String;
end;
```

Listing 7-1: Possible Numerical Definitions

In the first example, a number is defined as a record of two extended numbers. Operations on this type would be very fast, but may lack in other areas. The second example defines a number as two very large integer arrays and the third defines a number as two string variables. Each of these definitions could yield advantages and disadvantages and should be studied for possible use in the root finder.

Extensions to the MIMO QFT Toolbox

The current MIMO QFT toolbox provides basic manipulations needed for the MIMO QFT problem. However, it cannot be considered a full featured package in its present form. Several upgrades should be considered for this toolbox.

1. With the root finder upgrade mentioned previously, it MIMO toolbox could be extended to handle larger plant matrices with higher order transfer function elements. A reasonable goal is a 10 x 10 matrix of 6th order transfer functions.

2. Graphics capabilities now present in the MISO QFT Toolbox should be ported to the MIMO QFT toolbox. This includes the bounds generation, the L_0 formulation and the simulation graphics.

3. The current MIMO QFT Toolbox does not allow the development of the plant matrix from the actuator dynamic components and raw plant components. Nor does it allow the formation of a weighing matrix or the use of the Binet-Cauchey theorem. Future modifications to the MIMO QFT Toolbox should include these.

4. The current MIMO QFT Toolbox does not allow design via the *improved* design method. The next version of the toolbox should specifically allow this.

Future Tool Boxes

Finally, there is room for many more toolboxes. Every conceivable control system theory should be exploited in toolbox form using ICECAP-PC toolbox methodology. The MIMO QFT toolbox provides an excellent structural example for future development.

Appendix A: Bibliography

Ash, Raymond H. and Gerald R. Ash. "Numerical Computation of Root Loci Using the Newton-Raphson Technique," *IEEE Transactions on Automatic Control.* 376-382. October 1968.

Bailey, F. N. and others. "The Loop Gain-Phase Shaping Design Programs," *Quantitative Feedback Theory Symposium Proceedings* WL-TR-92-3063. 565-574. Wright-Patterson Air Force Base OH: Wright Laboratory (ASD), August 1992.

Ballance, D. J. and P. J. Gawthrop. "QFT, the UHB, and the Choice of the Template Nominal Point," *Quantitative Feedback Theory Symposium Proceedings* WL-TR-92-3063. 74-79. Wright-Patterson Air Force Base OH: Wright Laboratory (ASD), August 1992.

Barker, H. A. "User Interface Standards for Control System Design Applications," *1989 IEEE Control Systems Society Workshop on Computer-Aided Control System Design (CACSD).* 86-93. New York: The Institute of Electrical and Electronics Engineers, Inc., 1989.

Bell, Wayne. *An Object Oriented Computer Aided Design Program for Conventional Control Systems Analysis.* MS Thesis, AFIT/GE/ENG/92D. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.

Borland International Inc. *Tubro Pascal 6.0 Programmers Guide,* Scotts Valley, CA: Borland International Inc, undated.

Borland International Inc. *Tubro Pascal 6.0 Turbo Vision Guide,* Scotts Valley, CA: Borland International Inc, 1991.

Bossert, David Edward. *Design of Pseudo-Continuous Time Quantitative Feedback Theory Robot Controllers.* MS Thesis, AFIT/GE/ENG/89D-2. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

Bossert, David Edward, Captain US Air Force. Telephone Interview. Air Force Academy, 13 November 1992.

Burden, Richard L. and Douglas J. Faires. *Numerical Analysis.* Boston: PWS-Kent Publishing Company, 1981.

Chapra Stephen C. and Raymond P. Canale. *Numerical Methods for Engineers.* New York: McGraw-Hill Book Company, 1988.

Chiang, Richard Y. and Michael G. Safanov. *Robust Control Toolbox User's Guide.* Natick MA: The Math Works, Inc, 1 June 1988.

Chikofsky, E. J. *Software Development: Computer-Aided Software Engineering (CASE).* IEEE Computer Society Press Technology Series.

Curtis, Bill. "Prototyping Versus Specifying: A Multiproject Experiment," *Tutorial: Human Factors in Software Development.* pg 298.

DAC Micro Systems. *dCOM The Directory Commander.* Lancaster CA: DAC Micro Systems, undated.

D'Azzo, John J. and Constaintine H. Houpis. *Linear Control System Analysis & Design* (Third Edition). New York: McGraw-Hill Book Company, 1988.

Dongarra J. J. *Linpack Users Guide.* Philadelphia: SIAM, 1979.

Fisher, Jeff and Dale Gipson. "In Search of Elegance," *Computer Language*, 37-46 (November, 1992).

Frederick, D. K. and M. Rimer. "Benchmark Problems for Computer-Aided Control System Design," Computer Aided Design in Control Systems (IFAC 1988). 1988.

Fröberg, Carl_Erik. *Numerical Mathematics.* Menlo Park: The Benjamin/Cummings Publishing Company, Inc, 1985.

Gerald, Curtis F. *Applied Numerical Analysis.* Menlo Park: Addison-Wesley Publishing Company, 1978.

Grace, Andrew. *Optimization Toolbox User's Guide.* Natick MA: The Math Works, Inc., November 1990.

Horowitz, Isaac and Clayton Loecher, "Design of a 3x3 multivariable feedback system with large plant uncertainty," *International Journal of Control*, Vol 33 No 4:677-699, 1981.

Houpis, Constaintine. *Quantitative Feedback Theory Technique for Designing Multivariable Control Systems.* AFWAL-TR-86-3107. Wright-Patterson AFB OH: Air Force Wright Aeronautical Laboratories, 1987.

Houpis, Constantine H. and Gary B. Lamont. *Digital Control Systems: Theory, Hardware, Software* (Second Edition). New York: McGraw-Hill, Inc, 1992.

Houpis, Constantine H. AFIT Controls Department Head. Personal Interview. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 13 November 1992b.

IEEE. *Binary Floating-Point Arithmetic.* IEEE Std 754. New York: IEEE, 1985.

Integrated Systems, Inc. *MatrixX User's Guide Version 6.0.* Santa Clara CA: Integrated Systems, Inc, May 1986a.

Integrated Systems, Inc. *System_Build User's Guide.* Santa Clara CA: Integrated Systems, Inc, May 1986b.

Jones, John. "Stabilization of NonLinear Time-Varying Control Systems," *Southeastern Simulation Conference Proceedings*, 1993.

Kheir, Naim. *Systems Modeling and Computer Simulation.* New York: Marcel Dekker Inc., 1988.

Kobylarz, Thomas J. *Flight Controller Design with Nonlinear Aerodynamics, Large Parameter Uncertainty and Pilog Compensation.* MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.

Kreyszig, Erwin, *Advanced Engineering Mathematics* (Fifth Edition), New York, John Wiley & Sons, 1983.

Larimer, Stanley, *An Interactive Computer-Aided Design Program for Discrete and Continuous Control System Analysis and Systhesis.* MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1978.

Lempp, Peter and Rudolf Lauber. *Software Development: Computer-Aided Software Engineering (CASE).* IEEE Computer Society Press Technology Series.

Lial, Margaret L. and Charles D. Miller. *Algebra and Trigonometry* (Second Edition). Dallas: Scott, Foresman and Company, 1980.

MathSoft, Inc. *Mathcad 3.0 User's Guide.* Cambridge MA: MathSoft Inc, June 1991.

Microsoft Corporation. *Microsoft Windows User's Guide.* Document No. 620299-00 REV A. Microsoft Corporation, Redmond WA, 1990.

Mashiko, Susan and Gary Tarcjynski. *Development of a Computer Aided Design Package for Control System Design and Analysis for a Personal Compute.* MS Thesis, AFIT/ENG/86D8. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985.

Moler, Cleve and others. *PC-MATLAB for MS-DOS Personal Computers User's Guide Version 3.2-PC.* Sherborn MA: The Math Works, Inc, 8 June 1987.

Moore, Paul A. *Extension of the Software Development Workbench to Include Microcomputer Workstations.* MS Thesis, AFIT/GCS/ENG/84D-18. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1984.

O'Brian, Fredrick L, *Consolidated Computer Program for Control System Design.* MS Thesis, AFIT/GE/ENG/84D-18. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1977.

Press, William H. *Numerical Recipes: The Art of Scientific Computing* London: Cambridge University Press, 1989.

Pressman, Roger S, *Software Engineering: A Practitioner's Approach* (Second Edition). New York: McGraw-Hill Pub Co, 1987.

Rabinowitz, P. *A First Course in Numerical Analysis* (Second Edition). New York: McGraw-Hill, 1978.

Ravn, Ole. "On User-Friendly Interface Construction for CACSD Packages," *1989 IEEE Control Systems Society Workshop on Computer-Aided Control System Design (CACSD).* pp 35-40. New York: The Institute of Electrical and Electronics Engineers, Inc., 1989.

Reid, Gary J. *Linear System Fundamentals.* New York: McGraw-Hill Book Company, 1983.

Rumbaugh, James. *Object-Oriented Modeling and Design.* Englewood Cliffs: Prentice Hall, 1991.

Sating, Richard R. *Development of an Analog MIMO QFT Cad Package.* MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.

Smith, Sidney L. and Jane N. Mosier. *Guidelines for Designing User Interface Software.* ESD-TR-86-278. Contract MTR-10090. Bedford MA: MITRE, August 1986 (AD-A177 198).

Smith. *Matrix Eigensystem Routines: EISPACK Guide.* Berlin: Springer-Verlag, 1976.

Tannenbaum, Andrew S. *Stuctured Computer Organization.* Englewood Cliffs: Prentice Hall.

Thompson, Peter M. *User's Guide to Program CC, Version 3*. Contract F33657-83-0242. Hawthorne CA: Systems Technology, Inc., March 1985.

Yaniv, Oded. *Multiple Input Single Output (MISO) QFT-CAD Users Manual.* Tel-Aviv University, 1992.

Wheaton, David G. *Automatic Flight Control System Design for an Unmanned Research Vehicle using Discrete Quantitative Feedback Theory.* MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990.

Appendix B: Testing and Validation

This appendix contains several test results for numerical algorithms used in ICECAP-PC. This is a representative sample and not a complete list simply because there are far too many tests to include with any meaning.

1. Matrix Condition Number Test (pg B-4)

This test of the matrix condition number algorithm first finds the condition number of a given matrix per eq 2-2. It then finds the inverse of the matrix and determines the difference between the actual roundoff error and the projected roundoff error (eq 2-3). We found that for matrix inversion, the actual roundoff error sometimes exceeded the projected error. However, for other operations (not shown) the projected roundoff error gave a good maximum error estimate. We performed this test because at one point in the project we considered using the error anticipated by eq 2-2 to round off the resultant matrix elements. It remains an open option, but one we did not implement.

2. QR Shift Algorithm Test (pg B-5)

Our development of the QR Shift algorithm is based on the code given in Numerical Recipes for Pascal (Press, 1988). This source implements the QR Shift for the general real valued matrix that may contain complex eigenvalues in conjugate pairs. It recommends a three part sequence of balancing, upper Hessenberg conversion, and QR Shift (Sec II). However Numerical Recipes says nothing about a general complex valued matrix that may have single complex eigenvalues. No other reference that handled this case either. We instead build a test bench in Matlab to perform the QR Shift. The effort was a near failure until our discovery that after the QR Shift algorithm, complex eigenvalues can be hidden as eigenvalues of a 2x2 block along the main diagonal! With this discovery, we were able to modify the code from Numerical Recipes to handle the general complex valued matrix. This algorithm now appears in ICECAP-PC replacing that of previous versions that used roots of the characteristic equation. The listing for test 2 yielded this discovery!

3. QR Shift Code Modification Test (pg B-9)

Section 4.3 shows the testing performed on the final version of the ICECAP-PC QR Shift algorithm. It is included as an example of setting test points inside a numerical method to watch variables under iterations.

4. QR Shift Final Verification for multiple eigenvalues (pg B-29)

Section 4.4 shows some final validation tests for the QR Shift algorithm as implemented in ICECAP to determine behavior in the presence in multiple real eigenvalues. We found that in the presence of multiple real eigenvalues, the error associated with the calculated eigenvalues was symmetric about the actual eigenvalues. Therefore, we gave careful consideration to using the eigenvalue algorithm to perform root finding for polynomials via the fabrication of Jordan-Canonical matrices. However, as Jordan-Canonical matrices are notoriously ill-conditioned and imbalanced, this process would yield considerable error for high order polynomials. Therefore, standard root finding techniques are used for polynomials in ICECAP-PC.

5. LU Decomposition (pg B-30)

Section B-5 shows the final stage of testing for the LU Decomposition algorithm used for the general complex matrix. The purpose of this test was simply to take several matrices, decompose them into their upper and lower triangular components and then multiply these two components to see what error resulted. We did this for several matrices. This section shows a real matrix, a poorly condition Hilbert matrix and a general complex matrix.

6. Inversion of Plant Matrix (pg B-32)

This section shows the tests performed on the LU Decomposition algorithm for matrices of transfer function as implemented in the MIMO QFT toolbox. During the process of testing this code, we worked out a 3x3 matrix inversion by hand and compared it step by step with the computer answers. In all cases of discrepancy, the computer was right and I was wrong! The reader interested in LU Decomposition as a solution to the inverse of plant matrix P should consider working out such an example by hand. It then becomes quite interesting to watch root cancellation take place on a element by element basis.

7. The Root Finder (pg B-37)

Section B-7 shows our tests comparing our root finder with that of PC-Matlab. In all cases we exceed the accuracy attained in Matlab, often by several decimal places. However, in the process we noticed something that forced us to modify our algorithm. The ICECAP-PC root finder first calls the Laguerre method to make the initial cut at the roots. The Laguerre method returns a set of values whose error with the actual roots is

symmetrical with the actual roots in the case of root multiplicity. Hence coefficient reconstruction is extremely accurate after root finding with the Laguerre algorithm. The second step used by the ICECAP root finder is the polishing of the rough roots with either the Bairstow method in the case of real pairs or with Brents method in the case of single real roots or high root multiplicity. We were able to achieve extremely accurate roots with these two polishing methods. Improvements of 4-5 decimal places over Laguerre's method was common. However this added accuracy came at a loss of error symmetry about the actual values. ***Polynomial coefficient reconstruction is often degraded with the better roots found by the polishing methods!*** We finally modified our algorithm as follows.

8. Root Finder Coefficient Reconstruction Error Test (pg B-40)

After encountering the above mentioned problem we modified the root finder to take the first cut using Laguerre, store the Laguerre roots, take the second cut using Bairstow and Brent, and store the polished roots, reconstruct the polynomial coefficients with the Laguerre roots and find a reconstruction error, reconstruct the polynomial coefficients with the polished roots and find the polished reconstruction error, and compare the two errors taking the root set yielding the least reconstruction error. Very often ICECAP settles on the less accurate set of roots because the coefficient reconstruction error is minimized.

Matrix Condition Number Test

Fred L. Trevino, 1st Lt
EENG 799
Instructor: Prof Lamont

Comment:
The following matrix is a Hilbert (badly conditioned matrix) for n = 3

Matrix A

```
 1.0000  0.5000  0.3333
 0.5000  0.3333  0.2500
 0.3333  0.2500  0.2000
```

Matrix B = The Inverse Matrix Of Matrix A

Matrix B

```
  9.0000  -36.0000   30.0000
-36.0000  192.0000 -180.0000
 30.0000 -180.0000  180.0000
```

Matrix B

```
    9.000000000000000020    -36.000000000000000100     30.000000000000000100
  -36.000000000000000100    192.000000000000001000   -180.000000000000001000
   30.000000000000000100   -180.000000000000001000    180.000000000000001000
```

Comment:
Note that error has crept into the 17th bit position as predicted by Chapra

Comment:
Condition numbers can therefore be used to truncate error at known low bits

Comment:

The second test uses LU Decomposition to test the inverse of the 3x3 Hilbert Matrix

Matrix A

```
 1.00000000000000  0.50000000000000  0.33333333333333
 0.50000000000000  0.33333333333333  0.25000000000000
 0.33333333333333  0.25000000000000  0.20000000000000
```

Matrix B = The Inverse Matrix Of Matrix A

Matrix B

```
   8.999999999999999970   -35.999999999999999900    29.999999999999999900
 -35.999999999999999900   191.999999999999999000  -179.999999999999999000
  29.999999999999999900  -179.999999999999999000   179.999999999999999000
```

Comment:
it is only acurate to 15 significant bits

Matrix B

```
  9.00000000000000  -36.00000000000000   30.00000000000000
-36.00000000000000  192.00000000000000 -180.00000000000000
 30.00000000000000 -180.00000000000000  180.00000000000000
```

Comment:
Here we see that the calculation of the inverse is only accurate to 15 significant bits reguardless of which method we use for a 3x3 matrix. Freds eq predicts rouding error to the 19-3-16th bit. The fifteenth bit must be used for rounding, therefore rounding to 14 sig digits will produce the exact correct answer.

Eigenvalues using QR Shift

```
Fred L. Trevino
25 June 92
This is a test of Dr Jones QR Shift Algorithm for a general complex matrix
with single eigenvalues.  This will validate the formation of the Householder
matrix by the method taught by Dr Jones.
================================================================================

» a

a =

   1.0000 + 1.0000i   4.0000 - 2.0000i   3.0000 - 1.0000i
   4.0000 + 2.0000i   4.0000             6.0000 + 8.0000i
        0 + 1.0000i   3.0000             1.0000 + 7.0000i


» InMat = balance(a)

InMat =

   1.0000 + 1.0000i   4.0000 - 2.0000i   1.5000 - 0.5000i
   4.0000 + 2.0000i   4.0000             3.0000 + 4.0000i
        0 + 2.0000i   6.0000             1.0000 + 7.0000i

» InMat = hess(InMat)

InMat =

   1.0000 + 1.0000i   4.1079 - 1.3693i   1.3693 + 1.3693i
   4.3818 + 2.1909i   3.6667 + 0.8333i   4.9333 + 6.3000i
        0             3.0000 + 0.1667i   1.3333 + 6.1667i

» eigtst
     Undefined function or variable.
Symbol in question ==^P eigtst

» eigentst

----- Main Menu      -----

      1) Input Matrix
      2) Specify Ctr
      3) Run Test
      4) Display Result
      5) Quit


 Select a menu number:
 Enter Iteration Count:
Ctr =

    25


Ctr =

    25

Press Enter To Continue

----- Main Menu      -----

      1) Input Matrix
      2) Specify Ctr
      3) Run Test
      4) Display Result
      5) Quit


 Select a menu number: Running.....
Iteration:

i =

     1
```

```
Xmat =

   1.0000 + 1.0000i
   4.3818 + 2.1909i
        0


Zmat =

     1
     0
     0


Sigma =

   4.4737 + 2.3694i


VSquare =

   65.0787


Hmat =

  -0.2697 + 0.0000i  -0.9640 - 0.0852i        0
  -0.9640 + 0.0852i   0.2624                  0
        0                   0             1.0000


Rmat =

  -4.3069 - 2.7548i  -4.5713 - 0.7464i  -4.5882 - 6.8624i
   0.1008 - 0.3038i  -2.8810 + 1.8886i  -0.1419 + 0.4500i
        0             3.0000 + 0.1667i   1.3333 + 6.1667i


Qmat =

  -0.2605 - 0.0000i  -0.9570 - 0.0846i        0
  -0.9570 + 0.0846i   0.2677 - 0.0000i        0
        0                   0             1.0000


Y =

   5.5601 + 1.0455i   2.6650 + 2.8009i  -4.5882 - 6.8624i
   2.5712 - 1.9719i  -0.8935 + 0.7879i  -0.1419 + 0.4500i
  -2.8852 + 0.0942i   0.8032 + 0.0446i   1.3333 + 6.1667i

Iteration:

i =

     2


Xmat =

   5.5601 + 1.0455i
   2.5712 - 1.9719i
  -2.8852 + 0.0942i


Zmat =

     1
     0
     0


Sigma =

   6.3926 + 0.0737i
```

```
VSquare =

  153.4983


Hmat =

  -0.8778 + 0.0000i  -0.3717 - 0.3446i   0.4480 + 0.0567i
  -0.3717 + 0.3446i   0.8632 - 0.0000i   0.0991 - 0.0710i
   0.4480 - 0.0567i   0.0991 + 0.0710i   0.8914 + 0.0000i


Rmat =

  -7.8138 - 1.1923i  -1.3785 - 2.3781i   4.4828 + 8.7437i
  -0.4866 + 0.0394i  -2.6442 + 0.5048i   4.5173 + 1.8744i
   0.3728 + 0.2239i   1.9243 + 1.1579i  -1.3021 + 2.7179i


Qmat =

  -0.6718 + 0.0000i  -0.3309 - 0.3068i   0.3988 + 0.0505i
  -0.3309 + 0.3068i   0.8782 - 0.0000i   0.0882 - 0.0632i
   0.3988 - 0.0505i   0.0882 + 0.0632i   0.9033 + 0.0000i


Y =

   8.6650 + 4.4258i   0.8522 + 1.7579i   0.7215 + 6.9056i
   2.9433 - 0.4854i  -1.8690 + 1.0303i   3.6832 + 1.8959i
  -1.6245 + 1.2065i   1.3486 + 0.9858i  -0.7960 + 2.5438i


Iteration:

i =

    25


Xmat =

   7.6043 + 4.1451i
   0.1410 + 0.3151i
  -0.1389 - 0.1881i


Zmat =

     1
     0
     0


Sigma =

  -7.6043 - 4.1451i


VSquare =

    0.1704


Hmat =

   1.0000                  0                  0
        0  -0.3988 + 0.0000i   0.9256 + 0.2026i
        0   0.9256 - 0.2026i   0.3581 - 0.0000i


Rmat =

   7.6043 + 4.1451i  -2.6219 + 6.2994i  -0.4115 - 5.0033i
  -0.1467 - 0.3279i   3.5471 - 1.2151i  -0.4863 - 0.6414i
   0.1446 + 0.1957i   0.0817 + 2.9607i   1.7685 + 3.2964i
```

```
Qmat =

   1.0000                  0                  0
        0   -0.3441 - 0.0000i   0.8895 + 0.1947i
        0    0.8895 - 0.1947i   0.3832 + 0.0000i


Y =

   7.6043 + 4.1451i  -0.4380 - 2.2024i  -1.2632 - 8.0309i
  -0.1467 - 0.3279i  -1.7781 - 0.0577i   3.2053 - 0.6359i
   0.1446 + 0.1957i   2.1868 + 1.5689i   0.1739 + 3.9125i


----- Main Menu      -----

      1) Input Matrix
      2) Specify Ctr
      3) Run Test
      4) Display Result
      5) Quit


 Select a menu number: The Final Result Is:

Result =

   7.6043 + 4.1451i  -0.4380 - 2.2024i  -1.2632 - 8.0309i
  -0.1467 - 0.3279i  -1.7781 - 0.0577i   3.2053 - 0.6359i
   0.1446 + 0.1957i   2.1868 + 1.5689i   0.1739 + 3.9125i

%Now, the result of 25 iterations is as follows:
%=============================================================================
Result =

   7.6043 + 4.1451i  -0.4380 - 2.2024i  -1.2632 - 8.0309i
  -0.1467 - 0.3279i  -1.7781 - 0.0577i   3.2053 - 0.6359i
   0.1446 + 0.1957i   2.1868 + 1.5689i   0.1739 + 3.9125i

» eig(a)

ans =

  -3.4786 + 0.5744i
   1.8179 + 3.4037i
   7.6607 + 4.0219i  <- Note that this is close to element (1,1) of the result

» test = [Result(2,2) Result(2,3); Result(3,2) Result(3,3)]

test =

  -1.7781 - 0.0577i   3.2053 - 0.6359i  ┌──> This is the two by two matrix
   2.1868 + 1.5689i   0.1739 + 3.9125i  │     in the lower right hand corner

» eig(test)

ans =
                      ┌──────────> These are the two eigenvalues of the matrix
  -3.4530 + 0.5103i   │            in the lower right hand corner.  They are also
   1.8487 + 3.3445i   │            eigenvalues of the original matrix.

» save

Saving to: matlab.mat

» quit

 42652 flops.
```

The QR Shift Algorithm Implemented in ICECAP

Fred L. Trevino
EENG 799
Prof Gary I Lamont

```
: QRShift Test

Matrix C

 0.0000  1.0000  0.0000
 0.0000  0.0000  1.0000
 6.0000 -1.0000 -4.0000


Balanced Form of Matrix C

 0.0000  2.0000  0.0000
 0.0000  0.0000  1.0000
 3.0000 -1.0000 -4.0000


Hessenberg Form of Balanced Matrix C

 0.0000  0.0000  2.0000
 3.0000 -4.0000 -1.0000
 0.0000  1.0000  0.0000
TP 1
TP 2
TP 3
TP 3
TP 4
TP 6
TP 10
TP 12

its 1
nn: 3
--------------------------------
p:           0.2500
q:           0.0000
r:           0.7500
s:           1.3333
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           0.0000
y:          -4.0000
z:           0.0000
Eigenvalue 1           0.0000          0.0000
Eigenvalue 2           0.0000          0.0000
Eigenvalue 3           0.0000          0.0000
TP 18
TP 19
TP 20
TP 22
TP 23

its 1
nn: 3
--------------------------------
p:           0.0000
q:           0.0000
r:           0.7208
s:           0.7906
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.3162
y:           0.0000
z:           0.9487
Eigenvalue 1           0.0000          0.0000
Eigenvalue 2           0.0000          0.0000
Eigenvalue 3           0.0000          0.0000
TP 22
TP 23
```

```
its 1
nn: 3
--------------------------------
p:            0.7208
q:            0.0000
r:            0.7208
s:            0.7906
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.3162
y:            0.0000
z:            0.9487
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 22
TP 23

its 1
nn: 3
--------------------------------
p:            2.0000
q:            0.0000
r:            0.7208
s:            0.7906
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.3162
y:            0.0000
z:            0.9487
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 24
TP 25

its 1
nn: 3
--------------------------------
p:           -0.6000
q:            0.0000
r:            0.7208
s:            0.7906
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.3162
y:            0.0000
z:            0.9487
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 24
TP 25

its 1
nn: 3
--------------------------------
p:            3.0000
q:            0.0000
r:            0.7208
s:            0.7906
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.3162
y:            0.0000
z:            0.9487
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 24
TP 25
```

```
its 1
nn: 3
--------------------------------
p:          -1.8000
q:           0.0000
r:           0.7208
s:           0.7906
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.3162
y:           0.0000
z:           0.9487
Eigenvalue 1          0.0000          0.0000
Eigenvalue 2          0.0000          0.0000
Eigenvalue 3          0.0000          0.0000
TP 14

its 1
nn: 3
--------------------------------
p:           0.0000
q:           1.8000
r:           0.7208
s:           0.7906
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.3162
y:           0.0000
z:           0.9487
Eigenvalue 1          0.0000          0.0000
Eigenvalue 2          0.0000          0.0000
Eigenvalue 3          0.0000          0.0000
TP 16

its 1
nn: 3
--------------------------------
p:           0.0000
q:           1.8000
r:           0.0000
s:           0.7906
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.3162
y:           0.0000
z:           0.9487
Eigenvalue 1          0.0000          0.0000
Eigenvalue 2          0.0000          0.0000
Eigenvalue 3          0.0000          0.0000
TP 17

its 1
nn: 3
--------------------------------
p:           0.0000
q:           1.0000
r:           0.0000
s:           0.7906
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.8000
y:           0.0000
z:           0.9487
Eigenvalue 1          0.0000          0.0000
Eigenvalue 2          0.0000          0.0000
Eigenvalue 3          0.0000          0.0000
TP 18
TP 19
TP 21
TP 22
TP 22
TP 24
TP 24
```

```
TP 24
TP 26
TP 3
TP 3
TP 4
TP 6
TP 10
TP 12

its 2
nn: 3
--------------------------------
p:          -0.3060
q:           0.4113
r:          -0.2827
s:          11.1845
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:          -4.0000
y:          -0.6000
z:           0.6000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 20
TP 22
TP 23

its 2
nn: 3
--------------------------------
p:           1.4305
q:          -0.4614
r:           0.3172
s:          -0.5854
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.5227
y:          -0.7025
z:           0.4830
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 2
nn: 3
--------------------------------
p:          -0.5262
q:          -0.4614
r:           0.3172
s:          -0.5854
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.5227
y:          -0.7025
z:           0.4830
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 2
nn: 3
--------------------------------
p:          -0.4660
q:          -0.4614
r:           0.3172
s:          -0.5854
t:           0.0000
```

```
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.5227
y:           -0.7025
z:            0.4830
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 24
TP 25

its 2
nn: 3
================================
p:           -2.3056
q:           -0.4614
r:            0.3172
s:           -0.5854
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.5227
y:           -0.7025
z:            0.4830
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 24
TP 25

its 2
nn: 3
================================
p:           -0.5347
q:           -0.4614
r:            0.3172
s:           -0.5854
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.5227
y:           -0.7025
z:            0.4830
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 24
TP 25

its 2
nn: 3
================================
p:           -0.8321
q:           -0.4614
r:            0.3172
s:           -0.5854
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.5227
y:           -0.7025
z:            0.4830
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 14

its 2
nn: 3
================================
p:           -0.2603
q:            0.1412
r:            0.3172
s:           -0.5854
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
```

```
x:            1.5227
y:           -0.7025
z:            0.4830
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 16

its 2
nn: 3
================================
p:           -0.2603
q:            0.1412
r:            0.0000
s:           -0.5854
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.5227
y:           -0.7025
z:            0.4830
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 17

its 2
nn: 3
================================
p:           -0.6484
q:            0.3516
r:            0.0000
s:           -0.5854
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            0.4015
y:           -0.7025
z:            0.4830
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 18
TP 19
TP 21
TP 22
TP 22
TP 24
TP 24
TP 24
TP 26
TP 3
TP 3
TP 4
TP 6
TP 10
TP 12

its 3
nn: 3
================================
p:            0.8223
q:            0.1344
r:            0.0432
s:           37.4230
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:           -4.3027
y:           -0.4247
z:            0.7274
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 18
TP 19
TP 20
```

```
TP 22
TP 23

its 3
nn: 3
--------------------------------
p:            0.7514
q:            0.0811
r:            0.0261
s:            0.8344
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9856
y:            0.1611
z:            0.0518
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 3
nn: 3
--------------------------------
p:            1.2018
q:            0.0811
r:            0.0261
s:            0.8344
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9856
y:            0.1611
z:            0.0518
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 3
nn: 3
--------------------------------
p:            1.8097
q:            0.0811
r:            0.0261
s:            0.8344
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9856
y:            0.1611
z:            0.0518
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 3
nn: 3
--------------------------------
p:           -1.7892
q:            0.0811
r:            0.0261
s:            0.8344
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9856
y:            0.1611
z:            0.0518
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
```

```
TP 25

its 3
nn: 3
--------------------------------
p:            0.1380
q:            0.0811
r:            0.0261
s:            0.8344
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9856
y:            0.1611
z:            0.0518
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 3
nn: 3
--------------------------------
p:           -0.0545
q:            0.0811
r:            0.0261
s:            0.8344
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9856
y:            0.1611
z:            0.0518
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 14

its 3
nn: 3
--------------------------------
p:            0.0371
q:            0.0156
r:            0.0261
s:            0.8344
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9856
y:            0.1611
z:            0.0518
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 16

its 3
nn: 3
--------------------------------
p:            0.0371
q:            0.0156
r:            0.0000
s:            0.8344
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9856
y:            0.1611
z:            0.0518
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 17

its 3
nn: 3
```

```
--------------------------------
p:            0.7042
q:            0.2958
r:            0.0000
s:            0.8344
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            0.0527
y:            0.1611
z:            0.0518
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 21
TP 22
TP 22
TP 24
TP 24
TP 24
TP 26
TP 3
TP 3
TP 4
TP 6
TP 10
TP 12

its 4
nn: 3
--------------------------------
p:           -0.9840
q:            0.0151
r:           -0.0010
s:          308.7020
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:           -3.6282
y:           -1.3963
z:            1.0246
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 20
TP 22
TP 23

its 4
nn: 3
--------------------------------
p:            1.0249
q:           -0.0077
r:            0.0005
s:           -0.9841
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9999
y:           -0.0153
z:            0.0010
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 4
nn: 3
--------------------------------
p:            1.5475
q:           -0.0077
r:            0.0005
```

```
s:          -0.9841
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.9999
y:          -0.0153
z:           0.0010
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 4
nn: 3
--------------------------------
p:          -0.9831
q:          -0.0077
r:           0.0005
s:          -0.9841
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.9999
y:          -0.0153
z:           0.0010
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 4
nn: 3
--------------------------------
p:          -2.0252
q:          -0.0077
r:           0.0005
s:          -0.9841
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.9999
y:          -0.0153
z:           0.0010
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 4
nn: 3
--------------------------------
p:          -0.0247
q:          -0.0077
r:           0.0005
s:          -0.9841
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           1.9999
y:          -0.0153
z:           0.0010
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 4
nn: 3
--------------------------------
p:          -0.0010
q:          -0.0077
r:           0.0005
s:          -0.9841
```

```
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9999
y:           -0.0153
z:            0.0010
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 14

its 4
nn: 3
--------------------------------
p:            0.0002
q:           -0.0000
r:            0.0005
s:           -0.9841
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9999
y:           -0.0153
z:            0.0010
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 16

its 4
nn: 3
--------------------------------
p:            0.0002
q:           -0.0000
r:            0.0000
s:           -0.9841
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.9999
y:           -0.0153
z:            0.0010
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 17

its 4
nn: 3
--------------------------------
p:            0.8760
q:           -0.1240
r:            0.0000
s:           -0.9841
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            0.0002
y:           -0.0153
z:            0.0010
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 21
TP 22
TP 22
TP 24
TP 24
TP 24
TP 26
TP 3
TP 3
TP 4
TP 6
TP 10
```

```
TP 12

its 5
nn: 3
================================
p:          -0.9999
q:           0.0001
r:           0.0000
s:       69288.8960
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:          -3.1570
y:          -1.8431
z:           1.0001
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 20
TP 22
TP 23

its 5
nn: 3
================================
p:           1.0001
q:          -0.0000
r:          -0.0000
s:          -0.9999
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0001
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 5
nn: 3
================================
p:           1.6997
q:          -0.0000
r:          -0.0000
s:          -0.9999
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0001
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 5
nn: 3
================================
p:           0.7813
q:          -0.0000
r:          -0.0000
s:          -0.9999
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0001
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
```

```
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 5
nn: 3
--------------------------------
p:          -2.0001
q:          -0.0000
r:          -0.0000
s:          -0.9999
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0001
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 5
nn: 3
--------------------------------
p:          -0.0001
q:          -0.0000
r:          -0.0000
s:          -0.9999
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0001
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 5
nn: 3
--------------------------------
p:           0.0000
q:          -0.0000
r:          -0.0000
s:          -0.9999
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0001
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 14

its 5
nn: 3
--------------------------------
p:           0.0000
q:           0.0000
r:          -0.0000
s:          -0.9999
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0001
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 16
```

```
its 5
nn: 3
--------------------------------
p:            0.0000
q:            0.0000
r:            0.0000
s:           -0.9999
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            2.0000
y:           -0.0001
z:           -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 17

its 5
nn: 3
--------------------------------
p:            0.9625
q:            0.0375
r:            0.0000
s:           -0.9999
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            0.0000
y:           -0.0001
z:           -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 21
TP 22
TP 22
TP 24
TP 24
TP 24
TP 26
TP 3
TP 3
TP 4
TP 6
TP 10
TP 12

its 6
nn: 3
--------------------------------
p:           -1.0000
q:            0.0000
r:           -0.0000
s:  6298606684.8862
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:           -3.0160
y:           -1.9840
z:            1.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 20
TP 22
TP 23

its 6
nn: 3
--------------------------------
p:            1.0000
```

```
q:          -0.0000
r:           0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:           0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 6
nn: 3
--------------------------------
p:           1.7290
q:          -0.0000
r:           0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:           0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 6
nn: 3
--------------------------------
p:          -0.7146
q:          -0.0000
r:           0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:           0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 6
nn: 3
--------------------------------
p:          -2.0000
q:          -0.0000
r:           0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:           0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 6
nn: 3
--------------------------------
p:          -0.0000
q:          -0.0000
```

```
r:            0.0000
s:           -1.0000
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            2.0000
y:           -0.0000
z:            0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 6
nn: 3
================================
p:           -0.0000
q:           -0.0000
r:            0.0000
s:           -1.0000
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            2.0000
y:           -0.0000
z:            0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 14

its 6
nn: 3
================================
p:            0.0000
q:           -0.0000
r:            0.0000
s:           -1.0000
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            2.0000
y:           -0.0000
z:            0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 16

its 6
nn: 3
================================
p:            0.0000
q:           -0.0000
r:            0.0000
s:           -1.0000
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            2.0000
y:           -0.0000
z:            0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 17

its 6
nn: 3
================================
p:            0.9957
q:           -0.0043
r:            0.0000
s:           -1.0000
t:            0.0000
u:  0.55637508958790268E+0105
```

```
v:           0.0000
x:           0.0000
y:          -0.0000
z:           0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 21
TP 22
TP 22
TP 24
TP 24
TP 24
TP 26
TP 3
TP 3
TP 4
TP 6
TP 10
TP 12

its 7
nn: 3
--------------------------------
p:          -1.0000
q:           0.0000
r:           0.0000
s: 66626453394759628100.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:          -3.0002
y:          -1.9998
z:           1.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 20
TP 22
TP 23

its 7
nn: 3
--------------------------------
p:           1.0000
q:          -0.0000
r:          -0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 7
nn: 3
--------------------------------
p:           1.7320
q:          -0.0000
r:          -0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:          -0.0000
```

```
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 22
TP 23

its 7
nn: 3
--------------------------------
p:           0.7072
q:          -0.0000
r:          -0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 7
nn: 3
--------------------------------
p:          -2.0000
q:          -0.0000
r:          -0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 7
nn: 3
--------------------------------
p:          -0.0000
q:          -0.0000
r:          -0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 24
TP 25

its 7
nn: 3
--------------------------------
p:           0.0000
q:          -0.0000
r:          -0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
```

```
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 14

its 7
nn: 3
--------------------------------
p:           0.0000
q:          -0.0000
r:          -0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 16

its 7
nn: 3
--------------------------------
p:           0.0000
q:          -0.0000
r:           0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           2.0000
y:          -0.0000
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 17

its 7
nn: 3
--------------------------------
p:           1.0000
q:          -0.0000
r:           0.0000
s:          -1.0000
t:           0.0000
u:  0.55637508958790268E+0105
v:           0.0000
x:           0.0000
y:          -0.0000
z:          -0.0000
Eigenvalue 1            0.0000          0.0000
Eigenvalue 2            0.0000          0.0000
Eigenvalue 3            0.0000          0.0000
TP 18
TP 19
TP 21
TP 22
TP 22
TP 24
TP 24
TP 24
TP 26
TP 3
TP 3
TP 6
TP 7

its 7
nn: 3
--------------------------------
p:           0.5003
q:           0.2500
r:           0.0000
s:           2.9997
```

```
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:           -3.0003
y:           -1.9997
z:            0.5000
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2              0.0000          0.0000
Eigenvalue 3              0.0000          0.0000
TP 8

its 7
nn: 3
--------------------------------
p:            0.5003
q:            0.2500
r:            0.0000
s:            2.9997
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:           -3.0003
y:           -1.9997
z:            1.0003
Eigenvalue 1              0.0000          0.0000
Eigenvalue 2             -2.0000          0.0000
Eigenvalue 3             -3.0000          0.0000
TP 2
TP 4
TP 5

its 0
nn: 0
--------------------------------
p:            0.5003
q:            0.2500
r:            0.0000
s:            2.9997
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.0000
y:           -1.9997
z:            1.0003
Eigenvalue 1              1.0000          0.0000
Eigenvalue 2             -2.0000          0.0000
Eigenvalue 3             -3.0000          0.0000

its 0
nn: 0
--------------------------------
p:            0.5003
q:            0.2500
r:            0.0000
s:            2.9997
t:            0.0000
u:  0.55637508958790268E+0105
v:            0.0000
x:            1.0000
y:           -1.9997
z:            1.0003
Eigenvalue 1              1.0000          0.0000
Eigenvalue 2             -2.0000          0.0000
Eigenvalue 3             -3.0000          0.0000


Eigenvalues of Matrix C

 1.0000
-2.0000
-3.0000
```

Final Test of QR Shift For ICECAP-PC

Fred L. Trevino
EENG 799
Prof Gary I Lamont

```
: This is the first test matrix in Jordan Canonical Form

Matrix C

 0.0000  1.0000  0.0000
 0.0000  0.0000  1.0000
 6.0000 -1.0000 -4.0000


Eigenvalues of Matrix C

 1.00000000000000000
-3.00000000000000000
-2.00000000000000000

: The second test matrix is given in Numerical Recipes

Matrix E

 1.0000  2.0000   0.0000  0.0000  0.0000
-2.0000  3.0000   0.0000  0.0000  0.0000
 3.0000  4.0000  50.0000  0.0000  0.0000
-4.0000  5.0000 -60.0000  7.0000  0.0000
-5.0000  6.0000 -70.0000  8.0000 -9.0000


Eigenvalues of Matrix E

 50.00000000000000000
  2.00000000000000000 - 1.732050807568877 30j
  2.00000000000000000 + 1.732050807568877 30j
 -9.00000000000000000
  7.00000000000000000

: The third matrix is a complex matrix with distinct complex eigenvalues

Matrix A

 1.0000 + 1.0000j  4.0000 - 2.0000j  3.0000 - 1.0000j
 4.0000 + 2.0000j  4.0000            6.0000 + 8.0000j
 0.0000 + 1.0000j  3.0000            1.0000 + 7.0000j


Eigenvalues of Matrix A

-3.47855955887271427 + 0.57437430768745852j
 1.81789299597115922 + 3.40373402863058531j
 7.66066656290155505 + 4.02189166368195617j
```

LU Decompsition

Fred L. Trevino
EENG 799
Prof Gary I Lamont

Comment:
This is a test of the LU Decomposition Method. I will Decompose three matricies, multiply them back together and compare.

Matrix A

```
 0.000000000000000  1.000000000000000  0.000000000000000
 0.000000000000000  0.000000000000000  1.000000000000000
 6.000000000000000 -1.000000000000000 -4.000000000000000
```

The LU Decomposition is Given By:

L Component

```
 1.000000000000000  0.000000000000000  0.000000000000000
 0.000000000000000  1.000000000000000  0.000000000000000
 0.000000000000000  0.000000000000000  1.000000000000000
```

U Component

```
 6.000000000000000 -1.000000000000000 -4.000000000000000
 0.000000000000000  1.000000000000000  0.000000000000000
 0.000000000000000  0.000000000000000  1.000000000000000
```

The L and U Components Multiplied Together Equal:

Matrix B

```
 0.000000000000000  1.000000000000000  0.000000000000000
 0.000000000000000  0.000000000000000  1.000000000000000
 6.000000000000000 -1.000000000000000 -4.000000000000000
```

Matrix A

```
 0.000000000000000  1.000000000000000  0.000000000000000
 0.000000000000000  0.000000000000000  1.000000000000000
 6.000000000000000 -1.000000000000000 -4.000000000000000
```

Comment:
This was a Jordan Canonical Form Matrix

Comment:

Comment:
The second test is to to a 5x5 Hilbert matrix

Matrix I

```
 1.000000000000000  0.500000000000000  0.333333333333333  0.250000000000000  0.200000000000000
 0.500000000000000  0.333333333333333  0.250000000000000  0.200000000000000  0.166666666666667
 0.333333333333333  0.250000000000000  0.200000000000000  0.166666666666667  0.142857142857143
 0.250000000000000  0.200000000000000  0.166666666666667  0.142857142857143  0.125000000000000
 0.200000000000000  0.166666666666667  0.142857142857143  0.125000000000000  0.111111111111111
```

The LU Decomposition is Given By:

L Component

```
 1.000000000000000  0.000000000000000  0.000000000000000  0.000000000000000  0.000000000000000
 0.200000000000000  1.000000000000000  0.000000000000000  0.000000000000000  0.000000000000000
 0.500000000000000  1.250000000000000  1.000000000000000  0.000000000000000  0.000000000000000
 0.333333333333333  1.250000000000000  0.533333333333333  1.000000000000000  0.000000000000000
 0.250000000000000  1.125000000000000  0.200000000000000  0.642857142857143  1.000000000000000
```

U Component

```
 1.000000000000000  0.500000000000000  0.333333333333333  0.250000000000000  0.200000000000000
 0.000000000000000  0.066666666666667  0.076190476190476  0.075000000000000  0.071111111111111
 0.000000000000000  0.000000000000000 -0.011904761904762 -0.018750000000000 -0.022222222222222
 0.000000000000000  0.000000000000000  0.000000000000000 -0.000416666666667 -0.000846560846561
 0.000000000000000  0.000000000000000  0.000000000000000  0.000000000000000 -0.000011337868481
```

The L and U Components Multiplied Together Equal:

Matrix J

```
 1.000000000000000  0.500000000000000  0.333333333333333  0.250000000000000  0.200000000000000
 0.500000000000000  0.333333333333333  0.250000000000000  0.200000000000000  0.166666666666667
 0.333333333333333  0.250000000000000  0.200000000000000  0.166666666666667  0.142857142857143
 0.250000000000000  0.200000000000000  0.166666666666667  0.142857142857143  0.125000000000000
 0.200000000000000  0.166666666666667  0.142857142857143  0.125000000000000  0.111111111111111
```

Matrix I

```
 1.000000000000000  0.500000000000000  0.333333333333333  0.250000000000000  0.200000000000000
 0.500000000000000  0.333333333333333  0.250000000000000  0.200000000000000  0.166666666666667
 0.333333333333333  0.250000000000000  0.200000000000000  0.166666666666667  0.142857142857143
 0.250000000000000  0.200000000000000  0.166666666666667  0.142857142857143  0.125000000000000
 0.200000000000000  0.166666666666667  0.142857142857143  0.125000000000000  0.111111111111111
```

The Determinant Of Matrix I is 0.000000000003749

Comment:
Note how poorly conditioned this matrix is. Yet I did not loose even a single significant digit

Comment:

Comment:
The last test is with a complex matrix

Matrix C

```
 1.0000000000 + 2.0000000000j  4.0000000000 + 5.0000000000j  7.0000000000 + 3.0000000000j
 3.0000000000 + 4.0000000000j  7.0000000000 + 5.0000000000j  3.0000000000 + 5.0000000000j
 9.0000000000 + 4.0000000000j  3.0000000000 + 1.0000000000j  3.0000000000 + 6.0000000000j
```

The LU Decomposition is Given By:

L Component

```
 1.000000000000000                          0.000000000000000                          0.000000000000000
 0.443298969072165 + 0.247422680412371j  1.000000000000000                          0.000000000000000
 0.175257731958763 + 0.144329896907216j  0.769966722129784 + 0.245840266222962j  1.000000000000000
```

L Component

```
 9.0000000000 + 4.0000000000j  3.0000000000 + 1.0000000000j  3.0000000000 + 6.0000000000j
 0.0000000000                  5.9175257731 + 3.8144329896j  3.1546391752 + 1.5979381443j
 0.0000000000                  0.0000000000                  5.3040765391 - 0.4904328125j
```

The L and U Components Multiplied Together Equal:

Matrix D

```
 1.0000000000 + 2.0000000000j  4.0000000000 + 5.0000000000j  7.0000000000 + 3.0000000000j
 3.0000000000 + 4.0000000000j  7.0000000000 + 5.0000000000j  3.0000000000 + 5.0000000000j
 9.0000000000 + 4.0000000000j  3.0000000000 + 1.0000000000j  3.0000000000 + 6.0000000000j
```

Matrix C

```
 1.0000000000 + 2.0000000000j  4.0000000000 + 5.0000000000j  7.0000000000 + 3.0000000000j
 3.0000000000 + 4.0000000000j  7.0000000000 + 5.0000000000j  3.0000000000 + 5.0000000000j
 9.0000000000 + 4.0000000000j  3.0000000000 + 1.0000000000j  3.0000000000 + 6.0000000000j
```

Comment:
Again, I did not loose any significant digits in the process.

Inversion of Plant Matrix for MIMO QFT

Fred L. Trevino, Capt
EEE 799: Masters Thesis
Dr Gary I Lamont

The Matrix to be inverted is given by the following transfer functions:

MIMO Plant [1,1] Plant Matrix 1

$$\frac{1.0000}{s + 1.0000}$$

MIMO Plant [1,2] Plant Matrix 1

$$\frac{0.2000}{s^2 + 3.0000 + 2.0000}$$

MIMO Plant [2,1] Plant Matrix 1

$$\frac{0.5000}{s + 1.0000}$$

MIMO Plant [2,2] Plant Matrix 1

$$\frac{0.5000 (s + 0.0000)}{s^2 + 3.0000 + 2.0000}$$

The Combined LU Decomposed Matrix is given by the following transfer functions:
Note that the diagonal ones of the Lower Triangular matrix are assumed

LUD Plant [1,1] L/U Matrix: 1

$$\frac{1.0000}{s + 1.0000}$$

LUD Plant [1,2] L/U Matrix: 1

$$\frac{0.2000}{s^2 + 3.0000 + 2.0000}$$

LUD Plant [2,1] L/U Matrix: 1

$$\frac{0.5000}{1.0000}$$

LUD Plant [2,2] L/U Matrix: 1

$$\frac{0.5000 (s - 0.2000)}{s^2 + 3.0000 + 2.0000}$$

Now that the matrix is decomposed, we use forward and backsubstitution
to process the inverse. We input the columns of the identity matrix
one column at a time and get the inverse one column at a time. The following
text are the results of the internal multiplications, divisions, etc showing
how roots are internally cancelled and the growth of large order polynomials
is prevented.

Forward Substitution Process----------

Forward Substitution Process----------

*** Multiplication ***

First Operand: LUmat[2, 1]

$$\frac{0.5000}{1.0000}$$

Second Operand: BMat [1]

$$\frac{1.0000}{1.0000}$$

Result

$$\frac{0.5000}{1.0000}$$

*** Subtraction ***

First Operand

$$\frac{0.0000}{1.0000}$$

Second Operand

$$\frac{0.5000}{1.0000}$$

sum variable (sum = sum - a[i,j]*b[j])

$$\frac{-0.5000}{1.0000}$$

Back Substitution Process----------

Division
First Operand: LUmat[2, 2]

sum variable (sum = sum - a[i,j]*b[j])

$$\frac{-0.5000}{1.0000}$$

LUD Plant [2,2] L/U Matrix: 1

$$\frac{0.5000\ (\ s\ -\ 0.2000\ )}{s^2 + 3.0000s + 2.0000}$$

Answer: Element of Q

$$\frac{-1.0000\ (\ s^2 + 3.0000s + 2.0000\ )}{s\ -\ 0.2000}$$

Back Substitution Process----------

*** Multiplication ***

First Operand: LUmat[1, 2]

$$\frac{0.2000}{s^2 + 3.0000s + 2.0000}$$

Second Operand: BMat [2]

$$\frac{-1.0000\ (\ s^2 + 3.0000s + 2.0000\ )}{s\ -\ 0.2000}$$

Result

$$\frac{-0.2000}{s\ -\ 0.2000}$$

*** Subtraction ***

First Operand

$$\frac{1.0000}{1.0000}$$

Second Operand

$$\frac{-0.2000}{s\ -\ 0.2000}$$

sum variable (sum = sum - a[i,j]*b[j])

$$\frac{s + 0.0000}{s\ -\ 0.2000}$$

Division

First Operand: LUmat[1, 1]

$$\frac{s + 0.0000}{s\ -\ 0.2000}$$

Second Operand:

$$\frac{1.0000}{s\ +\ 1.0000}$$

Answer: Element of Q

$$\frac{s^2 + 1.0000s + 0.0000}{s\ -\ 0.2000}$$

Forward Substitution Process----------

Forward Substitution Process----------

Back Substitution Process----------

Division
First Operand: LUmat[2, 2]

One

$$\frac{1.0000}{1.0000}$$

LUD Plant [2,2] L/U Matrix: 1

$$\frac{0.5000\ (\ s\ -\ 0.2000\ )}{s^2 + 3.0000s + 2.0000}$$

Answer: Element of Q

$$\frac{2.0000\,(s^2 + 3.0000s + 2.0000)}{s - 0.2000}$$

Back Substitution Process----------

*** Multiplication ***

First Operand: LUmat[1, 2]

$$\frac{0.2000}{s^2 + 3.0000s + 2.0000}$$

Second Operand: BMat [2]

$$\frac{2.0000\,(s^2 + 3.0000s + 2.0000)}{s - 0.2000}$$

Result

$$\frac{0.4000}{s - 0.2000}$$

*** Subtraction ***

First Operand

$$\frac{0.0000}{1.0000}$$

Second Operand

$$\frac{0.4000}{s - 0.2000}$$

sum variable (sum = sum - a[i,j]*b[j])

$$\frac{-0.4000}{s - 0.2000}$$

Division
First Operand: LUmat[1, 1]

$$\frac{-0.4000}{s - 0.2000}$$

Second Operand:

$$\frac{1.0000}{s + 1.0000}$$

Answer: Element of Q

$$\frac{-0.4000\,(s + 1.0000)}{s - 0.2000}$$

The Inverted Matrix is given by the following transfer functions:

Answer: Element of Q

2

$$\frac{s + 1.0000s + 0.0000}{s - 0.2000}$$

Answer: Element of Q

$$\frac{-0.4000 (s + 1.0000)}{s - 0.2000}$$

Second Operand: BMat [2]

$$\frac{-1.0000 (s^2 + 3.0000s + 2.0000)}{s - 0.2000}$$

Second Operand: BMat [2]

$$\frac{2.0000 (s^2 + 3.0000s + 2.0000)}{s - 0.2000}$$

The following is the resulting Q Matrix Answer:

q-Plant [1,1] Q Matrix: 1

$$\frac{s^2 + 1.0000s + 0.0000}{s - 0.2000}$$

q-Plant [1,2] Q Matrix: 1

$$\frac{-0.4000 (s + 1.0000)}{s - 0.2000}$$

q-Plant [2,1] Q Matrix: 1

$$\frac{-1.0000 (s^2 + 3.0000s + 2.0000)}{s - 0.2000}$$

q-Plant [2,2] Q Matrix: 1

$$\frac{2.0000 (s^2 + 3.0000s + 2.0000)}{s - 0.2000}$$

Root Finder Test

```
Fred L. Trevino
EENG 799
Root Finding Test of IceCap-PC
IceCap-PC vs Matlab 3.26
```

```
ICECAP-PC
Polynomial A

GAIN  :   1.00000000000000000
          s^6      s^5      s^4       s^3       s^2       s^1       s^0
POLY  :  1.0000  12.0000  58.00000  144.00000  193.00000  132.00000  36.00000
ROOTS :  -3.00000000000000000
         -3.00000000000000000
         -2.00000000000000000
         -2.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
Internal root representation accurate within:  3.40000000E-4932

MATLAB
e =
1    12    58   144   193   132    36
» roots(e)

ans =
-3.00000000000005 + 0.000000280521561
-3.00000000000005 - 0.000000280521561
-1.99999999999996 + 0.000000231484971
-1.99999999999996 - 0.000000231484971
-0.99999999999999 + 0.000000068528331
-0.99999999999999 - 0.000000068528331

ICECAP-PC
Polynomial A

GAIN  :   1.00000000000000000
          s^7      s^6      s^5      s^4      s^3      s^2      s^1      s^0
POLY  :  1.0000  13.0000  70.0000  202.0000 337.0000  325.0000  168.0000  36.0000
ROOTS :  -3.00000000000000000
         -3.00000000000000000
         -2.00000000000000000
         -2.00000000000000000
         -1.00004935264587402
         -0.99972993915088197
         -1.00022070820324401
Internal root representation accurate within:  2.23355772E-0006



ICECAP-PC
Polynomial A

GAIN  :   1.00000000000000000
          s^6      s^5      s^4       s^3       s^2       s^1       s^0
POLY  :  1.0000  18.0000  123.0000  396.0000  615.0000  450.0000  125.0000
ROOTS :  -5.00000000000000000
         -5.00000000000000000
         -5.00000000000000000
         -1.00035554842761348
         -0.99964451789855957
         -1.00000000000000000
Internal root representation accurate within:  7.50811614E-0006

MATLAB
» a = [1 18 123 396 615 450 125]
» roots(a)
ans =

-5.00006006048980
-4.99996996975510 + 0.000052013708981
-4.99996996975510 - 0.000052013708981
-1.00000494089214 + 0.000008557997461
-1.00000494089214 - 0.000008557997461
```

```
-0.99999011821573


ICECAP-PC
Polynomial A

GAIN  :   1.00000000000000000
          s^6      s^5       s^4       s^3       s^2       s^1       s^0
POLY  :  1.0000   10.0000   41.0000   88.0000  104.0000   64.0000   16.0000
ROOTS :  -2.00000000000000000
         -2.00000000000000000
         -2.00000000000000000
         -2.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
Internal root representation accurate within:  3.40000000E-4932

» b = [1 10 41 88 104 64 16]
» roots(b)
ans =

-2.00024785812781 + 0.00024789014765i
-2.00024785812781 - 0.00024789014765i
-1.99975214187220 + 0.00024782611280i
-1.99975214187220 - 0.00024782611280i
-1.00000000000000 + 0.00000004785137i
-1.00000000000000 - 0.00000004785137i

ICECAP-PC
Polynomial A

GAIN  :    1.00000000000000000
          s^6      s^5       s^4        s^3        s^2        s^1        s^0
POLY  :  1.0000   21.0000  175.0000  735.00000 1624.0000  1764.0000  720.0000
ROOTS :  -6.00000000000000000
         -5.00000000000000000
         -4.00000000000000000
         -3.00000000000000000
         -2.00000000000000000
         -1.00000000000000000
Internal root representation accurate within:  3.40000000E-4932

» c = [1 21 175 735 1624 1764 720]
» roots(c)
ans =
-5.99999999999991
-5.00000000000039
-3.99999999999940
-3.00000000000039
-1.99999999999991
-1.00000000000001

ICECAP-PC
Polynomial A

GAIN  :  1.00000000000000000
          s^6      s^5       s^4       s^3       s^2       s^1       s^0
POLY  :  1.0000   6.0000   15.0000   20.0000   15.0000   6.0000   1.0000
ROOTS :  -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
Internal root representation accurate within:  3.40000000E-4932
```

```
MATLAB
» d = [1 6 15 20 15 6 1]
» roots(d)
ans =

-1.00331670554264
-1.00166155604821 + 0.002872350928871
-1.00166155604821 - 0.002872350928871
-0.99834164733085 + 0.002877899880591
-0.99834164733085 - 0.002877899880591
-0.99667688769924


ICECAP-PC
Polynomial A

GAIN  :  1.00000000000000000
         s^3                  s^2                  s^1                  s^0
POLY  :  1.00000000000000000  2.00000000000000000  3.00000000000000000  4.00000000000000000
ROOTS :  -0.17468540370464325-j1.54686890065397160
         -0.17468540370464325+j1.54686890065397160
         -1.65062919143938822
Internal root representation accurate within:  6.82119357E-0008

MATLAB
» f = [1 2 3 4]
» roots(f)
ans =

-1.65062919143939
-0.17468540428031 + 1.546868887231401
-0.17468540428031 - 1.546868887231401

ICECAP-PC
Polynomial A

GAIN  :  1.00000000000000000
         s^4                  s^3                  s^2                  s^1                  s^0
POLY  :  1.0000  2.00000000000000000  3.00000000000000000  4.00000000000000000  5.00000000000000000
ROOTS :  -1.28781545162200928-j0.85789686668854613
         -1.28781545162200928+j0.85789686668854613
         +0.28781548142433167-j1.41609309864941254
         +0.28781548142433167+j1.41609309864941254
Internal root representation accurate within:  3.65869539E-0007

MATLAB
» g = [1 2 3 4 5]
» roots(g)
ans =
 0.28781547955765 + 1.416093080171911
 0.28781547955765 - 1.416093080171911
-1.28781547955765 + 0.857896758328491
-1.28781547955765 - 0.857896758328491i
```

Root Finder Coefficient Reconstruction Error Test
Fred L. Trevino, Capt, USAF
Electrical Engineer

```
Unpolished Polynomial

Polynomial A
GAIN  :   1.0000
          s^6      s^5      s^4      s^3      s^2      s^1      s^0
POLY  :  1.0000  12.0000  58.0000  144.0000  193.0000  132.0000  36.0000
ROOTS :  -0.99999986114853990
         -1.00000013885151794
         -1.99999958484187777
         -2.00000041515827876
         -2.99999971614412451
         -3.00000028385566112
Internal root representation accurate within:  2.40238385E-0013 %

Unpolished Accuracy:  2.40238384741076E-0013

Polished Polynomial

Polynomial A
GAIN  :   1.0000
          s^6      s^5      s^4      s^3      s^2      s^1      s^0
POLY  :  1.0000  12.0000  58.0000  144.0000  193.0000  132.0000  36.0000
ROOTS :  -1.00000000000000000
         -1.00000000000000000
         -2.00000000000000000
         -2.00000000000000000
         -3.00000000000000000
         -3.00000000000000000
Internal root representation accurate within:  0.00000000E+0000 %

Polished Accuracy:  0.00000000000000E+0000

Choosing Polished Polynomial


--------------------------------------------------------------------------

Unpolished Polynomial

Polynomial A
GAIN  :   1.0000
          s^7      s^6      s^5      s^4      s^3      s^2      s^1      s^0
POLY  :  1.0000  13.0000  70.0000  202.0000  337.0000  325.0000  168.0000  36.0000
ROOTS :  -0.99995061525371413
         -1.00002469237289833
         -1.00002469786131161
         -1.99991445853344107
         -2.00008554146667941
         -2.99995722634657145
         -3.00004276816538401
Internal root representation accurate within:  7.31758333E-0009 %

Unpolished Accuracy:  7.31758333094579E-0009

Polished Polynomial

Polynomial A
GAIN  :   1.0000
          s^7      s^6      s^5      s^4      s^3      s^2      s^1      s^0
POLY  :  1.0000  13.0000  70.0000  202.0000  337.0000  325.0000  168.0000  36.0000
ROOTS :  -1.00000114885965188
         -1.00000082939761701
         -0.99999965957871331
         -2.00000000000000000
         -2.00000000000000000
         -3.00000000000000000
         -3.00000000000000000
Internal root representation accurate within:  5.89621054E-0005 %

Polished Accuracy:  5.89621054182185E-0005
```

```
Choosing Unpolished Polynomial

: ------------------------------------------------------------------

Unpolished Polynomial

Polynomial A
GAIN  :    1.0000
         s^6      s^5      s^4       s^3       s^2       s^1       s^0
POLY  :  1.0000  18.0000  123.0000  396.0000  615.0000  450.0000  125.0000
ROOTS :  -0.99997979938543333
         -1.00001010026207655
         -1.00001010058202872
         -4.99893003491269068
         -5.00053498242888536
         -5.00053498242888536
Internal root representation accurate within:  4.30870321E-0006 %

Unpolished Accuracy:  4.30870321250360E-0006

Polished Polynomial

Polynomial A
GAIN  :    1.0000
         s^6      s^5      s^4       s^3       s^2       s^1       s^0
POLY  :  1.0000  18.0000  123.0000  396.0000  615.0000  450.0000  125.0000
ROOTS :  -0.99999951613993230
         -1.00000033103686999
         -0.99999984246064752
         -5.00000171802838821
         -5.00000021689256729
         -4.99999930827796170
Internal root representation accurate within:  7.71537733E-0006 %

Polished Accuracy:  7.71537733115779E-0006

------------------------------------------------------------------

Unpolished Polynomial

Polynomial A
GAIN  :    1.0000
         s^6      s^5      s^4      s^3      s^2       s^1      s^0
POLY  :  1.0000  10.0000  41.0000  88.0000  104.0000  64.0000  16.0000
ROOTS :  -0.99999959752956686
         -1.00000040247108108
         -1.99971085500699006
         -1.99971085500699006
         -2.00000024217795799
         -2.00057804780741396
Internal root representation accurate within:  1.00259883E-0006 %

Unpolished Accuracy:  1.00259883497114E-0006

Polished Polynomial

Polynomial A
GAIN  :    1.0000
         s^6      s^5      s^4      s^3      s^2       s^1      s^0
POLY  :  1.0000  10.0000  41.0000  88.0000  104.0000  64.0000  16.0000
ROOTS :  -1.00000000000000000
         -1.00000000000000000
         -2.00076234204481064
         -1.99937451021959366
         -2.00000000000000000
         -2.00000000000000000
Internal root representation accurate within:  1.09291077E-0003 %

Polished Accuracy:  1.09291076660156E-0003

Choosing Unpolished Polynomial

------------------------------------------------------------------

Unpolished Polynomial
```

```
Polynomial A
GAIN  :    1.0000
         s^6      s^5      s^4       s^3       s^2       s^1       s^0
POLY  :  1.0000  21.0000  175.0000  735.0000  1624.0000  1764.0000  720.0000
ROOTS :  -0.99999999999999868
         -2.00000000000000659
         -2.99999999999998685
         -4.00000000000001295
         -4.99999999999999379
         -6.00000000000000114
Internal root representation accurate within:  1.59872116E-0013 %

Unpolished Accuracy:  1.59872115546023E-0013

Polished Polynomial

Polynomial A
GAIN  :    1.0000
         s^6      s^5      s^4       s^3       s^2       s^1       s^0
POLY  :  1.0000  21.0000  175.0000  735.0000  1624.0000  1764.0000  720.0000
ROOTS :  -2.00000000000000000
         -1.00000000000000000
         -4.00000000000000000
         -3.00000000000000000
         -6.00000000000000000
         -5.00000000000000000
Internal root representation accurate within:  0.00000000E+0000 %

Polished Accuracy:  0.00000000000000E+0000

-------------------------------------------------------------------

Unpolished Polynomial

Polynomial A
GAIN  :  1.0000
         s^6     s^5     s^4      s^3      s^2      s^1     s^0
POLY  :  1.0000  6.0000  15.0000  20.0000  15.0000  6.0000  1.0000
ROOTS :  -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
Internal root representation accurate within:  0.00000000E+0000 %

Unpolished Accuracy:  0.00000000000000E+0000

Polished Polynomial

Polynomial A
GAIN  :  1.0000
         s^6     s^5     s^4      s^3      s^2      s^1     s^0
POLY  :  1.0000  6.0000  15.0000  20.0000  15.0000  6.0000  1.0000
ROOTS :  -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
         -1.00000000000000000
Internal root representation accurate within:  0.00000000E+0000 %

Polished Accuracy:  0.00000000000000E+0000

Choosing Polished Polynomial

 -------------------------------------------------------------------

Unpolished Polynomial

Polynomial A
GAIN  :  1.0000
         s^3     s^2     s^1     s^0
POLY  :  1.0000  2.0000  3.0000  4.0000
ROOTS :  -0.17468540428030588-j1.54686888723139627
         -0.17468540428030589+j1.54686888723139628
```

```
        -1.65062919143938823
Internal root representation accurate within:  2.36419161E-0017 %

Unpolished Accuracy:  2.36419160587147E-0017

Polished Polynomial

Polynomial A
GAIN  :  1.0000
         s^3      s^2      s^1      s^0
POLY  :  1.0000   2.0000   3.0000   4.0000
ROOTS :  -0.17468540428030588-j1.54686888723139624
         -0.17468540428030588+j1.54686888723139624
         -1.65062919143938822
Internal root representation accurate within:  2.19008839E-0016 %

Polished Accuracy:  2.19008838842072E-0016

Choosing Unpolished Polynomial

================================================================

Unpolished Polynomial

Polynomial A
GAIN  :  1.0000
         s^4      s^3      s^2      s^1      s^0
POLY  :  1.0000   2.0000   3.0000   4.0000   5.0000
ROOTS :  -1.28781547955764792+j0.85789675832849048
         -1.28781547955764819-j0.85789675832849027
         +0.28781547955764808+j1.41609308017190785
         +0.28781547955764803-j1.41609308017190806
Internal root representation accurate within:  1.61971028E-0015 %

Unpolished Accuracy:  1.61971028480621E-0015

Polished Polynomial

Polynomial A
GAIN  :  1.0000
         s^4      s^3      s^2      s^1      s^0
POLY  :  1.0000   2.0000   3.0000   4.0000   5.0000
ROOTS :  -1.28781547955764797-j0.85789675832849028
         -1.28781547955764797+j0.85789675832849028
         +0.28781547955764797-j1.41609308017190794
         +0.28781547955764797+j1.41609308017190794
Internal root representation accurate within:  1.58293526E-0016 %

Polished Accuracy:  1.58293526465436E-0016

Choosing Unpolished Polynomial
```

Appendix C

ICECAP-PC Users Manual

Refer to Bell, 1992 for the ICECAP-PC Users Manual.

This section left intentionally blank

Appendix D

QFT Users Manual

ICECAP - PC

Version 10.0 - MS-DOS

Interactive Control Engineering Computer Analysis Package
for the Personal Computer

QFT TOOLBOX

Quantitative Feedback Theory (QFT)

User's Manual v 2.0

(Preliminary)

Professor Gary B. Lamont
Department of Electrical and Computer Engineering
School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB
Dayton, Ohio 45433-6583

PREFACE

WELCOME to the QFT Toolbox of ICECAP-PC. This Toolbox includes both multiple-input multiple-output (MIMO) and multiple-input single output (MISO) models in both continuous and discrete domains. ICECAP-PC is an ongoing development of a public domain computer-aided design (CAD) package for students, faculty and practitioners of control engineering and digital signal processing with special emphasis on education. Source code and executable files are available. If you are interested in adding additional code or have suggestions for improvement please contact:

Professor Gary B. Lamont
Department of Electrical and Computer Engineering
School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB OH 45433-6583
(513) 255-3450
(ARPANET lamont@afit.af.mil)

QFT Notation as used in ICECAP-PC is consistent with D'Azzo & Houpis[#] and Houpis & Lamont[#].

SYSTEM REQUIREMENTS

ICECAP-PC is designed to work on MS-DOS$_{tm}$ 80286/80386/80486 computers with a minimum of 640 KB of RAM and 5MB of hard disk space. ICECAP-PC supports all graphics terminals including Hercules$_{tm}$, EGA and VGA. Operation is greatly enhanced with expanded memory and a math co-processor. Installation of ICECAP-PC and its QFT Toolbox is done automatically with a provided install procedure (see ICECAP-PC manual).

Edited By: Gary B. Lamont , Wayne E. Bell, and Fred Trevino; 1992

TABLE OF CONTENTS

A. Introduction

The environmental objective of ICECAP-PC along with its associated toolboxes is to provide a 4th generation language (4GL) interface for control engineering students and practioners. Thus associated functions and operations used by this group do not have to be individually coded and combined over-and-over again, but are integrated into a window based and object-oriented implementation that is very user friendly as well as effective! The interface provides pull-down menus, speed-buttons, mouse support, "hot-keys" and on-line HELP.

ICECAP-PC and the QFT toolbox was developed using Borland's$_{tm}$ object-oriented TurboVision language (a 4th generation language) under Turbo Pascal Ver 6.0. Top pull-down menu selections produce dialog boxes from which the user defines an operation. The pull-down menu is always accessible providing a truly "event driven" and nimble interface. System parameters are quickly modified with recall of past inputs providing efficient interactive analysis and synthesis of the controller. The QFT toolbox is invoked by selecting either the *Toolbox - MIMO QFT* or *Toolbox - MISO QFT* menu options.

The overviews of the MIMO and MISO model QFT design techniques are presented in the following sections. For supporting theoretical studies please consult references.

B. The QFT MISO Design (QFD) Process

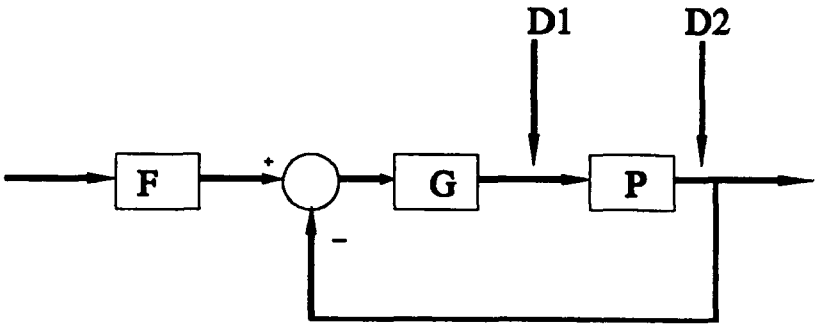


Figure 1: MISO QFT System

Consider designing a practical linear time invariant feedback controller for a plant model with uncertainty in parameter and disturbance. Due to Plant Uncertainty, there is a set {P} of plants. Consider, for example, a second-order plant model with the following variations:

$$P = \frac{k}{As^2 + Bs + C} \qquad (1)$$

$$A = [1\text{-}4]$$
$$B = [2\text{-}6]$$
$$C = [10\text{-}20]$$
$$k = [50\text{-}84]$$

The set {P} of plants consist of all possible combinations of plant variations which could be a very large number of specific plants. The QFT method, developed by Dr Isaac Horowitz, quantitatively defines the problem in the form of (1) sets $\{T_R\}$ of acceptable command or tracking input/output relations and (2) sets $\{T_D\}$ of acceptable disturbance input/output relations and (3) a set of {P} of possible plants. The design objective is to guarantee that the control ratio, $T_R = Y/R$, is a member of $\{T_R\}$ for all P in {P}. Although this technique can be used for a variety of system structures, this package only emphasizes structured uncertainty including non-minimum phase models.

The QFT design (QFD) approach is a frequency domain technique that provides robust performance despite plant uncertainties and disturbances. The general model has three inputs: a tracking input R(s), a plant disturbance D_1 and a measurement disturbance D_2 as shown in figure 3-10. ICECAP-PC currently handles only the case of plant disturbance D1. P is the symbol for the plant, G is the compensator to be designed and F is a pre-filter that also requires design. L = GP is defined as the loop transmission (open-loop transfer function). If only G as

a design parameter is available, it called a single degree of freedom control loop. If F is added, two degrees of freedom are available. Thus, the QFT method is a two degree of freedom design.

This approach has sufficient generality for modelling a variety of systems. The closed-loop transfer function for a tracking input is by definition

$$T_R(s) = \frac{FGP}{1+GP} = \frac{FL}{1+L} \qquad (2)$$

$$L = PG \text{ is Loop Transmission}$$

Specifications

To design a controller, various closed-loop performance bounds are specified which must be satisfied for all plant variations. The various specifications are generally in terms of the frequency response of the following tracking and disturbance response transfer functions.

Tracking Specification:

The closed loop tracking transfer function for plant P_j is given by

$$T_{R_j}(\omega) = \frac{FGP_j}{1+GP_j} = \frac{FL_j}{1+L_j} \quad \forall P_j \in \{P\} \qquad (3)$$

$$B_L(\omega) < |T_{R_j}(\omega)| < B_U(\omega)$$

where all T_{Rj} responses (one for each plant) must lie between the B_U and B_L specifications and where the maximum tracking error is defined as (D'Azzo, pg 429, 692)

$$\delta_r(\omega) = B_U(\omega) - B_L(\omega) \qquad (4)$$

There are two methods of deriving δ_r. In the first method, the time domain tracking specifications (Mp-max, Mp-min, Ts-max, Tx-min, etc) lead to two transfer functions, Tru and Trl, which describe the upper and lower bounds permitted in both time and frequency domains. δ_r is then defined as the difference between the frequency responses $\delta_r = B_U(w) - B_L(w) =$ Tru(w) - Trl(w). In the second method, $B_L(w)$ and $B_U(w)$ are input directly in the frequency domain as a set of data points.

Successful QFT design is enhanced if $\delta_R(w)$ monotonically increases with frequency. Using the augmented model (D'Azzo, 693), this is accomplished for high frequencies by adding a zero to Tru and a pole to Trl at w_h.

If the MISO tracking relationships are part of a larger MIMO problem, the tracking bounds B_u and B_l must be constricted to account for the tracking responses of other plant element modeled as disturbance inputs to the MISO loop. Currently, this is done manually off line and is not addressed in ICECAP-PC.

Stability Specifications (Phase Margin, Gain Margin)
The constraint on the distance from the -1+j0 point is given by (Yaniv, 2; D'Azzo, 309)

$$|1 + GP_j| \geq x \qquad (5)$$

where $x \leq 1$ is a chosen parameter. A larger x for a given frequency results in a smaller steady state sinusoidal error. Another parameter, chosen to reduce the ocsillatory nature of the design, is given by (D'Azzo, 312).

$$\left|\frac{GP_j}{1 + GP_j}\right| \leq y \qquad (6)$$

For an equivalent second order system, larger values of y result in smaller values of zeta.

These equations relate to the desired gain margin, phase margin, peak overshoot and M_l contour (nichols plot) as given by the following.

$$gm = \frac{1}{1 - x}$$
$$\gamma = 180^\circ - 2\cos^{-1}\left(\frac{x}{2}\right) \qquad (7)$$

Given a set {P} of all possible plants, the stability specifications describe the region in the frequency domain that none of the plants P in {P} should violate. In the QFT design technique, the set of {P} plants form a frequency domain template, or a region of possible responses over the parameter variation range. All plants P in {P} must remain outside the stability margins in the QFT design. Phase margin (τ), gain margin (gm) and peak overshoot (Mp) are all mathematically related and each can be derived from the other for a second order or equivalent second order system as shown in eq 8.

$$gm = -20\,Log_{10}\frac{M_p}{1 - M_p}$$

$$M_1 = 20\,Log_{10}\frac{10^{-\frac{gm}{20}}}{1 + 10^{-\frac{gm}{20}}} \qquad (8)$$

$$Mp = e^{\frac{Ln10M_1}{20}} = \frac{10^{-\frac{gm}{20}}}{1 - \frac{gm}{20}}$$

$$\gamma = 180^\circ - \cos^{-1}\sqrt{1-10^{-\frac{M_L}{10}}}$$

Disturbance Specification

The disturbance transfer functions for the two disturbance inputs of figure 1 are given by the relations of equations 9 and 10 (D'Azzo, 446). $M_D(w)$ is the upper disturbance bound defined in the specifications. Recent QFT designers have begun to define $M_D(w)$ as a constant (i.e -20db) rather than a transfer function. Using a constant allows for both easier and more conservative design.

Plant Disturbance D_1

$$T_{D_{1_j}} = \frac{P_j}{1 + L_j} \quad \forall\, P_j \in \{P\} \qquad (9)$$

$$| T_{D_{1_j}}(\omega) | < | M_D(\omega) |$$

Measurement Disturbance D_2

$$T_{D_{2_j}} = \frac{P_j}{1 + L_j} \quad \forall\, P_j \in \{P\} \qquad (10)$$

$$| T_{D_{2_j}}(\omega) | < | M_D(\omega) |$$

The Design Process

As mentioned above, the objective of the QFT technique is to find a G & F to guarantee that the closed-loop response is within prescribed limits of the various bounds despite plant uncertainty and disturbance inputs as represented in the set notation. The ICECAP-PC QFT package closely follows the design procedure specified by Dr Horowitz and Dr Houpis (D'Azzo, pg728). This procedure is summarized as follows:

1. Synthesize upper and lower tracking response transfer functions Tru and Trl to meet minimum and maximum specifications.
2. Synthesize T_D, the upper disturbance response transfer function.

3. Define a set of plants $\{P_j\}$ from all possible $\{P\}$ such that the frequency response of the set $\{P_j\}$ defines the perimeter of all possible frequency responses of $\{P\}$.
4. Select a representative nominal plant P_o from the set $\{P_j\}$. Normally, the best selection is the lower-left plant as seen on a nichols chart template.
5. Determine the disturbance bounds $B_D(w)$ on the loop transmission $L_0(w)$.
6. Determine the tracking bound $B_R(w)$.
7. Define the composite bounds as the most restrictive of the bounds determined in steps 6 and 7.
8. Design the loop transmission $L_0(w)$ for the nominal plant $P_0(w)$ to meet, as closely as possible, the composite bounds determined in step 7.
9. Synthesize the prefilter F(w) that satisfies eq 3-3.
10. Simulate system behavior to verify correct design.

At the conclusion of the design process, a formatted report is available at this point upon user request. This is a ascii text file containing all the design calculations that is easily edited with any number of text editors.

This CAD program allows one to view the high level pull-down menu capabilities of ICECAP-PC QFT for the various phases of design (File, Specs, Plant, Disturbance, Templates Bounds, L-Zero, Filter, Simulation). The various menu items corresponding to the high-level design steps are initiated by using the arrow keys, mouse or control keys. For example, the introductory menu can be accessed by pressing the first letter of the desired menu selection. For example, the "Specs" menu is pulled down by S. F1 displays a window with help information at the selected level. Although a detailed step-by-step example could be provided, the use of the keys or mouse is self-evident. The QFT MISO example in section B details each step of the design process as implemented in the ICECAP-PC QFT Toolbox.

Note that during the design process, it is helpful to develop a commented log file of the unfolding design. ICECAP-PC provides this capability allowing the user to open a text log file and enter comments at any point of the design.

Step 0. File Operations

ICECAP-PC data files are generated and stored in binary format allowing rapid access to large amounts of data. On the other hand, all output files such as time and frequency response data, etc are stored in ASCII form to allow easy interface to other graphics programs. Additionally, ICECAP-PC can read ASCII files generated from other popular engineering programs. Such files are checked for proper structure before the import process begins. ICECAP-PC sessions can be saved and recalled by user defined names thus returning the engineer to the place he/she left off during an interrupted design process. At the end of the design process, a printed report of the file contents presents the complete design record.

Step 1. QFD Specifications

Three operational domains are available consisting of the continuous, discrete, or sampled data domains (s, z or w). Specifications in each case are defined in either the time domain or frequency domain.

A priori synthesized tracking specifications can either be input as s-plane transfer functions or frequency domain data. For transfer function input, the upper tracking function bound (Tru) and the lower tracking function bound (Trl) are interactively synthesized or manually entered using the standard ICECAP-PC transfer function input. Transfer function, in general, are entered in one of two forms: factored (gain, zeros and poles) or polynomial (gain and coefficients). The time domain (Tru, Trl) or frequency domain specifications ($\delta_R(w)$) can be presented in a table or graphical form as requested. The frequency domain data can be directly entered as magnitude limits (max and min) vs frequency.

Selection from the SPEC pull-down menu produces a palette of options, each of which result in a dialog box that define specific actions. The options include system domain (s, z, or w), time-domain or frequency domain tracking and disturbance specification input. A summary of the time and frequency domain dialog commands are:

> Time Domain Tracking Specs Dialog Box
> (TF Specs, TRU and Trl; Display TF Specs; Table of Time-Domain Specs; Graph of Time-Domain Specs)
>
> Frequency Domain Specs
> (Display TF Specs; Table of TF Frequency Specs; Graph of TF Frequency Specs; Delta_R Frequency Specs and Graph; Phase Margin or M_L Specs).

1.1. Time Domain Tracking Specifications

Enter Tru(s) and Trl(s) bounds for step response tracking operations. Usually these bounds are defined in terms of transfer functions that are synthesized from time domain specifications such as rise time, settling time, overshoot (the conventional figures of merit). Alternately, you may synthesize the bounds interactively with the QFT toolboxes.

1.2. Frequency Domain Tracking Specifications

Enter Tru(w) and Trl(w) bounds for the frequency range of interest. These usually are generated in transfer function. $\delta_r(w)$ is the difference between Tru(w) and Trl(w). It is desirable that Tru and Trl be synthesized such that δ_r increases with frequency after Tru crosses the 0 db line. An increasing δ_r permits an effective high frequency design technique. δ_r can also be found from a given set of lower, $B_L(w)$ and upper, $B_U(w)$, bounds which are given for a finite number of frequencies (See Eq 3). Two other values that need consideration are the frequency at which $B_U(w)$ is -12db (w_{hR}) and the frequency at which B_U crosses the 0 db line (w_{zR}).

The discrete frequencies chosen are at the user's discretion. Normally the frequency range of interest is defined by the region two octaves below w_{zR} and two octaves above w_{hR}. Inside of this region, frequencies are normally chosen an octave apart. The magnitudes are entered in db and frequencies in rad/sec. For ultra-high frequencies, see section 4.2.

1.3. Stability Bound Specifications (Phase & Gain Margin)

Stability bounds are entered in one of three ways. Since stability bounds (phase margin, gain margin, tangent M_L contour, and peak overshoot) are mathematically related, entry of any one initiates automatic computation of the others. Entry of phase margin, peak overshoot, and tangent M_L contour is allowed, however computation of these three from a given gain margin is error prone since phase margin is very sensitive to changes in gain margin. Therefore entry of gain margin is not provided.

1.4. Disturbance Specifications

The disturbance specifications can also be input as a synthesized transfer function but usually are defined as a constant magnitude bound (db) or $|M_D(w)|$ in the frequency domain for the plant disturbance and for the measurement disturbance. Entry of disturbance bounds are done exactly as the tracking bounds.

Step 2. Plant Model Descriptions

Plant model parameter variations can be entered in four ways:

1 Individual Models - plant tfs encompassing the variations are entered separately.

2) Variation Models -
   (a) Nominal plant tf and parameter coefficient variations are entered.
   (b) Nominal plant tf and parameter pole/zero/gain variations are entered.

3) Plant Frequency Domain Models - frequency domain data is entered for each specified plant. (in progress)

Currently the first two methods are available. With approach (1), the number of plants is first defined and then each plant is entered separately by specifying the gain and the zeros and poles in factored or unfactored form using the standard ICECAP-PC input dialogue (order,gain, pole/zero-factored or coefficients-polynomial). Each plant transfer function can be displayed and checked for proper entry. Although not yet implemented, the user will have the ability to import plant models from external ASCII files in a future release.

Approach (2a) requests the entry of the maximum and minimum for each numerator and denominator coefficient. Currently the number of possible parameter variations is limited to four because of memory size.

Approach (2b) requests the variation (max,min) in each zero, in each pole and in the plant gain. The user is requested to answer the following question concerning the polynomial form: "Do you desire the transfer function roots (zeros/poles) to be represented as [s/a +1] or [s+a] for real roots or as $[s^2/(a^2 + w^2) + 2as/(a^2 + w^2) + 1]$ or $[s^2 + 2as + a^2 + w^2]$?"

Observe that a user of this package can build a program that uses symbols to characterize the coefficients or roots of the transfer function as related to some physical model. These symbols can then be instantiated with specific values to generate the specific plants. In this way, the QFT toolbox or any toolbox for that manner, could be used to interactively design controllers for a variety of models in a given application (automotive spring control, engine control, vehicle control)."

2.1. Plant Transfer Function Models

First enter the number of plants to be input (note that the 'tab' function highlights each section of the window). The plants are then entered using the standard ICECAP-PC transfer function input dialog (num order, denom order, num gain, zeros, denom gain, poles) for each factored form.

2.2. Plant Parameter Variations

A nominal transfer function is entered with nominal parameters. The number of plant variations is requested. Then the positive variation of each parameter (gain, zero, pole or coefficient) is entered in turn. The variation could be thought of as a uniform or a normal distribution. Depending upon the number of plants requested. a finite number (<= 625) of plants are defined.

Step 3. Disturbance Models

Plant disturbance models are entered using the standard ICECAP-PC format. The measurement disturbance model is not yet implemented. Gain is entered in real units not db.

Step 4. Plant Template Generation

Because of plant uncertainty there exists a set of complex numbers {P(w)} for any frequency w which are defined as 'plant templates'. A Plant template then is a plot on the Nichols chart over the range of plant uncertainty at a specific frequency w.

The commands are: Plant Frequency Responses; Frequency Data of Plants; Output Data of Plant Templates; Graphs of Plant Templates; U-Contour Generation.

4.1 Low, Intermediate and High Frequencies

If individual plants are entered, then the package generates all the frequency domain points for each plant at a given frequency. This data is then sorted by frequency to develop templates of uncertain plant response. The graphical structure of each template (one at each frequency) can also be displayed on the Nichols chart.

If the plant variation ranges were entered, then the package generates a "convex hull" around all the freq domain points and selects only those point on the boundary of the hull for generating the bounds $B_r(w)$. This process reduces the number of plants since the

maximum number of possible plants using the variation method is 625. This hull or template can also be displayed as a table or graphically on the Nichols chart.

The disturbance bounds are generated in a similar manner. However, in most cases, the disturbance bounds are generated from the disturbance specification of a constant over all frequencies. Various templates are usually generated for discrete values of frequency w an octave apart. From the templates,the plant bound $B_r(w)$ and disturbance bound $B_d(w)$ are generated. From $B_r(w)$ and $B_d(w)$, a combined or composite bound $B_c(w)$ is generated.

4.2. U-Contours for Ultra-High Frequency

To help ensure that the synthesized loop transmission L_0 does not yield roots of the closed loop system close to the imaginary axis, an additional constraint is imposed. This constraint is expressed as $|L/(1+L)| < Ml$ (the $\delta_2(w)$ constant) for all w and all plant variations. In some cases this results in an "ash can" structure. The synthesized loop transmission must not penetrate this region on the Nichols chart.

The U-Contour or ultra-high frequency bound (UHwB) which can be displayed in a table, graph and Nichols chart for the desired frequency range. It is based upon the desired phase angle and the maximum variation of magnitude at high frequencies.

Step 5. Tracking Bounds Generation

From the templates,the plant bound $B_r(w)$ and disturbance bound $B_d(w)$ are generated using a selection of two methods (manual, automatic). The manual process of graphically determining the $B_r(w)$ bound on the loop transmission is accomplished using the following steps which involve the movement of the various plant templates and the selection of a nominal plant:

1. Translate (do not rotate) the plant template for a specific frequency w to a major angle division on the Nichols chart.

2. Using the constant M contours on the Nichols chart determine the max and min values of $T_r(w)$ corresponding to the max and min values of M covered by the template, $\delta_M(w)$.

3. Move the template vertically up and down the Nichols chart until the difference, $\delta_M(w)$, is equal to $\delta_r(w)$. At that point the position of the nominal plant is a point on the bound $B_r(w)$. Usually the nominal plant is selected as the lower left point

of the template for each frequency. Thus, for the specific frequency this point position represents the bound of plant performance in order to meet the specifications.

4. Translate the template to the next major angle division and repeat step 2 until the template becomes tangent to the U-contour.

5. Draw a point through all points to construct $B_r(w)$, the tracking bound on $L_0(w)$, for the specific frequency w.

6. Repeat steps 1 to 5 for a discrete number of frequencies over the frequency band of interest.

The above algorithm for generating $B_r(w)$ is basically that has been implemented in this QFD - CAD package for the automated process (note that this is the same algorithm implemented in the original VAX ICECAP QFT package).

An educational interactive $B_r(w)$ generation mode is also available using movable templates on the Nichols Chart. Templates are entered for Nichols Chart display as tp#.dat. Template movement is possible with the use of the arrow keys. δ_r and δ_m are displayed so that the student can move the template to the point where $\delta_r \leq \delta_m$. Points on the $B_r(w)$ curve at a given frequency are stored with the press of the 'insert' key. The 'return' key provides exit from interactive bounds generation.

The plant disturbance bound $B_d(w)$ is determined by finding the minimum value of $L_0(w)$ that satisfies the following relation:

$$\left|\frac{P_0(\omega)}{\delta_3(\omega)}\right| < \left|\frac{P_0(\omega)}{P(\omega)+L_0(\omega)}\right| \qquad \text{Eq 10}$$

across the range of plant uncertainty embodied in P(w). The tracking bounds can be listed and graphed.

For the measurement disturbance the associated disturbance bound is found from the inequality

$$\left|\frac{P_0(\omega)}{\delta_3(\omega)\cdot P(\omega)}\right| < \left|\frac{P_0(\omega)}{P(\omega)+L_0(\omega)}\right| \qquad \text{Eq 11}$$

using basically the same method.

The combined or composite bound B_c is generated by comparing the tracking and disturbance bounds and defining the highest boundary at each major angle division for a select number of frequencies which are an octave apart. All bounds can be displayed on a Nichols chart for the synthesis of L_0 in the next step. See Section 4.2 for information on the U-Contour.

Step 6. Loop Transmission Design

In order to develop the loop transmission, the Loop transmission, L_0, can be generated using a variety of techniques. The "loop shaping" is accomplished by keeping the value of $L_0(w)$ above the composite bound B_c at each distinct frequency while not violating the U-contour.

Currently, no CAD algorithm for completely automated design has been implemented. The controller designer can determine the form manually from the Nichols chart and enter the transfer function as indicated. This is accomplished by fitting a curve between the composite bounds and the specific 'M_L' contour on the Nichols chart. An initial L_0 is selected and then modified by adding zeros and poles to meet the above criteria. A good initial L_0 is the nominal plant P_0 itself. Using the nominal plant results in a minimum order controller. The nominal plant is normally selected as the plant in the lower left corner of the templates on the Nichols chart. Although, any plant even a different one from the set of plant variations transfer functions may be chosen.

If the designed controller satisfies the bounds, then the closed-loop system output should be within the tracking tolerances as well as rejecting the disturbances within the required specification. This performance requirement can be analyzed by doing a closed-loop time or frequency domain analysis at this point in the design process. If not within specifications then a filter (prefilter) must be designed (step 8).

6.1. Loop Transmission - Manual Design

The designer inputs the L_0, the loop transmission transfer function from the manual design process. The manual design process consists of starting with an initial form such as plant P_0. Then determining if it meets the bound specifications. This is easily accomplished by either using a Nichols chart or a Bode diagram. In a Bode diagram, the composite bounds, $B_C(w)$, give the necessary magnitude and angle conditions for which L_0 must be designed. Overdesign is minimized by placing L_0 as close as possible to the bound. Zeros and poles are added to $L_0(w)$ in order to meet and approach the composite bound. The U-contour corresponds to the phase margin bound on the Bode diagram while the composite bound $B_c(w)$ on the Nichols chart defines the amplitude bound.

6.2 Loop Transmission - Interactive Design

After entering an initial $L_0(w)$, additional poles and zeros are added interactively. After each addition/subtraction of poles and zeros, an updated graph is generated. A recommended process is to locate the frequency at which the worst violation of the $B_C(w)$ bounds occurs. After finding this frequency, install one zero in $L_0(w)$ for each 12 db of violation at a frequency 1 octave (approx) below the violation frequency. An equal number of poles also needs to be installed 1 octave below this frequency to offset the zeros. Repeat this process until the $B_c(w)$ bounds are met while remaining outside and in close proximity to the U-Contour to achieve minimum overdesign.

Poles are entered with 'shift p' and zeros are entered with 'shift z' commands. Entering the 'return key' updates the Nichols chart with the new poles and zeros and stores the data to file. Use of the 'esc key' exits the procedure.

6.3 Loop Transmission - Automated Design

An automated design of loop transmission is under current development. This process will provide the necessary tools for the user to define a desired $L_0(w)$ at a distinct set of frequencies, a loop transmission order constraint, and a cost function such as a weighted least squares solution. The optimization process generates the transfer function coefficients that minimize the cost function.

Step 7. The Controller

From L_0 and the nominal plant, the compensator G is generated from the equation Lo = G * Po. This G is then employed as the robust controller. The CLTF time responses can be generated for checking the design.

Step 8. Filter Generation

The design of the controller G only guarantees the frequency responses for any given p in {P} lies within a δ_R without reguard to the acutal position of δ_R in the frequency domain. A prefilter translates the magnitude progression to lie between B_U and B_L frequency responses (D'Azzo, 725).

In order to meet the original tracking frequency specifications for $T_r(w)=F\cdot Lo(w)/(1+L_0(w))$, a filter is employed to position the frequency domain closed-loop response within the desired envelope of Tru(w) and Trl(w). In some cases this is only a gain.

The process consists of the following 4 steps:

1. Place the nominal point of the plant template P(w) on the point $L_0(w)$. Determine tmax and tmin at that point using the M-contours. The entire perimeter (each plant) of the template must be inspected in order to determine the true t_{min} and t_{max} because the curvilinear nature of the M_L Contour can be deceptive.

2. Obtain the values of Tru and Trl for various frequencies w.

3. From the values obtained in steps 1 and 2, plot [Tru-t_{max}] and [Trl-t_{min}] vs w. These values are the upper and lower bounds on the filter (prefilter). The command Calculate Filter Transfer function generates this table.

4. Synthesis a filter that lies within the frequency plots by the use of straight line approximations. For a step forcing function, the following limit as s goes to zero (or time goes to infinity) must be satisfied:

   lim [F(s)] = 1
   s -> infinity

Filter Input: The filter transfer function is entered based upon manual synthesized design. The Filter can be Displayed and a Frequency Response generated.

Step 9. Complete Closed-Loop Simulation

The closed-loop system with the Filter, F, and the Controller, G, is simulated (step-input) for each plant and each disturbance to ascertain if the system meets the original specifications. Table and graphical data are presented as requested for the time and frequency domains. Non-linear function will be added in the future as appropriate.

9.1. Simulation in the Time Domain

Again the ICECAP-PC simulated input type (step, impulse, sine wave, ramp, user defined, random signal) is requested over the range of time entered for tracking and disturbances. This data can then be displayed using a table or the graph selection for up to ten data sets. They can also be compared to time domain specifications.

9.2. Simulation in the Frequency Domain

The frequency domain data (table and graph) is presented over the range of frequency entered using standard ICECAP-PC deluge for tracking and disturbance models. This

data can then be displayed using the graph selection for up to ten data sets. They can also be compared to frequency domain delta specifications.

Step 10. The Report

A text file suitable for editing is generated using the various data files from each step as appropriate to the selected process for identifying the plant variations. Graphical presentations can be printed using screen dumps and then incorporated into the printed report.

C. QFT MISO Design Example

This discussion is based upon the QFT design approach (see qft tool box) and the report file (QFT_REPORT.TXT) that contains the qft design datafiles that can be generated from the qft toolbox.

Step 1. Specifications:

A second order plant is modeled as

Plant tf =ka / s(s+a) with variations $1 < k < 5$, $2 < a < 10$

The tracking response specifications are

for upper tracking bound Tru - Mp = 1.2 (1.58db - overshoot)
- ts = 1.65 sec (settling time)
for lower tracking bound Trl - ts = 1.65 sec (settling time)

The plant disturbance specification is

max output value to unit step - |Td| = 0.1 (-20 db);

For a Mp = 1.2, zeta = 0.4559 and for ts = 1.65, wn = 5.3169. Thus, a Ts was selected as 1.75 with ts < Ts and zeta = 0.48 with wn = 4.7619 resulting in the following Tru (qfttru):

** Upper Tracking Transfer Function - qfttru **

Numerator: order = 1
constant/gain = 1.889600

polynomial	roots
1.000000	-12.000000 j 0.000000
12.000000	

Denominator: order = 2
constant/gain = 1.000000

polynomial	roots
1.000000	-2.285700 j -4.177500
4.571400	-2.285700 j 4.177500

22.675931

In a similar trail and error process, Trl evolved as

** Lower Tracking Transfer Function - qfttrl **

Numerator: order = 1
constant/gain = 1.1600E4

polynomial	roots
1.000000	

Denominator: order = 4
constant/gain = 1.00000

polynomial	roots
1.00000E+0000	-4.00000E+0000 j 0.00000E+0000
1.15250E+0002	-4.00000E+0000 j 0.00000E+0000
1.59900E+0003	-7.25000E+0000 j 0.00000E+0000
7.51600E+0003	-1.00000E+0002 j 0.00000E+0000
1.16000E+0004	

** Time domain tracking specifications - qfttrut

Time	Magnitude
0.00	0.000000
1.00	1.060103
2.00	1.004076
3.00	0.998920
4.00	1.000034
5.00	0.999972
6.00	0.999967
7.00	0.999968
8.00	0.999968
9.00	0.999968
10.00	0.999968

The desired Tru model specifications are achieved as shown:

RISE TIME: TR= 2.99722818757E-0001
DUPLICATION TIME: TD= 3.98639562532E-0001
PEAK TIME: TP= 6.54828325989E-0001

SETTLING TIME: TS= 1.65130077524E+0000
PEAK VALUE: MP= 1.19723535520E+0000
FINAL VALUE: FV= 9.99967774642E-0001

Similarly, ts for Trl is 1.65.

The frequency domain specifications at the frequencies of interest (observe that selecting these frequencies is an iterative process although octave values are suggested as a minimum):

** Freq domain tracking specifications - qfttruf

Frequency	Magnitude	Phase
0.50	0.058664	-3.433671
1.25	0.359712	-9.197165
2.50	1.270365	-23.060350
5.00	0.581211	-73.185940
10.00	-9.666784	-109.602844
20.00	-18.898683	-107.343144
30.00	-23.253644	-102.916918
35.00	-24.785234	-101.344557
40.00	-26.073534	-100.086578
80.00	-32.421198	-95.248695

** Freq domain tracking specifications - qfttrlf

Frequency	Magnitude	Phase
0.50	-0.155383	-18.481695
1.25	-0.937225	-45.206616
2.50	-3.354850	-84.468469
5.00	-9.873879	-140.135077
10.00	-21.877692	-196.165662
20.00	-37.819826	-238.764487
30.00	-48.111949	-257.923966
35.00	-52.151961	-264.547541
40.00	-55.705826	-270.106893
80.00	-75.101888	-297.756713

** Freq domain tracking error spec - qftdelr (δ_r)

Frequency	Magnitude	Phase

0.50	0.214047	15.048024
1.25	1.296937	36.009451
2.50	4.625216	61.408118
5.00	10.455090	66.949137
10.00	12.210908	86.562818
20.00	18.921143	131.421343
30.00	24.858305	155.007048
35.00	27.366726	163.202984
40.00	29.632293	170.020315
80.00	42.680690	202.508018

The magnitude values should always be increasing for increasing frequency.

Step 2. Plant model variations:

Six plants were chosen to represent the plant variation model. The coefficient variation or pole/zero variation model could have been entered instead.

```
** QFT Plant Explicit Plant Models -  6

 QFTTF1

Numerator:        order = 0
 constant/gain =  2.000000
  polynomial                          roots
   1.000000

Denominator:      order = 2
 constant/gain =  1.000000
  polynomial                          roots
   1.000000                    0.000000  j  0.000000
   2.000000                   -2.000000  j  0.000000
   0.000000

 QFTTF2

Numerator:        order = 0
 constant/gain =  10.000000
  polynomial                          roots
   1.000000
```

```
Denominator:      order = 2
 constant/gain =  1.000000
  polynomial                        roots
   1.000000                    0.000000  j  0.000000
   2.000000                   -2.000000  j  0.000000
   0.000000

 QFTTF3

Numerator:        order = 0
 constant/gain =  30.000000
  polynomial                        roots
   1.000000

Denominator:      order = 2
 constant/gain =  1.000000
  polynomial                        roots
   1.000000                    0.000000  j  0.000000
   6.000000                   -6.000000  j  0.000000
   0.000000

 QFTTF4

Numerator:        order = 0
 constant/gain =  50.000000
  polynomial                        roots
   1.000000

Denominator:      order = 2
 constant/gain =  1.000000
  polynomial                        roots
   1.000000                    0.000000  j  0.000000
   10.000000                  -10.000000  j  0.000000
   0.000000

 QFTTF5

Numerator:        order = 0
 constant/gain =  10.000000
```

```
  polynomial                          roots
   1.000000

Denominator:      order = 2
 constant/gain =  1.000000
  polynomial                          roots
   1.000000                      0.000000  j  0.000000
   10.000000                    -10.000000  j  0.000000
   0.000000

  QFTTF6

Numerator:        order = 0
 constant/gain =  6.000000
  polynomial                          roots
   1.000000

Denominator:      order = 2
 constant/gain =  1.000000
  polynomial                          roots
   1.000000                      0.000000  j  0.000000
   6.000000                     -6.000000  j  0.000000
   0.000000
```

Step 3. Disturbance model:

```
  ** Disturbance Transfer Function Spec - distf **

Numerator:        order = 0
 constant/gain =  0.100000
  polynomial          roots
   1.000000

  Denominator:      order = 0
  constant/gain =  1.000000
   polynomial                         roots
    1.000000
```

** Freq domain disturbance specifications - distff

Frequency	Magnitude	Phase
0.50	-20.000000	0.000000
1.25	-20.000000	0.000000
2.50	-20.000000	0.000000
5.00	-20.000000	0.000000
10.00	-20.000000	0.000000
20.00	-20.000000	0.000000
30.00	-20.000000	0.000000
35.00	-20.000000	0.000000
40.00	-20.000000	0.000000
80.00	-20.000000	0.000000

In order to generate the high frequency u-contour,
V = Pmax - Pmin is generated.

** QFT U-Contour Parameters **

frequency	Pmax(db)	Pmin(db)	V(db)
0.50000	19.98916	5.75731	14.23185
1.25000	11.97387	-3.37030	15.34417
2.50000	5.75731	-12.04544	17.80275
5.00000	-0.96910	-22.58278	21.61368
10.00000	-9.03090	-34.14973	25.11883
20.00000	-19.03090	-46.06381	27.03291
30.00000	-25.56303	-53.08351	27.52048
35.00000	-28.12412	-55.75628	27.63216
40.00000	-30.36629	-58.07264	27.70635
80.00000	-42.21153	-70.10571	27.89418

U-Contour V Height = 27.894180 or 28 db

Mp U-Contour = 3.84 db for a phase margin of 40 degrees

Step 4. Plant template generation:

The plant templates are developed from the six plants. Figure B.1 depicts graphical the 10 templates.

** Plant Templates for Specified Frequencies

tp1.dat

Frequency	Magnitude	Phase
0.50	5.757311	-104.036243
0.50	19.736711	-104.036243
0.50	19.969945	-94.763642
0.50	19.989156	-92.862405
0.50	6.009756	-92.862405
0.50	5.990545	-94.763642
0.50	5.757311	-104.036243

** Plant Templates for Specified Frequencies

tp2.dat

Frequency	Magnitude	Phase
1.25	-3.370301	-122.005383
1.25	10.609100	-122.005383
1.25	11.856680	-101.768289
1.25	11.973866	-97.125016
1.25	-2.005534	-97.125016
1.25	-2.122720	-101.768289
1.25	-3.370301	-122.005383

Fig B.1

** Plant Templates for Specified Frequencies

tp3.dat

Frequency	Magnitude	Phase
2.50	-12.045439	-141.340192
2.50	1.933961	-141.340192
2.50	5.325358	-112.619865
2.50	5.757311	-104.036243
2.50	-8.222090	-104.036243
2.50	-8.654042	-112.619865
2.50	-12.045439	-141.340192

** Plant Templates for Specified Frequencies

tp4.dat

Frequency	Magnitude	Phase
5.00	-22.582780	-158.198591
5.00	-8.603380	-158.198591
5.00	-2.290273	-129.805571
5.00	-0.969100	-116.565051
5.00	-14.948500	-116.565051
5.00	-16.269673	-129.805571
5.00	-22.582780	-158.198591

** Plant Templates for Specified Frequencies

tp5.dat

Frequency	Magnitude	Phase
10.00	-34.149733	-168.690068
10.00	-20.170333	-168.690068
10.00	-11.792964	-149.036243
10.00	-9.030900	-135.000000
10.00	-23.010300	-135.000000
10.00	-25.772364	-149.036243
10.00	-34.149733	-168.690068

** Plant Templates for Specified Frequencies

tp6.dat

Frequency	Magnitude	Phase
20.00	-46.063814	-174.289407
20.00	-32.084414	-174.289407
20.00	-22.873040	-163.300756
20.00	-19.030900	-153.434949
20.00	-33.010300	-153.434949
20.00	-36.852440	-163.300756
20.00	-46.063814	-174.289407

** Plant Templates for Specified Frequencies

tp7.dat

Frequency	Magnitude	Phase
30.00	-53.083509	-176.185925
30.00	-39.104109	-176.185925
30.00	-29.712758	-168.690068
30.00	-25.563025	-161.565051
30.00	-39.542425	-161.565051
30.00	-43.692159	-168.690068
30.00	-53.083509	-176.185925

** Plant Templates for Specified Frequencies

tp8.dat

Frequency	Magnitude	Phase
35.00	-55.756280	-176.729512
35.00	-41.776880	-176.729512
35.00	-32.346087	-170.272421
35.00	-28.124120	-164.054604
35.00	-42.103520	-164.054604
35.00	-46.325487	-170.272421
35.00	-55.756280	-176.729512

** Plant Templates for Specified Frequencies

tp9.dat

Frequency	Magnitude	Phase
40.00	-58.072644	-177.137595
40.00	-44.093243	-177.137595
40.00	-34.636608	-171.469234
40.00	-30.366289	-165.963757
40.00	-44.345689	-165.963757
40.00	-48.616008	-171.469234
40.00	-58.072644	-177.137595

** Plant Templates for Specified Frequencies

tp10.dat

Frequency	Magnitude	Phase
80.00	-70.105713	-178.567904
80.00	-56.126313	-178.567904
80.00	-46.605535	-175.710847
80.00	-42.211533	-172.874984
80.00	-56.190933	-172.874984
80.00	-60.584935	-175.710847
80.00	-70.105713	-178.567904

Step 5. Bounds (tracking, disturbance, composite):

The tracking bounds are achieved using the specified geometric approach which results in the following frequency tables for the 10 frequencies of interest.

** Tracking Bounds for Specified Frequencies

trb1.dat

Frequency	Magnitude	Phase
0.50	30.366686	0.000000
0.50	30.366686	-10.000000
0.50	29.866686	-20.000000
0.50	29.866686	-30.000000
0.50	29.116686	-40.000000
0.50	28.116686	-50.000000

0.50	27.116686	-60.000000
0.50	25.366686	-70.000000
0.50	22.866686	-80.000000
0.50	19.616686	-90.000000
0.50	18.116686	-100.000000
0.50	21.116686	-110.000000
0.50	24.866686	-120.000000
0.50	27.116686	-130.000000
0.50	28.616686	-140.000000
0.50	29.616686	-150.000000
0.50	30.116686	-160.000000
0.50	30.616686	-170.000000
0.50	30.616686	-180.000000

** Tracking Bounds for Specified Frequencies

trb2.dat

Frequency	Magnitude	Phase
1.25	14.158996	0.000000
1.25	13.908996	-10.000000
1.25	13.408996	-20.000000
1.25	12.721496	-30.000000
1.25	12.158996	-40.000000
1.25	11.971496	-50.000000
1.25	11.533996	-60.000000
1.25	10.908996	-70.000000
1.25	10.033996	-80.000000
1.25	8.846496	-90.000000
1.25	7.471496	-100.000000
1.25	7.283996	-110.000000
1.25	8.971496	-120.000000

** Tracking Bounds for Specified Frequencies

trb3.dat

Frequency	Magnitude	Phase
2.50	1.357881	0.000000
2.50	1.076631	-10.000000

2.50	0.670381	-20.000000
2.50	0.139131	-30.000000
2.50	-0.485869	-40.000000
2.50	-1.204619	-50.000000
2.50	-1.923369	-60.000000
2.50	-2.345244	-70.000000
2.50	-2.829619	-80.000000
2.50	-3.313994	-90.000000
2.50	-3.657744	-100.000000
2.50	-3.720244	-110.000000
2.50	-3.907744	-120.000000

** Tracking Bounds for Specified Frequencies

trb4.dat

Frequency	Magnitude	Phase
5.00	-9.496843	0.000000
5.00	-9.809343	-10.000000
5.00	-10.153093	-20.000000
5.00	-10.496843	-30.000000
5.00	-10.824968	-40.000000
5.00	-11.153093	-50.000000
5.00	-11.449968	-60.000000
5.00	-11.715593	-70.000000
5.00	-11.934343	-80.000000
5.00	-11.981218	-90.000000
5.00	-11.817155	-100.000000
5.00	-11.067155	-110.000000
5.00	-10.379655	-120.000000

** Tracking Bounds for Specified Frequencies

trb5.dat

Frequency	Magnitude	Phase
10.00	-11.639968	0.000000
10.00	-11.843093	-10.000000
10.00	-12.061843	-20.000000
10.00	-12.280593	-30.000000

10.00	-12.499343	-40.000000
10.00	-12.702468	-50.000000
10.00	-12.905593	-60.000000
10.00	-13.061843	-70.000000
10.00	-13.202468	-80.000000
10.00	-13.311843	-90.000000
10.00	-13.272780	-100.000000
10.00	-13.147780	-110.000000
10.00	-12.444655	-120.000000

** Tracking Bounds for Specified Frequencies

trb6.dat

Frequency	Magnitude	Phase
20.00	-21.967134	0.000000
20.00	-22.154634	-10.000000
20.00	-22.264009	-20.000000
20.00	-22.295259	-30.000000
20.00	-22.248384	-40.000000
20.00	-22.123384	-50.000000
20.00	-21.920259	-60.000000
20.00	-21.639009	-70.000000
20.00	-21.310884	-80.000000
20.00	-20.912446	-90.000000
20.00	-20.474946	-100.000000
20.00	-20.014009	-110.000000
20.00	-19.326509	-120.000000

** Tracking Bounds for Specified Frequencies

trb7.dat

Frequency	Magnitude	Phase
30.00	-35.676283	0.000000
30.00	-35.863783	-10.000000
30.00	-35.895033	-20.000000
30.00	-35.707533	-30.000000
30.00	-35.363783	-40.000000
30.00	-34.801283	-50.000000

30.00	-34.051283	-60.000000
30.00	-33.082533	-70.000000
30.00	-31.926283	-80.000000
30.00	-30.582533	-90.000000
30.00	-29.113783	-100.000000
30.00	-27.582533	-110.000000
30.00	-26.098158	-120.000000

** Tracking Bounds for Specified Frequencies

trb8.dat

Frequency	Magnitude	Phase
35.00	-56.928155	0.000000
35.00	-57.178155	-10.000000
35.00	-57.178155	-20.000000
35.00	-56.928155	-30.000000
35.00	-56.178155	-40.000000
35.00	-55.428155	-50.000000
35.00	-54.178155	-60.000000
35.00	-52.428155	-70.000000
35.00	-49.928155	-80.000000
35.00	-46.428155	-90.000000
35.00	-41.303155	-100.000000
35.00	-35.803155	-110.000000
35.00	-31.365655	-120.000000

The disturbance bounds are achieved using the specified geometric approach which results in the following frequency tables for the 10 frequencies of interest.

** Disturbance Bounds for Specified Frequencies

db1.dat

Frequency	Magnitude	Phase
0.50	4.841138	-10.000000
0.50	4.866419	-20.000000
0.50	4.932439	-30.000000
0.50	5.022113	-40.000000
0.50	5.132764	-50.000000

0.50	5.261063	-60.000000
0.50	5.403127	-70.000000
0.50	5.554625	-80.000000
0.50	5.710920	-90.000000
0.50	5.867215	-100.000000
0.50	6.190783	-110.000000
0.50	6.990566	-120.000000
0.50	7.250768	-123.521237

** Disturbance Bounds for Specified Frequencies

db2.dat

Frequency	Magnitude	Phase
1.25	-5.818170	-10.000000
1.25	-5.883702	-20.000000
1.25	-5.882905	-30.000000
1.25	-5.799389	-40.000000
1.25	-5.476635	-50.000000
1.25	-5.099829	-60.000000
1.25	-4.680040	-70.000000
1.25	-4.230305	-80.000000
1.25	-3.765184	-90.000000
1.25	-3.300063	-100.000000
1.25	-2.850329	-110.000000
1.25	-2.430539	-120.000000
1.25	-2.292340	-123.521237

** Disturbance Bounds for Specified Frequencies

db3.dat

Frequency	Magnitude	Phase
2.50	-17.831487	-10.000000
2.50	-18.133736	-20.000000
2.50	-18.301196	-30.000000
2.50	-18.332603	-40.000000
2.50	-18.227750	-50.000000
2.50	-17.987368	-60.000000
2.50	-17.610768	-70.000000

2.50	-16.966317	-80.000000
2.50	-16.202078	-90.000000
2.50	-14.493796	-100.000000
2.50	-12.649904	-110.000000
2.50	-11.053478	-120.000000
2.50	-10.557684	-123.521237

** Disturbance Bounds for Specified Frequencies

db4.dat

Frequency	Magnitude	Phase
0.50	-3.468183	-10.000000
0.50	-3.544164	-20.000000
0.50	-3.592646	-30.000000
0.50	-3.612203	-40.000000
0.50	-3.602262	-50.000000
0.50	-3.563115	-60.000000
0.50	-3.495909	-70.000000
0.50	-3.331169	-80.000000
0.50	-2.909641	-90.000000
0.50	-2.488113	-100.000000
0.50	-2.080328	-110.000000
0.50	-1.699344	-120.000000
0.50	-1.573820	-123.521237

** Disturbance Bounds for Specified Frequencies

db5.dat

Frequency	Magnitude	Phase
0.50	-16.829739	-10.000000
0.50	-16.967571	-20.000000
0.50	-17.032351	-30.000000
0.50	-17.022558	-40.000000
0.50	-16.938420	-50.000000
0.50	-16.781924	-60.000000
0.50	-16.556893	-70.000000
0.50	-16.269100	-80.000000
0.50	-15.926364	-90.000000

0.50	-15.475263	-100.000000
0.50	-14.839083	-110.000000
0.50	-14.203693	-120.000000
0.50	-13.403941	-123.521237

** Disturbance Bounds for Specified Frequencies

db6.dat

Frequency	Magnitude	Phase
0.50	-45.452773	-10.000000
0.50	-45.591908	-20.000000
0.50	-45.493507	-30.000000
0.50	-45.153305	-40.000000
0.50	-44.556190	-50.000000
0.50	-43.674382	-60.000000
0.50	-42.464581	-70.000000
0.50	-40.865973	-80.000000
0.50	-38.807586	-90.000000
0.50	-36.252004	-100.000000
0.50	-33.312533	-110.000000
0.50	-30.336900	-120.000000
0.50	-29.356833	-123.521237

The composite bounds--the combination of the tracking and disturbance bounds--are as follows:

** Composite Bounds for Specified Frequencies

cb1.dat

Frequency	Magnitude	Phase
0.50	30.366686	-10.000000
0.50	30.366686	-20.000000
0.50	29.866686	-30.000000
0.50	29.866686	-40.000000
0.50	29.116686	-50.000000
0.50	28.116686	-60.000000
0.50	27.116686	-70.000000
0.50	25.366686	-80.000000
0.50	22.866686	-90.000000

0.50	19.616686	-100.000000
0.50	18.116686	-110.000000
0.50	21.116686	-120.000000
0.50	24.866686	-123.521237

** Composite Bounds for Specified Frequencies

cb2.dat

Frequency	Magnitude	Phase
1.25	14.158996	-10.000000
1.25	13.908996	-20.000000
1.25	13.408996	-30.000000
1.25	12.721496	-40.000000
1.25	12.158996	-50.000000
1.25	11.971496	-60.000000
1.25	11.533996	-70.000000
1.25	10.908996	-80.000000
1.25	10.033996	-90.000000
1.25	8.846496	-100.000000
1.25	7.471496	-110.000000
1.25	7.283996	-120.000000
1.25	8.971496	-123.521237

** Composite Bounds for Specified Frequencies

cb3.dat

Frequency	Magnitude	Phase
2.50	1.357881	-10.000000
2.50	1.076631	-20.000000
2.50	0.670381	-30.000000
2.50	0.139131	-40.000000
2.50	-0.485869	-50.000000
2.50	-1.204619	-60.000000
2.50	-1.923369	-70.000000
2.50	-2.345244	-80.000000
2.50	-2.829619	-90.000000
2.50	-3.313994	-100.000000
2.50	-3.657744	-110.000000

2.50	-3.720244	-120.000000
2.50	-3.907744	-123.521237

** Composite Bounds for Specified Frequencies

cb4.dat

Frequency	Magnitude	Phase
5.00	-3.468183	-10.000000
5.00	-3.544164	-20.000000
5.00	-3.592646	-30.000000
5.00	-3.612203	-40.000000
5.00	-3.602262	-50.000000
5.00	-3.563115	-60.000000
5.00	-3.495909	-70.000000
5.00	-3.331169	-80.000000
5.00	-2.909641	-90.000000
5.00	-2.488113	-100.000000
5.00	-2.080328	-110.000000
5.00	-1.699344	-120.000000
5.00	-1.573820	-123.521237

** Composite Bounds for Specified Frequencies

cb5.dat

Frequency	Magnitude	Phase
10.00	-11.639968	-10.000000
10.00	-11.843093	-20.000000
10.00	-12.061843	-30.000000
10.00	-12.280593	-40.000000
10.00	-12.499343	-50.000000
10.00	-12.702468	-60.000000
10.00	-12.905593	-70.000000
10.00	-13.061843	-80.000000
10.00	-13.202468	-90.000000
10.00	-13.311843	-100.000000
10.00	-13.272780	-110.000000
10.00	-13.147780	-120.000000
10.00	-12.444655	-123.521237

** Composite Bounds for Specified Frequencies

cb6.dat

Frequency	Magnitude	Phase
20.00	-21.967134	-10.000000
20.00	-22.154634	-20.000000
20.00	-22.264009	-30.000000
20.00	-22.295259	-40.000000
20.00	-22.248384	-50.000000
20.00	-22.123384	-60.000000
20.00	-21.920259	-70.000000
20.00	-21.639009	-80.000000
20.00	-21.310884	-90.000000
20.00	-20.912446	-100.000000
20.00	-20.474946	-110.000000
20.00	-20.014009	-120.000000
20.00	-19.326509	-123.521237

** Composite Bounds for Specified Frequencies

cb7.dat

Frequency	Magnitude	Phase
30.00	-35.676283	0.000000
30.00	-35.863783	-10.000000
30.00	-35.895033	-20.000000
30.00	-35.707533	-30.000000
30.00	-35.363783	-40.000000
30.00	-34.801283	-50.000000
30.00	-34.051283	-60.000000
30.00	-33.082533	-70.000000
30.00	-31.926283	-80.000000
30.00	-30.582533	-90.000000
30.00	-29.113783	-100.000000
30.00	-27.582533	-110.000000
30.00	-26.098158	-120.000000
30.00	-24.738783	-130.000000
30.00	-23.473158	-140.000000
30.00	-21.215345	-150.000000
30.00	-19.879408	-160.000000
30.00	-19.066908	-170.000000

** Composite Bounds for Specified Frequencies

cb8.dat

Frequency	Magnitude	Phase
35.00	-56.928155	0.000000
35.00	-57.178155	-10.000000
35.00	-57.178155	-20.000000
35.00	-56.928155	-30.000000
35.00	-56.178155	-40.000000
35.00	-55.428155	-50.000000
35.00	-54.178155	-60.000000
35.00	-52.428155	-70.000000
35.00	-49.928155	-80.000000
35.00	-46.428155	-90.000000
35.00	-41.303155	-100.000000
35.00	-35.803155	-110.000000
35.00	-31.365655	-120.000000

35.00	-28.240655	-130.000000
35.00	-26.037530	-140.000000
35.00	-24.342217	-150.000000
35.00	-21.318780	-160.000000
35.00	-20.139092	-170.000000
35.00	-19.600030	-180.000000

Step 6. Loop transmission design:

The loop transmission is designed to have a phase angle above gamma - 180 = -140 over the region of frequency 3 to 40 rad/sec which prohibits Lo from penetrating the u-contour. Also, Lo was designed to have a magnitude greater than the bounds at the specified discrete frequencies. The bode plot provided by the qft tool box reflects this characteristic. In addition, the angle contributions are determined as each new pole or zero is add in the Lo design process using the bode plot. An iterative design technique is employed starting with

$$Lo(s) = Po/s = 2/s(s+2)$$

The resulting Lo meeting the bounding criteria using the QFD approach is

** Loop Transmission - Lo **

Numerator: order = 4
constant/gain = 701766.000000

polynomial	roots
1.00	-0.010000 j 0.000000
10562.0	-12.000000 j 0.000000
5626705.62	-550.000000 j 0.000000
66056266.00	-10000.000000 j 0.000000
660000.00	

Denominator: order = 7
constant/gain = 1.000000

polynomial	roots
1.0	0.000000 j 0.000000
8727.00	0.000000 j 0.000000
51772450.00	-2.000000 j 0.000000
16238510000.00	-125.000000 j 0.000000
1257270000000.00	-200.000000 j 0.000000
2450000000000.00	-4200.000000 j -5600.000000

0.000000 -4200.000000 j 5600.000000
0.000000

The zero at -10000 was added to decrease the gain. The complex pole has a zeta of 0.6 and a wn = 7000 rad/sec

The following frequency response can be check for achieving the desired bounds:

** Freq domain Loop Transmission - lof

Frequency	Magnitude	Phase
0.10	45.564071	-98.160021
1.10	23.592022	-114.803398
2.10	15.989122	-128.097662
3.10	10.663229	-134.869957
4.10	6.586885	-137.915513
5.10	3.339406	-138.960580
9.10	-5.072261	-136.351593
10.10	-6.499835	-135.269468
19.10	-14.389152	-128.424274
20.10	-14.963042	-128.047181
29.10	-18.955878	-126.971790
30.10	-19.310471	-127.041855
34.10	-20.614006	-127.574123
35.10	-20.915351	-127.760558
39.10	-22.041838	-128.673153
40.10	-22.306226	-128.936752

Figure B.2 presents the combined curves for the composite bounds, Lo and the U-contour

Step 7. Controller G generation:

Using Lo and the nominal plant Po, the controller is generated

** Controller - Gcontr **

Numerator: order = 4
constant/gain = 701766.000000
polynomial roots
1.000000 -0.010000 j 0.000000

```
10562.010000                    -12.000000  j  0.000000
5626705.620000                  -550.000000  j  0.000000
66056266.000000                      -10000.000000  j  0.000000
660000.000000
```

Fig B.2

```
Denominator:      order = 5
 constant/gain =  2.000000
  polynomial                          roots
        1.000000              0.000000  j  0.000000
     8725.000000                -125.000000  j  0.000000
    51755000.000000             -200.000000  j  0.000000
 16135000000.000000            -4200.000000  j-5600.000000
1225000000000.000000           -4200.000000  j 5600.000000
        0.000000
```

The CLTFs with G and each plant can be generated and analyzed for a unit step input.

Step 8. Filter generation:

** QFT Data for Filter Generation **

frequency	Tmax	Tmin	Tru	Trl	Tru-Tmax	Trl-Tmin
0.50	0.05	0.00	0.06	-0.16	0.01	-0.16
1.25	0.29	0.00	0.36	-0.94	0.07	-0.94
2.50	1.14	0.01	1.27	-3.35	0.13	-3.36
5.00	3.56	-0.02	0.58	-9.87	-2.97	-9.85
10.00	2.36	-3.74	-9.67	-21.88	-12.03	-18.13
20.00	0.98	-14.00	-18.90	-37.82	-19.88	-23.82
30.00	1.11	-18.72	-23.25	-48.11	-24.37	-29.39
35.00	1.06	-20.42	-24.79	-52.15	-25.85	-31.74
40.00	1.03	-21.87	-26.07	-55.71	-27.11	-33.84
80.00	1.31	-29.82	-32.42	-75.10	-33.73	-45.28

The filter was design to "fit" between the values of the last two columns of the previous table. Using a trail and error procedure, the following filter tf was developed:

** Prefilter - filter **

```
Numerator:        order = 1
 constant/gain = 116.667000
  polynomial                          roots
   1.000000                  -18.000000  j  0.000000
  18.000000
```

Denominator: order = 3
constant/gain = 1.000000

polynomial	roots
1.000000	-3.000000 j 0.000000
83.000000	-10.000000 j 0.000000
940.000000	-70.000000 j 0.000000
2100.000000	

Observe that filter frequency data at the specified frequencies is within the desired magnitude band.

** Freq domain Filter Data - filterd

Frequency	Magnitude	Phase
2.50	1.142266	0.005701
5.00	3.555355	-0.023358
10.00	2.359959	-3.743955
20.00	0.982747	-13.995255
30.00	1.111816	-18.724763
35.00	1.064837	-20.416449
40.00	1.033052	-21.867131
80.00	1.306490	-29.822368

Step 9. Simulation (time and frequency domains):

*** Tracking Simulation ***

Observe that frequency domain responses are within the desired tracking tolerances specified by Tru and Trl. Figure B.3 presents the 6 variations. The simulations for Plant 1 are shown in the following tables:

** Frequency Domain Tracking Simulations - simf#

simf1.dat

Frequency	Magnitude	Phase
0.00	0.004601	0.000000
1.00	-0.299118	-24.962418
2.00	-0.979253	-47.659076
3.00	-1.653054	-68.522305

4.00	-2.186886	-90.022440
5.00	-2.885382	-114.795202
10.00	-16.505601	-205.594432
20.00	-34.408601	-231.877907
30.00	-43.727907	-241.745678
40.00	-50.185303	-250.958689
50.00	-55.297283	-260.066176
60.00	-59.636560	-268.824671
70.00	-63.468612	-277.061248
80.00	-66.934951	-284.701878
90.00	-70.119377	-291.733938
100.00	-73.075825	-298.177534

**** Disturbance Simulation ***

Observe that time domain unit step responses are within the desired disturbance tolerance specified by td. Figure B.4 depicts the six variations. The following list is for plant 1 time domain simulation.

Fig B.3

Fig B.4

** Time Domain Disturbance Simulation - dsimt#

dsimt1.dat

Time	Magnitude
0.00	-0.000000
1.00	0.049418
2.00	0.051887
3.00	0.051440
4.00	0.050916
5.00	0.050407
6.00	0.049905
7.00	0.049408
8.00	0.048917
9.00	0.048430
10.00	0.047948

Observe that frequency domain responses are within the desired disturbance tolerance specified by td.

** Frequency Domain Disturbance Simulations - dsimf#

dsimf1.dat

Frequency	Magnitude	Phase
0.10	-25.568491	4.995521
1.10	-25.339810	-7.540779
2.10	-24.848301	-16.181341
3.10	-24.084997	-26.969107
4.10	-23.199018	-41.781977
5.10	-22.642605	-62.204749
10.10	-31.735471	-142.160328
20.10	-45.244328	-165.337401
30.10	-52.592365	-170.916968
40.10	-57.703980	-173.558937
50.10	-61.641599	-175.130722
60.10	-64.850193	-176.178110
70.10	-67.560114	-176.923008
80.10	-69.906538	-177.474652

90.10 -71.975765 -177.894259

****

Conclusions - Thus a robust controller has been designed, analyzed and simulated for the given second-order system with plant coefficient variation and plant disturbances.

(This example was based upon the work of Dennis Trosen in the AFIT course EENG660 as taught by Professor Houpis)

D. Overview of QFT MIMO Design Process

The QFT MIMO synthesis problem requires conversion into a number of MISO single-loop feedback problems in which parameter uncertainly, external disturbances, and performance tolerances are derived from the original MIMO problem. The combined solutions to these MISO single-loop problems achieve the desired performance for the MIMO plant. The basic approach is a point-wise frequency domain MISO synthesis technique. A 3x3 MIMO feedback structure is shown in figure 2.

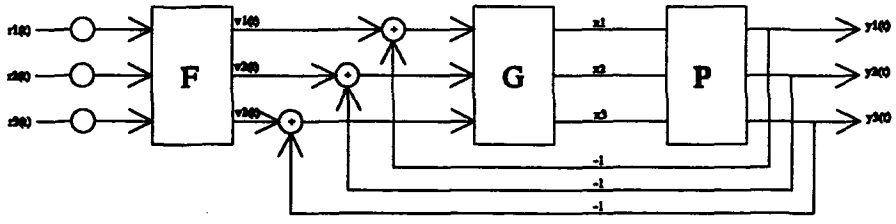


Figure 2: Three by three MIMO Design Structure

Plant models for figure 2 are developed in one of two formats: differential equation form and state space form. The general state-space model is manipulated as follows:

The state-space model representation for a LTI MIMO system is given by

$$\begin{aligned} \dot{X}(t) &= AX(t) + BU(t) \\ Y(t) &= CX(t) \end{aligned} \qquad (13)$$

where X is an m vector, Y is an n vector and U is an r vector. A,B, and C are constant matrices of the proper dimension.

The plant transfer-function matrix P(s) is defined in state space form as

$$P(s) = C[sI - A] - B \tag{14}$$

When the system of figure 2 is defined in differential equation form, we start with the following relations:

$$\begin{aligned} a_1(s)y_1(s) + b_1(s)y_2(s) + c_1(s)y_3(s) &= d_1u_1(s) + e_1u_2(s) + f_1u_3(s) \\ a_2(s)y_1(s) + b_2(s)y_2(s) + c_2(s)y_3(s) &= d_2u_1(s) + e_2u_2(s) + f_2u_3(s) \\ a_3(s)y_1(s) + b_3(s)y_2(s) + c_3(s)y_3(s) &= d_3u_1(s) + e_3u_2(s) + f_3u_3(s) \end{aligned} \tag{15}$$

This set of differential equations can be represented in matrix notation as

$$\begin{vmatrix} a_1(s) & b_1(s) & c_1(s) \\ a_2(s) & b_2(s) & c_2(s) \\ a_3(s) & b_3(s) & c_3(s) \end{vmatrix} Y(s) = \begin{vmatrix} d_1 & e_1 & f_1 \\ d_2 & e_2 & f_2 \\ d_3 & e_3 & f_3 \end{vmatrix} U(s) \tag{16}$$

which yields the following:

$$\begin{aligned} M(s)Y(s) &= NU(s) \\ Y(s) &= M^{-1}NU(s) \\ Y(s) &= P(s)U(s) \\ P(s) &= M^{-1}N \end{aligned} \tag{17}$$

This plant transfer function matrix $P(s) = [p_{ij}(s)]$ is a member of the set $P = \{P(s)\}$ of possible plant matrices which are functions of the uncertain plant parameters. If the equivalent plant matrix P resulting from the three matrices is not square, a weighing matrix W can be used to form an effective square plant; however, the use of a weighing matrix is not implemented in the ICECAP-PC MIMO QFT toolbox.

In CACSD practice, one of three explicit methods can be used to define the region of plant uncertainty. The first is based upon the physical modeling of various plants representing the variety of possible plants. The second includes the selection of only a finite set of P matrices, representing the extreme boundaries of plant pole/zero uncertainty. The third considers the variations in plant coefficients by considering a preselected number of plants to represent the maximum variations. A convex hull is then closed around these plants to derive the minimum number of plant models to represent the variation.

An mxm MIMO closed-loop system can be represented by three mxm transfer function matrices, F, G, and P. There are m^2 closed-loop system transfer functions $t_{ij}(s)$ (transmissions) contained within its system transmission matrix or system tracking matrix. $T_R(s) = \{t_{ij}(s)\}$, relates the outputs $y_i(s)$ to the inputs $r_j(s)$, that is, $Y_i(s) = t_{ij}(s)r_j(s)$. In a quantitative problem statement

there are tolerance bounds on each $t_{ij}(s)$, giving a set of m^2 acceptable regions $T_{ij}(s)$ which are to be specified in the design, thus $t_{ij}(s) \in T_{ij}(s)$ and $T(s) = \{T_{ij}(s)\}$. These regions may also be directly given in the frequency domain.

The following system equations define the input/output relation of Fig 2:

$$\begin{aligned} Y &= PX \\ X &= GU \\ U &= FR - Y \end{aligned} \tag{18}$$

In these equations G(s) is the matrix of compensator transfer functions and is often simplified so that it is diagonal. F(s) is the matrix of refilter transfer functions which may also be a diagonal matrix. The combination of these equations yields a 2 degree-of-freedom feedback structure

$$Y = [I + PG]^{-1} PGFX \tag{19}$$

where the system tracking control ratio relating r to y is

$$T_R = [I + PG]^{-1} PGF \tag{20}$$

The disturbance model is given as

$$T_D = [I + PG]^{-1} P = \{d_{ij}\} \tag{21}$$

The MIMO design objective is to determine a F and G for all plants in {P} such that

(1) the closed-loop control ratio is stable and
(2) the norm of $t_{ij}(w)$ is bounded: $a_{ij}(w) \le t_{ij}(w) \le b_{ij}(w)$ for $w \le w_o$
(3) the disturbance, eq 3-16, is bounded (disturbance rejection).

A linear mapping from a MIMO system structure results in m^2 MISO equivalent systems, each with two inputs and one output. One input is designated as a "desired" tracking input and the other as a disturbance input. To develop this mapping consider the inverse of the plant matrix represented by

$$P^{-1} = \begin{vmatrix} P^*_{11} & P^*_{12} & \cdots & P^*_{1m} \\ P^*_{21} & P^*_{22} & \cdots & P^*_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ P^*_{m1} & P^*_{m2} & \cdots & P^*_{mm} \end{vmatrix} \tag{22}$$

The m^2 effective plant transfer functions are formed by defining

$$q_{ij} = \frac{1}{p^*_{ij}} \tag{23}$$

A Q matrix is defined as:

$$P^{-1} = \begin{vmatrix} q_{11} & q_{21} & \cdots & q_{1m} \\ q_{21} & q_{22} & \cdots & q_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ q_{m1} & q_{m2} & q_{m3} & q_{mm} \end{vmatrix} = \begin{vmatrix} p^*_{11} & p^*_{12} & \cdots & p^*_{1m} \\ p^*_{21} & p^*_{22} & \cdots & p^*_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ p^*_{m1} & p^*_{m2} & \cdots & p^*_{mm} \end{vmatrix} \tag{24}$$

where $\mathbf{P} = [p_{ij}]$, $\mathbf{P}^{-1} = [p^*_{ij}] = [1/q_{ij}]$, and $\mathbf{Q} = [q_{ij}] = [1/p^*_{ij}]$.
the matrix $\mathbf{P}^{-1}$ is partitioned to form

$$P^{-1} = [p^*_{ij}] = [\frac{1}{q_{ij}}] = \Lambda + B \tag{25}$$

where $\Lambda = \{\lambda_{ij}\}$ is the diagonal part and $\mathbf{B} = \{b_{ij}\}$ is the off-diagonal component of $\mathbf{P}^{-1}$. Thus, $\lambda_{ii} = 1/q_{ii} - p_{ii}$, $b_{ij} = 1/q_{ij} = p_{ij}$ for $i \neq j$. Pre-multiplying eq 20 by $\mathbf{P}^{-1}[\mathbf{I}+\mathbf{PG}]$ yields

$$\begin{aligned} P^{-1}[I + PG]\, T_R &= P^{-1}[I + PG]\,[I + PG]^{-1} PGF \\ [P^{-1} + G]\, T_R &= GF \end{aligned} \tag{26}$$

If we let $\mathbf{P}^{-1} = \Lambda + \mathbf{B}$ where Λ contains the diagonal terms and $\mathbf{B}$ contains the off diagonal terms, eq 26 becomes

$$\begin{aligned} [\Lambda + B + G]\, T_R &= GF \\ [\Lambda + G]\, T_R &= GF - BT \\ T_R &= [\Lambda + G]^{-1}[GF - BT] \end{aligned} \tag{27}$$

Each of the m^2 matrix elements on the right side of eq 27 can be interpreted as a MISO problem. A fixed point mapping based on Schauder's fixed point theorem (D'Azzo, pg 699) is defined by $\mathbf{Y}(\mathbf{T_R})$ as:

$$Y(T_R) = [\Lambda + G]^{-1}[GF - BT] \tag{28}$$

where $\mathbf{G} = \{g_{ij}\}$ is assumed to be diagonal and each member of $\mathbf{T_R}$ is from the acceptable set $\{\mathbf{T_R}\}$. If this mapping has a fixed point, i.e., $\mathbf{T_R} \in \{\mathbf{T_R}\}$ such that $\mathbf{Y}(\mathbf{T_R}) = \{\mathbf{T_R}\}$, then this $\mathbf{T_R}$ is a solution of eq 27.

The control ratios for the desired tracking of the inputs by the corresponding outputs for each feedback loop of eq 28 have the form

$$y_{ii} = w_{ii}(v_{ij} + d_{ij}) = y_{r_i} + y_{d_{ij}} \tag{29}$$

where

$$\begin{aligned} w_{ii} &= \frac{q_{ii}}{1 + g_{ii}q_{ii}} \\ v_{ij} &= g_{ii}f_{ii} \end{aligned} \tag{30}$$

The interaction between the loops has the form

$$d_{ij} = -\sum_{k \neq i} \frac{t_{kj}}{q_{ik}} \qquad k = 1,2,\cdots,m \tag{31}$$

and appears as a "disturbance" input in each of the feedback loops.

E. The MIMO QFT CAD Toolbox

The heart of the MIMO-QFT toolbox interface is the dialog box which is a pop-up window containing input lines, radio buttons, and check boxes. *Input lines* allow text entry such as comment lines and transfer function definitions. ICECAP-PC input lines contain history windows by which past inputs are recalled and edited with mouse or the down arrow. *Checkboxes* are a multiple selection tools consisting of brackets that are checked when selected ([X][ ][X]). For example, if the user desires to see or define any number of transfer functions at the same time he/she may do so by selecting from among a 5x5 array of checkboxes. *Radiobuttons* are singular selection tools consisting of parenthesis that are dotted when selected [( )( )( )]. For example, when the user is allowed to select from one and only one frequency range, he/she makes his/her selection on a checkbox.

The MIMO QFT toolbox allows the design of systems up to 5x5 in dimension. It provides up to 20 plant cases of the **P** matrix. Thus, the file structure is conceptually a 5x5 p_{ij} transfer function matrix 20 levels deep with each level containing a plant case **P**. The toolbox stores a Tru, Trl, Td, L_0, G, and FCTF for each transfer function element.

There are two ways to define transfer function elements and matrices: polynomial format and factored format. Transfer function entry is done on an input line very similar to the most popular matrix/engineering programs with the exceptions that 1) complex factors may be entered directly and 2) brackets are not used to enclose the entry. Differentiation between the two forms is simply a matter of radio button selection. To enter a transfer function $(12.02s^2 + 4.32s + 5) / (4s + 2)$ one would enter the following line:

12.02 4.32 5; 4 2

Note that the numbers are not encased by right and left brackets like other popular programs. After such entry, the transfer function is displayed on a scrollable screen in standard coefficient form. To enter a transfer function with 3 zeros at -2, -5, and -7 and four poles -3±2j, -8, -1 and -9 with a gain of 1, one would enter the following:

-2 -5 -7; -3j2 -8 -1 -9

Note the direct entry of the single complex variable. The entry of complex is always done with an i or j preceding the complex element with no spaces allowed. The entry of spaces i a complex number is taken to be a second root at an imaginary location. The entry of the complex conjugate is done automatically by the computer. Manual entry of the complex conjugate is an error and results in an extra undesired root. After such entry, the transfer function is displayed

on a scrolable screen in either factored form ()()()/()()() or in list form showing the gain and root locations of the numerator and denominator as selected by the user.

Several p_{ij} (in fact all of the P matrix) can be defined at one time by simple checkbox selection from among a 5x5 array of elements that indicate which p_{ij} is to be defined.

The heart of the MIMO process is the accurate calculation of $\mathbf{P}^{-1}$ and its decomposition into MISO equivalent loops. The MIMO QFT toolbox uses the LU Decomposition process to find $\mathbf{P}^{-1}$ and **Q**. LU Decomposition offers several advantages over the classical Adj/Det technique.

1. It provides a single general method for the inversion of all **P**

2. Root cancellation occurs at very low orders preventing the growth of unnecessarily large order polynomials.

3. Because of this, root finding is much simpler and more accurate and root cancellation can be done with far tighter constraints than any other technique.

4. There is no need to factor out common denominators and numerators in **P**. These are canceled internally with no loss of accuracy.

LU Decomposition also provides quick and accurate calculation of **P** matrix determinant so that the engineer can inspect the **P** matrix for singularity and for non-minimum phase conditions.

The MIMO QFT process as implemented in ICECAP-PC follows the following menu and process structure:

1. File

Log File On: Activates the ICECAP-PC log file. All subsequent information displayed to screen is also sent to the log file. The log file is a text file named LOGFILE.TXT and as such can be viewed, printed or edited with any good text editor.

Log File Off: Turns log file off.

DOS Shell: Provides Temporary exit to DOS. Once in the DOS environment, return to the ICECAP-PC MIMO QFT program is initiated with the 'Exit' DOS command. Caution: Do not attempt to edit an active log file by using this option.

Exit: Exits the ICECAP-PC MIMO QFT toolbox.

2. Set

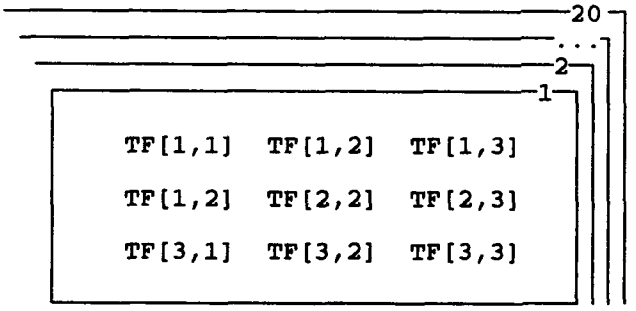
Clear Display: Clears the screen.

CommentLine: Presents direct entry of comments into the design session. Comments can be up to 132 characters long. If the log file is active, comments are also sent to it.

Header: Provides a dialog box to record the user/student name, title and other header information that should appear on top of log files. The header information is saved to disk and recalled each time the log file is turned on with the header option enabled.

View Options: Allows user to customize the session including the selection of high resolution screens, Number of decimal places displayed, fixed decimals or scientific notation display selection, and complex letter (i or j) selection.

Select Current Plant: Provides selection of the current plant and plant dimension. The MIMO-QFT toolbox allows up to 20 plant matrix definitions for both P and Q matricies. Each plant is a transfer function matrix. The complete plant set of 1-20 plants defines the region of plant paramater uncertainty.



Select Frequency Range: The MIMO-QFT toolbox provides three frequency ranges each containing 16 frequencies 1 octave apart. Plant templates are generated over the selected frequency range. The three ranges are:

1) .015 - 512 Rad/Sec
2) .500 - 16,000 Rad/Sec
3) 4096.000 - 134,000,000 Rad/Sec

Select Number Of Plants: Provides selection of the number of plants used in the MIMO QFT design process. The user can select from 1-20 plants used. Plant template generation depends on the proper setting of this option.

3. Specs

Define Tru: Define the upper tracking response transfer function (Tru) for transfer function elements of the plant matrix. Tru is normally defined element by element on an individual basis, however, several transfer function elements may have the same Tru. Therefore, Tru can be defined for several elements at a time by simply checking the appropriate boxes. Transfer function input can be in either polynomial or factored form as shown below.

To enter this: s^2+3s+5 / s^3+6s^2+7s+8
Check Poly :
And Type This: 1 3 5; 1 6 7 8

To enter this : (s+4)(s+6) / (s+8)(s-4+j2)(s-4+j2)
Check Factored:
Type this : -4 -6; -8 4j2
Note that the complex conjugate is automatically entered

Display Tru: Display Tru for any p_{ij} element/elements desired.

Edit Tru: Edit Tru for any transfer function element of the plant matrix.

Define Trl: Define the lower tracking response transfer function (Trl) for transfer function elements of the plant matrix. Trl cannot be defined for off-diagonal elements. Trl is normally defined element by element on an individual basis, however, several transfer function elements may have the same Trl. Therefore, Trl can be defined for several diagonal elements at a time by simply checking the appropriate boxes. Transfer function input can be in either polynomial or factored form as shown below.

To enter this: s^2+3s+5 / s^3+6s^2+7s+8
Check Poly :
And Type This: 1 3 5; 1 6 7 8

To enter this : (s+4)(s+6) / (s+8)(s-4+j2)(s-4+j2)
Check Factored:
Type this : -4 -6; -8 4j2

Note that the complex conjugate is automatically entered

Display Trl: Display Trl for any p_{ij} element(s) desired. Since Trl is undefined for off-diagonal elements, the dislay of such Trl is disallowed.

Edit Trl: Edit Trl for any transfer function element of the plant matrix.

Define Td: Define the disturbance tracking response transfer function (Td) for transfer function elements of the plant matrix. Td is normally defined element by element on an individual basis, however, several transfer function elements may have the same Td. Therefore, Td can be defined for several elements at a time by simply checking the appropriate boxes. Transfer function input can be in either polynomial or factored form as shown below. However, it is common practice to simply define Td as some constant (ie .1 for -20db)

To enter this: s^2+3s+5 / s^3+6s^2+7s+8
Check Poly :
And Type This: 1 3 5; 1 6 7 8

To enter this : (s+4)(s+6) / (s+8)(s-4+j2)(s-4+j2)
Check Factored:
Type this : -4 -6; -8 4j2
Note that the complex conjugate is automatically entered

Display Td: Display Td for any p_{ij} element(s) desired.

Edit Td: Edit Td for any transfer function element in the plant matrix.

Define Stability Specs: Stability specifications may be defined as either a phase margin (τ), a M_L contour, or a peak overshoot(Mp). Specifications are entered one for each row in the design and multiple rows may be define simultaneously. The entry of gain margin is specifically disallowed because 1) the sensitivity of phase margin with respect to gain margin and 2) the ill conditioned nature of calculating the phase margin given a gain margin which is error prone. Using any definition provided results in the printing of all specs.

Display Stability Specs: Display stability specifications for any number of row(s) of the **P** matrix. The display will show phase margin, peak overshoot, Ml contour, and gain margin.

4. Plants

Define Plant Elements p: Define the plant transfer function p_{ij} for transfer function elements of the plant matrix P. p_{ij} is normally defined element by element on an individual basis, however, several transfer function elements may have the same definition. Therefore, p_{ij} can be defined for several elements at a time by simply checking the appropriate boxes. Transfer function input can be in either polynomial or factored form as shown below. The current **P** is indicated in the dialog box and may be set using the *Select Current Plant* option.

To enter this: s^2+3s+5 / s^3+6s^2+7s+8
Check Poly :
And Type This: 1 3 5; 1 6 7 8

To enter this : (s+4)(s+6) / (s+8)(s-4+j2)(s-4+j2)
Check Factored:
Type this : -4 -6; -8 4j2
Note that the complex conjugate is automatically entered

Define Plant Variation: Provides an automated method for generating 20 plant cases. The user defines two transfer functions at the minima and maxima in variation for any given plant p_{ij}. Entry is allowed in polynomial or factored form by selection. The MIMO-QFT toolbox will automatically generate 125 plant cases for each plant element p_{ij}, envelop these plants (p_{ij}) with a convex hull, and choose 20 plants on its perimeter.

Copy Plant Matrix P:

Display Plant Elements p: Provides for the display of 1-25 plants p_{ij} in a given plant **P** as selected in *Select Current Plant* (described above).

Display Plant Matrix P: Allows the rapid display of all p_{ij} in a given plant case **P**. This is a separate menu item from the *Display Plants* option because the selection of several p_{ij} in a matrix can be a time consuming task. Radio button selection allows selection of the desired plant case and **P** dimension.

Edit Plant Elements p:

Find Plant Determinant: Calculates and displays the determinant for a selected **P**. P is chosen via radiobutton selection. In the QFT design process, all **P** must be checked for

non-singularity and for minimum phase conditions. Calculation of the determinant is via LU Decomposition.

High Freq Sign Check: Calculates and displays the sign of diagonal elements p_{ii} of **P** for all plant **P** cases.

5. Q Matrix

Define Plant Elements q: Define the plant transfer function q_{ij} for transfer function elements of the plant matrix Q. THE DIRECT DEFINITION OF Q IS AN ALTERNATIVE TO THE NORMAL INVERSION OF P. q_{ij} is normally defined element by element on an individual basis, however, several transfer function elements may have the same definition. Therefore, q_{ij} can be defined for several elements at a time by simply checking the appropriate boxes. Transfer function input can be in either polynomial or factored form as shown below. The current **P** is indicated in the dialog box and may be set using the *Select Current Plant* option.

To enter this: s^2+3s+5 / s^3+6s^2+7s+8
Check Poly :
And Type This: 1 3 5; 1 6 7 8

To enter this : (s+4)(s+6) / (s+8)(s-4+j2)(s-4+j2)
Check Factored:
Type this : -4 -6; -8 4j2
Note that the complex conjugate is automatically entered

Form Q Matrix: Inverts the selected matrix **P** via LU decomposition and then reciprocates the elements of $\mathbf{P}^{-1}$ to form the matrix **Q.**

Verify Plant Inversion: Multiplies **P** x $\mathbf{P}^{-1}$ in order to verify the accuracy of the inversion process. Ideally in the absence of zero roundoff error, the result will be an identity matrix. In practice, finite roundoff error is detected via this process. Scientific notation display should be used with this option to detect low levels of roundoff.

Check Diagonal Dominance: Calculates the magnitude of each $p_{\frac{i}{j}}$ in a selected **P** via L^Hospitals rule and displays the result along with an analysis as to the diagonal dominance of the plant **Q.** A dialog box provides radiobutton selection among 20 **Q.**

Display Plant Elements q: Provides a dialog box with checkbox input for the display of 1-25 plants q_{ij} in a given plant **Q** as selected in *Select Current Plant* (described above).

Display Plant Matrix Q: Allows the rapid display of all q_{ij} in a given plant case **Q**. This is a separate menu item from the *Display Plants* option because the selection of several q_{ij} in a matrix can be a time consuming task. Radio button selection allows selection of the desired plant case and **Q** dimension.

Edit Plant Elements q:

E. MIMO Design Example

Students Name
Teachers Name
Class

: This log provides demonstration of the entry modes and operation of
: MIMO-QFT toolbox functions and operation.

: Step 1: Define Tru, Trl, and Td for plant P. Only plant P1 is shown below:
: Note that in this example, Tru and Trl are the same for all rows.

MIMO Tru [1,1]

$$\frac{1.8896\ (s + 12.0000)}{(s + 2.2857 \pm j4.1775)}$$

MIMO Tru [2,2]

$$\frac{1.8896\ (s + 12.0000)}{(s + 2.2857 \pm j4.1775)}$$

MIMO Tru [3,3]

$$\frac{1.8896\ (s + 12.0000)}{(s + 2.2857 \pm j4.1775)}$$

: Tru for off diagonal elements is defined for all off diagonal plants as:

MIMO Tru [1,2]

$$\frac{0.1000}{1.0000}$$

MIMO Tru [1,3]

$$\frac{0.1000}{1.0000}$$

MIMO Tru [2,1]

$$\frac{0.1000}{1.0000}$$

MIMO Tru [2,3]

$$\frac{0.1000}{1.0000}$$

MIMO Tru [3,1]

$$\frac{0.1000}{1.0000}$$

MIMO Tru [3,2]

$$\frac{0.1000}{1.0000}$$

: Trl is only defined for diagonal elements:

MIMO Trl [3,3]

$$\frac{11600.0000}{(s + 4.0000)(s + 4.0000)(s + 7.2500)(s + 100.0000)}$$

MIMO Trl [3,3]

$$\frac{11600.0000}{(s + 4.0000)(s + 4.0000)(s + 7.2500)(s + 100.0000)}$$

MIMO Trl [3,3]

$$\frac{11600.0000}{(s + 4.0000)(s + 4.0000)(s + 7.2500)(s + 100.0000)}$$

: If definition of Trl for off diagonal is attempted, the following results:

Trl Undefined for Off Diagonal Transfer Function Element [1, 2]

: Td is defined at .1 for all plants p(ij)

MIMO Td [1,1]

$$\frac{0.1000}{1.0000}$$

MIMO Td [1,2]

$$\frac{0.1000}{1.0000}$$

MIMO Td [1,3]

$$\frac{0.1000}{1.0000}$$

MIMO Td [2,1]

$$\frac{0.1000}{1.0000}$$

MIMO Td [2,2]

$$\frac{0.1000}{1.0000}$$

MIMO Td [2,3]

$$\frac{0.1000}{1.0000}$$

: Stability bounds are defined as followed:

Stability Bounds for Row 1

Phase Margin: 40.0000°
Gain Margin: 4.3116 db
ML Contour: 3.8387 db
Mp : 1.5557

Stability Bounds for Row 2

Phase Margin: 35.0000°
Gain Margin: 3.9378 db
ML Contour: 4.8282 db
Mp : 1.7434

Stability Bounds for Row 3

Phase Margin: 35.0000°
Gain Margin: 3.9378 db
ML Contour: 4.8282 db
Mp : 1.7434

: Plant Case P_1 is defined as follows:

MIMO Plant [1,1] Plant Matrix 1

$$\frac{2.0000\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)(s + 5.0915)}$$

MIMO Plant [1,2] Plant Matrix 1

$$\frac{-10.1961\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)(s + 5.0915)}$$

MIMO Plant [1,3] Plant Matrix 1

$$\frac{5.4902\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)(s + 5.0915)}$$

MIMO Plant [2,1] Plant Matrix 1

$$\frac{-10.1961\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)(s + 5.0915)}$$

MIMO Plant [2,2] Plant Matrix 1

$$\frac{-2.3529\ (s - 17.0000)(s + 3.0000)}{(s + 2.0000)(s + 0.0000)(s + 5.0915)}$$

MIMO Plant [2,3] Plant Matrix 1

$$\frac{5.8824\ (s + 0.3333)(s + 3.0000)}{(s + 2.0000)(s + 0.0000)(s + 5.0915)}$$

MIMO Plant [3,1] Plant Matrix 1

$$\frac{5.4902\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)(s + 5.0915)}$$

MIMO Plant [3,2] Plant Matrix 1

$$\frac{5.8824\ (s + 0.3333)(s + 3.0000)}{(s + 2.0000)(s + 0.0000)(s + 5.0915)}$$

MIMO Plant [3,3] Plant Matrix 1

$$\frac{-4.7059\ (s + 1.8889)(s + 3.0000)}{(s + 2.0000)(s + 0.0000)(s + 5.0915)}$$

: Inspection of the Determinant of P1 reveals it is non-singular and minimum phase

Determinant of Plant Matrix 1

$$\frac{-47.0588\ (s + 3.0000)(s + 3.0000)(s + 3.0000)}{(s + 2.0000)(s + 2.0000)(s + 2.0000)(s + 0.0000)(s + 0.0000)(s + 0.0000)(s + 5.0915)}$$

Matrix Q will contain minimum phase elements upon P inversion

: The Q matrix is formed via LU Decomposition and is :

q-Plant [1,1] Q Matrix: 1

$$\frac{2.0000}{(s + 2.0000)(s + 0.0000)}$$

q-Plant [1,2] Q Matrix: 1

$$\frac{3.0000\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)}$$

q-Plant [1,3] Q Matrix: 1

$$\frac{1.0000\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)}$$

q-Plant [2,1] Q Matrix: 1

$$\frac{3.0000\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)}$$

q-Plant [2,2] Q Matrix: 1

$$\frac{5.0000\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)}$$

q-Plant [2,3] Q Matrix: 1

$$\frac{4.0000\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)}$$

q-Plant [3,1] Q Matrix: 1

$$\frac{1.0000\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)}$$

q-Plant [3,2] Q Matrix: 1

$$\frac{4.0000\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)}$$

q-Plant [3,3] Q Matrix: 1

$$\frac{10.0000\ (s + 3.0000)}{(s + 2.0000)(s + 0.0000)}$$

: The inversion process can be checked by multiplying P x inv P

Identity[1, 1]

$$\frac{1.0000E+0000}{1.0000E+0000}$$

Identity[1, 2]

0.0000E+0000

1.0000E+0000

Identity[1, 3]

1.6263E-0019

(s + 5.0915E+0000)

Identity[2, 1]

-4.3368E-0019 (s + 4.7624E+0008)

(s + 5.0915E+0000)

Identity[2, 2]

1.0000E+0000

1.0000E+0000

Identity[2, 3]

7.2810E-0011

(s + 5.0915E+0000)

Identity[3, 1]

4.6623E-0011

(s + 5.0915E+0000)

Identity[3, 2]

$$\frac{-2.9216E-0011}{(s + 5.0915E+0000)}$$

Identity[3, 3]

$$\frac{1.0000E+0000}{1.0000E+0000}$$

: This accuracy will be improved with more sophistication in the algorithm but is acceptable for now

: The next step is to check the diagonal dominance of the Q matrix

Transfer Function Magnitudes at w = ∞

0.0E+0000 0.0E+0000 0.0E+0000

0.0E+0000 0.0E+0000 0.0E+0000

0.0E+0000 0.0E+0000 0.0E+0000

Diagonal Dominance Test Passed For Q Plant 1

: Since Q is diagonally dominant, this Q is ready for MISO manipulation.

: Q1 (1,1) is now shown to be

q-Plant [1,1] Q Matrix: 1

$$\frac{2.0000E+0000}{(s + 2.0000E+0000)(s + 0.0000E+0000)}$$

: Solution of this MISO problem is shown in the MISO QFT handbook

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1992	Masters Thesis

4. TITLE AND SUBTITLE
An Object-Oriented Computer Aided Design Program for Modern Control Systems Analysis

5. FUNDING NUMBERS

6. AUTHOR(S)
Air Force Institute of Technology, WPAFB, OH 45433-6583

AFIT/GE/ENG/92D-36

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt Stuart Sheldon
WL/FIGL
Wright-Patterson AFB OH 45433

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This investigation is a continuation of the ICECAP-PC research project conducted under Prof. Gary B. Lamont at the Air Force Institute of Technology. It is an ongoing development of a public domain Computer Aided Design (CAD) package for Control Engineering students and faculty.with a special emphasis on education. This investigation focuses on three areas. First, an object-oriented environment is specified, designed and implemented. The functional structure of ICECAP-PC is then converted into an object-oriented structure because object-orientation provides an advanced logic and implementation structure and effectively addresses common software engineering issues. Additionally, a new interface featuring pull-down menus, mouse support, and single stroke access provides optimal user friendliness. Second, the numerical methods used for modern control capabilities are updated providing state of the art numerical capabilities. Third, using a program extension known as a toolbox, a basic MIMO (Multiple Input Multiple Output) faculty is created to augment the MISO capabilities. The MIMO QFT toolbox allows the entry and manipulation of up to a 3x3 MIMO plant matrix of transfer functions for the QFT solution to MIMO control systems problems.

14. SUBJECT TERMS
Control System Engineering, Control Systems, Computer Aided Design (CAD), Numerical Methods, Computer Aided Control System Design (CACSD)

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	UL