DOCTOR OF PHILOSOPHY

Computer-aided control system design using optimization methods.

Grace, A.C.W.

Award date:
1989

Awarding institution:
Bangor University

Computer-Aided Control System Design Using Optimization Methods

Thesis submitted in candidature for the Degree of PhD

A.C.W.Grace

1989

University of Wales, Bangor.

Abstract

Control System Design methods are presented in terms of optimization techniques that incorporate Multi-Objective design criteria. Computer-Aided Control System Design (CACSD) environments make the approach easy-to-use and accessible to the practising control engineer.

Two CACSD environments have been developed using different versions of the MATLAB package, one interfacing the ADS optimization package to an upgraded FORTRAN version of MATLAB, the other using Non-linear Programming algorithms coded in the PRO-MATLAB command language. In both environments, optimization problems are entered interactively and in a flexible manner using simple interpreted commands and programs.

A Control System Design method has been implemented using optimal control theory and integral quadratic measures of control. The theory has been developed to incorporate a large number of design options, control structures and disturbance types. An evolutionary design process is used so that the control order and number of design criteria are systematically increased to incorporate more complex control structures and a wide set of performance objectives. In the later stages of this evolutionary design process, a Multi-Objective design strategy, known as the Goal Attainment method, is used to address multiple performance objectives.

Acknowledgments

A research project of this nature demands the help and support of many people. In particular, I would like to thank my academic supervisor, Prof. P.J.Fleming, for his guidance throughout this project. Not only for his help in the initial stages of the project, with numerous lectures and discussions in both control theory and optimization, but also for his enthusiasm and motivation in the later stages of the project. I am also indebted to him for his assistance with the preparation of this thesis.

I would also like to express my thanks to Mr. P.Smith and Mr. S.Winter of Royal Aerospace Establishment (RAE), Bedford who provided me with insights into non-linear simulation and the design of flight control systems.

I gratefully acknowledge the financial support of the Science and Engineering Research Council (SERC) and to RAE as part of a Cooperative Award in Science and Engineering (CASE). I would also like to thank SERC for the funding of numerous Vacation Schools and workshops which I found most useful.

Contents

3. CONTROL SYSTEM DESIGN

OVERVIEW

AS OUR ABILITY to model increasingly complex systems improves so does the feasibility of designing the system to operate with better design characteristics and capabilities. The relationship between design objectives and design parameters is, however, often complex and non-linear. Such a situation arises in Control System Design where the design requirements are not easily expressed as functions of the design parameters. The tendency has been to simplify the problem in terms of the model being used, the design objectives and/or the control configuration. This results in designs which, when applied to the actual system, may not function effectively or provide good overall performance.

Optimization methods offer the ability to consider performance objectives whose mathematical relationship with the design parameters need not be explicitly expressed. Instead performance objectives are iteratively evaluated for specific values of the design parameters and optimized with respect to the problem formulation being considered. This permits more realistic models and design objectives to be considered and allows more appropriate control structures to be used.

The application of optimization techniques is, however, generally not an easy process and may typically require extensive coding in a high-level language for evaluation of the performance objectives and linking to numerical optimization subroutines. The resulting program will typically be non-conducive to interaction on the part of the designer resulting in further editing and compilation cycles for small variations in the problem statement. This hinders the designer's ability to explore the effect of variations in control-configurations, performance objectives and model parameters. Further, since the optimization routine cannot usually guarantee a global solution, different starting points or optimization strategies may have to be tried requiring further time consuming editing and compilation cycles.

The aim of this research has been to provide a design environment which allows optimization problems to be coded in an interactive, flexible and efficient manner and to provide a Control System Design methodology which fits into this framework. Three main research areas have been studied:- Computer Aided Control System Design (CACSD), Optimization and Control System Design, which together form the basis of an overall control design strategy. This thesis has been organized to reflect these three main distinct but related topics.

Part 1 describes aspects and future trends within the field of CACSD. Integrated design environments and the MATLAB package are highlighted as examples of state-of-the-art design environments in this field. An environment based on an upgraded FORTRAN version of MATLAB, linked to an optimization package ADS, is described.

In Part 2, nonlinear optimization techniques are discussed for unconstrained, least squares and constrained problems which have been implemented as part of an Optimization Toolbox, coded in the PRO-MATLAB command language. A method of multi-objective optimization is described, known as the Goal Attainment method. Sequential Quadratic Programming (SQP), which is a modern, highly-effective, non-linear programming strategy, is focused on and refinements to the method are presented for the Goal Attainment method.

In Part 3, a Control System Design methodology is described using integral quadratic measures of control. Various design options and disturbance types are handled using a number of control configurations. In particular, the design of servomechanisms using a generalized feedforward/feedback two-degree-of-freedom control structure is described. Multi-objective optimization is proposed as part of an evolutionary and interactive design process to incorporate increasingly complex controllers and a wider set of performance objectives. Three examples are presented to demonstrate the design methodology.

Part 1

CACSD

Summary - Computer-Aided Control System Design (CACSD) plays a critical role in the implementation and application of control theory. Trends and future directions within this field will be discussed, looking at, in particular, integrated design environments and possible further evolutionary changes to the MATLAB package. A design environment using a FORTRAN version of MATLAB and an optimization package, ADS, is presented as an integrated design environment for optimization applications of Control System Design.

1.1 INTRODUCTION

THE INCREASING speed of modern computers and the evolving trend towards powerful computing systems, such as networked workstations, has resulted in a shift of attention within engineering software away from algorithm efficiency towards aspects of functionality and user-friendliness. Issues such as inter-package communication, data structures and user-interfaces are now important considerations for any CACSD environment.

Control System Design is by its nature a complex procedure which has prompted a plethora of control design methods and associated software packages. Many of these methods have been developed by academia where the resulting user-interfaces, reliability, maintenance and applicability of the software have been key issues in the poor integration of these packages within the control community and into industry. This has been one of the factors in the continued use of often heuristic methods for Control System Design within industry and the development of mathematically tractable but sometimes impractical methods of Control System Design within academia.

This part does not intend to give a full review of existing software and the reader is referred to Fleming and De Oliveira [1] and [2-6] for useful introductions to the subject. References [14-39] give specific details with respect to a large number of existing packages.

1.2 INTEGRATED DESIGN ENVIRONMENTS

Over the last three decades a great deal of progress has been made in the development, refinement and optimization of engineering software in the form of numerical subroutines, libraries and packages. Much of this software has been written in FORTRAN which still remains the fastest compiled language for numerical applications on most machines. Attempts have been made to harness these powerful numerical routines through the use of integrated environments which provide, among other things, a user-friendly interface to the numerical routines, database handling facilities, graphics facilities and high-level command language capabilities.

The MATLAB package [18-20] is an example, of such an integrated environment in which the numerical libraries, LINPACK [34] and EISPACK [35], for linear algebra have been integrated into a user-friendly environment. Although such environments are a significant improvement over numerical libraries, they are limited by the amount of facilities that can be reasonably programmed into one package. Attempts are now being made to develop environments which offer further integration by the inter-linking of other packages as well as numerical subroutines. Such environments provide a common software base providing tools and libraries of numerical algorithms and the ability to link to existing packages. This avoids excessive duplication of software and helps to provide a common base to facilitate the transfer of design methods into industry. The advantage to the control engineer is that the environment gives him the ability to work at a high-level of abstraction so that he can concentrate on important aspects of the design problem.

An integrated design environment has been developed as part of the SERC's Special Initiative in the field of Computing and Design Techniques for Control Engineering (CDTCE). This is called ECSTASY [15] (Environment for Control System Theory, Analysis and Synthesis). The aim of the

project is to provide an embryo infrastructure which consists of six major design tools as shown in Fig. 2.1. Further facilities are provided by packages which are linked to the infrastructure and database through inter-process communication mechanisms. For a fuller description of the ECSTASY environment, see [16].

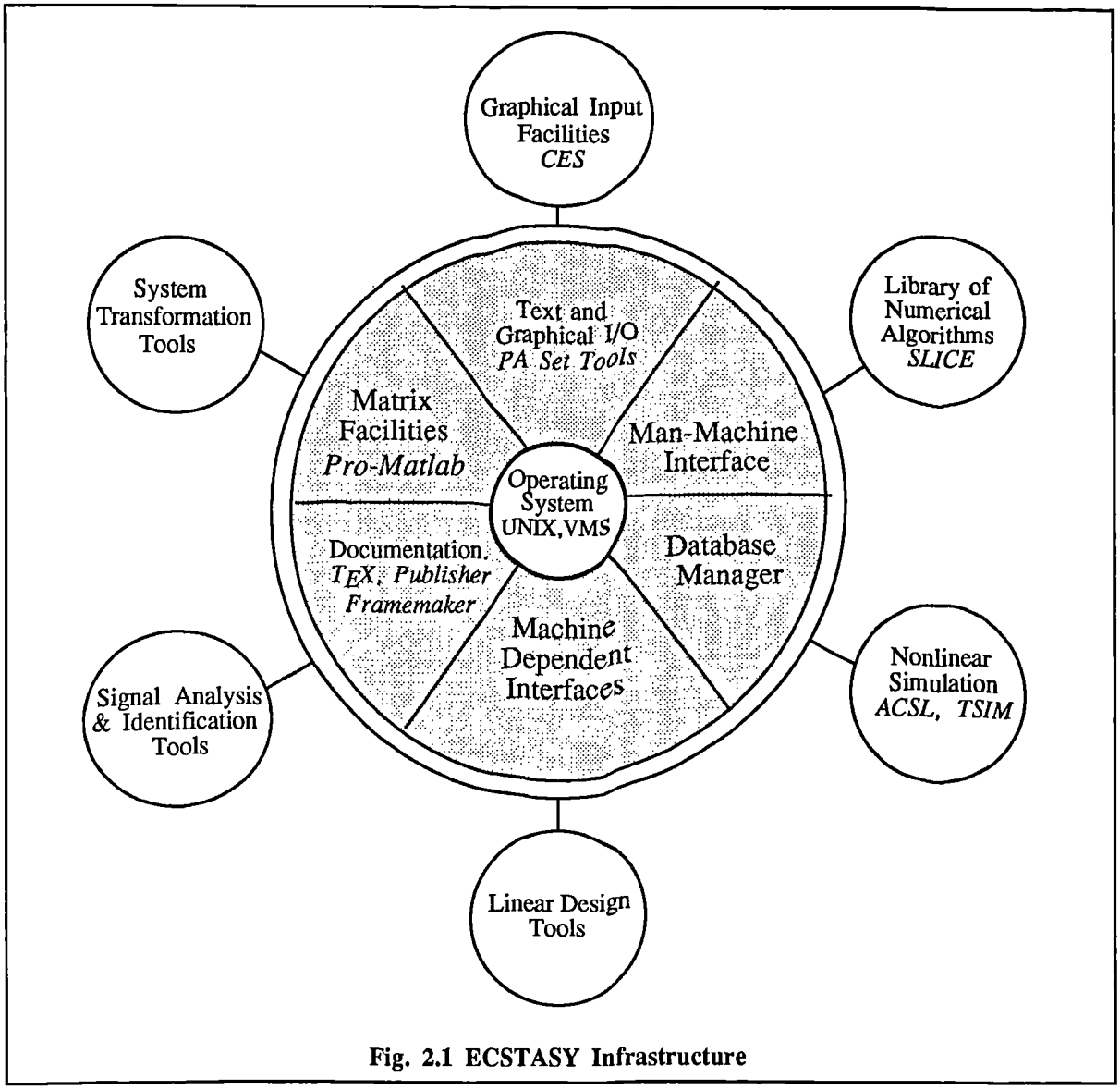

Fig. 2.1 ECSTASY Infrastructure

ECSTASY aims to provide a flexible environment capable of covering features such as system identification, simulation and Control System Design. It is one step towards a totally integrated design environment which would allow, for instance, control implementation details to be taken into consideration at the system design level. MATRIX-X [24], CTRL-C [21] and DELIGHT [28] are other packages which are aiming at providing an integrated design environment by incorporating a wider range of functions by linking to other packages. It is hoped that ECSTASY will provide better flexibility in terms of data communication enabling a wider range of facilities to be linked into the environment.

Whilst recognizing and fully supporting the aims and objectives of the ECSTASY environment it may nevertheless have several shortcomings. The cost of ECSTASY is likely to be high due to the necessity of having to buy supporting packages and because the infrastructure itself contains a number of expensive packages (PA Set Tools [17], PRO-MATLAB [20]). The environment will also have large

memory requirements which may result in a degradation of computational speed and limit its use to mainframes and workstations. The database aspects of the package will also tend to slow the package down due to type checking, parsing, handshaking and general housekeeping.

Depending on the operating system being used there are many different inter-process communication mechanisms which can be used. Data requirements and other issues will dictate the type of inter-process communication mechanism to be used. For instance, for optimization applications a fast data link with high precision is beneficial due to the iterative nature of the method. In such cases it may be more appropriate to directly link the application software to the source package. It is therefore important for integrated environments such as ECSTASY to provide a choice of inter-package communication methods so that a particular method can be chosen depending on the application being considered. The actual mechanism could then be chosen to create a balance between precision, speed and ease of use.

Many new features, such as the incorporation of new data structures and modern computing approaches cannot be wholly achieved through inter-communication between packages. Further advancements in CACSD will also involve the evolution of design packages such as MATLAB.

1.3 MATLAB

MATLAB is increasingly becoming the de-facto standard for linear control system design. Released in 1980, by Cleve Moler and originally intended for linear algebra, the application to control design was quickly seized upon by the control community and a number of packages appeared as derivatives of MATLAB such as CTRL-C, MATRIX-X and IMPACT [22].

The original version of MATLAB [18] is public-domain software written in FORTRAN. Subsequently, it has been rewritten in C, offering improved graphics facilities, faster computation and improved programming facilities including user-defined functions. It has been marketed commercially as PC-MATLAB [19] and PRO-MATLAB, the former targeted for implementation on IBM-PCs and the latter for mainframes and workstations, although both offer almost identical facilities.

PRO-MATLAB follows an "open system" philosophy by which it provides the user with a set of low-level functions and the means to construct higher-level functions by enabling the user to create M-files (distinguished by having the extension ".m"), which subsequently are added to the set of available commands and user-functions.

M-files have led to the provision of Toolboxes, which are marketed with PRO-MATLAB. These consist of M-files, with specific objectives. CACSD-oriented Toolboxes include a Control System Toolbox, an Identification Toolbox, a Signal Processing Toolbox, a Robust Control Toolbox, a State Space Identification Toolbox and a Multivariable Frequency Domain (MFD) Toolbox.

The success and wide-spread use of MATLAB can be attributed to a number of features which make the package easy-to-use and highly functional. They are as follows:

- MATLAB is interpretive so that there is no wasted time in compilation. This allows for better interaction on the part of the user. PRO-MATLAB is especially applicable to a window-based environment where one window can be used for editing a user-defined function while another can be used for testing it. Although interpretive languages are slower than compiled languages, MATLAB is not wholly interpretive, as user-defined functions are semi-compiled on their first call, offering some speed-up on subsequent calls.
- Data management, storage and housekeeping is external to the user, allowing the user to concentrate at a higher level of abstraction than most so-called high-level languages such as C and FORTRAN.

- MATLAB has at its core a large library of numerically reliable and optimized software based on the LINPACK and EISPACK subroutine libraries.
- A major feature of MATLAB is that it is extendable allowing the user to write routines which automatically become part of the command set. This is especially important in the realm of Control System Design where individual methods and applications are prevalent.
- The simple syntax and the use of default commands reduces unnecessary coding.

One criticism of MATLAB is that it is not a typed language. That is variables do not have to be given a type and are all assumed to be subsets of the complex double-precision matrix. Rimvall [10], however, has indicated that typing may be a severe restriction on flexibility. Strong typing may not be appropriate to languages such as MATLAB where its high level commands and data-structure restrict program size and the quantity of variables. Variable confusions are further reduced if the programmer uses sensible naming conventions. Further, since MATLAB is interpretive, typing problems are easily resolved, unlike compiled languages where complex debugging procedures may be necessary.

1.3.1 Possible Improvements To MATLAB

Although MATLAB has proved a highly successful and popular package, there are areas in which improvements would help to enhance the package and help to maintain its position as a state-of-the-art software package. The next few sub-sections will address how possible achievements could be achieved within MATLAB.

A main criticism of MATLAB is its restrictive data structures which do not allow any data types other than subsets of the complex matrix and character strings. Lack of database management has also been cited as a weakness of MATLAB where the commands *load* and *save* are deemed inadequate for handling large amounts of data.

Other possibilities for improvement are an increase in run-time efficiency by further compilation of the MATLAB M-files. Error diagnostics and help facilities are also areas in which improvements would help the user.

Inter-process communication mechanisms within the MATLAB infrastructure have the potential to make MATLAB a highly functional integrated design environment which can link to a whole range of facilities in a flexible way.

Modern computing approaches such as graphical input, symbolic processing, Intelligent Knowledge Based Systems (IKBS), object oriented, functional and logical programming concepts are also deemed as possibilities for further investigation.

The realization of these improvements can be achieved in a number of ways. Besides rewriting MATLAB from scratch, which would be an expensive solution and contravene the principles of evolutionary change and reusability within software, the alternatives are as follows:

- Link to other packages which have the desired facilities.
- Try to simulate improvements within the constraints of the MATLAB syntax and data structures by creating extra M-files which perform the necessary tasks.
- Add improvements to the source code of MATLAB.

Each of these options has its advantages and disadvantages. Linking to other packages preserves modularity but is also likely to be slow due to communication overheads. This may not resolve problems such as restrictive data structures.

Simulating improvements through the use of additional M-files , where possible, is an attractive solution since it requires no re-release of the package though this is likely to be less efficient and more restrictive than incorporating new features within the source code.

Adding new features to the source code seems an attractive solution though this may slow the interpreter down and upgrading might make new software un-executable on old versions of the package.

The following sections discuss aspects of MATLAB which could be improved and changes which might be implemented within MATLAB.

1.3.2 Data Structures

The only data structure which exists within MATLAB is the double precision complex matrix although strings may be thought of as a different data type. Mansour *et al* [11] and (see also Maciejowski [9]) have outlined a proposed list of data structures which have been implemented in the IMPACT package and might be used within control. They are as follows:

- Polynomials.
- Transfer Functions.
- Complex Matrices.
- Linear Systems Descriptions in the Time Domain.
- Linear Systems Descriptions in the Frequency Domain.
- Non-linear Systems Descriptions.
- Domains. (1-D structures containing ordered discrete values)
- Trajectories (e.g. Time and Frequency data)
- Non-numeric Structures.

Many of these structures can be simulated within MATLAB. For instance, a polynomial can be simulated using a row vector and a transfer function can be simulated using one row for the numerator and the other for the denominator. This situation, however, is far from satisfactory as there is no indication from the display or operation of the data what kind of data structure is being used.

Rimvall has also suggested the overloading of operators such as, *, /, +, and - so that they have different meanings depending on the data structure being used. This has been implemented together with extensions on data structures in the package IMPACT [19], a MATLAB based package. However, such a feature is not as yet commercially available in a MATLAB based package.

Some data structures are more difficult to incorporate than others. For instance, non-linear descriptions and symbolic processing may best be left to specialist packages such as simulation packages (ACSL [32], SIMNON [33], TSIM [34] etc.) and symbolic processing packages (MACSYMA [27], REDUCE [26]).

MATLAB has proved very successful within the realm of linear control design. Data structures such as transfer functions, pole-zero representations and state-space systems should therefore be inherently part of the package.

A first step for providing new data structures within MATLAB would be to provide flexible input/output format. For example, this would allow polynomials to be entered and displayed as polynomials and not simply as row or column vectors.

In order to incorporate additional data structures within MATLAB the source code must be changed to some extent. One desirable feature would be to allow variables to be labelled with information concerning how they are to displayed on the screen. This, although not a complete solution, would be one step in the right direction. An example of a command defining the output format of variable containing the state space representation of a controller (named CONT1) might be as follows:

```
label(CONT1, 'format short variable s transfer_function', 'Controller 1')
```

This would mean that the variable CONT1 would always be labelled as a transfer function, an example of such an output might be, for example:

```
CONT1 = ( Controller 1)
        2.1234s + 4.3333
  ---------------------------------
  3.454s^2 + 2.3333s + 5.4444
```

This format is much easier to understand when compared with an output consisting of a matrix of numbers, however problems may arise in user-defined functions if it is required to write flexible code which allows state-space or polynomial descriptions to be used interchangeably. There should therefore be rules, so that variables can inherit output formats. For example, the command:

```
TF1=feedback(SYSTEM,CONT1)
```

would need to ensure that the variable TF1 has the same format as the variables SYSTEM and CONT1. This might be achieved by a simple command such as

```
TF1=inherit(SYSTEM)
```

within the .m file feedback.

MATLAB's Data Structure.

All data elements within MATLAB are treated as matrices, the elements of which are stored in a large data stack of high precision numbers. A stack containing the position of the first element of each variable in the main data stack (i.e. an array of pointers) has reference to other stacks which refer to the variable names, row sizes, column sizes and a flag for indicating whether the matrix is real or complex. This type of database resembles to some extent that of a relational database with columns of data relating to each other. To incorporate new data structures and other facilities in MATLAB other columns of data could be added to the database with information regarding further attributes and relational aspects of the data. In this way pointers could be set up to display formats and other related data so that clustering of data could be implemented. This would then allow structured information to be defined. This might take the form of the structure command as in C. An example of this might be to define a state space system in the following way:

```
structure(SYSTEM)
{A;B;C;D}
```

A structure would become a data type whose elements are a series of pointers to other data elements. Referencing elements in the structure could be done as in C. Thus SYSTEM.A would refer to the A matrix of structure SYSTEM.

When functions are called, the structures could behave like macros so that stepr(SYSTEM) is equivalent to stepr(A,B,C,D). Grouping of data in this way should also serve to reduce complexity of calling functions where there are a lot of variables. One problem encountered in trying to implement

optimization algorithms is the large amount of data that must be temporarily stored (about 10-30 variables). If one uses the principle of not having any global variables, adding long lists of variable argument lists to functions can become cryptic and the principle of information hiding intended for user-defined functions is lost. Grouping of data into one matrix is complicated and time inefficient as is temporarily storing to files. A good solution would be to use pre-defined structures.

There are many other relational aspects of data that could be incorporated within the MATLAB data-structure. For a more detailed discussion of relational database aspects in control the reader is referred to Breuer *et al* [7] [8].

1.3.3 Databases

One of the attractions of MATLAB is the internal database handling of the matrices. All alterations and housekeeping of the database are maintained within the MATLAB operating system. However, MATLAB offers very few facilities on a wider scale for data management. Although the requirements for data management in control are relatively small compared with many applications there are several areas where database management is an important issue.

One of these areas is in the modelling process where changes to the model may be made by a number of workers and at intermittent intervals of time. If other design processes are being carried out in conjunction with the modelling process or when a number of co-workers are working on the same problem it becomes important to document details of changes in terms of time, date, type of change and by whom the change was made.

Another area where data management may be a problem is within the realm of control system design when a large number of linear models are obtained from the non-linear model at various operating points in preparation for control system design. The control design, may, also create data problems if different controllers and control structures are used. This might be the case if a control designer were experimenting with different controllers in order to view trade-offs between them or to implement them in a gain-scheduled controller.

It is clear that for the proper management of such data, data management systems should be inherent within the data model. Non-linear modelling is really within the realms of system identification work and database management aspects of this should be contained within the modelling and simulation package being used.

The data management of linear models and controllers can be simulated within the constraints of the MATLAB command language using M-files. These would perform the necessary housekeeping using the commands *load* and *save*.

The policy which is used in the software used for Control System Design examples given in Part 3 is to use a different directory for each linearized model, under a main directory for the overall model. Each directory contains files which contain the data from different designs resulting from changes in design features or controller configuration. Files and directories are created automatically at the end of each design and named depending on characteristics of the design. In this way an update of all designs is maintained. In each file the data is recorded for further reference.

Although this method provides some form of data management additional features would greatly facilitate and enhance the handling and storing of data. One such improvement would be to have flexible data memory handling. At present when a file is to be examined the data is loaded into the system overwriting any existing data with the same name. When examining differences between designs it would be an advantage to just switch database from, for example, internal memory to a file.

Another weakness of MATLAB is its inability to temporarily save variables in user-defined functions from one call to the next. Although this can achieved by saving to file this is slow when compared to saving in internal memory.

1.3.4 Help and Error Diagnostics

MATLAB has a clever way of incorporating the documentation accompanying the M-files with the help diagnostics which appear under the help command. However, the help command gives the user no control over how much or how little information is received. Further, the help is no way structured making it difficult to get an overview of related commands.

Error diagnostics are generally not clear. For instance, if a matrix is of the wrong size or an element which does not exist has been addressed, then the name of the matrix should be displayed along with its contents. At present the only output which is provided is the line number where the error occurred.

Another problem involves correction of syntax errors. These are often hard to find due to lack of information and thus tedious to correct. A solution to this might be to allow syntax errors to be corrected semi-automatically. This could be achieved by displaying the line at fault at the cursor so that if the user wishes to make any changes he can do so without having to use the editor. The user would then have the option of typing a carriage return, in which case no change would be made. Otherwise the file would be rewritten with the correction made.

More information should be made available when an error condition occurs and all associated variables should be displayed. A query feature could be made available giving progressively more detail of the error to avoid clogging the screen.

Due to the ever-growing number of user-defined functions the chances of giving a variable the same name as a user-defined function become increasingly high. This can result in errors which are difficult to debug and may not be apparent until later use of a user-defined function. A useful feature would be to give a warning whenever a variable is created which conflicts with a user-defined function name. A good naming convention also relieves this problem. Such a naming convention might be as follows:

- user-defined functions - lower case letters having at least three letters.
- script files - first letter upper case
- global variables & function arguments -upper case
- temporary miscellaneous variables - lower case, maximum of two letters.
- integers - begin variable name with i or I
- strings - begin variable name with s or S
- polynomials - begin variable name with p or P

One additional feature which would be useful would be the facility to call a user-defined function without parameters and have MATLAB prompt you for the variables whilst in interactive mode. This overcomes having to remember the order of often long lists of variables.

1.3.5 Compilation

Because MATLAB is basically an interpretive language, the run time efficiency is slow when compared to compiled languages. Whilst this is not a problem in most applications, when iteration is demanded MATLAB is particularly slow. One such application is optimization. It would therefore be of a great advantage if once a function has been tried and tested it could be compiled and linked into source code.

Most of the constructs within MATLAB (IF,WHILE,FOR) are contained within the source code (C or FORTRAN), so that a translator would not be difficult to implement. However, where compilation is likely to prove difficult is in the database handling of user-defined functions. For instance, a matrix of arbitrary size can be created within a user-defined function. However dynamic creation of variables like this is not generally or easily performed in high-level compiled languages. Therefore, some form of kernel database handling system must be inherent within the compiled code.

Compiled routines must be linked into the main database either by communication mechanisms (e.g. UNIX pipes) or by directly linking the compiled routines into the source code.

1.3.6 Linking To Other Numerical Libraries

Several lessons can be learned from the enormous popularity of MATLAB. Essentially, MATLAB is a package which has linked to a library of routines (LINPACK [36] and EISPACK [37,38]) and provided a user-friendly and easy-to-use environment. One of the criticisms of libraries such as the NAG library [35] is that there is no way to easily and quickly write application software or test programs. A framework such as that provided by MATLAB would be an ideal environment for such experimentation.

A possibility for incorporating libraries such as NAG into an interactive design environment such as MATLAB might be to write an auto-linking program which reduces the amount of mundane tasks concerned with aspects of memory allocation, compilation and coding. Linking directly to whole libraries as has been done with LINPACK and EISPACK would be prohibitive due to large memory requirements. However, an auto-linking compilation process could allow commands to be directly typed in using a MATLAB command which would invoke a process to link with the required subroutine.

As an example of how this could be achieved, an example will be taken from the NAG library. The MATLAB command which initiates the link will be performed using only constructs which are available within the MATLAB command set. The example chosen is a subroutine which involves finding a solution of a set of real linear equations using Crout's factorization method.

The command would be carried out using a MATLAB function of the form:

```
<>[X,A]=NAG('F04AAF',A,int(rows(A)),B,int(rows(B)),int(cols(A)),int(cols(B)),X,..
                int(rows(X)),WSPACE,int(IFAIL))
```

Associated with this command would be the user-defined function NAG.m which would first store the variables to a named file and then invoke the auto-link and compilation process. Once compiled the function would become part of the command set so that no future compilation would be necessary until the compiled subroutine is deleted. A user-defined function could then be set up to simplify the syntax of such a command. For the above function the user-defined function would be:

MATLAB Function using NAG Routine

```
function [X,U]=solve(A,B)
% SOLVE Calculates the approximate solution for a set of real
% real linear equation with multiple right-hand sides by
% Crout's factorization method.
% X-is a contains the solution vectors.
% U-contains the Crout factorization.
X=zeros(B);
WKSPACE=zeros(cols(A));
IFAIL=0;
[X,A]=NAG('F04AAF',A,int(rows(A)),B,int(rows(B)),int(cols(A)),int(cols(B)),
X,int(rows(X)),WSPACE,int(IFAIL))
```

The general purpose MATLAB file for invoking the auto-linking program to any of the library routines is:

General Purpose MATLAB Auto-linking Routine

```
function[b1,b2,b3,b4,b5,b6,b7,b8,b9,b10]=NAG(a1,a2,a3,a4,a5,a6,a7,a8,a9,a
10,a11,a12,a13,a14,a15,a16,a17,a18,a19,a20,a21,a22]
% NAG Invokes auto-linking compilation of NAG subroutines.
cmd='save LINKFILE'
%save correct number of arguments
for i=1:nargin
   cmd=[cmd,' a',num2str(i)];
end
exec('cmd')
% Invoke auto-linker compiler
! autolink
% load back variables returned form subroutine and contained in file LINKFILE
load LINKFILE
```

The command autolink invokes a process to write a program which links with the NAG routine and stores the returned variables in a file LINKFILE.

Since MATLAB is not a typed language it is necessary to provide an integer function to indicate when a variable is of type integer - this would be done by conversion of the integer to a character string which is preceded with a marker (e.g. "*"). Thus the file would of the form:

```
function in=int(a)
A=round(a)
in=['*',num2str(A)]
```

The command rows and cols simply returns the number of rows or columns of the matrix using the MATLAB command size.

This proposal could form a useful connection to a whole range of FORTRAN or other high-level language. However, such a connection due to its dependence on files as the communicating mechanism would be slow. For faster implementation, other inter-process communication mechanisms need to be explored, NAG optimization routines would also not be able to be implemented in this way since they require a separate FORTRAN subroutine to calculate the cost functional and constraints which must be called at every iteration.

1.3.7 Modern Computing Approaches

Modern computing approaches and the trend towards data-driven languages have been noted by some authors as the *Future for CACSD*. Shepherd [42], for example has highlighted the need for CACSD databases to support objects and graphical input/output. An object-oriented approach [see, for example, [44]) lends itself particularly to modern graphical input/output techniques such as iconic and pictorial representation. Although the introduction of objects requires a restructuring of the data structure aspects of MATLAB, as outlined in Section 1.3.3, some principles of object-oriented programming could be implemented without recourse to rewriting the package.

Objects are groups of procedures and data which communicate via message passing. One important aspect of object-oriented programming is the concept of class and inheritance. Class refers to protocols and characteristics of an object. Inheritance allows one to create objects which can inherit characteristics from other classes of objects. Variables are created by making instances of a class. A totally object-oriented approach may not be appropriate for a package such as MATLAB, however the concept could be applied by building up objects from user-defined functions and data and thus used as part of an iconic graphical interface.

Consider for example, the construction of a control configuration in preparation for parameter optimization. The problem here is data handling since there may be a number of control configurations which need to be tested. To invoke the necessary control structure requires the execution of both procedures and data to construct the necessary matrices into a general problem formulation. Under an object-oriented system the control configuration could be thought of as an object (as opposed to both procedures and data). By interface to a graphical input/output facility icons could then be used to represent abstract data types such as controller configuration, model choice and design method.

1.4 CACSD ENVIRONMENT FOR OPTIMIZATION APPLICATIONS

The incorporation of parameter optimization algorithms into a flexible CACSD environment is seen as the gateway to the widespread use of optimization techniques in Control System Design. A major part of this project has therefore been concerned with the development of such an environment which allows optimization problems to be coded in an interactive and flexible manner using a simple command language. One such environment has involved the upgrading and interconnection of a FORTRAN version of MATLAB [18] to an optimization package, ADS [39].

1.4.1 Integrating Optimization Software

The coupling of optimization sub-programs to an interactive design package, such as MATLAB, is not as straightforward as, for example, linking MATLAB to other numerical subroutines. This is because the majority of optimization programs are not directly subordinate to the main program but act themselves as the controlling program by calling user-supplied subroutines which are required to return the appropriate information to the optimization program. This makes the introduction of such programs into an environment such as MATLAB a difficult process since, for flexible programming, performance indices and constraints need to be evaluated using MATLAB user-defined functions. In order to overcome these difficulties, pseudo subroutines must be created and complicated inter-process communication techniques must be applied in order for information to be transferred between MATLAB and the optimizer. The problem is further compounded due to the requirement for a fast and accurate data link and by the generally large amount of data associated with the optimizer in terms of gradient matrices, constraint vectors and optimization parameters.

Some of these problems may be overcome by packages such as ADS which allows the optimizer to be subordinate to the calling program so that the Optimizer is called on an iterative basis by a controlling program which supplies values for the objective function and constraints. The problem here is that the ADS optimization subroutines retain parameters from one iteration to the next which are not stored in the calling program. This means that ADS cannot be spawned from MATLAB as can other processes. Therefore, the ADS package must be directly coupled into MATLAB or connected using slow and complicated inter-process communication methods.

As the source code to PRO-MATLAB is not available this makes direct linking to ADS very difficult. Further, since PRO-MATLAB was not available at the outset of this project, a public-domain FORTRAN version Of MATLAB has been used. This has the advantage that subroutines may be directly and easily linked into the package.

1.4.2 The Design Environment

Fig. 1.2 shows how the design environment is structured. The FORTRAN version of MATLAB was upgraded and called MATLANG because of its language capabilities. MATLANG parallels to a large extent the development of PRO-MATLAB offering the following additional features to the FORTRAN version of MATLAB.

- User-defined functions called as in PRO-MATLAB.
- IF,FOR,WHILE loops can be written over more than one line.
- Variables can have up to 10 characters (instead of 4).
- Graphics facility for TEK4010.
- On-line Programming without recourse to a text editor.

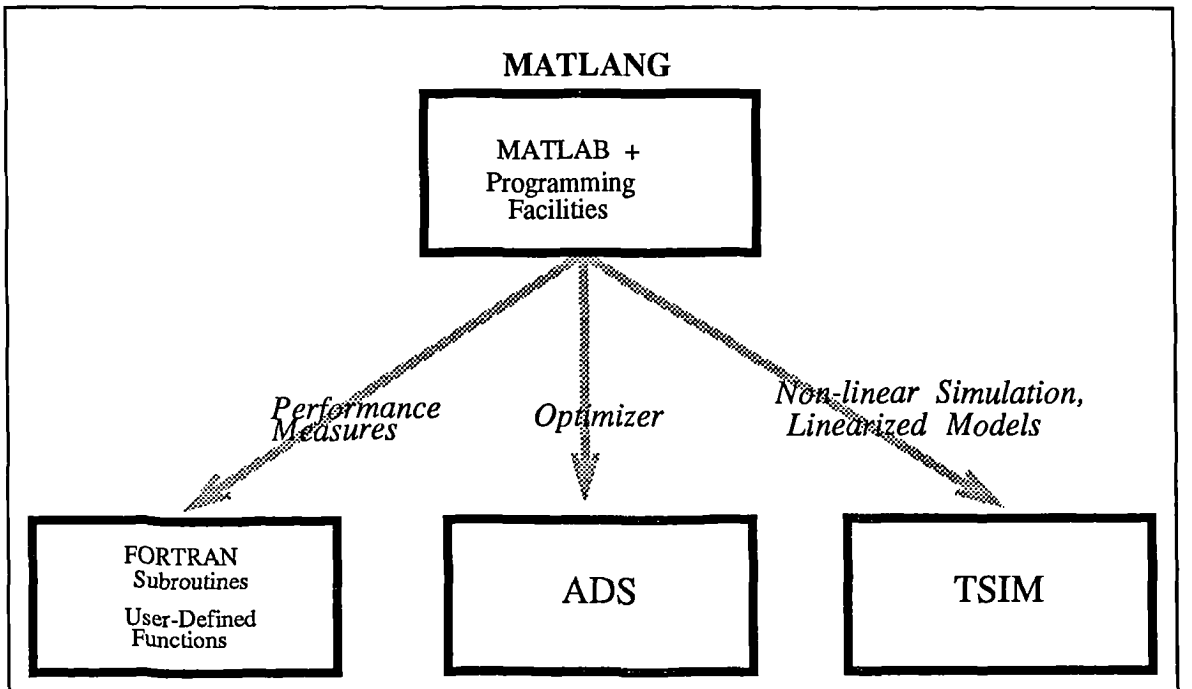

Fig. 2.2 MATLANG CACSD Environment

This version of MATLAB has the advantage that FORTRAN subroutines can be directly linked into the package. This offers the advantage of speed up over PRO-MATLAB user-defined functions. Several FORTRAN subroutines were written and added to the package so that responses for time and frequency could be obtained quickly for inclusion into performance indices for optimization problems. The package has been linked to a non-linear dynamic simulator, TSIM [34] for access to non-linear simulations and direct down loading of linearized models. To preserve modularity, this link is performed through files.

The key features of the design environment are that optimization may be easily and interpretively entered into the program. The user has a number of options in the choice of optimization algorithm. The optimization may be interrupted during execution and the user may change algorithms or reconfigure the optimization problem in a flexible manner. For alternative descriptions of the environment see also Grace and Fleming [40,41].

An example of an optimization program to minimize Rosenbrock's function (see also Part 2):

$$f(x) = 100*(x_2-x_1^2)^2 + (1-x_1))^2 \qquad (1.1)$$

is shown below.

MATLANG Optimization Example

```
X=<-1.2,1,0>                                    %Initalize X
PARA=<0,0,5,4,0>                %Set Algorithm Choice and Parameters
while PARA(1)<>3                          %Check Termination Paramneter
   F=100*(X(2)-X(1)**2)**2  +  (1-X(1))**2;          %Evaluate Function
   <X,PARA>=optim(F,X,PARA);          %Recursively Call Optimizer
end
```

Notice the call to the optimizer <X,PARA>=optim(F,X,PARA); which calls the ADS optimization package on an iterative basis. The variable PARA contains information regarding algorithm choices and relevant optimization parameters.

1.4.3 ADS Optimization Package

ADS (Automated Design Suite) is an optimization suite of programs written in FORTRAN and developed by G.N.Vanderplaats at the Naval Postgraduate School, Monterey, California. The optimization suite offers two significant advantages over existing optimization libraries such as NAG:- flexibility of algorithm choice and improved program control

ADS Algorithm Choice

The program has been modularized so that the user has options over the algorithms to be used at three levels of calculation. These levels have been named *Strategy, Optimizer* and *One-Dimensional Search.*

At the *Strategy* level the user decides whether to use, for instance, Sequential Unconstrained Minimization [46,47], Sequential Linear Programming [48,49] or Sequential Quadratic Programming (see Section 2.6.1) to solve a constrained optimization problem. Essentially, this either converts a constrained optimization problem into an unconstrained optimization problem or tries to deal with the constraints directly. If the problem is unconstrained the user may ignore the *Strategy* algorithm. The *Strategy* algorithm may also be ignored when the algorithm to be used does not first convert a constrained optimization problem to an unconstrained optimization problem. This may be the case using either the Method of Feasible Directions [50,51] or a Random Search which are chosen at the *Optimizer* level.

At the *Optimizer* level the method for solution of an easier sub-problem is chosen. This may be an unconstrained optimization problem, such as a Quasi-newton methods (cf. Section 2.2.1), including the Broyden-Fletcher-Golfarb-Shanno (BFGS) method. Alternatively a constrained sub-problem is chosen, such as, a Quadratic Programming [52] method.

Having found a search direction at the *Strategy* and *Optimizer* levels, a line search is performed to try to locate the minimum along the line determined by the current point and the search direction. This algorithm may be chosen at the *One-Dimensional Search* algorithm level. The algorithms available consist of interpolation and extrapolation methods, and a number of search algorithms (see Section 2.3.3 for comparison of methods).

Although not every permutation of algorithms within each of the three levels is possible, the user still has a wide choice of combinations available. In this way, different combinations can be tested to find the most suitable for the class of problem being considered.

ADS Program Control Flow

ADS is called on an iterative basis by a controlling program. This means that ADS can be interactively interrupted and the algorithm changed or problem formulation updated. This contrasts to many optimization subroutines, such as those provided by NAG, where the function to be minimized must be evaluated in a user-supplied FORTRAN subroutine. The subroutine name is passed as a parameter to the optimizer and is called when needed from the optimizer. This means that the control resides totally within the optimizer itself. The disadvantage of this is that it makes interaction difficult. Further, if the performance indices are functions of variables other than purely the design

variables, then they must be stored and loaded by file or hard coded into the compiled function. This is the case in control where the cost will usually be a function of both the design parameters (controller) and the system parameters (plant).

1.4.4 Present Software Status.

The package has proved a useful test-bed for optimization algorithms and for the development of Control System Design methodologies. However, PRO-MATLAB was thought to offer improved facilities in terms of graphics, software support and speed due to improved numerical routines. Software development has therefore continued with PRO-MATLAB and the optimization algorithms have been developed as MATLAB user-defined functions. MATLANG is not however totally redundant and is proving a useful test-bed for optimization algorithms and for direct linking with existing FORTRAN and C subroutines.

1.5 REVIEW

The emergence of packages such as MATLAB and the tend towards integration of existing software within CACSD has been highlighted. Improvements to the MATLAB package have been suggested in the form of simple realizable changes to an already useful package. An interactive design environment has been described for optimization application of Control System Design by the integration of an optimization package, ADS, and an upgraded FORTRAN version of MATLAB. In the next part we will examine how optimization algorithms can be directly implemented in the MATLAB command language.

1.6 REFERENCES

Overviews of CACSD, General References

[1] Fleming P.J. and De Oliveira C., "Computer Aided Control System Design - A Tutorial," Proc. Control 88, Univ. of Oxford, April 1988.

[2] Hammond P.H., "Developments In Computer-Aided Control System Design," Computer-Aided Design, Vol.18 No.10, pp.552-557, 1986.

[3] IEEE Control Systems Magazine, (Special Issue on CACSD), Vol.2, No.4, 1982.

[4] Astrom, K.J., "Computer-Aided Modeling, Analysis And Design In Computer-Aided Control System Design," IEEE Control Systems Magazine, Vol.3, No.2, pp.4-16, May 1983.

[5] Birdwell J.D., "Future Directions In Computer-Aided Control System Design Software Development," IEEE Control Systems Magazine, Vol 3, No.2, pp.11-14, Feb 1983.

[6] Polak E., "Optimization-Based Computer-Aided-Design of Control Systems," Proc. JACC, University of Virginia, Charlotesville, VA, Vol. 1., June 1981.

Aspects Of CACSD

[7] Breuer P.T, Maciejowski J.M., Phaal P., "Definition and Implementation of a Data Model for Computer-Aided Control Engineering," Proc. 10th World Congress on Automatic Control, IFAC, Munich, 1987.

[8] Breuer P.T. and Maciejowski J.M. "An Extended Relational DBMS for Control Engineering," 7th International Conf. on Systems Analysis and Optimization, Antibes, France, 1986.

[9] Maciejowski, J.M., "Data Structures for Control System Design," Proc. EUROCON 84, 1984.

[10] Rimvall, M. "Man-Machine Interfaces and Implementational Issues In Computer-Aided Control Systems Design," DSc Dissertation, Swiss Federal Institute of Technology, Zurich, 1986.

[11] Mansour, M., Rimvall, M. and Schaufelberger, W. "Computer-Aided Design of Control Systems, and Integrated Approach," Proc. 3rd IFAC/IFIP Symposium on Computer-Aided Design in Control and Engineering Systems (CADCE'85), Copenhagen, Denmark, pp.28-33, 1985.

[12] Rimvall M., "CACSD Software and Man-Machine Interfaces of Modern Control Environments," Trans. Inst. M.C., Vol.9, No.2, 1987.

[13] "Future Needs in Computer Aided Control Systems Design," Proc. SERC Workshop, Cambridge, Dec. 1988.

Selected CAD Packages, Integrated Environments and Numerical Libraries

[14] "ELCS - Extended List of Control Software," No.4; (D.Frederick, C.Herget, R.Kool and M.Rimvall, eds.) ETH Zurich, Switzerland, Jan 1988.

[15] Munro N. and Hammond P.H., "Computing and Design Techniques for Control Engineering - A UK Initiative," Proc. IMACS 12th World Congress, Paris, Vol.5, pp.97-10, 1988.

[16] "ECSTASY User's Manual," Rutherford Appleton Laboratory, Didcot, UK, 1989.

[17] PA Set Tools, PA Consulting Group, CAD Cambridge Lab., Melbourn, Royston, Herts., 1988.

[18] Moler, C., "MATLAB User's Guide," Department of Computer Science, University of New Mexico, Albuerque, 1980.

[19] Moler, C., Little, J.,Bangert , S.N. and Kleinman, S., "PC-MATLAB User's Guide," The MathWorks, Inc., South Natick, MA 01760,USA, 1985.

[20] Moler, C., Little, J., Bangert, S.N., "PRO-MATLAB User's Guide," The MathWorks, Inc., South Natck, MA 01760,USA, 1987.

[21] "CTRL-C User's Guide," Systems Control Technology, Inc, 4.0 Edition, September, 1986.

[22] Rimvall M., "IMPACT, Interactive Mathematical Program for Automatic Control Theory, a Preliminary User's Manual," Department of Automatic Control, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1983.

[23] Shah S.C., Floyd M.A. and Lehman L.L.,"MATRIX-X:Control Design and Model Building CAE Capabilities," Computer-Aided Control Systems Engineering (M. Jamshidi and C.J. Herget,eds.), pp 181-207, North-Holland Elsevier Science Publishers, Amsterdam, 1985.

[24] "MATRIX-X:User's Guide, MATRIX-X:Reference Guide, MATRIX-X: Training Guide, Command Summary and On-line Help," Integrated Systems, Inc., Palo Alto, CA 94031.

[25] SLICE (Principal Developers: Control Systems Research Group, Kingston Polytechnic), NAG Central Office, Oxford.

[26] Hearn A.C.,"REDUCE User's Manual," The Rand Corporation, Santa Monica, 1985.

[27] MACSYMA: Symbolics Inc., Cambridge, MA 02143, 1986.

[28] Polak E., Siegel P., Wuu T., Nye W.T. and Mayne D.Q., "DELIGHT.MIMO: An Interactive Optimization-Based Multivariable Control System Design Package," IEEE Control Systems Magazine, pp. 9-14, Dec 1982

[29] Nye W.T.,"DELIGHT: An Interactive System for Optimization-Based Engineering Design," Ph.D. Thesis, Dept. EECS, Berkeley, Univ. California, 1983.

[30] Weislander, J.,"IDPAC Commands -User's Guide," Department of Automatic Control, Lund Institute of Technology,Lund,Sweden, Report CODEN: LUTFD2/(TFRT-3157)/1-108/1980.

[31] Fleming P.J., "SUBOPT - A CAD Program for Suboptimal Regulators," Proc. Inst. Meas. Control Workshop on "Computer Aided Control System Design," 19-21 September, 1984, Sussex, U.K., pp.13-20.

[32] "ACSL Users' Guide," Mitchell and Guathier Associates, Inc., Concord, MA 01742.

[33] "SIMNON User's Guide," Dept. of Automatic Control, Lund Inst. of Technology, Lund, Sweden, 1988.

[34] "TSIM Users' Guide and Reference Manual," Cambridge Control Ltd, High Cross, Madingley Road, Cambridge. 1986.

[35] "The NAG Fortran Library Manual," The Numerical Algorithms Group, Mayfield House, 256 Banbury Road, Oxford, U.K.

[36] Dongarra J.J. et al., "LINPACK Users' Guide," Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[37] Smith B.T. et al., "Matrix Eigensystem Routines - EISPACK Guide," Lecture Notes in Computer Science, Vol. 6, second edition, Springer Verlag, 1976.

[38] Garbow B.S. et al., "Matrix Eigensystem Routines - EISPACK Guide Extension," Vol. 51, Springer Verlag, 1977.

[39] Vanderplaats, G.N., "ADS - A Fortran Program for Automated Design Synthesis," Naval Postgraduate School, Monterey,CA, USA, 1983.

[40] Grace, A.C.W. and Fleming P.J., "A Design Environment for Control System Design via Multi-Objective Optimization", Proc. 12th IMACS World Congress, Paris, Vol.2, pp.572-574, July 1988.

[41] Grace, A.C.W., Fleming, P.J. "Design Environment for Optimization Applications of Control System Design", in "Advanced Computing Concepts and Techniques in Control Engineering" NATO ASI Series(F) Vol 47, Springer Verlag, pp.495-512, 1988

Computing Aspects

[42] Shepherd, D., "Alternatives to ECSTASY," IEE Colloquium Digest No.1987/97, pp.5/1-5/4

[43] Maskell K. "Building Software Bridges," Systems International, pp.63-64, Jan 1987.

[44] Bell D., Murray I., and Pugh I., "Software Engineering - A Programming Approach," Prentice-Hall, 1987.

[45] "Software Development Environments: A Tutorial," IEEE Computer Society, Los Alamitos, CA, USA, (Wasserman A.I. ed.), Catalog No. EHO 187-5, 1981.

ADS Optimization Algorithms

[46] Fiacco, A.V. and McCormick, G.P. "Nonlinear Programming: Sequential Unconstrained Minimization Techniques," John Wiley and Sons, 1968.

[47] Frasad, B., "A Class of Generalized Variable Penalty Methods for Nonlinear Programming," Journal of Optimization Theory and Applications, Vol.35, No.2, pp.159-182, 1981

[48] Kelley, J.E., "The Cutting Plane Method for Solving Convex Programs," J.SIAM, pp.703-712, 1960

[49] Moses, F., "Optimum Structural Design Using Linear Programming," Proc. A.S.C.E., Vol.90, pp.89-104, 1964.

[50] Zoutendijk, M., "Methods of Feasible Directions," Elsevier Publishing Co., Amsterdam, 1960

[51] Vanderplaaats, G.N. and Moses, P., "Structural Optimization by Methods of Feasible Directions," J. of Computers and Structures, Vol.3, Pergamon Press, pp.739-755, 1973.

[52] Himmelblau, D.M., "Applied Nonlinear Programming," McGraw-Hill, 1972.

Part 2

OPTIMIZATION

Summary - Optimization methods which have been implemented as part of a MATLAB Optimization Toolbox will be discussed. The Toolbox includes unconstrained, least squares, constrained and multi-objective implementations. Algorithms have been chosen for their efficiency and include the BFGS method for unconstrained optimization, Gauss-Newton and Levenberg-Marquardt for least squares optimization and Sequential Quadratic Programming (SQP) for constrained optimization. Efficient line search procedures have been developed for unconstrained problems which are compared against procedures in the ADS package. Multi-objective optimization is addressed using the Goal Attainment method allowing design problems to be expressed in a convenient and solvable format. SQP has been tailored for the solution of Goal Attainment problems by exploiting characteristics of the objective function and constraints.

2.1 INTRODUCTION

ADVANCES in modern computing and the ability to describe increasingly complex systems have prompted the development and use of optimization techniques as a way of performing complex design tasks. This has enabled the improvement of cost, reliability and performance in a wide range of applications. As engineering design is automated to an ever-increasing level, realistic measures of performance, which may be in the form of multiple non-linear objectives, need to be incorporated into the design. In tandem with this requirement, is the need to model the system accurately so that reliable measures of system performance can be obtained. These measures may be functions of complex relationships such as the results of expensive computer simulations. It therefore becomes profitable to invest in careful problem formulation and algorithm efficiency. Hence, attention will be focussed on efficient solution procedures for a number of representative optimization design problems.

In Part 1 the difficulty of linking existing FORTRAN optimization code into a interactive environment was outlined. A number of optimization methods have therefore been programmed in the MATLAB command language which form an Optimization Toolbox (a users' manual is given in Appendix A). These routines are easy-to-use and avoid the inherent difficulties associated with communication to MATLAB with existing optimization software. The disadvantage is that, due to the interpretive nature of MATLAB, the code is slower to execute than, for instance, a FORTRAN realization of the same algorithm. For large problems, where matrix calculations dominate the CPU time, the optimization routines, which use the efficient MATLAB matrix algorithms, are able to approach the speed of FORTRAN implementations. Besides, since cost functionals and constraints are often computationally expensive to evaluate, slower optimization execution time is compensated for by the iterative efficiency of the methods employed. Moreover, the ease of programming using MATLAB and the associated optimization routines, considerably reduces the time to code and refine optimization problems. This allows a high-level of interaction on the part of the user.

Each of the routines will be presented in terms of a short general introduction, followed by a more detailed description about practical aspects of the implementation. For an introduction to the subject, the books by Gill and Murray [1], and Fletcher [2] are recommended. Brayton *et al* [3] give a good overview of the state of the art in optimization while Mayne *et al* [4], and Nye and Tits [5] give useful insights into the subject.

2.1.1 Parametric Optimization

The aim in parametric optimization is to find a set of design parameters, $x=\{x_1, x_2, \ldots x_n\}$, which can in some way be defined as optimal. In a simple case this could be the minimization or maximization of some system characteristic which is dependent on x. In a more advanced formulation the objective function , $f(x)$, to be minimized or maximized could be subject to constraints which may be in the form

of equality constraints, $g_i(x)=0$ $(i=1...m_e)$, inequality constraints, $g_i(x)\leq 0$ $(i=m_e+1...m)$, and/or parameter bounds, x_l, x_u. A General Problem (GP) description can therefore be stated as

GP

$$\underset{x\in\Re^n}{\text{minimize}} \quad f(x)$$

$$\text{subject to:} \quad g_i(x) = 0, \quad i=1, .., .m_e$$
$$g_i(x) \leq 0 \quad i=m_e+1, \ ..., \ m$$
$$x_l \leq x \leq x_u \tag{2.1}$$

where x is the vector of design parameters ($x\in\Re^n$), f the objective function ($f:\Re^n\Rightarrow\Re$) and g is the vector of equality and inequality constraints ($g:\Re^n\Rightarrow\Re^m$).

The efficient and accurate solution of this problem is not only dependent on the size of the problem in terms of the number of constraints and design variables but also on characteristics of the objective function and constraints. When both the objective function and the constraints are linear functions of the design variable the problem is known as a Linear Programming problem (LP). Quadratic Programming (QP) concerns the minimization or maximization of a quadratic objective function which is linearly constrained. For both the LP and QP problems reliable solution procedures are readily available. More difficult to solve is the Non-linear Programming (NP) problem in which the objective function and constraints may be non-linear functions of the design variables. Solution of the NP problem generally requires an iterative procedure to establish a direction of search, at each major iteration. This is generally achieved by the solution of an LP, a QP or an unconstrained sub-problem.

2.2 UNCONSTRAINED OPTIMIZATION

Although there exist a wide spectrum of methods for unconstrained optimization, they can be broadly categorized in terms of the derivative information that is, or is not, used. Search methods which use only function evaluations (e.g. the simplex search of Nelder and Mead [61]) are most suitable for problems which are very non-linear or have a number of discontinuities. Gradient methods are generally more efficient when the function to be minimized is continuous in its first derivative. Higher order methods, such as Newton's method, are only really suitable when the second order information is readily and easily calculated since calculation of second order information, using numerical differentiation, is computationally expensive.

Gradient methods use information about the slope of the function to dictate a direction of search where the minimum is thought to lie. The simplest of these is the method of steepest descent in which a search is performed in a direction, $-\nabla f(x)$ (where $\nabla f(x)$ is the gradient of the objective function). This method is very inefficient when the function to be minimized has long narrow valleys as, for example, is the case for Rosenbrock's function.

$$f(x) = 100*(x_2-x_1^2)^2 + (1-x_1))^2 \tag{2.2}$$

The minimum of this function is at $x=\{1,1\}$, $f(x)=0$. A contour map is shown in Fig. 2.1 which shows the solution path to the minimum for a steepest descent implementation starting at the point {-1.9,2}. The optimization was terminated after 1000 iteration; still a considerable distance form the minimum. The black areas are where the method is continually zig-zagging from one side of the valley to another.

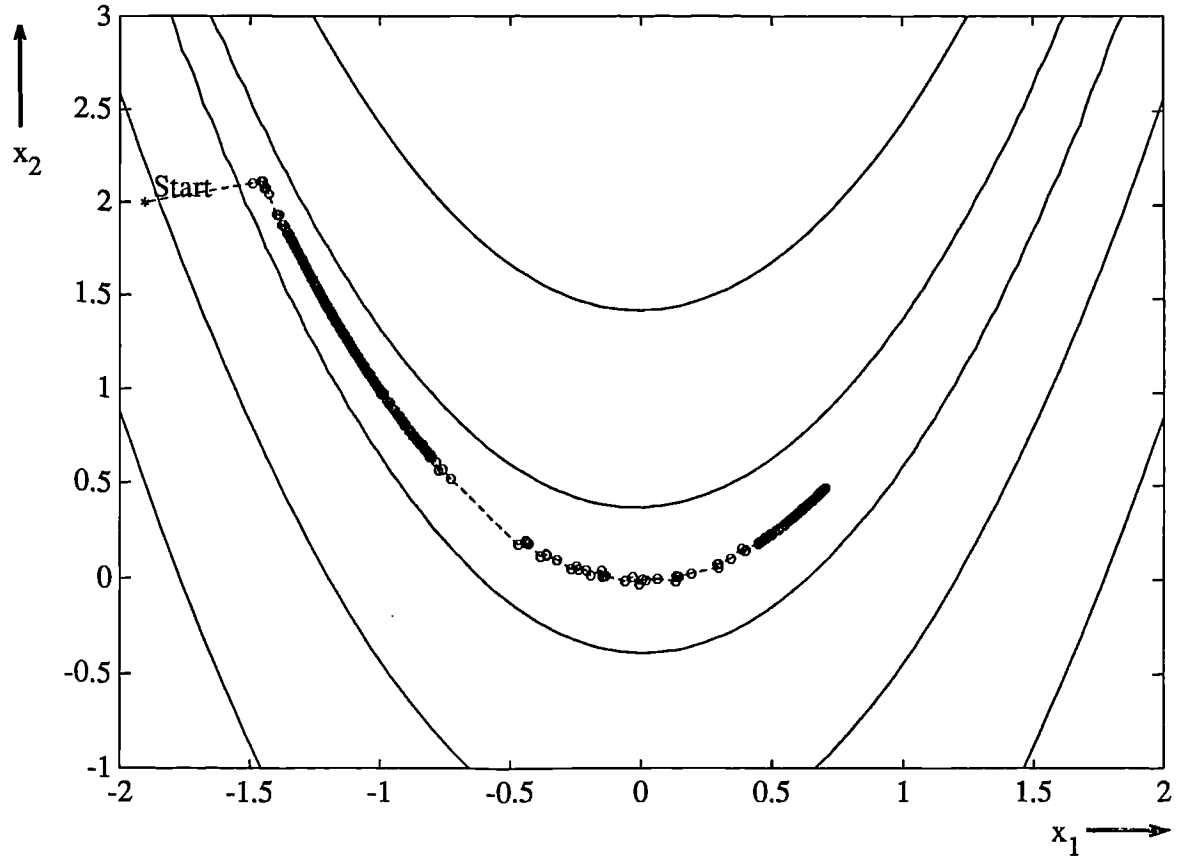

Fig. 2.1 Steepest Descent Method on Rosenbrock's Function (Eq. 2.2)

This type of function (Eq. 2.2) is notorious in unconstrained examples because of the way the curvature bends around the origin (also known as the banana function). Eq. 2.2 will be used throughout this part to illustrate the use of a variety of optimization techniques. The contours have been plotted in exponential increments due to the steepness of the slope surrounding the U-shaped valley.

2.2.1 Quasi-Newton Methods

Of the methods which use gradient information, the most favoured are the quasi-Newton methods. These methods build up curvature information at each iteration to formulate a quadratic model problem of the form:

$$\underset{x \in \Re^n}{\text{minimize}} \quad \{ f(x) = \tfrac{1}{2}\, x^T H x + b^T x + c \} \tag{2.3}$$

where the Hessian matrix, H, is a positive definite symmetric matrix, b is a constant vector and c is a constant. Optimal solution for this problem occurs when the partial derivatives of x go to zero, i.e.

$$\nabla f(x) = Hx^* + b = 0. \tag{2.4}$$

The optimal solution point, x^*, may thus be written as

$$x^* = -H^{-1} b. \tag{2.5}$$

Newton-type methods (as opposed to quasi-Newton methods) calculate H directly and proceed in a direction of descent using a line search method to locate the minimum after a number of iterations. Calculating H numerically involves a large amount of computation. Quasi-Newton methods avoid this, by using the observed behaviour of $f(x)$ and $\nabla f(x)$ to build up curvature information, in order to make an approximation to H using an appropriate updating technique.

A large number of Hessian updating methods have been developed. It is now generally recognized that the formula of Broyden [6], Fletcher [7], Goldfarb [8] and Shanno [9] (BFGS) is the most effective for use in a general purpose method. The formula is given by

BFGS

$$H_{k+1} = H_k + \frac{q_k q_k^T}{q_k^T s_k} - \frac{H_k^T H_k}{s_k^T H_k s_k}$$

where $s_k := x_{k+1} - x_k$

$q_k := \nabla f(x_{k+1}) - \nabla f(x_k)$ (2.6)

As a starting point H_0 can be set to any symmetric, positive definite matrix; for example, the unit matrix, I. If one wants to avoid the inversion of the Hessian H it is possible to derive an updating method in which the direct inversion of H is avoided by using a formula which makes an approximation of the inverse Hessian, H^{-1}, at each update. A well known procedure is the DFP formula of Davidon [10], Fletcher and Powell [11]. This uses the same formula as the above BFGS (Eq. 2.6) method except that q_k is substituted for s_k.

The gradient information is either supplied through analytically calculated gradients, or derived by partial derivatives using a numerical differentiation method via finite differences. This involves perturbing each of the design variables, x, in turn and calculating the rate of change in the objective function.

At each major iteration, k, a line search is performed in the direction:

$$d = - H_k^{-1} \nabla f(x_k) \tag{2.7}$$

The quasi-Newton method is illustrated by the solution path on Rosenbrock's function (Eq. 2.2) in Fig. 2.2. The method is able to follow the shape of the valley and converges to the minimum after 140 function evaluations using only finite difference gradients.

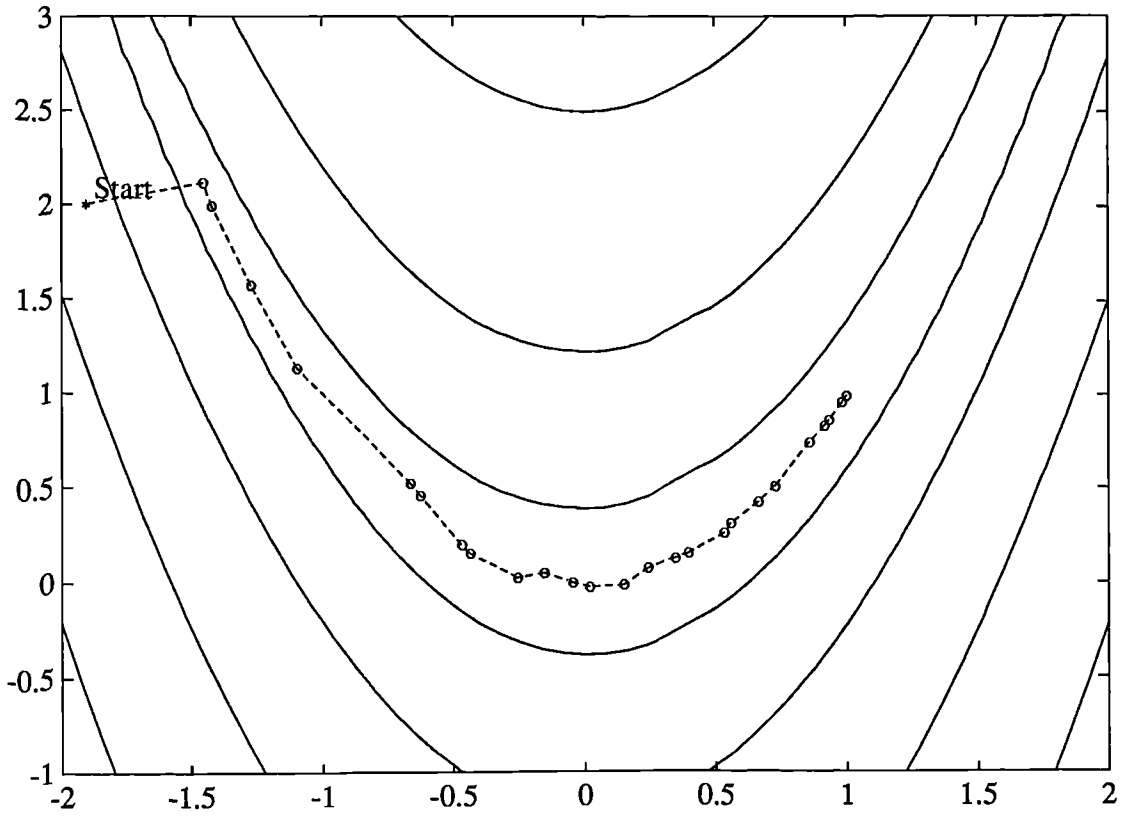

Fig. 2.2 BFGS Method on Rosenbrock's Function

2.2.2 Line Search

Most unconstrained and constrained methods use the solution of a sub-problem to yield a search direction in which the solution is estimated to lie. The minimum along the line formed from this search direction is generally approximated using a search procedure (e.g. Fibonacci, Golden Section) or by a polynomial method involving interpolation or extrapolation (e.g. Quadratic, Cubic). Polynomial methods approximate a number of points with a univariate polynomial whose minimum can be calculated easily. Interpolation refers to the condition that the minimum is bracketed (i.e. the minimum lies in the area spanned by the available points), whereas extrapolation refers to a minimum located outside the range spanned by the available points. Extrapolation methods are generally considered unreliable for estimating minima for non-linear functions. However, they are useful for estimating step length when trying to bracket the minimum as will be shown in Section 2.3.2. Polynomial interpolation methods are generally the most effective in terms of efficiency when the function to be minimized is continuous. The problem is to find a new iterate $\mathbf{x}_{k+1}$ of the form

$$\mathbf{x}_{k+1} = x_k + \alpha^* d \tag{2.8}$$

where x_k denotes the current iterate, d the search direction obtained by an appropriate method and α^* is a scalar step length parameter which is the distance to the minimum.

Quadratic Interpolation

Quadratic interpolation involves a data fit to a univariate function of the form

$$f(x) = a\alpha^2 + b\alpha + c \tag{2.9}$$

where an extremum occurs at a step length of

$$\alpha^* = -\frac{b}{2a} \tag{2.10}$$

This point may be a minimum or a maximum. It is a minimum when interpolation is performed (i.e. using a bracketed minimum) or when a is positive. Determination of coefficients, a and b, can be found using any combination of three gradient or function evaluations. It may also be carried out with just two gradient evaluations. The coefficients are determined through the formulation and solution of a linear set of simultaneous equations. Various simplifications in the solution of these equations can be achieved when particular characteristics of the points are used. For instance, the first point can generally be taken as $\alpha=0$. Other simplifications can be achieved when the points are evenly spaced. A general problem formula is as follows:

Given three unevenly spaced points $\{x_1, x_2, x_3\}$, and their associated function values $\{f(x_1), f(x_2), f(x_3)\}$, the minimum resulting from a second-order fit is given by:

Quadratic Interpolation

$$x_{k+1} = \frac{1}{2}\,\frac{\beta_{23}f(x_1) + \beta_{31}f(x_2) + \beta_{12}f(x_3)}{\gamma_{23}f(x_1) + \gamma_{31}f(x_2) + \gamma_{12}f(x_3)}$$

where: $\beta_{ij} := x_i^2 - x_j^2$

$\gamma_{ij} := x_i - x_j$ $\quad \{i,j\}=\{2,3\}, \{3,1\}, \{1,2\}$ (2.11)

In order for interpolation to be performed, as opposed to extrapolation, it is necessary for the minimum to be bracketed so that the points can be arranged in order that:

$$f(x_2) < f(x_1) \text{ and } f(x_2) < f(x_3).$$

Cubic Interpolation

Cubic interpolation is useful when gradient information is readily available or when more than three function evaluations have been calculated. It involves a data fit to the univariate function,

$$f(x) = a\alpha^3 + b\alpha^2 + c\alpha + d, \tag{2.12}$$

where the local extrema are roots of the quadratic equation,

$$3a\alpha^{*2} + 2b\alpha^* + c = 0. \tag{2.13}$$

In order to find the minimum extremum the root should be taken which gives $6a\alpha^*+2b$ as positive. Coefficients, a and b, can be determined using any combination of four gradient or function evaluations, or alternatively, with just three gradient evaluations. The coefficients are calculated by the formulation

and solution of a linear set of simultaneous equations. A general formula, given two points,{ x_1, x_2 }, their corresponding gradients with respect to x, { $\nabla f(x_1)$, $\nabla f(x_2)$ }, and associated function values, $\{f(x_1), f(x_2)\}$ is:

Cubic Interpolation

$$x_{k+1} = x_2 - (x_2 - x_1)\frac{\nabla f(x_2) + \beta_2 - \beta_1}{\nabla f(x_2) - \nabla f(x_1) + 2\beta_2}$$

where:
$$\beta_1 = \nabla f(x_1) + \nabla f(x_2) - 3\frac{f(x_1) - f(x_2)}{x_1 - x_2}$$

$$\beta_2 = \{\beta_1^2 - \nabla f(x_1)\nabla f(x_2)\}^{1/2} \qquad (2.14)$$

2.3 QUASI-NEWTON IMPLEMENTATION

A quasi-Newton algorithm has been implemented as part of an Optimization Toolbox (Appendix A). The algorithm consists of two phases:

(1) Determination of a direction of search
(2) Line search procedure.

Implementation details of the two phases will be discussed below.

2.3.1 Hessian Update

The direction of search is determined by a choice of either the BFGS (Eq. 2.6) or the DFP method given in Section 2.2.1. The Hessian, $\boldsymbol{H}$, is always maintained to be positive definite so that the direction of search, $\boldsymbol{d}$, is always in a descent direction. This means that for some arbitrarily small step, α, in the direction, $\boldsymbol{d}$, the objective function will decrease in magnitude. Positive definiteness of $\boldsymbol{H}$ is achieved by ensuring that $\boldsymbol{H}$ is initialized to be positive definite and thereafter $q_k^T s_k$ (from Eq. 2.6) is always positive. The term $q_k^T s_k$ is a product of the line search step length parameter, α_k, and a combination of the search direction, $\boldsymbol{d}$, with past and present gradient evaluations:

$$q_k^T s_k = \alpha_k (\nabla f(x_{k+1})^T \boldsymbol{d} - \nabla f(x_k)^T \boldsymbol{d}). \qquad (2.15)$$

The condition that $q_k^T s_k$ is positive is always achieved by ensuring that a sufficiently accurate line search is performed. This is because the search direction, $\boldsymbol{d}$, is a descent direction so that α_k and $-\nabla f(x_k)^T \boldsymbol{d}$ are always positive. Thus, the possible negative term $\nabla f(x_{k+1})^T \boldsymbol{d}$ can be made as small in magnitude as required by increasing the accuracy of the line search.

2.3.2 Line Search Procedures

Two line search strategies are used depending on whether gradient information is readily available or whether it must be calculated using a finite difference method. When gradient information is available the default is to use a cubic polynomial method. When gradient information is not available the default is to use a mixed quadratic and cubic polynomial method.

Cubic Polynomial Method

In the proposed cubic polynomial method a gradient and a function evaluation is made at every iteration, k. At each iteration an update is performed when a new point is found, x_{k+1}, which satisfies the condition that

$$f(x_{k+1}) < f(x_k). \tag{2.16}$$

At each iteration a step, α_k, is attempted to form a new iterate of the form

$$x_{k+1} = x_k + \alpha_k d. \tag{2.17}$$

If this step does not satisfy the condition (Eq. 2.16) then α_k is reduced to form a new step, α_{k+1}. The usual method for this reduction is to use bisection (i.e. to continually halve the step length until a reduction is achieved in $f(x)$). However, it has been found, through numerical experimentation, that this procedure is slow when compared to an approach which involves using gradient and function evaluations together with cubic interpolation/extrapolation methods to identify estimates of step length.

When a point is found which satisfies the condition (Eq. 2.16) an update is performed if $q_k^T s_k$ is positive. If it is not, then further cubic interpolations are performed until the univariate gradient term $\nabla f(x_{k+1})^T d$ is sufficiently small so that $q_k^T s_k$ is positive.

It is usual practice to reset α_k to unity after every iteration. However, it should be noted that the quadratic model (Eq. 2.3) is generally only a good one near to the solution point. Therefore, α_k is modified at each major iteration to compensate for the case when the approximation to the Hessian is monotonically increasing or decreasing. To ensure that, as x_k approaches the solution point, the procedure reverts to a value of α_k close to unity, the values of $q_k^T s_k$, $-\nabla f(x_k)^T d$ and α_{k-1} are used to estimate the closeness to the solution point and thus to control the variation in α_k.

After each update procedure, a step length α_k is attempted, following which a number of scenarios are possible. Consideration of all the possible cases is quite complicated and so they are represented pictorially in Fig. 2.3, where the left hand point on the graphs represents the point x_k. The slope of the line bisecting each point represents the slope of the univariate gradient, $\nabla f(x)^T d$ which is always negative for the left hand point. The right hand point is the point x_{k+1} after a step of α_k is taken in the direction d. Cases 1 and 2 show the procedures which are performed when $\nabla f(x_{k+1})^T d$ is positive. Cases 3 and 4 show the procedures performed when $\nabla f(x_{k+1})^T d$ is negative. The notation $\min\{a,b,c\}$ refers to the smallest value of the set $\{a,b,c\}$.

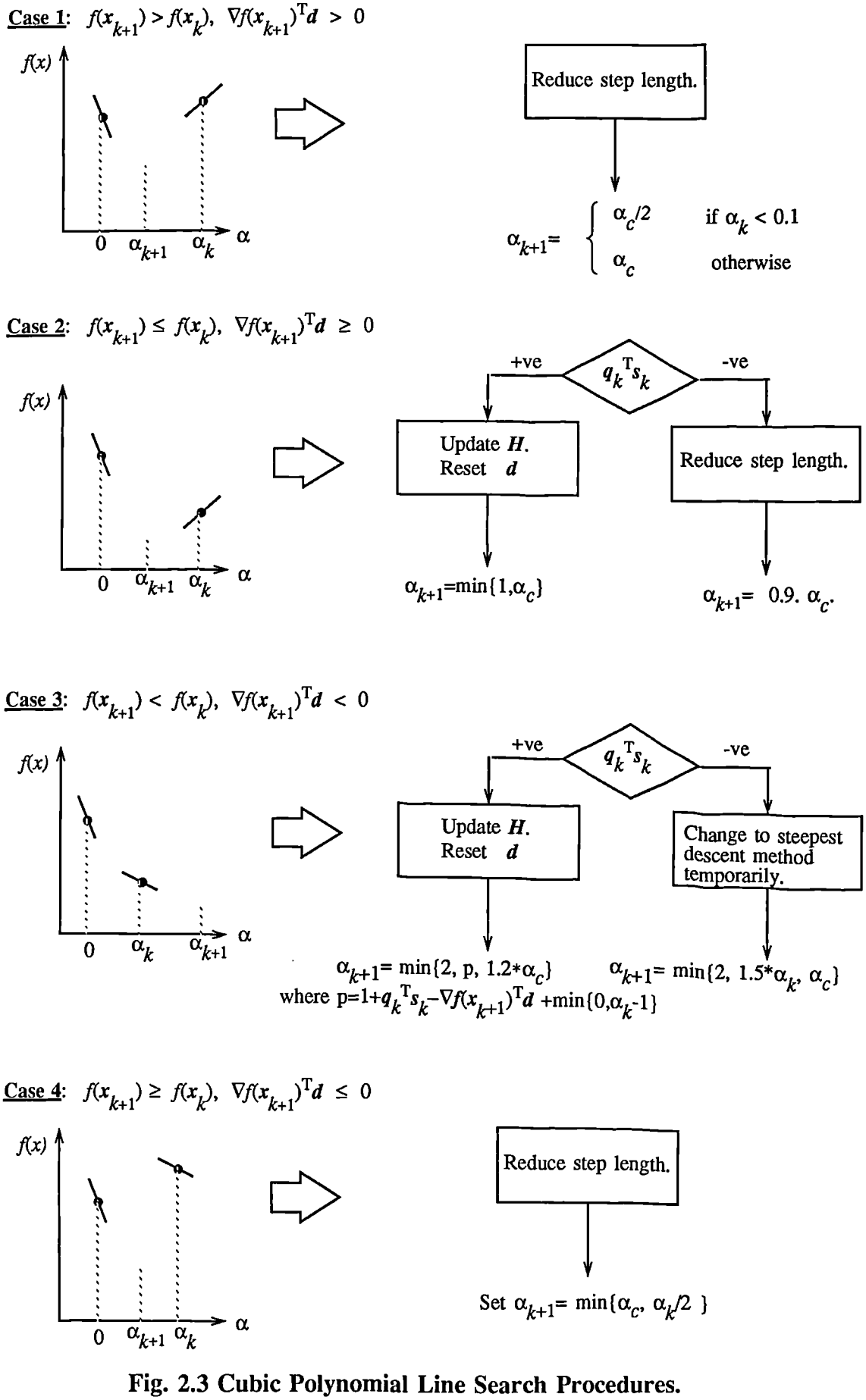

Fig. 2.3 Cubic Polynomial Line Search Procedures.

At each iteration a cubicly interpolated step length α_c is calculated which is used to adjust the step length parameter α_{k+1}. Occasionally for very non-linear functions α_c may be negative in which case α_c is given a value of $2*\alpha_k$. The methods for changing the step length have been refined over a period of time by considering a large number of test problems.

Certain robustness measures have also been included so that, even in the case when false gradient information is supplied, a reduction in $f(x)$ may be achieved by taking a negative step. This is realized by setting $\alpha_{k+1} = -\alpha_k/2$ when α_k falls below a certain threshold value (e.g. 1e-8). This is important for the case when extremely high precision is required if only finite difference gradients are available. A further robustness measure is incorporated for use on discrete functions. This is used when the step length falls below a second threshold (e.g. 1e-12) and consists of calculating a random search direction, d, and step length, α_{k+1}.

Mixed Cubic/Quadratic Polynomial Method

The use of the cubic interpolation/extrapolation method has proved successful for a large number of optimization problems. However, when analytic derivatives are not available the evaluation of finite difference gradients is computationally expensive. Therefore, another interpolation/extrapolation method has been implemented so that gradients are not needed at every iteration. The usual approach in these circumstances, when gradients are not readily available, is to use a quadratic interpolation method. The minimum is generally bracketed using some form of bisection method. This method, however, has the disadvantage that all the available information about the function is not used. For instance, a gradient calculation is always performed at each major iteration for the Hessian update. Therefore, given three points which bracket the minimum, it is possible to use cubic interpolation which is likely to be more accurate than using quadratic interpolation. Further efficiencies are possible if, instead of using bisection to bracket the minimum, extrapolation methods similar to those used in the cubic polynomial method are used.

Hence, the method which has been used in the MATLAB implementation is to find three points which bracket the minimum and to use cubic interpolation to estimate the minimum at each line search. The estimation of step length, at each minor iteration, j, is shown in Fig. 2.4 for a number of point combinations. The left hand point in each graph represents the function value $f(x_k)$ and univariate gradient $\nabla f(x_k)$ obtained at the last update. The right hand points represent the points accumulated in the minor iterations of the line search procedure. The terms α_q and α_c refer to the minimum obtained from a respective quadratic and cubic interpolation or extrapolation. For highly non-linear functions, α_c and α_q may be negative in which case they are set to a value of $2*\alpha_k$ so that they are always maintained to be positive. Cases 1 and 2 use quadratic interpolation with two points and one gradient to estimate a third point which brackets the minimum. If this should fail, cases 3 and 4 represent the possibilities for changing the step length when at least three points are available.

When the minimum is finally bracketed cubic interpolation is achieved using one gradient and three function evaluations. If the interpolated point is greater than any of the three used for the interpolation, then, it is replaced with the point with the smallest function value. Following the line search procedure the Hessian update procedure is performed as for the cubic polynomial line search method.

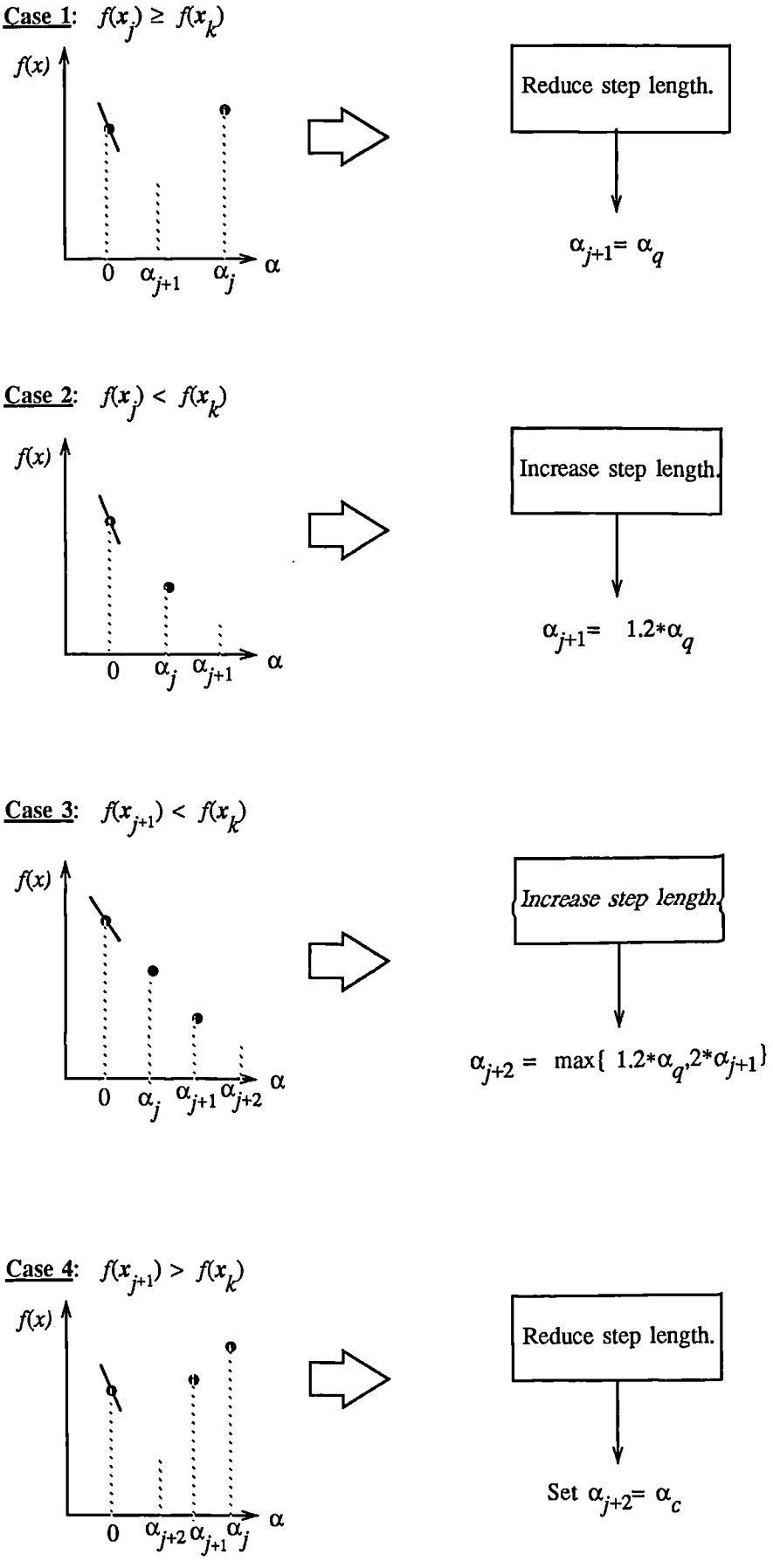

Fig. 2.4 Line Search Procedures With Only Gradient For The First Point

2.3.3 Comparison of Methods

The BFGS method (Eq. 2.6) and line search procedures described in Section 2.3.2 and implemented in MATLAB were compared against a BFGS implementation and line search strategies contained in the ADS Optimization package [39], (see also, Section 1.4.3). Four test problems were chosen of varying difficulty and non-linearity. Each problem was tested with ten different starting values.

In order to make a comparison in terms of efficiency, the normal stopping criteria of the packages were not used. Instead the optimization was terminated when the values of $f(x)$ dropped below a given threshold value. The problems, solutions and stopping criteria are given below:

Problem 1: $\min f(x) = 100*(x_2-x_1^2)^2 + (1-x_1))^2$

solution at $x = \{1,1\}$, $f(x) = 0$,
termination criterion, $f(x) <$ 1e-5.

Problem 2: $\min f(x) = e^{x1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)) - x_1$

solution $x = \{0.62580,-1.1258\}$, $f(x) = -0.56662$,
termination criterion, $f(x) < -0.56663$.

Problem 3: $\min f(x) = 100*(x_3-((x_1+x_2)/2)^2 + (1-x_1)^2 + (1-x_2)^2$

solution at $x = \{1,1,1\}$, $f(x) = 0$,
termination criterion, $f(x) <$ 1e-4.

Problem 4: $\min f(x) = (x_1^2+x_2^2+x_1x_2)^2 + \sin^2(x_1) + \cos^2(x_2)^2$

solutions at $x = \{-0.1554,0.6946\}$ and $x = \{0.1554,-0.6946\}$, $f(x) = 0.7732$,
termination criterion, $f(x) < 0.7733$.

The number of function and gradient evaluations for each problem are shown in Table 2.1 with starting values appearing in the first column. The first two problems used both gradient and function evaluations and the number of evaluations of each have been shown in separate columns. The last two problems used finite difference gradients and thus only the number of function evaluations have been entered. The finite difference interval was set to 1e-8 for each package.

The results are split into columns, where the first four columns represent the ADS implementation using different line search strategies, and the last two columns represent the MATLAB implemented strategies. The line search strategies are as follows:

(1) Golden Section search.

(2) Golden Section search followed by Cubic Interpolation.

(3)Polynomial Interpolation after bracketing the minimum.

(4) Polynomial interpolation/extrapolation without bracketing the minimum.

(5) MATLAB implemented Cubic Polynomial Interpolation/Extrapolation method.

(6) MATLAB implemented Mixed Polynomial Interpolation/Extrapolation method.

ADS reported a number of failures in which case f has been entered into the appropriate column (failure taken to mean function evaluations exceeding 2000 iterations). In order to make an appropriate comparison, only those values in which all the routines were successful have been used for the total and the average.

The MATLAB routines showed the least number of failures with the greatest efficiency being achieved by the mixed polynomial method.

Table 2.1 Function and gradient evaluations for two BFGS implementations

	ADS								MATLAB			
	1		2		3		4		5		6	
Problem No.	*Golden Section*		*Golden Section + Cubic Inter.*		*Bracketed Polynomial Interpolation*		*Unbracketed Polynomial Inter/Extrap*		*Cubic Inter/Extrap*		*Mixed Polynomial Inter/Extrap*	
x	func	grad	func	grad	func	grad	func	grad	func	grad	func	grad
1 {-1.2,1}	546	26	313	25	112	27	108	27	40	39	76	23
{2,2}	261	13	f	f	61	14	56	14	28	27	35	10
{-2,-2}	573	28	354	27	129	31	120	31	45	44	90	27
{2,-2}	170	9	138	12	37	8	45	11	20	19	15	4
{-2,2}	734	35	511	40	150	36	139	35	52	51	93	28
{0,0}	287	14	176	14	60	15	60	15	25	24	50	16
{10,10}	518	24	353	24	88	25	78	25	96	95	76	22
{-10,10}	1406	63	849	61	270	64	312	79	109	108	169	50
{10,-10}	670	27	353	24	88	25	81	18	90	89	138	41
{-10,-10}	1467	66	796	58	286	68	336	83	104	103	47	14
2 {-1,1}	128	6	85	6	28	6	28	7	11	10	18	6
{1,1}	101	5	67	5	26	5	23	6	16	15	21	6
{1,-1}	95	5	62	5	26	5	20	5	8	7	13	3
{-2,-2}	213	9	141	9	39	9	73	16	22	21	54	18
{2,-2}	110	5	80	5	28	5	33	7	8	7	20	5
{0,0}	104	5	70	5	25	5	24	6	8	7	12	4
{10,10}	469	18	333	18	93	19	84	22	58	57	50	15
{5,5}	288	11	194	11	63	13	59	15	22	21	38	11
{3,3}	137	6	98	6	48	10	57	14	18	17	48	14
{5,-5}	419	17	232	14	50	10	40	10	18	17	38	11
3 {-1.2,1,0}	*f*		*f*		*f*		*f*		89		96	
{1,1,0}	*f*		*f*		*f*		640		53		59	
{1,-1,0}	*f*		*f*		*f*		*f*		77		80	
{-2,-2,2}	*f*		*f*		*f*		*f*		173		146	
{2,-2,2}	*f*		*f*		*f*		*f*		73		106	
{0,0,0}	*f*		*f*		*f*		*f*		81		74	
{10,10,10}	*f*		*f*		1273		374		341		127	
{5,5,5}	649		424		215		211		189		111	
{3,3,3}	413		276		146		*f*		169		106	
{5,-5,-5}	*f*		*f*		*f*		*f*		97		103	
4 {-1,1}	112		78		34		27		16		16	
{1,1}	111		75		34		28		22		23	
{1,-1}	113		78		34		29		16		16	
{-2,-2}	138		96		42		41		64		35	
{2,-2}	112		77		36		32		25		23	
{10,10}	103		104		41		60		82		61	
{5,5}	103		75		38		45		61		48	
{3,3}	98		99		48		59		43		33	
{5,-5}	126		102		42		60		37		36	
{3,0.1}	122		87		41		46		52		29	
TOTAL	*9803*	374	*6500*	369	*2251*	386	*2358*	432	*1377*	752	*1496*	318
AVERAGE	***327.7***	**17.2**	***216.7***	**19.4**	***75.0***	**20.3**	***78.6***	**22.7**	***45.9***	**39.5**	***49.9***	**16.7**

Note: *f* indicates failed optimization.

2.4 LEAST SQUARES OPTIMIZATION

The line search procedures used in conjunction with a quasi-Newton method have proved very successful. They may also be used as part of a non-linear least squares optimization routine. In the least squares problem a function, $F(x)$, is minimized which is a sum of squares

LS

$$\underset{x\in\Re^n}{\text{minimize}}\ F(x) = \sum_{i=1}^{m} f_i(x)^2 = f(x)^T f(x) \tag{2.18}$$

Problems of this type occur in a large number of practical applications especially when fitting model functions to data i.e. nonlinear parameter estimation. They are also prevalent in control where it is desired that the output, $y(x,t)$, follow some continuous model trajectory, $\phi(t)$. This problem can be expressed as

$$\underset{x\in\Re^n}{\text{minimize}} \int_{t_0}^{t_1} (y(x,t) - \phi(t))^2 \, dt \tag{2.19}$$

When the integral is discretized using a suitable quadrature formula, Eq. 2.19 can be formulated as a least squares problem

$$\underset{x\in\Re^n}{\text{minimize}}\ F(x) = \sum_{i=1}^{m} (\bar{y}(x,t_i) - \bar{\phi}(t_i))^2 \tag{2.20}$$

where $\bar{y}$ and $\bar{\phi}$ represent the weights of the quadrature scheme.

In problems of this kind the residual, $\|f(x)\|$, is likely to be small at the optimum since it is general practice to set realistically achievable target trajectories. Although the function in LS (Prob. 2.18) can be minimized using a general unconstrained minimization technique as described in Section 2.2, certain characteristics of the problem can often be exploited to improve the iterative efficiency of the solution procedure. In particular, the gradient and Hessian matrix of LS (Prob. 2.18) have a special structure. Denoting the $m\times n$ Jacobian matrix of $f(x)$ as $J(x)$, the gradient, $\nabla F(x)$, and the Hessian, $H(x)$ of $F(x)$ are defined as

$$\begin{aligned} \nabla F(x) &= J(x)^T f(x) \\ H(x) &= J(x)^T J(x) + Q(x) \end{aligned} \tag{2.21}$$

$$\text{where } Q(x) = \sum_{i=1}^{m} f_i(x) H_i(x)$$

The matrix $Q(x)$ has the property that when the residual, $\|f(x)\|$, tends to zero then as x_k approaches the solution it also tends to zero. When $\|f(x)\|$ is small at the solution, a very effective method is to use the Gauss-Newton direction as a basis for an optimization procedure.

2.4.1 Gauss-Newton Method

In the Gauss-Newton method a search direction, d_k is obtained at each major iteration, k, which is a solution of the linear least squares problem:

Gauss-Newton

$$\underset{x\in\Re^n}{\text{minimize}} \quad \| J(x_k)d_k - f(x_k)\|_2^2 \tag{2.22}$$

the direction derived from this method is equivalent to the Newton direction when the terms of $Q(x)$ can be ignored. The search direction d_k can be used as part of a line search strategy to ensure that at each iteration the function F(x) decreases.

To consider the efficiencies that are possible with the Gauss-Newton method, Fig. 2.5 shows the path to the minimum on Rosenbrock's function (Eq. 2.2) when posed as a least squares problem. The Gauss-Newton method converged after only 48 function evaluations using finite difference gradients compared to 140 iterations using an unconstrained BFGS method.

The Gauss-Newton method often encounters problems when the second order term $Q(x)$ in Eq. 2.21 is significant. A method which overcomes this problem is the Levenberg-Marquardt method.

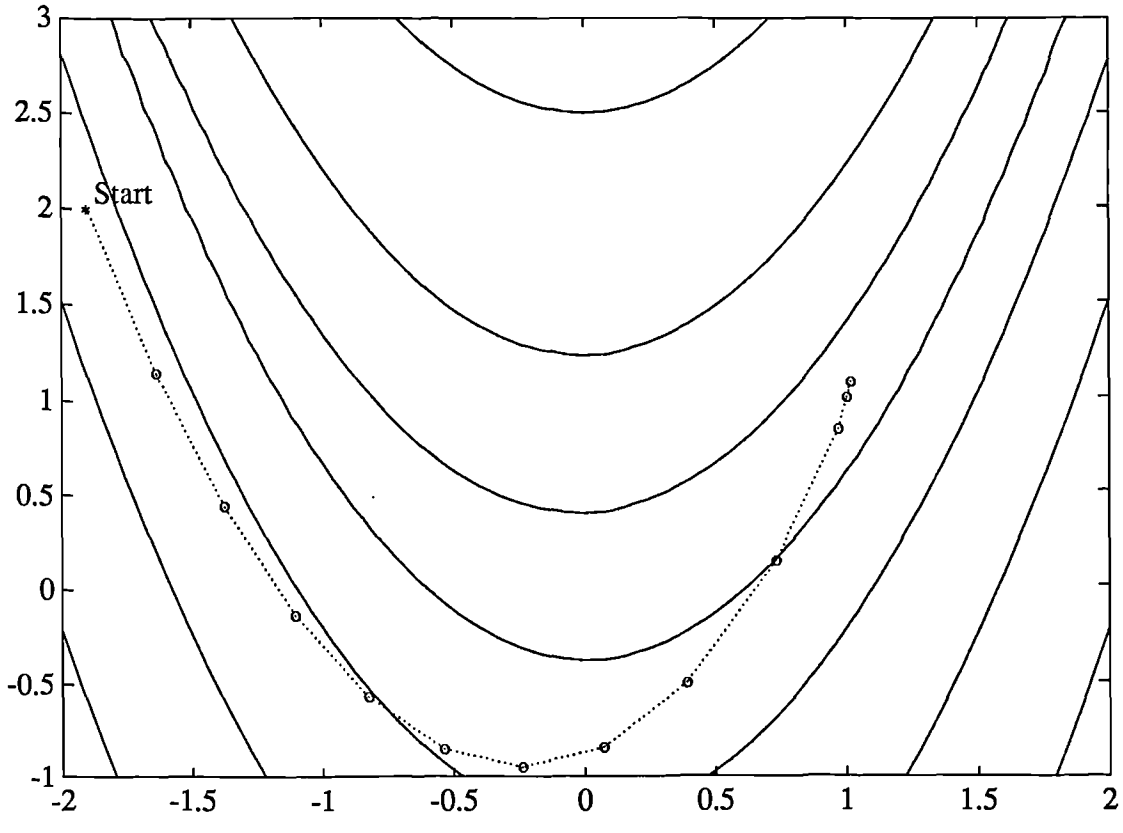

Fig. 2.5 Gauss-Newton Method on Rosenbrock's Function

2.4.2 Levenberg-Marquardt Method

The Levenberg-Marquardt [30,31] method uses a search direction which is a solution of the linear set of equations:

Levenberg-Marquardt

$$(J(x_k)^T J(x_k) + \lambda_k I)\, d_k = - J(x_k)\, f(x_k) \tag{2.23}$$

where the scalar, λ_k, controls both the magnitude and direction of d_k. When λ_k is zero, the direction d_k, is identical to that of the Gauss-Newton method. As λ_k tends to infinity, d_k tends towards a vector of zeros and a steepest descent direction. This implies that for some sufficiently large λ_k the term $F(x_k+d_k) < F(x_k)$. The term λ_k can therefore be controlled to ensure descent even when second order terms which restrict the efficiency of the Gauss-Newton method are encountered.

The Levenberg-Marquardt method therefore uses a search direction which is a cross between the Gauss-Newton direction and the steepest descent. This is illustrated in Fig. 2.6 below. The solution for Rosenbrock's function (Eq. 2.2) converged after 90 function evaluations compared to 48 for the Gauss-Newton method. The poorer efficiency is partly because the Gauss-Newton method is generally more effective when the residual is zero at the solution. However, such information is not always available beforehand, and occasional poorer efficiency of the Levenberg-Marquardt method is compensated for by its increased robustness.

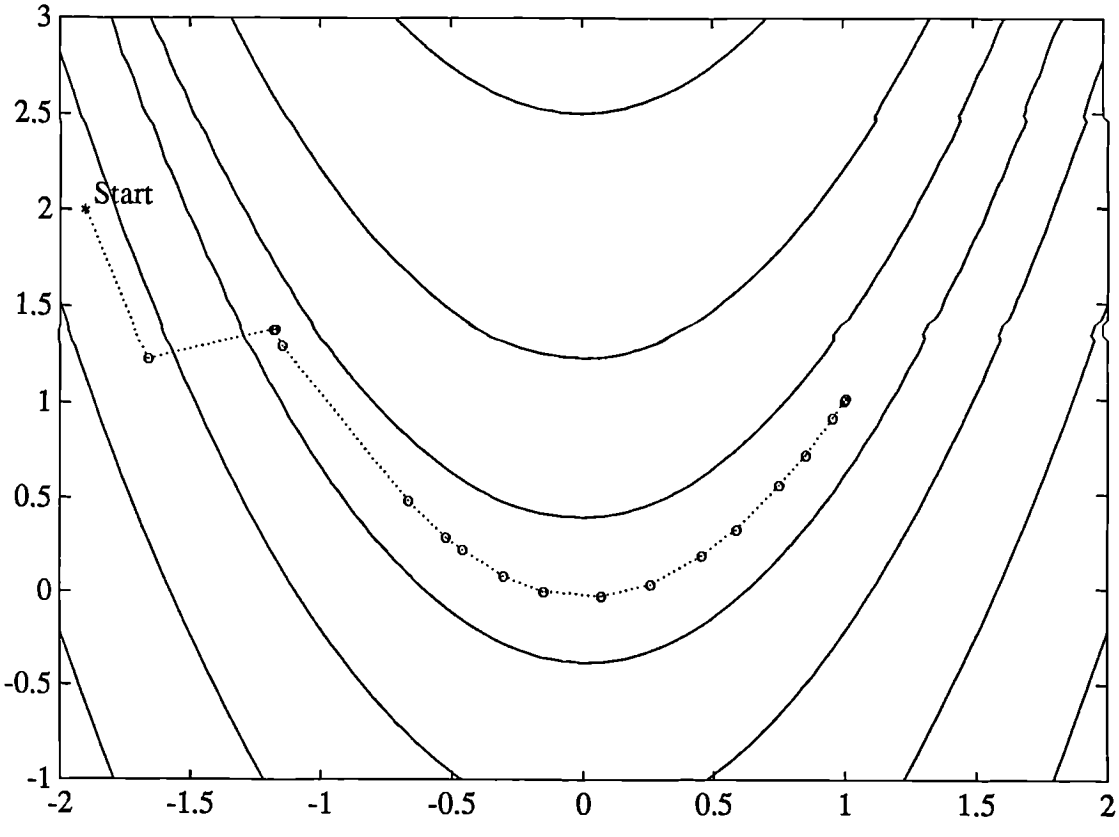

Fig. 2.6 Levenberg-Marquardt Method on Rosenbrock's Function

2.5 LEAST SQUARES IMPLEMENTATION

For a general survey of nonlinear least squares methods see Dennis [33]. Specific details on the Levenberg-Marquardt method can be found in Moré [32]. Both the Gauss-Newton method and the Levenberg-Marquardt method have been implemented in the MATLAB command language (see Appendix A). Details of the implementations will be discussed below.

2.5.1 Gauss-Newton Implementation

The Gauss-Newton method has been implemented using similar polynomial line search strategies discussed for unconstrained optimization. In solving the linear least squares problem (Prob. 2.18) exacerbation of the conditioning of the equations is avoided by using the QR decomposition of $J(x_k)$ and applying the decomposition to $f(x_k)$ (using the MATLAB \ operator). This is in contrast to inverting the explicit matrix, $J(x_k)^T J(x_k)$, which can cause unnecessary errors to occur.

Robustness measures are included in the method which consist of changing the algorithm to the Levenberg-Marquardt method when either the step length goes below a threshold value (in this implementation 1e-15) or when the condition number of $J(x_k)$ exceeds 1e10 (the condition number is a ratio of the largest singular value to the smallest).

2.5.2 Levenberg-Marquardt Implementation

The main difficulty in the implementation of the Levenberg-Marquardt method is an effective strategy for controlling the size of λ_k at each iteration so that it is efficient for a broad spectrum of problems. The method which is used in this implementation is to estimate the relative non-linearity of $F(x)$ using a linear predicted sum of squares $F_p(x_k)$ and a cubicly interpolated estimate of the minimum $F_k(x^*)$. In this way the size of λ_k is determined at each iteration.

The linear predicted sum of squares is calculated as

$$F_p(x_k) = J(x_{k-1})^T d_{k-1} + f(x) \tag{2.24}$$

and the term $F_k(x^*)$ is obtained by cubicly interpolating the points $F(x_k)$ and $F(x_{k-1})$. A step length parameter α^* is also obtained form this interpolation which is the estimated step to the minimum. If $F_p(x_k)$ is greater than $F_k(x^*)$ then λ_k is reduced, otherwise it is increased. The justification for this is that the difference between $F_p(x_k)$ and $F_k(x^*)$ is a measure of the effectiveness of the Gauss-Newton method and the linearity of the problem. This determines whether to use a direction approaching the

steepest descent direction or the Gauss-Newton direction. The formulas for the reduction and increase in λ_k, which have been developed through consideration of a large number of test problems, are shown in Fig. 2.7 below:

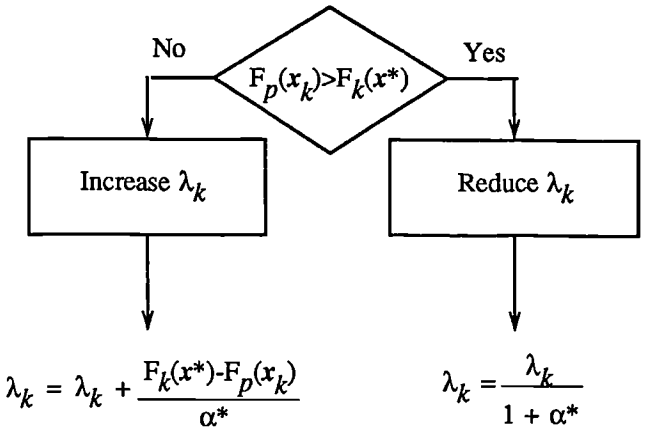

Fig. 2.7 Updating λ_k

Following the update of λ_k, a solution of Eq. 2.23 is used to obtain a search direction, d_k. A step length of unity is then taken in the direction d_k which is followed by a line search procedure similar to that discussed for the unconstrained implementation. The line search procedure ensures that $F(x_{k+1}) < F(x_k)$ at each major iteration and the method is therefore a descent method.

The implementation has been successfully tested on non-linear problems with up to 58 design variables and 100 sums of squares. It has been found that the method is considerably more robust than the Gauss-Newton method, and iteratively more efficient than an unconstrained method.

2.6 CONSTRAINED OPTIMIZATION

In constrained optimization, the general aim is to transform the problem into an easier sub-problem which can then be solved and used as the basis of an iterative process. A characteristic of a large class of early methods is the translation of the constrained problem to a basic unconstrained problem by using a penalty function for constraints which are near or beyond the constraint boundary. In this way the constrained problem is solved using a sequence of parameterized unconstrained optimizations which in the limit converge to the constrained problem. These methods are now considered relatively inefficient and have been replaced, by and large, by methods which have focussed on the solution of the Kuhn-Tucker (KT) equations. The KT equations are necessary conditions for optimality for a constrained optimization problem. If the problem is a so-called convex programming problem, that is $f(x)$ and $g_i(x)$, $i = 1, ..., m$, are convex functions, then the KT equations are both necessary and sufficient for a global solution point.

Referring to GP (Prob.2.1) the Kuhn-Tucker equations can be stated as

KT

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) = 0$$

$$\lambda_i^* \, g_i(x^*) = 0 \qquad i=1,...,m$$

$$\lambda_i^* \geq 0 \qquad i=m_e+1,...,m \qquad (2.25)$$

The first equation describes a cancelling of the gradients between the objective function and the active constraints at the solution point. In order for the gradients to be cancelled, Lagrangian Multipliers (λ_i, $i=1,..,m$) are necessary to balance the deviations in magnitude of the objective function and constraint gradients. Since only active constraints are included in this cancelling operation, constraints which are not active must not be included in this operation and so are given Lagrangian multipliers equal to zero. This is stated implicitly in the last two equations of Eq. 2.25.

The solution of the KT equations forms the basis to many Non-linear Programming algorithms. These algorithms attempt to compute directly the Lagrangian multipliers. Constrained quasi-Newton methods, in particular, guarantee superlinear convergence by accumulating second order information regarding the KT equations using a quasi-Newton updating procedure. These methods are commonly referred to as Sequential Quadratic Programming (SQP) methods since a QP sub-problem is solved at each major iteration (also known as Iterative Quadratic Programming, Recursive Quadratic Programming and Constrained Variable Metric methods).

2.6.1 Sequential Quadratic Programming (SQP)

SQP methods to a large extent represent state-of-the-art in non-linear programming methods. Schittowski [35], for instance, has implemented and tested a version which out-performs every other tested method in terms of efficiency, accuracy and percentage of successful solutions, over a large number of test problems.

Based on the work of Biggs [13], Han [14] and Powell [15,16], the method allows one to closely mimic Newton's method for constrained optimization just as is done for unconstrained optimization. At each major iteration an approximation is made of the Hessian of the Lagrangian function using a quasi-Newton updating method. This is then used to generate a QP sub-problem whose solution is used to form a search direction for a line search procedure. An overview of SQP can be found in Fletcher [2], Gill *et al* [1], Powell [17] and Schittowski [21], however, the general method will be stated here.

Given the problem description in GP (Prob. 2.1) the principal idea is the formulation of a QP sub-problem based on a quadratic approximation of the Lagrangian function:

$$L(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \tag{2.26}$$

Here Prob. 2.1 is simplified by assuming that bound constraints have been expressed as inequality constraints. The QP sub-problem is obtained by linearizing the non-linear constraints:

QP sub-problem

$$\underset{x\in\Re^n}{\text{minimize}} \quad \frac{1}{2} d^T H_k d + \nabla f(x_k)^T d$$

$$\nabla g_i(x)^T d + g_i(x) = 0 \qquad i=1, \ldots, m_e$$

$$\nabla g_i(x)^T d + g_i(x) \le 0 \qquad i=m_{e+1}, \ldots, m \tag{2.27}$$

This sub-problem may be solved using any QP algorithm (see, for instance, Section 2.7.2 below). The solution is used to form a new iterate:

$$x_{k+1} = x_k + \alpha_k d_k$$

The step length parameter, α_k, is determined by an appropriate line search procedure so that a sufficient decrease in a merit function is obtained (see below in Section 2.7.3). The matrix H_k is a positive definite approximation of the Hessian matrix of the Lagrangian function (Eq. 2.26). H_k can be updated by any of the quasi-Newton methods, although the BFGS method (see below in Section 2.7.1) appears to be the most popular.

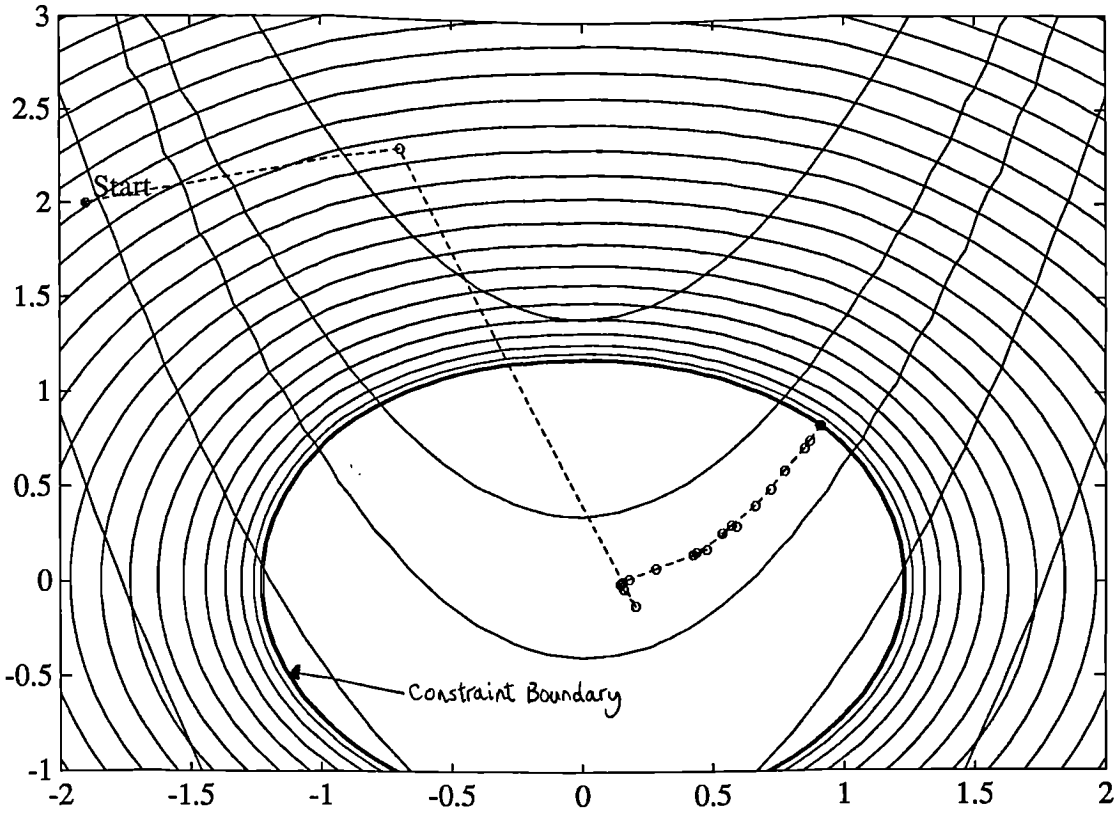

Fig. 2.8 SQP Method On Non-linearly Constrained Rosenbrock's Function

A non-linearly constrained problem can often be solved in fewer iterations using SQP than an unconstrained problem. One of the reasons for this is that, due to limits on the feasible area, the optimizer can make well informed decisions regarding directions of search and step length.

Consider Rosenbrock's function (Eq. 2.2) with an additional non-linear inequality constraint.

$$g(x) \leq x_1^2 + x_2^2 - 1.5 \tag{2.28}$$

This was solved by an SQP implementation in 96 iterations compared to 140 for the unconstrained case. Fig. 2.8 shows the path to the solution point, x={0.9072,0.8228}, starting at x={-1.9, 2}.

2.7 SQP IMPLEMENTATION

The MATLAB SQP implementation consists of three main stages which will be discussed briefly in the following sub-sections, they are:

(1) Updating of the Hessian matrix of the Lagrangian function.

(2) Quadratic Programming problem solution.

(3) Line search and merit function calculation.

2.7.1 Updating The Hessian Matrix

At each major iteration, H, a positive-definite quasi-Newton approximation of the Hessian of the Lagrangian function, is calculated using the BFGS method

Hessian Update (BFGS)

$$H_{k+1} = H_k + \frac{q_k q_k^T}{q_k^T s_k} - \frac{H_k^T H_k}{s_k^T H_k s_k}$$

where $s_k := x_{k+1} - x_k$

$$q_k := \nabla f(x_{k+1}) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x)_{k+1} - \left(\nabla f(x_k) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x)_k \right) \tag{2.29}$$

where λ_i $(i=1,...,m)$ is an estimate of the Lagrangian multipliers. Powell [15] recommends keeping the Hessian positive definite even though it may be positive indefinite at the solution point. A positive definite Hessian is maintained providing $q_k^T s_k$ is positive at each update and that H is initialized with a positive definite matrix. When $q_k^T s_k$ is not positive, q_k is modified on an element by element basis so that $q_k^T s_k > 0$. The general aim of this modification is to distort the elements of q_k, which contribute to a positive definite update, as little as possible. Therefore, in the initial phase of the modification, the most negative diagonal element of $q_k s_k^T$ is repeatedly halved. This procedure is continued until the minimum diagonal element of $q_k s_k^T \geq$ *-1e-5 or if* $q_k s_k^T$ *becomes positive. If after this procedure,* $q_k^T s_k$ is still not positive, q_k is modified using a vector, v, multiplied by a constant, w, so that::

$$q_k = q_k + w.v \tag{2.30}$$

where

$$v_i = \begin{cases} \nabla g_i(x)_{k+1} g_i(x)_{k+1} - \nabla g_i(x)_k g_i(x)_k , & \text{if } q_{ki}\, w_i < 0 \text{ and } q_{ki}\, s_{ki} < 0 \\ 0 & \text{otherwise} \end{cases} \qquad i=1, ..., m$$

and w is systematically increased until $q_k^T s_k$ becomes positive.

2.7.2 Quadratic Programming Solution

At each major iteration of the SQP method a QP problem is solved of the form

QP

$$\underset{x\in\Re^n}{\text{minimize}} \ \frac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x$$

$$A_i x = b \qquad i=1, \ldots, m_e$$

$$A_i x \le b \qquad i=m_e, \ldots, m \tag{2.31}$$

where A_i refers to the ith row of the $m\times n$ matrix A.

A number of methods have been implemented and tested including one using Wolfe's procedure [28] which uses a Simplex [27] (or other) Linear Programming algorithm. The method which was finally adopted due to its fast convergence was an active set strategy (also known as a projection method) similar to that of Gill *et al*, described in [1] and [29]. It has been modified for both LP and QP problems.

The solution procedure involves two phases: the first phase involves the calculation of a feasible point (if one exists), the second phase involves the generation of an iterative sequence of feasible points which converge to the solution. In this method an active set is maintained, $\mathring{A}_k$, which is an estimate of the active constraints (i.e. which are on the constraint boundaries) at the solution point. Virtually all QP algorithms are active set methods. This point is emphasized because there exist many different methods which are very similar in structure but which are described in widely different terms.

$\mathring{A}_k$ is updated at each iteration, k, and this is used to form a basis for a search direction, $\mathbf{d}_k$. It should be noted that equality constraints always remain in the active set, $\mathring{A}_k$. The notation for the non-italicized variables, $\mathbf{d}_k$ and k, is used here in order to distinguish them from d_k and k in the major iterations of the SQP method. The search direction, $\mathbf{d}_k$, is calculated which minimizes the objective function while remaining on any active constraint boundaries. The feasible subspace for $\mathbf{d}_k$ is formed from a basis, Z_k, whose columns are orthogonal to the estimate of the active set, $\mathring{A}_k$ (i.e. $\mathring{A}_k Z_k=0$). Thus a search direction, which is formed from a linear summation of any combination of the columns of Z_k, is guaranteed to remain on the boundaries of the active constraints.

The matrix Z_k is formed from the last m-l columns of the QR decomposition of the matrix $\mathring{A}_k$, where l is the number of active constraints and $l < m$. i.e. Z_k is given by:

$$Q\,\mathring{A}_k^{\mathrm{T}} = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

$$\text{and } Z_{kij} = Q_{ij} \qquad i=1{:}n \ \ j=m\text{-}l{:}m. \tag{2.32}$$

Having found Z_k a new iterate x_{k+1} is sought of the form:

$$x_{k+1} = x_k + \alpha \mathbf{d}_k \tag{2.33}$$

where $\mathbf{d}_k$ is the search direction formed from a linear combination of the columns of Z_k, i.e. $\mathbf{d}_k = p^{\mathrm{T}} Z_k$, where p is a vector of constants.

The value of the objective function at the next iterate, k+1, can thus be given as:

$$f(p)_{k+1} = \frac{1}{2}\,(x_k + pZ_k)^T H\,(x_k + pZ_k) + c^T(x_k + pZ_k). \tag{2.34}$$

Differentiating this with respect to p yields:

$$\nabla f(p)_{k+1} = Z_k{}^T H\, Z_k p + Z_k\,(Hx_k + c\,). \tag{2.35}$$

$\nabla f(p)_{k+1}$ is referred to as the projected gradient of the objective function since it is the gradient projected in the subspace defined by Z_k. The term, $Z_k{}^T H\, Z_k$, is called the projected Hessian. Assuming the Hessian matrix, H, is positive definite (which is the case in this implementation of SQP) then the minimum of the function, $f(p)_{k+1}$, in the subspace defined by Z_k, occurs when $\nabla f(p)_{k+1}=0$, which is the solution of the system of linear equations

$$Z_k{}^T H\, Z_k p = -\,Z_k\,(Hx_k + c\,). \tag{2.36}$$

A step is then taken of the form:

$$x_{k+1} = x_k + \alpha d_k \qquad \text{where } d_k = p^T Z_k. \tag{2.37}$$

At each iteration, because of the quadratic nature of the objective function, there are only two choices of step length. A step of unity along $\mathbf{d}_k$ is the exact step to the minimum of the function restricted to the null space of $\mathring{A}_k$. If such a step can be taken, without violation of the constraints, then this is the solution to QP (Prob. 2.31). Otherwise, the step along $\mathbf{d}_k$ to the nearest constraint is less than unity and a new constraint will be included in the active set at the next iterate. The distance to the constraint boundaries in any direction, $\mathbf{d}_k$, is given by:

$$\alpha_i = \frac{-(A_i x_k - b_i)}{A_i\, \mathbf{d}_k} \qquad i=1, \ldots, m \tag{2.38}$$

which is defined for constraints not in the active set and where the direction, $\mathbf{d}_k$, is towards the constraint boundary i.e. $A_i \mathbf{d}_k > 0$, $i=1,\ldots,m$.

When n independent constraints have been included in the active set, without location of the minimum, Lagrange multipliers, λ_k, are calculated which satisfy the non-singular set of linear equations:

$$\mathring{A}_k \lambda_k = Hx_k + c,. \tag{2.39}$$

If all elements of λ_k are positive, x_k is the optimal solution of QP (Prob. 2.31). However, if any component of λ_k is negative, and it does not correpond to an equality constriant, then the corresponding element is deleted from the active set and a new iterate is sought.

Initialization

The algorithm requires a feasible point to start. If the current point from the SQP method is infeasible then a feasible point can be found by solving the linear programming problem

LP Feasibility Phase

$$\underset{\gamma\in\Re,\ x\in\Re^n}{\text{minimize}}\ \gamma$$

$$A_i x = b \qquad i=1, \ldots, m_e$$

$$A_i x - \gamma \le b \qquad i=m_{e+1}, \ldots, m \tag{2.40}$$

where, again, the notation A_i indicates the ith row of the matrix A. A feasible point (if one exists) to Prob. 2.40 can be found by setting x with a value which satisfies the equality constraints. This can be achieved by solving an under or over-determined set of linear equations formed from the set of equality constraints. If there is a solution to this problem then the slack variable, γ, is set to the maximum inequality constraint at this point.

The above QP algorithm is modified for LP problems by setting the search direction to the steepest descent direction,

$$\mathbf{d}_k = Z_k^T Z_k g_k, \tag{2.41}$$

at each iteration, where g_k is the gradient of the objective function (equal to the coefficients of the linear objective function).

If a feasible point is found using the above LP method, the main QP phase is entered. The search direction, $\mathbf{d}_k$, is initialized with a search direction, $\mathbf{d}_1$, found from solving the set of linear equations,

$$H\mathbf{d}_1 = -g, \tag{2.42}$$

where g_k is the gradient of the objective function at the current iterate x_k (i.e. $Hx_k + c$). A step of unity is the unconstrained minimum of $f(x)$.

If a feasible solution is not found for the QP problem, the direction of search for the main SQP routine, d_k, is taken as one which minimizes γ.

2.7.3 Line Search and Merit Function

The solution to the QP sub-problem produces a vector d_k which is used to form a new iterate,

$$x_{k+1} = x_k + \alpha_k d_k. \tag{2.43}$$

The step length parameter, α_k, is determined in order to produce a sufficient decrease in a merit function. The merit function used by Han [15] and Powell [15] of the form:

Merit Function

$$\psi(x) = f(x) + \sum_{i=1}^{m_e} r_i\, g_i(x) + \sum_{i=m_e}^{m} r_i\, \max\{0, g_i(x)\} \tag{2.44}$$

has been used in this implementation. Powell recommends setting the penalty parameter:

$$r_{ki} = \max\{\lambda_i, \tfrac{1}{2}(r_{(k-1)i} + \lambda_i)\}, \qquad i = 1, ..., m \tag{2.45}$$

This allows a positive contribution form constraints which are inactive in the QP solution but were recently active. In this implementation, initially the penalty parameter r_i is set to:

$$r_i = \frac{\| \nabla f(x) \|}{\| \nabla g_i(x) \|} \qquad i = 1, ..., m \tag{2.46}$$

where $\| . \|$ represents the Euclidean norm.

This ensures larger contributions to the penalty parameter from constraints with smaller gradients which would be the case for active constraints at the solution point.

2.7.4 Constrained Example

In order to demonstrate the software and the ease and efficiency with which optimization problems can be coded and solved, a constrained optimization example will be considered. For a full description on how to use the software, refer to Appendix A. The test problem under consideration is taken from the NAG user manual [45] and is of the form:

Example Problem

$$\underset{x}{\text{minimize}} \quad f(x) = (x_1-1)^2 + (x_1-x_2)^2 + (x_2-x_3)^3 + (x_3-x_4)^4 + (x_4-x_5)^4$$

subject to the constraints

$$g_1(x): \quad x_1 + x_2^2 + x_3^3 - 2 - 3 = 0$$

$$g_2(x): \quad x_2 - x_3^2 + x_4 + 2 - 2 \geq 0$$

$$g_3(x): \quad 0 \leq x_1 x_5 \leq 2$$

and subject to the bounds

$$0 \leq x_1 \leq 1.5$$

$$0 \leq x_3 \leq 1.5$$

$$0 \leq x_4 \leq 1.5$$

$$0 \leq x_5 \leq 1.5$$

starting from the initial guess x = {0.5, 1, 0.6, 1.4, 1 }

(2.47)

The MATLAB program required to solve this problem is as follows:

MATLAB Implemented Solution Procedure

```
PARA=0;                                   %Reset Optimization Parameters
PARA(13)=1                                %Set number of equality constraints to 1
X=[0.5,1,0.6,1.4,1];                      %Initialize Design Variables
VLB=zeros(X);                             %Set upper and lower bounds
VUB=1.5*ones(X);

while PARA(1)~=1                          %Check Termination Parameter
                                          %Evaluate F
  F=(X(1)-1)^2+(X(1)-X(2))^2+(X(2)-X(3))^3+(X(3)-X(4))^4+(X(4)-X(5))^4;
  G(1,1)=X(1)+X(2)^2+X(3)^3-2-3*sqrt(2);  %Evaluate Constraints
  G(1,2)=-(X(2)-X(3)^2+X(4)+2-2*sqrt(2));
  G(1,3)=-X(1)*X(5);
  G(1,4)=-2-G(3);
  [X,PARA]=constr(X,F,G,PARA,VLB,VUB);    %Recursively Call Optimizer
end
```

The optimization routine (constr) is called on an iterative basis following calculation of the objective function and gradients. In the above example, default optimization parameters have been used which can be overridden by entering the appropriate values in the vector PARA. In this example, gradients are calculated using a finite difference method.

Results

After 68 function evaluations the optimization terminated with the following results:

```
X =
  1.2264e+00  1.4150e+00  1.4445e+00  1.5000e+00  1.4993e+00
```

Objective Function Value:

```
F =
  8.6808e-02
```

Constraint Values:

```
G =
  -8.8818e-15 -2.2204e-15 -1.8388e+00 -1.6117e-01
```

(i.e. $g_1(x)$ = -8.8818e-15, $g_2(x)$=-2.2204e-15, $g_3(x)$=1.8388e+00)

Comparison of Results

The NAG results given in the reference manual are as follows:

After 160 Function Evaluations the Estimate of the Solution is:

x={1.2264, 1.4150, 1.4445, 1.5000, 1.5000}

Objective Function Value:

f(x)=8.6812e-02

Constraint Values:

$g_1(x)$ =9.8595e-5

$g_2(x)$=-2.3336e-5

$g_3(x)$=1.8396

The NAG routine uses a sequential augmented Lagrangian method, the minimization sub-problem is solved using a Quasi-Newton method.

The method used in the MATLAB implementation is Sequential Quadratic Programming. The equality constraint has been met to a greater degree and the objective function value is less than that of the NAG routine. The number of function evaluations is significantly less than that of the NAG routine which demonstrates the iterative efficiency of the SQP method.

The MATLAB implementation does not have the speed advantage of a compiled language. However, the flexibility of an interpretive language more than compensates for the additional computing speed. For example, the FORTRAN implementation for this problem in the NAG user manual has nearly 200 lines of source code compared to the 14 lines of MATLAB code.

2.8 MULTI-OBJECTIVE OPTIMIZATION

The rigidity of the mathematical problem posed by the general optimization formulation given in GP (Prob. 2.1) is often remote from that of a practical design problem. It is rarely the case that a single objective with several hard constraints adequately represents the problem being faced. More often there will be a vector of objectives, $f(x)=\{f_1(x), f_2(x), \ldots f_m(x)\}$, which must be traded off against each other in some way. The relative importance of these objectives will not generally be known until the system's best capabilities can be determined and trade-offs between the objectives can be fully understood. As the number of objectives increases, trade-offs between these objectives are likely to become complex and less easily quantified. There is, therefore, much reliance on the intuition of the designer and his ability to express preferences throughout the optimization cycle. Thus, requirements for a multi-objective design strategy are that it enables a natural problem formulation to be expressed, yet, is easily solvable using numerical algorithms. In this way, the designer can alter his preferences throughout the optimization cycle and enter them into a numerically tractable and realistic design problem.

This section will begin with an introduction to multi-objective Optimization, looking at a number of alternative methods. Attention will be focussed on the Goal Attainment method which can be posed as a non-linear programing problem. Algorithm improvements to the SQP method will be presented for use with the Goal Attainment method.

2.8.1 Introduction To Multi-Objective Optimization

It is appropriate initially to introduce some concepts concerned with multi-objective optimization. Multi-objective optimization is concerned with the minimization of a vector of objectives $f(x)$ which may be the subject to a number of constraints or bounds:

$$\begin{array}{lll} \textbf{MO} & \underset{x\in\Re^n}{\text{minimize}}\ f(x) & \\ \text{subject to:} & g_i(x) = 0, & i=1, .., .m_e \\ & g_i(x) \le 0 & i=m_e+1, \ldots, m \\ & x_l \le x \le x_u & \end{array} \tag{2.48}$$

It is important to note that since $f(x)$ is a vector, then, if any of the components of $f(x)$ are competing, there is no-unique solution to this problem. Instead, the concept of non-inferiority [47]] (also called Pareto optimality [46], [48]) must be used to characterize the objectives. A non-inferior solution is one in which an improvement in one objective requires a degradation of another. To define this concept more precisely, consider a feasible region, Ω, in the parameter space $x\in\Re^n$ which satisfies all the constraints, i.e.

$$\begin{array}{lll} & \Omega : x & \\ \text{subject to:} & g_i(x) = 0, & i=1, .., .m_e \\ & g_i(x) \le 0 & i=m_e+1, \ldots, m \\ & x_l \le x \le x_u & \end{array} \tag{2.49}$$

This allows us to define the corresponding feasible region for the objective function space, Λ,

$$\Lambda = f(x) \text{ subject to } x \in \Omega. \tag{2.50}$$

The performance vector, $f(x)$, therefore maps parameter space into objective function space as is represented for a two-dimensional case in Fig. 2.9 below.

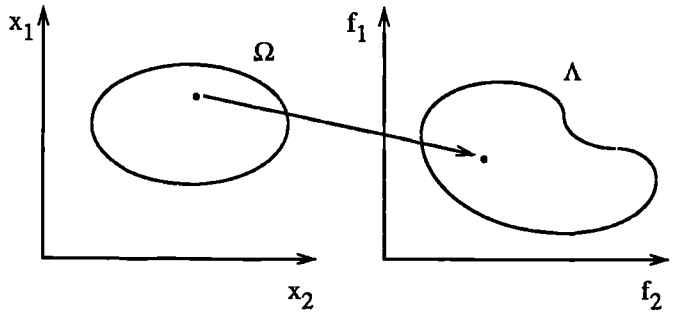

Fig. 2.9 Mapping from parameter space into objective function space.

A non-inferior solution point may now be defined.

Definition: A point a $x^* \in \Omega$ is a non-inferior solution if and only if for some neighborhood of x^* there does not exist a Δx such that $(x^*+\Delta x) \in \Omega$ and

$$f_i(x^*+\Delta x) \leq f_i(x^*), \qquad i=1, .., .m$$
$$f_j(x^*+\Delta x) < f_j(x^*) \qquad \text{for some } j. \tag{2.51}$$

In the two dimensional representation of Fig. 2.10 the set of non-inferior solutions lie on the curve between C and D. Points A and B represent specific non-inferior points.

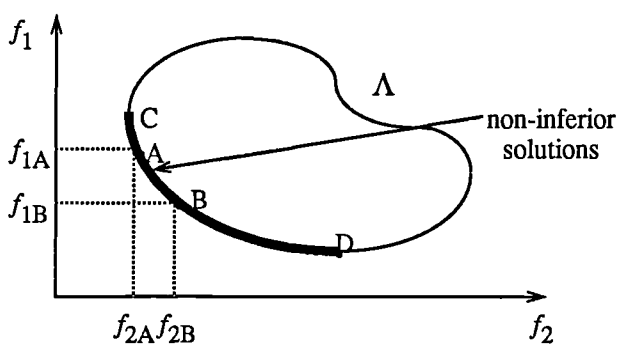

Fig. 2.10 Set of Non-inferior Solutions.

A and B are clearly non-inferior solution points since an improvement in one objective, f_1, requires a degradation in the other objective, f_2; i.e. $f_{1B}<f_{1A}$, $f_{2B}>f_{2A}$.

Since any point in Ω which is not a non-inferior point represents a point in which improvement can be attained in all the objectives, it is clear that such a point is of no value. Multi-objective optimization is therefore, concerned with the generation and selection of non-inferior solution points. The techniques for multi-objective optimization are wide and varied and all the methods cannot be covered within the scope of this thesis. However, some of the techniques will be described below.

Weighted Sum Strategy

The weighted sum strategy converts the multi-objective problem of minimizing the vector $f(x)$ into a scalar one by constructing a weighted sum of all the objectives

Weighted Sum

$$\underset{x\in\Omega}{\text{minimize}}\ F(x) = \sum_{i=1}^{m} w_i f_i(x)^2 \tag{2.52}$$

The problem can then be optimized using a standard unconstrained optimization algorithm. The problem here is in attaching weighting coefficients to each of the objectives. The weighting coefficients do not necessarily correspond directly to the relative importance of the objectives or allow trade-offs between the objectives to be expressed. Further, the non-inferior solution boundary may be non-convex so that certain solutions would not be accessible.

This can be illustrated geometrically. Consider the two objective case in Fig. 2.11. In the objective function space a line, L, $w^T f(x) = c$ is drawn. The minimization of Prob. 2.53 can be interpreted as finding the value of c for which L just touches the boundary of Λ as it proceeds outwards from the origin. Selection of weights w_1 and w_2, therefore, define the slope of L which in turn leads to the solution point where L touches the boundary of Λ.

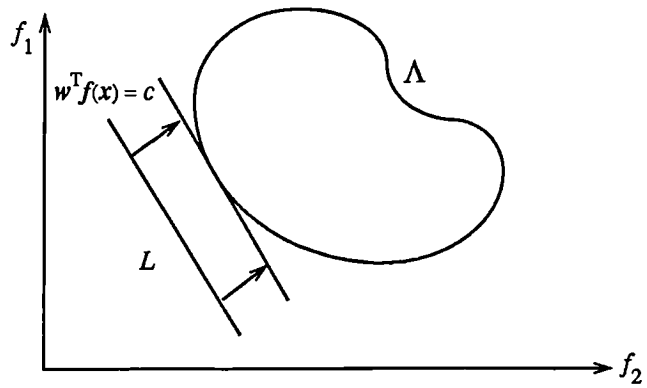

Fig. 2.11 Geometrical Representation of the Weighted Sum Method.

The aforementioned convexity problem arises when the lower boundary of Λ is non-convex as shown in Fig. 2.12. In this case the set of non-inferior solutions between A and B is not available.

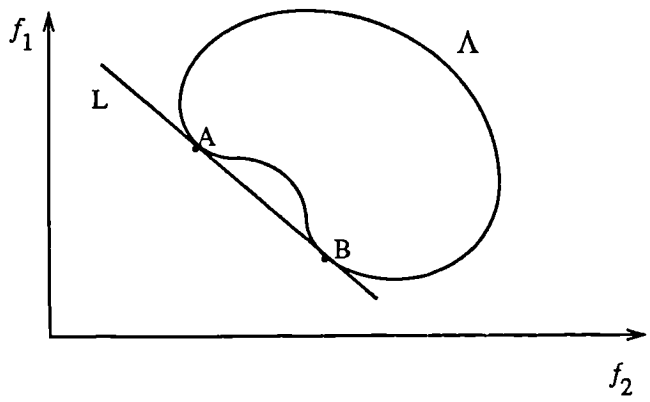

Fig. 2.12 Non-convex Solution Boundary.

ε-constraint method

A procedure which overcomes some of the convexity problems of the weighted sum technique is the ε-constraint method. This involves minimizing a primary objective, f_p and expressing the other objectives in the form of inequality constraints

ε-constraint

$$\underset{x\in\Omega}{\text{minimize}}\ f_p(x)$$

$$\text{subject to: } f_i(x) \le \varepsilon_i \qquad i = .1, \ldots, m,\ i \neq p \tag{2.53}$$

Fig. 2.13 shows a 2-dimensional Representation of the ε-constraint method for a two objective problem

$$\underset{x\in\Omega}{\text{minimize}}\ f_1(x) \qquad \text{subject to: } f_2(x) \le \varepsilon_2$$

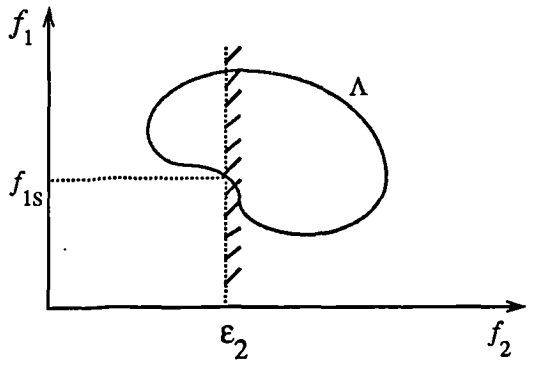

Fig. 2.13 Geometrical Representation of ε-Constraint Method

This approach is able to identify a number of non-inferior solutions on a non-convex boundary that would not be obtainable using the weighted sum technique, for instance, at the solution point $f_1=f_{1s}$ and $f_2=\varepsilon_2$. A problem with this method is, however, a suitable selection of ε to ensure a feasible solution.

A further disadvantage of this approach is that the use of hard constraints is rarely adequate for expressing true design objectives. Similar methods exist, such as that of Waltz [55] which prioritize the objectives. The optimization proceeds with reference to these priorities and allowable bounds of acceptance. Here the difficulty is in expressing such information at early stages of the optimization cycle.

In order for the designers true preferences to be put into a mathematical description would require that the designer express a full table of his preferences and satisfaction levels for a range of objective value combinations. A procedure must then be realized which is able to find a solution with reference to this. Such methods have been derived for discrete functions using the branches of statistics known as decision theory and game theory (for a basic introduction, see [52]). Implementation for continuous functions requires suitable discretization strategies and complex solution methods. Since it is rare for the designer to know such detailed information anyway, this method is deemed impractical for most practical design problems, however, it is seen as a possible area for further research.

What is required is a formulation which is simple to express, which retains the designers preferences and which is numerically tractable.

2.8.2 Goal Attainment Method

The method which has been favoured and adopted here is the Goal Attainment method of Gembicki [50]. This involves expressing a set of design goals, $\mathbf{f}^* = \{f_1^*, f_2^*, \ldots f_m^*\}$, which are associated with a set of objectives, $\mathbf{f}(x) = \{f_1(x), f_2(x), \ldots f_m(x)\}$. The problem formulation allows the objectives to be under- or over-achieved which enables the designer to be relatively imprecise about initial design goals. The relative degree of under- or over-achievement of the goals is controlled by a vector of weighting coefficients, $\mathbf{w} = \{w_1, w_2, \ldots w_m\}$ and is expressed as a standard optimization problem using the following formulation :

Goal Attainment

$$\underset{\gamma \in \Re,\, x \in \Omega}{\text{minimize}} \quad \gamma$$

$$f_i(x) - w_i \gamma \leq f_i^* \qquad i = 1, .., m \tag{2.54}$$

The term $w_i \gamma$ introduces an element of *slackness* into the problem which would otherwise impose that the goals should be rigidly met. The weighting vector, $\mathbf{w}$, enables the designer to express a measure of the relative trade-offs between the objectives. For instance, setting the weighting vector, $\mathbf{w}$, equal to the initial goals indicates that the same percentage under or over-attainment of the goals, $\mathbf{f}^*$, will be achieved. Hard constraints may be incorporated into the design by setting a particular weighting factor to zero (i.e. $w_i = 0$). The Goal Attainment method therefore provides a convenient intuitive interpretation of the design problem that is solvable using standard optimization procedures. Illustrative examples of the use of Goal Attainment method in Control System Design can be found in Fleming [53,54].

The Goal Attainment method is represented geometrically in Fig. 2.14 for the 2-dimensional problem

$$\underset{\gamma, x \in \Omega}{\text{minimize}} \ \gamma \qquad \text{subject to:} \quad f_1(x) - w_1 \gamma \leq f_1^*$$
$$f_2(x) - w_2 \gamma \leq f_2^*$$

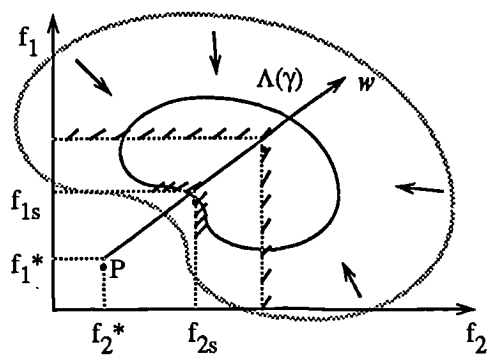

Fig. 2.14 Geometrical Representation of Goal Attainment Method

Specification of the goals, $\{f_1^*, f_2^*\}$, defines the goal point, P. The weighting vector defines the direction of search from P to the feasible function space, $\Lambda(\gamma)$. During the optimization γ is varied which changes the size of the feasible region. The constraint boundaries will converge to the unique solution point $\{f_{1s}, f_{2s}\}$.

2.8.3 Algorithm Improvements For Goal Attainment Method

The Goal Attainment method has the advantage that it may be posed as a non-linear programming problem. There are also characteristics of the problem which can be exploited in a non-linear programming algorithm. In Sequential Quadratic Programming the choice of merit function for the line search is not easy because, in many cases, it is difficult to weigh up the relative importance between improving the objective function, and reducing constraint violations. This has resulted in a number of different schemes for constructing the merit function (see, for example, Schittowski [35]). In Goal Attainment programming there may be a more appropriate merit function which can be achieved by posing Prob. 2.55 as the minimax problem:

$$\underset{x\in\Re^n}{\text{minimize}} \ \max_i \{ \Lambda_i \}$$

$$\text{where} \quad \Lambda_i = \frac{f_i(x) - f^*_i}{w_i} \qquad i=1, .., m. \tag{2.55}$$

Following the argument of Brayton *et al* [56] for minimax optimization using SQP, using the merit function of Eq. 2.45 for the Goal Attainment problem of Eq. 2.55, gives:

$$\psi(x,\gamma) = \gamma + \sum_{i=1}^{m} r_i \max\{ 0, f_i(x) - w_i\gamma - f^*_i \}. \tag{2.56}$$

When the merit function of Eq. 2.57 is used as the basis of a line search procedure, then, although $\psi(x,\gamma)$ may decrease for a step in a given search direction, the function $\max(\Lambda_i)$ may paradoxically increase. This would be accepting a degradation in the worst case objective. Since the worst case objective is responsible for the value of the objective function γ we would be accepting a step which would ultimately increase the objective function to be minimized. Conversely, $\psi(x,\lambda)$ may increase when $\max(\Lambda_i)$ decreases implying a rejection of a step which improves the worst case objective.

Following the lines of Brayton *et al* [56] a solution is therefore to set $\psi(x)$ equal to the worst case objective, i.e.

$$\psi(x) = \max_i \{ \Lambda_i \}). \tag{2.57}$$

A problem in the Goal Attainment method is that it is common to use a weighting coefficient equal to zero in order to incorporate hard constraints. The merit function of Eq. 2.58 then becomes infinite for arbitrary violations of the constraints. To overcome this problem while still retaining the features of Eq. 2.58 the merit function is combined with that of Eq. 2.45 giving:

$$\psi(x) = \sum_{i=1}^{m} \begin{cases} r_i \max\{ 0, f_i(x) - f^*_i \}, & \text{if } w_i = 0 \\ \max_i \{ \Lambda_i \}, \ i=1, .., m & \text{otherwise.} \end{cases} \tag{2.58}$$

Another feature which can be exploited in SQP is the objective function γ. We know from the KT equations (Eq. 2.25) that:

$$\nabla\gamma + \sum_{i=1}^{m} \lambda_i^* \nabla(\Lambda - \gamma) = 0 \tag{2.59}$$

The gradient vector $\nabla\gamma$ is a vector of zeros (except for the element corresponding to γ which is equal to unity). Also, the elements of $\nabla(f_i(x) - w_i\,\gamma - f_i^*)$ corresponding to the gradient of γ are all less than or equal to zero. Therefore, assuming $w_i>0$, we can conclude that:

$$\sum_{i=1}^{m} \lambda_i = 1. \tag{2.60}$$

The Lagrange function can therefore be written as

$$\begin{aligned} L(x,\lambda) &= f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \\ &= \gamma + \sum_{i=1}^{m} \lambda_i\,(\Lambda - \gamma) \\ &= \sum_{i=1}^{m} \lambda_i\,(\Lambda). \end{aligned} \tag{2.61}$$

It follows that the approximation to the Hessian of the Lagrangian, $\boldsymbol{H}$, should have zeros in the rows and columns associated with the variable γ. By initializing $\boldsymbol{H}$ as the identity matrix this property would not appear. $\boldsymbol{H}$ is therefore initialized and maintained to have zeros in the rows and columns associated with γ.

These changes make the Hessian, $\boldsymbol{H}$, indefinite, therefore $\boldsymbol{H}$ is set to have zeros in the rows and columns associated with γ, except for the diagonal element which is set to a small positive number (e.g. 1e-10). This allows the fast converging positive definite QP method described in Section 2.7.2 to be used.

The above modifications have been implemented as part of the Optimization Toolbox, described in Appendix A. It has been found that the above modifications make the method more robust. However, due to the rapid convergence of the SQP method, the requirement that the merit function (Eq. 2.59) strictly decreases sometimes requires more function evaluations than an implementation of SQP using the merit function of (Eq. 2.45). The choice of which merit function to use is therefore left as an option for the user.

2.9 REVIEW

A number of different optimization strategies have been discussed. The algorithms used (e.g. BFGS, Levenberg-Marquardt and SQP) have been chosen for their robustness and iterative efficiency. The choice of problem formulation (e.g. unconstrained, least squares, constrained, minimax or multi-objective) depends on the problem being considered and the required execution efficiency. The overall aim of this part of the research has been to develop a set of tools which are readily accessible to control engineers (and other workers) and which allow optimization problems to be coded in a way which is natural to the problem at hand. The MATLAB environment has been used to implement the programs due to its Control System Design and other utilities. This has enabled the development of an Optimization Toolbox in which problems can be coded easily and efficiently. In Part 3 it is seen how these utilities may be used, within the context of Control System Design, to design robust and effective controllers.

2.10 REFERENCES

Overviews of Optimization, General References

[1] Gill, P.E., Murray, W. and Wright, M.H., "Practical Optimization", Academic Press, London, 1981.

[2] Fletcher, R., "Practical Methods of Optimization", Vol. 1, Unconstrained Optimization, and Vol. 2, Constrained Optimization, John Wiley and Sons. 1980.

[3] Brayton, R.K., Hachtel, G.D. and Sangiovanni-Vincentelli, A.L., "A survey of optimization techniques for integrated-circuit design," Proc. of IEEE, Vol.69, No.10, pp.1334-1363, 1981.

[4] Mayne, D., Polak, E. and Sangiovanni-Vincentelli, "Computer-Aided Design via optimization: A review," Automatica, Vol.18, pp.881-907, 1980.

[5] Nye, W.T. and Tits, A.L. "An application-oriented, optimization-based methodology for interactive design of engineering systems," Int. J. Control, Vol.43, No.6, pp.1693-1721, 1986.

Quasi-Newton Updating Methods

[6] Broyden, C.G, "The convergence of a class of double-rank minimization algorithms," J. of the Inst. of Mathematics and its Applic., Vol. 6, pp. 76-90, 1970.

[7] Fletcher, R., "A new approach to variable metric algorithms," Computer Journal, Vol. 13, pp. 317-322, 1970.

[8] Golfarb, D., "A family of variable metric updates derived by variational means," Mathematics of Computing, Vol. 24, pp. 23-26, 1970.

[9] Shanno, D.F., "Conditioning of quasi-Newton methods for function minimization," Mathematics of Computation, Vol. 24, pp. 647-656, 1970.

[10] Davidon, W.C., "Variable metric method for minimization", A.E.C. Research and Development Report, ANL - 5990, 1959.

[11] Fletcher, R., Powell M.J.D., "A rapidly convergent descent method for minimization," Computer Journal, Vol. 6, pp. 163-168, 1963.

[12] Powell, M.J.D "On the convergence of the variable metric algorithm," J.Inst. Maths. Applics., Vol.7, pp.21-36, 1971.

Sequential Quadratic Programming

[13] Biggs, M.C., "Constrained minimization using recursive quadratic programming," in Towards Global Optimization (L.C.W.Dixon and G.P.Szergo, eds.), North-Holland, pp.341-349, 1975.

[14] Han, S.P., "A globally convergent method for nonlinear programming," J. of Optimization Theory and Applications, Vol. 22, pp. 297 , 1977.

[15] Powell, M.J.D. "A fast algorithm for nonlinearly constrained optimization calculations," Numerical Analysis, ed. G.A.Watson, Lecture Notes in Mathematics, Springer Verlag, Vol. 630, 1978.

[16] Powell, M.J.D. "The convergence of variable metric methods for nonlinearly constrained optimization calculations," Nonlinear Programming 3, (O.L. Mangasarian, R.R. Meyer and S.M. Robinson, eds.), Academic Press, 1978.

[17] Powell, M.J.D., "Variable metric methods for constrained optimization," in Mathematical Programming: The State of the Art, (A.Bachem, M.Grotschel and B.Korte, eds.) Springer Verlag, pp.288-311, 1983.

Test Problems, Comparative Studies

[18] Hock, W. and Schittowski, "Test Examples for Nonlinear Programming Codes," Lecture Notes in Economics and Mathematical Systems, Vol. 187, Springer Verlag, 1981.

[19] Hock, W. and Schittowski, "Nonlinear Programming Codes," Lecture Notes in Economics and Mathematical Systems, Vol. 183, Springer Verlag, 1980.

[20] Hock, W. and Schittowski, "More Test Examples for Nonlinear Programming Codes," Lecture Notes in Economics and Mathematical Systems, Vol. 282, Springer Verlag 1987.

[21] Hock, W. and Schittowski, K., "A comparative performance evaluation of 27 nonlinear programming codes," Computing Vol. 30., pp.335, 1983.

[22] Crowder, H., and Saunders, P.B., "Reporting computational experiments with mathematical software," Math. Programming. Vol.5, pp.316-319, 1978. (Also in A.C.M. Trans. Math. Software Vol.5, pp.193-203, 1979.

[23] Lenard, M.L., and Minkoff, M. "Randomly generated test problems for positive definite quadratic programming," A.C.M. Trans. Math. Software, Vol.10, pp.86-96, 1984.

[24] Lootsma, F.A., "Comparative performance evaluation: Experimental design and generation of test problems in non-linear optimization," Computational Mathematical Programming (NATO ASI Series),pp, 249-260, K.Schittowski (ed.) Springer, 1985.

[25] Minkoff, M. "Methods of evaluating nonlinear programming," in Nonlinear Programming 4, ed. O.L. Mangasarian, Academic Press, pp.519-548, 1981.

[26] Mulvey, J.M. (ed.), "Evaluating Mathematical Programming Techniques," Springer Verlag, 1982.

LP and QP methods

[27] Dantzig, G., "Linear programming and extensions", Princeton Universtiy Press, Princeton, 1963.

[28] Wolfe, P., "The simplex method for quadratic programming," Econometrica, Vol.27 pp.382-398, 1959.

[29] Gill, P.E., Murray, W., Saunders, M.A. and Wright, M.H. "Procedures for optimization problems with a mixture of bounds and general linear constraints," ACM Trans. Math. Software, Vol.10, pp.282-298, 1984.

Nonlinear Least Squares Methods

[30] Levenberg, K., "A method for the solution of certain problems in least squares", Quart. Apl. Math. Vol.2, pp.164-168, 1944.

[31] Marquardt, D., "An algorithm for least-squares estimation of nonlinear parameters," SIAM J. Appl. Math. Vol.11, pp. 431-441, 1963.

[32] Moré, J.J., "The Levenberg-Marquardt algorithm: implementation and theory," Numerical Analysis, (G. A. Watson, ed.) Lecture Notes in Mathematics 630, Springer-Verlag, pp.105-116, 1977.

[33] Dennis, J.E., Jr. "Nonlinear Least Squares," State of the Art in Numerical Analysis (D. Jacobs, ed.), Academic Press. pp. 269-312, 1977.

Software for Optimzation

[34] Waren, A.D., Hung S., and Lasdon L.S., "The status of nonlinear programming software: an update," Operations Research, Vol.35, No.4, pp.489-503, 1987.

[35] Schittowski,K., "NLQPL: A FORTRAN-subroutine solving constrained nonlinear programming problems", Annals of Operations Research, Vol. 5, 485-500, 1985.

[36] Gill, P.E., Murray, M.A.Saunders,M.A. and Wright, M.H., "User's guide for NPSOL (Version 4.0): A Fortran Package for Nonlinear Programming," Technical Report SOL 86-2, Systems Optimization Laboratory, Dept.. of Operations Research, Stanford Univ., Stanford, Calif. USA, 1986.

[37] Powell, M.J.D., "VMCWD: A Fortran subroutine for constrained optimization," Report DAMPT/1982/NA4, Cambridge Univ., 1983.

[38] Powell, M.J.D., "ZQPCVX: A Fortran subroutine for convex quadratic programming," Report DAMPT/1983/NA17, Cambridge Univ., 1983.

[39] Rosenthal, R. "Review of the GAMS/MINOS modelling language and optimization program," OR/MS Today. Vol.3, pp.24-32, 1986.

[40] Vanderplaats, G.N., "COPES/ADS- A Fortran control program for engineering synthesis using the ADS optimization program," User's Manual, Engineering Design Optimization, inc.., 1275 Camino Rio Verde, Santa Barabara, CA 93111, USA., 1985.

[41] Drud, A., "CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems", Math. Program., Vol.27, pp.153-191, 1985.

[42] Ecker, J.G., and Kupferschmid "An ellipsoid algorithm for nonlinear programming," Math. Program. Vol.27, pp.83-106, 1983.

[43] Lasdon, L.S., and Waren, A.D., "GRG2 User's Guide," CIS-86-01. Dept. of Computer Information Science, Cleveland State Univ., Cleveland, Ohio, USA, 1986.

[44] Nye., W., Polak, E., Sangiovanni-Vincentelli, A. and Tits, A., "DELIGHT: An optimization-based Computer-Aided-Design system," Proc. IEEE Int. Symp. on Circuits and Systems, Chicago, 1981.

[45] NAG Fortran Library Manual, Mark 12, Vol.4 E04UAF pp.16

Multi-Objective Optimization

[46] Censor, Y., "Pareto optimality in multiobjective problems," Appl. Math. Optimiz., vol. 4, pp. 41-59, 1977.

[47] Zadeh, L.A., "Optimality and non-scalar-valued performance criteria, ", IEEE Trans. Automat. Contr., vol. AC-8, pp. 1, 1963.

[48] Da Cunha, N.O., and Polak, E. "Constrained minimization under vector-valued criteria in finite dimensional spaces," J.Math. Anal. Appl., Vol. 19, pp. 103-124, 1967.

[49] Mukai, H. "Algorithms for multicriterion optimization," IEEE Trans. Autom. Contr., Vol. AC-25, pp.421-432, 1980.

[50] Gembicki, F.W. "Vector optimization for control with performance and parameter sensitivity indices," Ph.D. Dissertation, Case Western Reserve Univ., Cleveland, Ohio, USA., 1974.

[51] Lightner, M.R., and Director S.W., "Multiple criterion optimization for the design of electronic circuits," IEEE Trans. Circuits and Systems, Vol. CAS-28, No.3, pp.169-179, 1981.

[52] Hollingdale S.H., "Methods of operational analysis," in Newer Uses of Mathematics (James Lighthill, ed.), Penguin Books, 1978

[53] Fleming, P.J., "Application of multi-objective optimisation to compensator design for SISO control systems," Electronics Letters, Vol.22, No.5, pp.258-259, 1986.

[54] Fleming, P.J., "Computer-Aided Control System Design of regulators using a Multiobjective Optimization approach," Proc. IFAC Control Applications of Nonlinear Porg. and Optim., Capri, Italy, pp.47-52, 1985.

[55] Waltz, F,M., "An engineering approach: Hierarchical optimization criteria," IEEE Trans., Vol. AC-12, April, 1967, pp.179-180.

Minimax Optimization

[56] R.K.Brayton, S.W.Director, G.D.Hachtel, an L.Vidigal, "A new algorithm for statistical circuit design based on quasi-Newton methods and function splitting," IEEE Trans. Circuits Syst., Vol. CAS-26, pp. 784-794, Sept. 1979.

[57] Hald J., and Madsen K., "Combined LP and quasi-Newton methods for minimax optimization," Math. Program., Vol. 20, No. 1, pp. 49-62, 1981.

[58] Madsen K., Schjaer-Jacobsen H., and Voldby J. "Automated minimax design of networks," IEEE Trans. Circuits and Systems, Vol. CAS-22, pp. 791-795, Oct. 1975.

[59] Madsen,K. and Schjaer-Jacobsen, H., "Singularities in minimax optimization of networks", IEEE Trans. Circuits and Systems, Vol. CAS-23, no. 7, 456-460, 1976.

[60] Madsen, K. and Schjaer-Jacobsen, H., "Algorithms for worst case tolerance optimization", IEEE Trans. Circuits and Systems, Vol. CAS-26, Sept 1979.

Miscellaneous

[61] Nelder, J.A., and Mead, R., "A simplex method for function minimization," Computer Journal, Vol.7, pp.308-313

Part 3

CONTROL SYSTEM DESIGN

Summary - Control System Design (CSD) methods which are used in conjunction with the optimization techniques described in Part 2 are presented. Integral quadratic measures are used as the primary measure of system performance. Problem formulations have been developed to cover a large number of design options and disturbance types. In particular, a method of incorporating control derivative energy terms is used in order to avoid excessive actuator rate saturation. A method for the design of servomechanisms is also presented using a general feed-forward/feedback two-degree-of-freedom control structure. Application of multi-objective optimization to CSD is presented as part of an evolutionary and interactive design process. Examples are given which demonstrate these techniques.

3.1 CONTROL IN PERSPECTIVE

THE FIELD of control provides a wide variety of challenging and potentially rewarding problems covering many engineering disciplines and having a rich background of intellectual depth. Affecting many aspects of modern industrialized life control systems are an integral yet often invisible part of many modern systems. Applications range from a simple thermostat that regulates temperature in buildings to production of consumer goods in a flexible manufacturing plant and from the anti-skid braking systems in modern cars to robot arms in industrial plants.

Although this research is of a general nature, the methodology has principally been aimed at flight control systems. In this field, in particular, CSD plays a very important role. The aircraft auto-pilot reduces the work load on the pilot, the control of high-performance jet engines increases fuel efficiency and power ratings, the trajectory optimization of landing and take-off manoeuveres help to conserve fuel. Recently, feedback control has made it possible to design aircraft that are aerodynamically unstable (e.g. X-29) which makes them highly manoeuverable and responsive to pilot command. The control of the angle of thrust makes it possible to design Vertical Short Take-Off and Landing (VSTOL) aircraft which are also highly manoeuverable in the air.

The control system, therefore, plays a vital role by providing stability and improving performance characteristics over the uncontrolled system. It is surprising then that the underlying mechanism for this achievement is not any physical component but may typically be a set of computer instructions. The potential is therefore to improve the system as a whole without changing any of the system components but by the implementation of better control design techniques. An integrated approach, in which CSD is tied in with the design of sensing, actuation and system components, offers further rewards in terms of improved performance since the control system can then be designed as an integral part of the system as a whole.

Better techniques for the implementation of control strategies and algorithms through improved actuation and on-line computation offer challenges to control designers to meet the full potential of these devices. Improved sensing helps to provide an increase in the availability of system information which needs to be incorporated as part of an effective CSD method. Control System designers are also now being challenged to consider problems such as partially modelled systems or systems where there are large uncertainties and non-linearities using multi-variable, robust, adaptive and fault-tolerant control algorithms.

Although there exist a wide number of both analytical and numerical techniques for CSD encompassing many branches of mathematics and which have been tailored for the design of particular control problems, there is still much scope to find methods which are both practical in terms of implementation and which give the desired performance characteristics when applied to the actual system.

3.1 INTRODUCTION

We consider here systems or sub-systems which can be described by continuous variables and which can be described by differential equations. CSD will therefore be considered in terms of a mathematical description or model. The advantages of being able to model the system, over an on-line technique such as self-tuning or adaptive control, are that the CSD procedure may be tested and tuned within the safety of the model before implementation. The on-line speed advantage over adaptive controllers allows a wider set control parameters to be considered and, additional system information to be utilized. A drawback of basing the design on a modelled system arrives when the system is subject to uncertain change such as aging effects. However, if bounds can be expressed on these uncertainties then they can be compensated for by incorporation of robustness measures or sensitivity reduction in the CSD procedure.

In the order to handle uncertainty and to improve system performance further, requires the use of better control structures and design techniques. To achieve this, it seems likely that there will be a merging of off-line and on-line techniques. This means that multi-loop adaptive control might form for the basis of an otherwise fixed-gain or gain-scheduled global controller. This requires incorporation of effective control algorithms and more realistic design criteria. Thus a control strategy, utilizing improved control structures such as a feed-forward/feedback dynamic controller, and incorporating multi-objective performance objectives, is the means to providing robust, efficient and effective control.

This part will therefore focus on the improvement of control structure and the use of multiple performance objectives in the design method. It is felt that off-line linear CSD strategies will continue to form an integral part of future CSD methods as the foundation for gain-scheduled and adaptive algorithms which can tackle non-linearities and uncertainties. Linear CSD will therefore be considered and envisaged as part of a gain-scheduled or adaptive control strategy which is capable of handling non-linearities, uncertainties, multiple modes of operation and differing operating conditions.

Integral quadratic measures of performance will be used extensively. This is motivated by the well-established numerical solution of such problems and the flexibility with which this criterion can be transformed to incorporate broader control objectives such as stability, speed of response, reduction of interaction in multivariable systems, sensitivity reduction, actuator limits, integrity with respect sensor or actuator failure, as well as differing operating conditions and disturbance types. It is felt that due to the plethora of design choices in terms of controller structures, design objectives and operating conditions it is important to base the Control Design Method on a well established and numerically robust framework in which the problem can be solved. Having laid the foundation for this framework, techniques will be developed for covering other performance objectives, and for incorporating these in a multiple-objective design strategy.

In order to tackle complex problems, the concept of design by evolution will be introduced. In this application, this concerns the systematic increase in both control order and problem complexity in order to incorporate more appropriate control structures and a wider set of design objectives. This facilitates a structured approach to CSD and has distinct computational advantages. Moreover, insight is gained with regard to the trade-offs, associated with control order, and conflicting requirements for control energy restrictions and performance improvement. A number of design examples are given which illustrate this approach focussing on servo design using a two-degree-of-freedom control structure and multi-objective performance criteria.

3.2 INTEGRAL QUADRATIC MEASURES OF CONTROL

A typical control problem is subject to a multitude of design choices involving performance criteria, operating points, model uncertainties, non-linearities, initial conditions, disturbance characteristics an control structures. It therefore becomes important to couch the problem within a well established numerical procedure. One such procedure is based on linear optimal theory in which integral quadratic measures are used with a control configuration of arbitrary size.

A linear time invariant plant description will be used of the form.

$$\begin{aligned} \dot{\mathbf{x}}_p &= \mathbf{A}_p\mathbf{x}_p + \mathbf{B}_p\mathbf{u}_p \\ \mathbf{y}_p &= \mathbf{C}_p\mathbf{x}_p \end{aligned} \tag{3.1}$$

where $\mathbf{A}_p$, $\mathbf{B}_p$, $\mathbf{C}_p$ are real constant matrices and $\mathbf{x}_p$ $\{\mathbf{x}_p(t)\in\Re^n\}$ is termed the state vector of the system, and $\mathbf{u}_p$ $\{\mathbf{u}_p(t)\in\Re^m\}$ and $\mathbf{y}_p$ $\{\mathbf{y}_p(t)\in\Re^r\}$ are the inputs and outputs of the system respectively. The plant is represented pictorially in Fig. 3.1 below. As in Part 2, the notation of boldface lower-case letters for vectors and boldface upper-case letters for matrices will be used where possible.

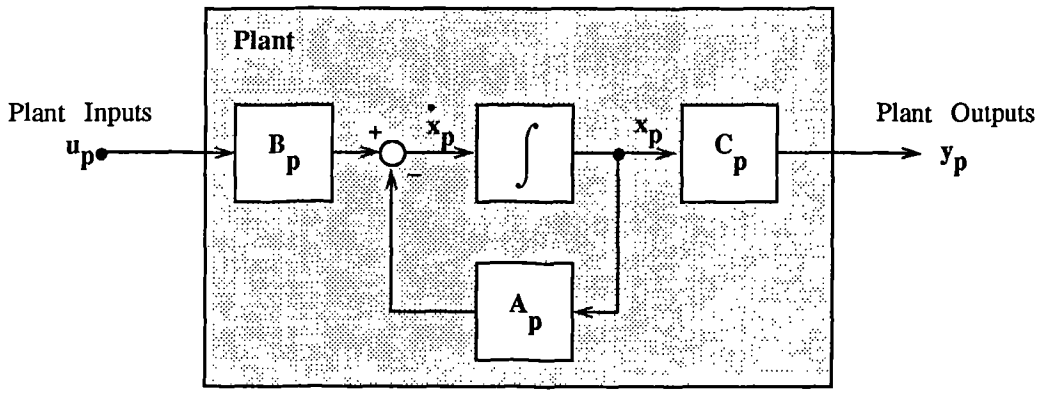

Fig. 3.1 State Space Plant Description

In this Section we will be concerned with *parametric Linear Quadratic* problems in which a parametrized controller (i.e. a controller composed of a number of design parameters) is optimized with respect to a performance objectives consisting of integral quadratic measures. A common form of this is the basic Linear Quadratic Regulator (LQR) problem which is concerned with finding a control $\mathbf{u}_p$ to minimize a cost function, J, composed of integral squared error terms i.e.

Linear Quadratic Regulator Problem

$$\underset{\mathbf{u}_p\in\Re^m}{\text{minimize}} \quad J$$

$$\text{where } J = \int_0^\infty (\mathbf{x}_p^T\mathbf{Q}_p\mathbf{x}_p + \mathbf{u}_p^T\mathbf{R}_p\mathbf{u}_p)\,dt$$

$$\text{and} \quad \mathbf{x}_p(0) = \mathbf{x}_{p0}. \tag{3.2}$$

The introduction of the term $\mathbf{R_p}$ limits the control energy to the input of the system. If this were not present then the real parts of the eigen-values of the closed loop system might tend to $-\infty$ making the control unrealizable.

In order to make the control realizable for a number of initial conditions the control $\mathbf{u_p}$ is generally parametrized as a function of the outputs or states using a matrix of control gains. The way in which the control gains are parameterized gives rise to a number of different control structures and problem formulations.

3.3 CONTROLLER STRUCTURES

3.3.1 Full State Feedback

Given the plant description in Eq. 3.1, the solution to Prob. 3.2 is a full state feedback of the form

$$\mathbf{u_p^* = K_s x_p} \tag{3.3}$$

which together with the plant is represented in Fig. 3.2 below.

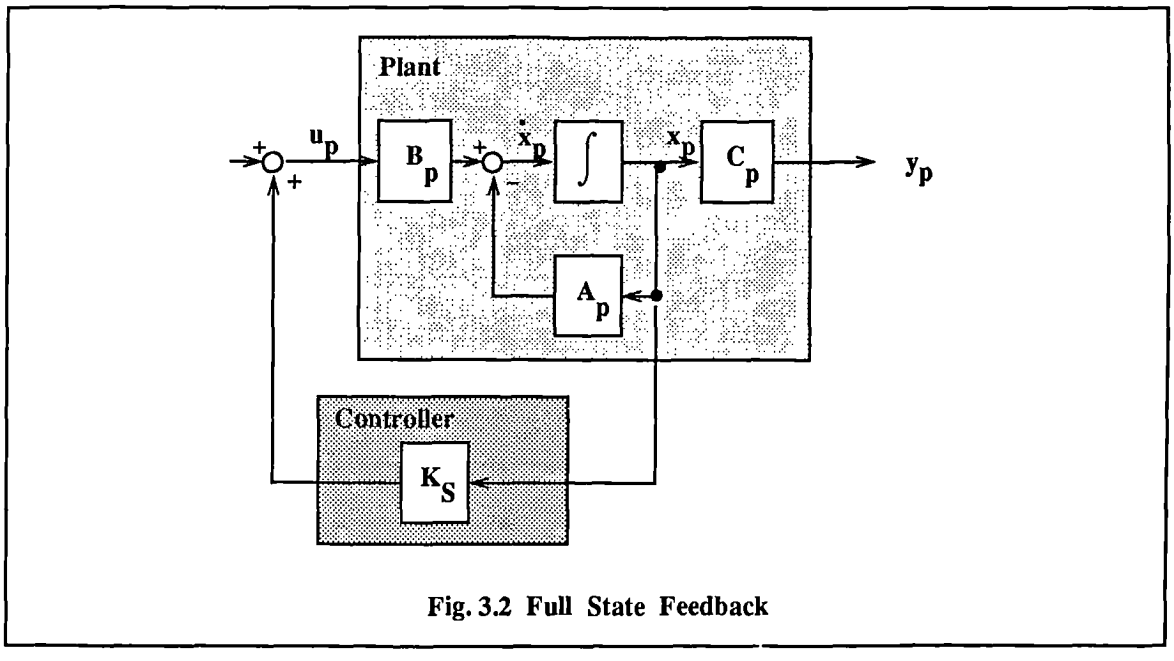

Fig. 3.2 Full State Feedback

The solution to this problem is well known and is found is for example in [3]

$$\mathbf{K_s = -R^{-1}B^TP} \tag{3.4}$$

where $\mathbf{P}$ is an $n \times n$ symmetric matrix which is the unique positive semi-definite solution of the algebraic Ricatti equation:

$$\mathbf{A_p^TP + PA_p - PB_pR^{-1}B_p^TP + Q = 0} \tag{3.5}$$

It should be noted that the optimal controller, K_S, is not initial condition dependent, that is K_S will be optimal for any set of initial conditions. This point is emphasized because other forms of controllers are initial condition dependent (including observer based designs and output feedback based designs). The implementation of the full state controller has the disadvantage, however, that the states must be available for feedback which is not always possible. The following formulations overcome this:

3.3.2 Output Feedback

The method of Levine and Athans [6] and Kosut [7] for optimal constant gain output feedback requires that only the plant outputs be available for feedback. For the plant description (Eq. 3.1) the problem is to find the controller $\mathbf{u}_p$ where

$$\mathbf{u}_p = K_o \mathbf{y}_p \tag{3.6}$$

which minimizes the cost function in Eq. 3.5. The controller and plant are represented in Fig. 3.3 below.

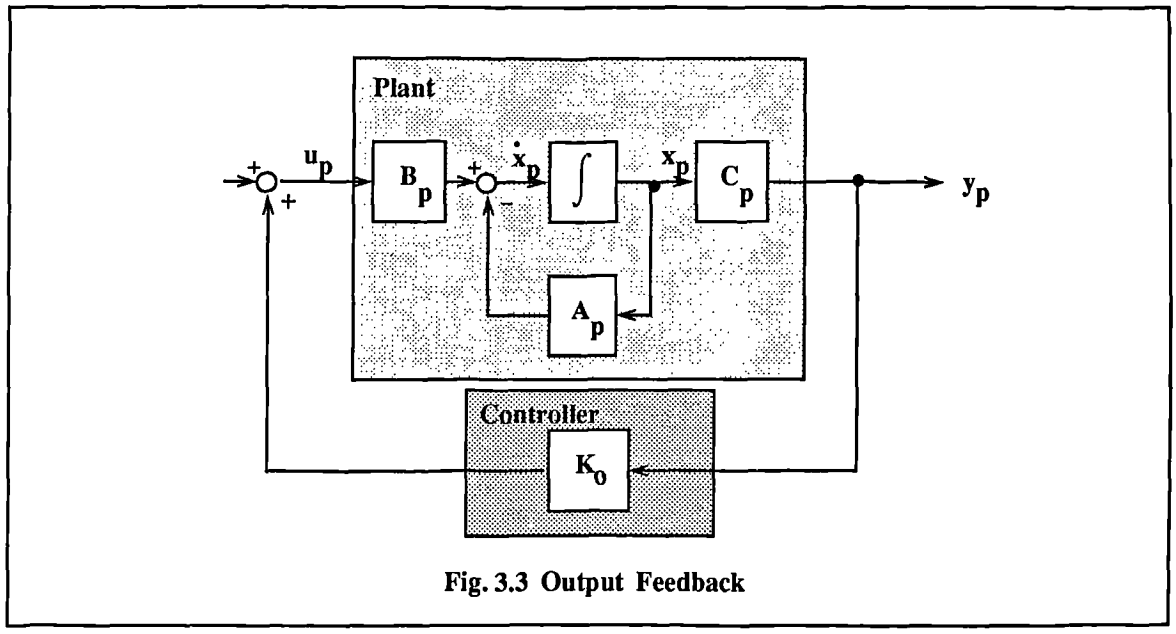

Fig. 3.3 Output Feedback

3.3.3 Dynamic Output Feedback

Johnson and Athans [8], Levine *et al* [9], Basuthakur and Knapp [10] and Wenk and Knapp [11] extended output feedback to the design of dynamic output feedback of fixed order. For the plant description of (Eq. 3.1). the controller $\mathbf{u}_p$ description is of the form:

$$\begin{aligned} \mathbf{u}_p &= D_c \mathbf{y}_p + C_c \mathbf{x}_c \\ \dot{\mathbf{x}}_c &= A_c \mathbf{y}_p + D_c \mathbf{x}_c \end{aligned} \tag{3.7}$$

where $\mathbf{x}_c$ is the compensator state vector $\{\mathbf{x}_c(t) \in \Re^{nc}\}$. The controller is represented in Fig. 2.4.

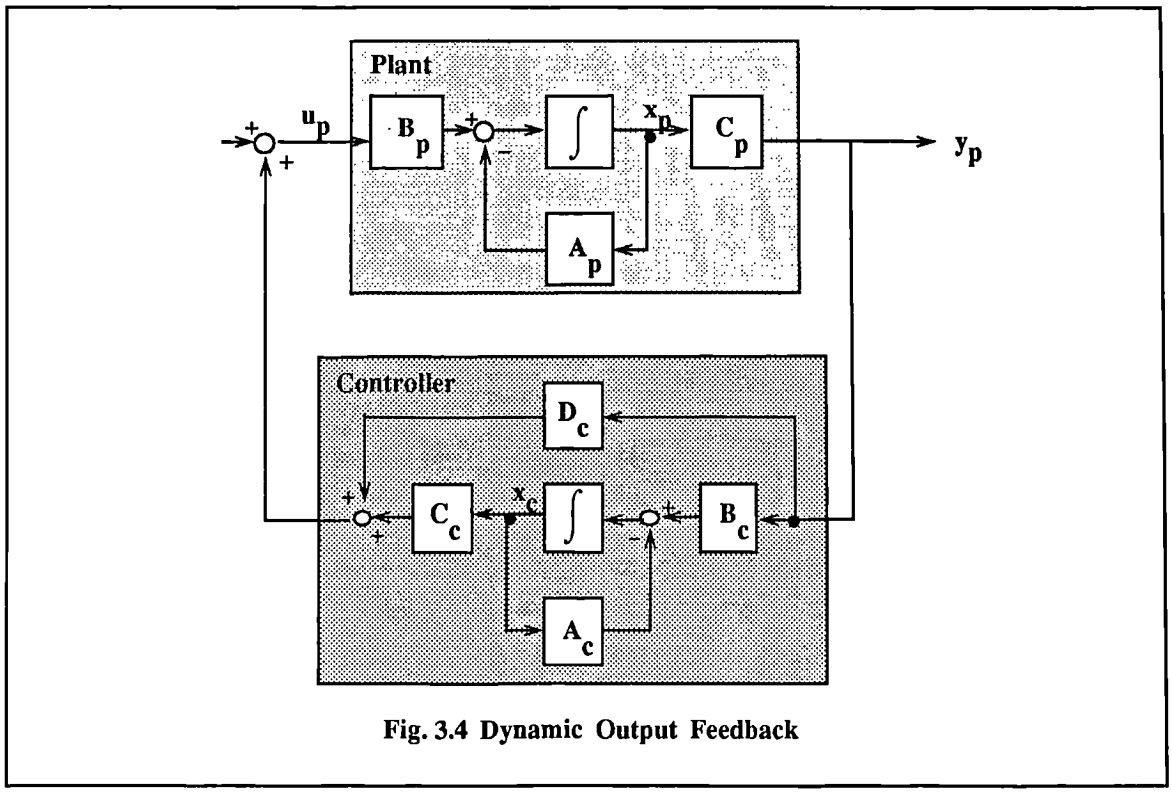

Fig. 3.4 Dynamic Output Feedback

This can be converted to an equivalent output feedback problem by forming the following set of closed loop equations:

$$\dot{x} = (A + BKC)x = \bar{A}x, \tag{3.8}$$

where $A=\begin{bmatrix} A_p & 0 \\ 0 & 0 \end{bmatrix}$ $B=\begin{bmatrix} B_p & 0 \\ 0 & I \end{bmatrix}$ $K=\begin{bmatrix} D_c & C_c \\ B_c & A_c \end{bmatrix}$ $C=\begin{bmatrix} C_p & 0 \\ 0 & I \end{bmatrix}$ $x=\begin{bmatrix} x_p \\ x_c \end{bmatrix}$.

The cost function to be minimized is taken as the same as in Prob. 3.2 i.e

$$J = \int_0^\infty (x_p{}^T Q_p x_p + u_p{}^T R_p u_p)\, dt. \tag{3.9}$$

In the original problem formulation Johnson and Athans [8] considered a different cost function of the form:

$$J = \int_0^\infty (x_p{}^T Q_p x_p + y_p{}^T (D_c{}^T R_p D_c + B_c{}^T R_c B_c) y_p + x_c{}^T (C_c{}^T R_p C_c + A_c{}^T R_c A_c) x_c)\, dt. \tag{3.10}$$

The inclusion of the second and third terms in Eq. 3.10 penalize the elements of the controller in order to limit the control gains. However, since the object of limiting the control gains is to limit the control energy in order to avoid excessive actuator saturation, it is argued that the term

$\int_0^\infty u_p^T R_p u_p dt$ in Prob. 3.2 is sufficient to minimize this. Obviously this condition may not be sufficient if the elements of R_p are negative or are all zero. It is therefore necessary to impose the conditions that the elements of R_p are both positive and the diagonal elements are finite. The system should also be controllable and observable. There may be some rare cases where although the integral quadratic control energy is low the elements of the controller are apparently high, however, the realization of large control values is not usually a problem using modern electronics. Further, since the state space description is non-unique, scaling of the matrices may be achieved. The advantage of using the cost function in Eq. 3.9 is that it simplifies the equations. Further, Eq. 3.9 can be related to the control energy in more direct terms giving the designer more insight to the precise functioning of the weighting matrices. This is especially important in the light of the criticism of LQR methods that the weighting matrix coefficients are difficult to chose with respect to actual design requirements.

3.3.4 General LQR Problem Solution

Following the lines of Fleming [18] and implemented in the program SUBOPT [19], all of the above problems are expressed in a general form. Initially we group all the matrices into a general description using the matrices $\bar{A}$ and $\bar{Q}$:

$$\dot{x} = (A + BKC)x = \bar{A}x \qquad x(0) = x_0 \tag{3.11}$$

$$J = \int_0^\infty (x^T(Q + C^T K^T R K C)x \; dt = \int_0^\infty x^T \bar{Q} x \; dt. \tag{3.12}$$

The construction of the matrices **A,B,C,Q,R** and **x** is dependent on the control structure being used (cf. Eq.3.6 and Eq. 3.8).

The solution to this problem is a set of non-linear matrix equations which can be derived along the lines of Levine and Athans [6] and Kosut [7]. The preferred approach is to use a method used originally by Mendel [17] and Newmann [14]. This involves calculating the cost function and gradients explicitly. From Eq. 3.11 and Eq. 3.12 it is well known that the cost functional, J, is given by:

$$J=\mathrm{tr}(PX), \tag{3.13}$$

where $X_0 = E\{x_0 x_0^T\}$ and $X = X_0$ when there are no disturbances acting on the system.

The matrix, **P**, is a solution of the Lyapunov matrix equation:

$$P\bar{A} + \bar{A}^T P + \bar{Q} = 0$$

$$(\text{i.e } P(A + BKC) + (A + BKC)^T P + Q + C^T K^T R K C = 0) \tag{3.14}$$

Gradient Calculation

While Eq. 3.13 and Eq. 3.14 allow the explicit calculation of J, gradient calculations enable the efficient solution of Prob. 3.2 using an unconstrained gradient optimization method. The problem can be considered as an equality constrained problem composed of the cost function in Prob. 3.2 and a set of equality constraints (Eq. 3.14). Forming the Lagrangian function we get:

$$L(\mathbf{K},\Lambda)=\mathrm{tr}\{\mathbf{X}\mathbf{P} + (\ \mathbf{P}\bar{\mathbf{A}} + \bar{\mathbf{A}}^{\mathrm{T}}\mathbf{P}\ + \bar{\mathbf{Q}}\ )\Lambda\} \tag{3.15}$$

where Λ is a symmetric matrix of Lagrangian multipliers. The necessary conditions for a minimum are given by (cf. Mendel[17]):

$$\nabla L(\Lambda)=\nabla L(\mathbf{P})=\nabla L(\mathbf{K})=0 \tag{3.16}$$

The partial derivatives $\nabla L(\Lambda)$, $\nabla L(\mathbf{P})$, $\nabla L(\mathbf{K})$ can be calculated using gradient matrix operations (see, [25-27] and Appendix B) giving:

$$\nabla L(\Lambda) = \bar{\mathbf{A}}\Lambda\mathbf{A}^{\mathrm{T}} + \bar{\mathbf{A}}\Lambda + \mathbf{X} \tag{3.17}$$

$$\nabla L(\mathbf{P}) = \mathbf{P}\bar{\mathbf{A}} + \bar{\mathbf{A}}^{\mathrm{T}}\mathbf{P}\ + \bar{\mathbf{Q}} \tag{3.18}$$

$$\nabla L(\mathbf{K}) = 2(\mathbf{B}^{\mathrm{T}}\mathbf{P}\Lambda\mathbf{C}^{\mathrm{T}} + \mathbf{R}\mathbf{K}\mathbf{C}\Lambda\mathbf{C}^{\mathrm{T}}). \tag{3.19}$$

It should be noted that $\nabla L(\mathbf{P})$ is identical to Eq. 3.14, also, $\nabla L(\Lambda)$ is a Lyapunov matrix equation which can be calculated efficiently using the Schur form of $\bar{\mathbf{A}}$ used in the calculation of Eq. 3.14 (see, for instance, [23]). The gradient matrix, $\nabla L(\mathbf{K})$ is equivalent to $\nabla J(\mathbf{K})$ since Eq. 3.14 is solved at each iteration.

The advantage of calculating the cost, J, and gradient, $\nabla J(\mathbf{K})$, explicitly are twofold. Firstly, it facilitates the use of a well established gradient optimization (e.g. BFGS). Secondly, the cost function may be used in another problem formulation, such as, in a multi-objective strategy.

3.4 DISTURBANCE REJECTION

The objective of LQR designed controllers is to bring a plant which is in a non-zero state with initial conditions $\mathbf{x}(0) = \mathbf{x}_0$, to a zero state $\mathbf{x}=0$ and $\dot{\mathbf{x}}=0$. In a practical situation this type of problem is not usually of this form. A more typical situation is that the plant is being continually or intermittently disrupted by a set of external disturbances. The disturbances cause unwanted transients in the output which it is desired to reject.

The vector $\mathbf{x}_0$ is a vector of the plant and controller initial conditions ($\mathbf{x}_{p0}$ and $\mathbf{x}_{c0}$), it may also be chosen in order to reflect disturbances acting on the plant. The next few Sections look at how such disturbances can be modelled and introduced into a general problem formulation. Section 3.4.1 looks at, in particular, the case of an output disturbance acting on the plant. In this case the general solution procedure must be modified since the controller itself is a function of the disturbance being fed back to the plant. Section 3.4.2 looks at how disturbances can be modelled and put into a general form.

3.4.1 Impulse Disturbances

The disturbances will be assumed to be of two types:- those acting on the plant itself, x_{d0}, and those acting on the output, y_{d0}, as shown in Fig. 3.5. In the case when the disturbances are Dirac impulses (i.e $x_{d0}\delta(t)$, $y_{d0}\delta(t)$) they can be modelled as initial conditions on the closed loop plant. To understand why this is possible MacFarlane [5] shows that the response of a system with no input released from an initial condition x(0) is the same as that of a system, initially at rest, that is subject to Dirac impulses whose weights are the same as the components of x(0) . i.e

$$\{\ \dot{x} = Ax \qquad x(0)=x_0\ \} \text{ is equivalent to } \{\ \dot{x} = Ax + x_0\delta(t) \quad x(0)=0\ \} \tag{3.20}$$

where the Dirac impulse $\delta(t)$ occurs at t=0.

Output Disturbance

A disturbance at the output as shown in Fig. 3.5 cannot be directly handled by setting constant initial conditions on the plant. This is because the disturbance acts initially on the controller so that the initial conditions on the plant and controller states are dependent on the controller itself. We must therefore add terms to the gradient, $\nabla J(K)$ and initial condition matrix X_0.

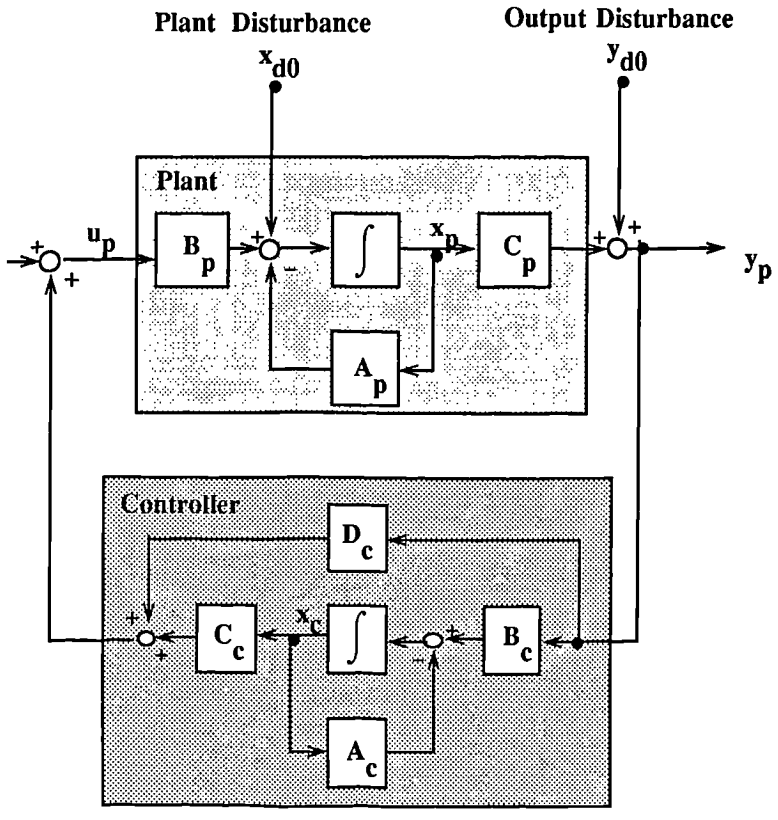

Fig. 3.5 Output Disturbance

Defining statistical matrices $\mathbf{X_d}$ and $\mathbf{Y_d}$ as follows:

$$\mathbf{X_d} = E\{\mathbf{x_d}\mathbf{x_d}^T\}$$
$$\mathbf{Y_d} = E\{\mathbf{y_d}\mathbf{y_d}^T\}, \tag{3.21}$$

where $\mathbf{x_d} = \begin{bmatrix} \mathbf{x_{d0}} \\ \mathbf{0}_{n_c} \end{bmatrix}$, $\mathbf{y_d} = \begin{bmatrix} \mathbf{0}_n \\ \mathbf{y_{d0}} \end{bmatrix}$,

and $\mathbf{x_{d0}}$, $\mathbf{y_{d0}}$ are statistically independent i.e. $E\{\mathbf{y_{d0}}\}=0$ $E\{\mathbf{x_{d0}}\}= 0$ and $E\{\mathbf{x_{d0}}\mathbf{y_{d0}}^T\}=0$.

Since the disturbances are independent, X in Eq. 3.13 can be augmented as follows

$$\mathbf{X} = \mathbf{X_0} + \mathbf{X_d} + \mathbf{B}\mathbf{K}\mathbf{Y_d}\mathbf{K}^T\mathbf{B}. \tag{3.22}$$

Thus, using gradient matrix operations and the formulas given in Appendix B, a new gradient vector is found of the form:

$$\nabla J(\mathbf{K}) = 2(\mathbf{B}^T\mathbf{P}\Lambda\mathbf{C} + \mathbf{R}\mathbf{K}\mathbf{C}\Lambda\mathbf{C}^T) + 2\mathbf{B}^T\mathbf{P}\mathbf{B}\mathbf{K}^T\mathbf{Y_d}. \tag{3.23}$$

This problem has been considered by Kuhn and Schmidt [20] using a different approach but resulting in the same equations.

3.4.2 Stochastic Problem(LQG)

The LQR problem is commonly referred to as the deterministic problem because the disturbances are statistically determined *a priori* to the problem solution. In the case when the disturbances acting on the system , $\mathbf{x_{d0}}$, $\mathbf{y_{do}}$ are Gaussian white noise the problem is generally referred to as the LQG problem. For the LQG problem a similar and related problem to the LQR problem can be solved. In the LQG problem the plant states are excited by zero-mean white noise where the closed loop state equations (cf. Eq. 3.11 and Fig. 3.5) are given by:

$$\dot{\mathbf{x}} = \bar{\mathbf{A}}\mathbf{x} + \mathbf{x_d} \tag{3.24}$$

and the system output vector is given by:

$$\mathbf{y_p} = \mathbf{C_p}\mathbf{x} + \mathbf{y_{d0}}, \tag{3.25}$$

where in this case $\mathbf{x_d}$and $\mathbf{y_{d0}}$ (cf. Eq. 3.21) are zero-mean white-noise processes with

$$E\{\mathbf{x_d}(t)\mathbf{x_d}(t+\tau)\}=\mathbf{X_s}\delta(t)$$
$$E\{\mathbf{y_d}(t)\mathbf{y_d}(t+\tau)\}=\mathbf{Y_s}\delta(t)$$
$$E\{\mathbf{x_d}(t)\mathbf{y_d}(t+\tau)\}=0 \qquad \text{where } \mathbf{x_d} = \begin{bmatrix} \mathbf{x_{d0}} \\ \mathbf{0}_{n_c} \end{bmatrix}, \quad \mathbf{y_d} = \begin{bmatrix} \mathbf{0}_n \\ \mathbf{y_{d0}} \end{bmatrix}, \tag{3.26}$$

The cost function of concern is of the form:

$$J_s = \lim_{t\to\infty} E(\mathbf{x}^T\mathbf{Q}\mathbf{x}), \tag{3.27}$$

and J_s, is given by (see, for instance, Kwakernaak and Sivan [24]):

$$J_s=tr(P_sQ) \tag{3.28}$$

where

$$\bar{A}^TP_s + P_s{}^T\bar{A} + X_s + BKY_sK^TB^T = 0. \tag{3.29}$$

There is thus a close correspondence between the stochastic problem and the deterministic problem. The problem may be solved using the same procedures as described for the deterministic problem. By augmenting the matrices $\bar{Q}$ and X it is even possible to consider a plant with both stochastic and deterministic disturbances.

3.4.3 Disturbance Modeling

We have seen how the problem may be formulated to tackle either impulse type disturbances or Gaussian white noise disturbances. The equations represent a standard form. Generally the disturbances acting on the system will be coloured in some way. Also, the plant may have a D matrix in the state space description. Fig. 3.6 shows how the Disturbance models can be mapped into the standard problem formulation where the plant equations are this time given as A_m, B_m, C_m, D_m. The following matrix achieve the required mapping to the standard form:

$$A_p = \begin{bmatrix} A_m & B_mC_i & C_x & 0 \\ 0 & A_i & 0 & 0 \\ 0 & 0 & A_x & 0 \\ 0 & 0 & 0 & A_o \end{bmatrix} \quad B_p = \begin{bmatrix} B_m \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad C_p = \begin{bmatrix} C_m \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \quad x_{do} = \begin{bmatrix} B_mD_iv_o+D_xv_x \\ B_iv_i \\ B_xv_x \\ B_ov_o \end{bmatrix}$$

$$y_{do}= [\ D_iD_mv_i + D_ov_o]. \tag{3.30}$$

3.4.4 Choosing Initial Conditions

The matrix X in Eq. 3.13 and Eq. 3.30 is calculated from a combination of disturbance estimates and initial conditions which are assumed to be statistically independent. X_0 is a matrix related to the initial conditions of the plant and controller. Since it is normally assumed that the controller has initial conditions, $x_{c0}=0$, then only the plant initial conditions are of concern. Denoting X_{p0} as the $n\times n$ submatrix of X_0 then the following strategies for calculation of X_{p0} have been suggested:

- Set $X_{p0}=E\{x_{p0}x_{p0}{}^T\}$ when there is statistical information available.
- Set $X_{p0}=I$ to minimize an average set of initial conditions (see Kleinman and Athans(1968)[26])
- Set $X_{p0}=P_{11}$, where P_{11} is the $n\times n$ submatrix of P; this tends to minimize the "worst case" cost. (see Fleming [25]).

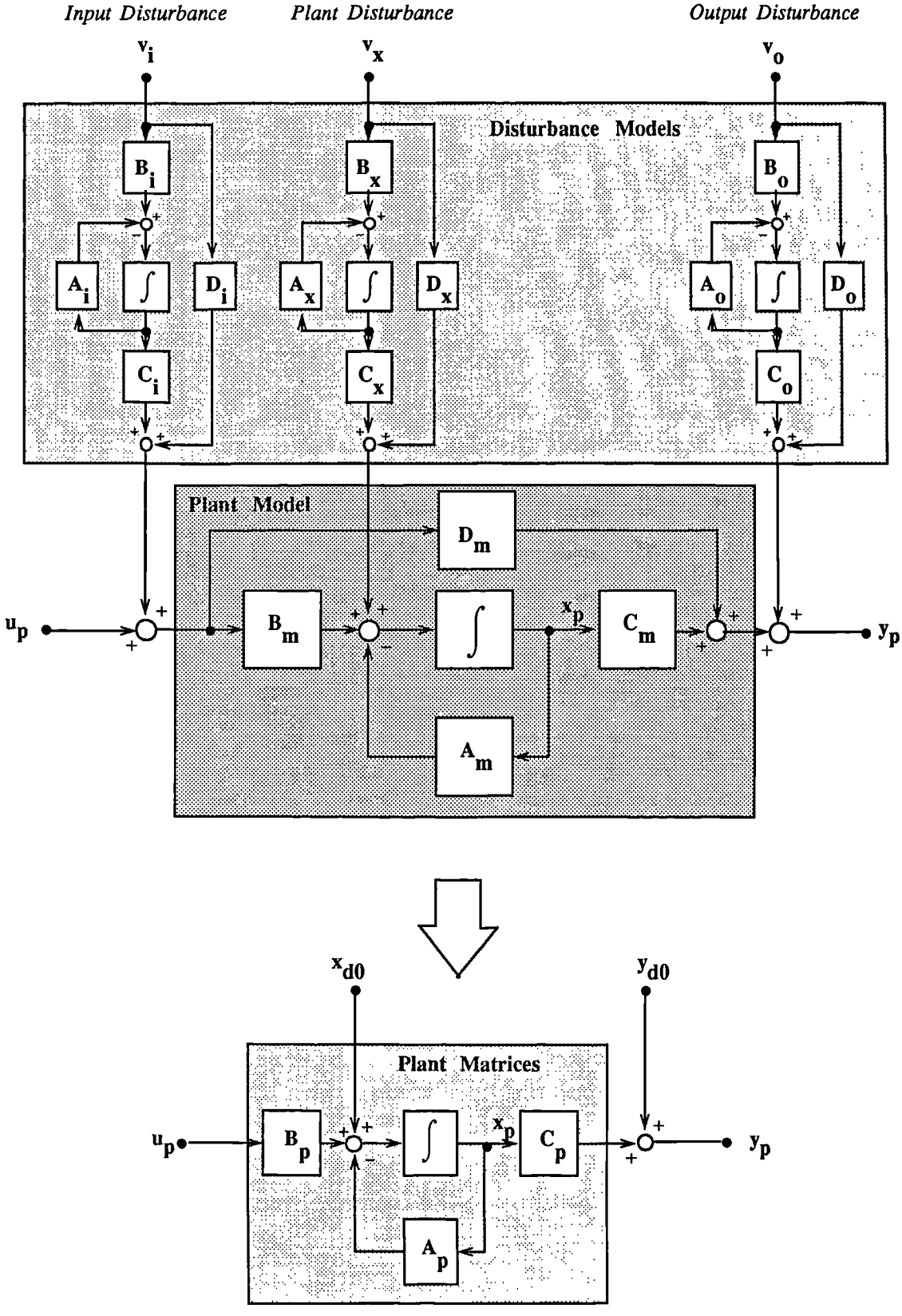

Fig. 3.6 Augmenting The Plant Matrices To Include Disturbance Models and a plant D matrix

3.4.5 Canonical Form

A state space description of a system is not unique. This means that not every element of the dynamic output feedback controller structure of Eq. 3.7 is independent. In an optimization solution procedure this can increase the computation time and, can also lead to numerical difficulties since part of the controller can become decoupled form the plant. A canonical form for $\mathbf{A_c}$, $\mathbf{B_c}$, $\mathbf{C_c}$ and $\mathbf{D_c}$ is therefore used which has only $n_c(m+r) + mr$ free elements. This is a *minimum parameter description* since it is the minimum number of elements required to describe an arbitrary multi-input multi-output system of order n_c. The idea is taken from Kuhn and Schmidt [20], Martin and Bryson [21] and Sirisena and Choi [16]. We use a different canonical form here which is derived from similarity transformations using the controllability matrices of $\mathbf{A_c}$ and $\mathbf{B_c}$. The canonical description of the compensator is given by:

$$\mathbf{A_c} = \begin{bmatrix} 0 & 0 & a_{1,1} & \cdots & a_{1,m} \\ 1 & 0 & a_{2,1} & \cdots & a_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 1 & a_{nc-1,1} & \cdots & a_{nc-1,m} \\ 0 & 0 & a_{nc,1} & \cdots & a_{nc,m} \end{bmatrix} \quad \mathbf{B_c} = \begin{bmatrix} 1 & 0 & \cdots & b_{1,m} \\ 0 & 1 & \cdots & b_{2,m} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & b_{nc-1,m} \\ 0 & 0 & \cdots & b_{nc,m} \end{bmatrix} \tag{3.31}$$

where the matrices $\mathbf{C_c}$ and $\mathbf{D_c}$ are free to vary. For examples of this canonical form, for both SISO and MIMO systems, refer to the design examples in Section 3.10. In order to fix the necessary elements of the control matrices a set of masking matrices are used so that any elements of $\mathbf{A_c}$, $\mathbf{B_c}$, $\mathbf{C_c}$ and $\mathbf{D_c}$ can be fixed or free to vary. This allows the controller to be set to any appropriate form, such as a decentralized control structure with each output feeding back independently. The reason for using this canonical description, as opposed to any other, is to facilitate the mapping of other controllers into this form using similarity transformations. This enables other controllers to be used as starting values for the optimization procedure whcih may assist in fast convergence and the avoidance of local minima.

3.4.6 Evolutionary Controller Mapping

In order to make use of control gains from previous design phases an evolutionary design procedure is used in which a controller of order n_c is mapped onto a controller of order n_c+1. This is achieved by the matrix mapping shown in Fig. 3.7

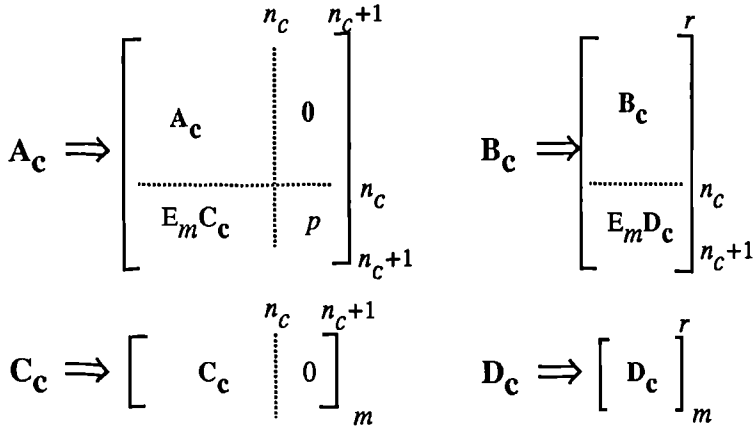

Fig. 3.7 Mapping Of A Lower-Order Controller Into A Higher-Order Controller

where E_m is a row vector of ones, {1,1,..1}, of size m. The scalar p should be chosen as any finite negative number (e.g. -10), it represents the position of the additional pole on the real axis. In the s-domain, the above mapping corresponds to making a series connection of a unity gain controller whose poles and zeros are cancelled.

Having performed the mapping, it is then necessary to transform the new controller to the required canonical form prior to optimization. This can be achieved using similarity transformations based on forming the controllability matrix of $\mathbf{A_c}$, $\mathbf{B_c}$

The controllability matrix of $\mathbf{A_c}$ $\mathbf{B_c}$ is given as:

$$\mathbf{C_o} = [\ \mathbf{B_c},\ \mathbf{A_cB_c},\ \mathbf{A_c^2B_c},\ ...,\ \mathbf{A_c^{n_c}}\ \mathbf{B_c}], \tag{3.32}$$

where the order of $\mathbf{A_c}$ is n_c+1.

A canonical form can then be obtained using the following similarity transformations:

$$\mathbf{A_t} = \mathbf{C_o \backslash A_c C_o} \qquad \mathbf{B_t} = \mathbf{C_o \backslash B_c} \qquad \mathbf{C_t} = \mathbf{C_c C_o} \qquad \mathbf{D_t} = \mathbf{D_c}, \tag{3.33}$$

where the operator $\mathbf{C_o \backslash A_c}$ indicates a non-zero solution, $\mathbf{F}$, to the equation, $\mathbf{C_o F} = \mathbf{A_c}$. In the case when m=1 (i.e. SISO or SIMO systems) the matrices $\mathbf{A_t}$, $\mathbf{B_t}$, $\mathbf{C_t}$, $\mathbf{D_t}$ will be in the required canonical form, otherwise certain rows and columns of the matrices must be removed which correspond to the rows in the matrix $\mathbf{A_t}$ which contain all zero elements.

When m>1 (i.e. MIMO or MISO system), it has been found, for some systems, that the required canonical form cannot always be achieved using this procedure. In this case different values for the variable p in Fig. 3.7 are tried until the required canonical form is found. Further research is necessary to discover whether a more robust method can be found for performing this task.

To summarize, incrementing the controller matrices, $\mathbf{A_c}$, $\mathbf{B_c}$, $\mathbf{C_c}$ and $\mathbf{D_c}$ involves the following steps:

(1) Map lower-order controller into higher-controller using matrix augmentations given in Fig. 3.7.

(2) Using similarity transformations (Eq. 3.32 and Eq. 3.33) put $\mathbf{A_c}$, $\mathbf{B_c}$ and $\mathbf{C_c}$ in the required canonical form.

(3) For MIMO or MISO controllers, remove rows and columns $\mathbf{A_t}$, $\mathbf{B_t}$, $\mathbf{C_t}$ (Eq. 3.33) which correspond to rows of zeros in $\mathbf{A_t}$. If the controller is still not in the required form try a different value for p in Fig. 3.7 and go to Step(1).

3.5 ADDITIONAL DESIGN OPTIONS

Prob. 3.2 represents the standard LQR problem. Due to the explicit expression for J and $\nabla J(\mathbf{K})$ and by augmenting the matrices **A**, **Q** and **X** it is possible to consider additional design options.

3.5.1 Control Derivative Measures

We present here a method for the incorporation of derivative control energy terms into the cost function in Prob. 3.2 without causing an increase in problem order.

Introduction

The incorporation of the integral term $\int_0^\infty (\mathbf{u}_p{}^T \mathbf{R}_p \mathbf{u}_p)\, dt$ in Prob. 3.2 reduces the amount of control energy applied to the actuators or inputs of the system. This is required since all actuators or inputs are limited in terms of the amount of energy that can be transferred to the system.

In a practical system the actuator limits generally occur in the form of magnitude and rate limits. Magnitude limits are caused by a maximum absolute value of the input that can be applied to the system before saturation occurs. For instance, an accelerator pedal in a car cannot be pressed past a fixed point. Rate limits are caused by a maximum rate of change that can applied to the system before saturation occurs. In the car example rate limits occur during acceleration of the car when, pressing the accelerator down past a certain rate, does not make the car accelerate any faster.

Actuator saturation of this type can cause non-linear effects in the system which can lead to instability and performance degradation. Whilst in many cases driving the actuators to these limits may produce a system with better system characteristics than a system whose actuation is under-utilized (i.e. which never saturates) the amount of saturation should be controlled so that non-linear effects do not undermine the performance characteristics of the linear model.

In the basic linear regulator problem the measure $\int_0^\infty (\mathbf{u}_p{}^T \mathbf{R}_p \mathbf{u}_p)\, dt$ is used to limit control energy. However, while this measure is appropriate for controller magnitude limits, it is not particularly good for constraining rate saturation. A better term for controlling rate saturation is the cost function:

$$J_d = \int_0^\infty (\dot{\mathbf{u}}_p{}^T \mathbf{S}_p \dot{\mathbf{u}}_p)\, dt, \tag{3.34}$$

where $\dot{\mathbf{u}}_p$ indicates the rate of change of the control input with respect to time.

The addition of this term in the cost function has the effect of limiting the derivative control energy and can be used to reduce the amount of actuator rate limit saturation. Eq. 3.34 is used to form the augmented cost function:

$$J = \int_0^\infty (\mathbf{x}_p{}^T \mathbf{Q}_p \mathbf{x}_p + \mathbf{u}_p{}^T \mathbf{R}_p \mathbf{u}_p + \dot{\mathbf{u}}_p{}^T \mathbf{S}_p \dot{\mathbf{u}}_p)\, dt. \tag{3.35}$$

The addition of this extra term in the cost function requires additional terms in the general LQR

solution of Section 3.4.4. The additional terms make use of the equivalence:

$$\dot{u}_p = -KC\dot{x}_p = -KC(A + BKC)x_p. \tag{3.36}$$

Eq. 3.34 may thus be written as:

$$J_d = \int_0^\infty x^T(C^TK^T(A + BKC)^TS_p(A + BKC)KC) x \, dt. \tag{3.37}$$

Eq. 3.37 is still of the same form as Eq. 3.11 and Eq. 3.12, so that $\bar{Q}$ can be augmented to include the additional terms. Alternatively we can express the cost function explicitly for use in other problem formulations (e.g. multi-objective), i.e

$$J_d = tr(P_dX),$$

where P_d is a solution of the Lyapunov matrix equation (3.38)

$$P_d(A + BKC) + (A + BKC)^TP_d + C^TK^T(A + BKC)^TS_P(A + BKC)KC = 0$$

the partial derivative of J_d can be calculated using the method of Lagrangian multipliers and matrix gradient operations (see [27-29] and Appendix B):

$$\nabla J_d(K)=2(B^TP_d\Lambda_dC+ \quad D_d) \tag{3.39}$$

where

$$D_d =S_pKC((A\Lambda_d(A^TC^T+C^TK^TB^T)+BKC\Lambda_d(A^TC^T+C^TK^TB^T))+ B^TC^TK^TS_pKC((A +BKC)\Lambda_d)) C^T.$$

and Λ_d is a solution of the Lyapunov equation

$$\Lambda_d(A + BKC)^T + (A + BKC)\Lambda_d + X_0 = 0. \tag{3.40}$$

The above cost function may also be used in the standard problem using the following augmentations:

$$\bar{Q} = \bar{Q} + C^TK^T(A + BKC)^TS_P(A + BKC)KC \tag{3.41}$$

$$\nabla J(K) = \nabla J(K) + D \tag{3.42}$$

where

$$D =S_pKC((A\Lambda(A^TC^T+C^TK^TB^T)+BKC\Lambda(A^TC^T+C^TK^TB^T))+ B^TC^TK^TS_pKC((A +BKC)\Lambda)) C^T.$$

Output Disturbance with Control Derivative Measures

When using derivative measures, in the case of an output disturbance acting on the system, there is a direct coupling of the disturbance to the plant input via the matrix D_c. The cost function and gradient matrix should therefore be augmented as follows:

$$J=J + tr\{E(y_0y_0^T)D_c^TS_pD_c\} \tag{3.43}$$

$$\nabla J(K) = \nabla J(K) + 2S_pD_cY_d. \tag{3.44}$$

3.5.2 Sensitivity Measures

Many authors have considered the reduction in sensitivity for the LQR design (see for instance, [44-46]). The approach we adopt here is to consider the sensitivity in terms of an additional cost function which can be added to the general problem formulation or used within a multi-objective design method. Sensitivity will be considered with respect to the disturbances, initial conditions and plant matrices. A scalar measure of sensitivity will be obtained using a norm of the sensitivity matrices. This operation uses the trace operator so that the gradient matrices can be acquired with respect to the controller parameters. Thus, we will consider a sensitivity measure of the following form :

$$J_s = \mathrm{tr}\,\{S^T E S\}, \tag{3.45}$$

where **S** is matrix of sensitivity terms and **E** is a weighting matrix which can be used to address individual elements or to weight particular elements which have more importance.

The sensitivity matrices (**S** in Eq. 3.45) that we will be considering are with respect to the initial condition matrix, $\mathbf{X_0}$, the disturbance matrices, $\mathbf{X_d}$, $\mathbf{Y_d}$, and the plant matrices $\mathbf{A_p}$, $\mathbf{B_p}$, $\mathbf{C_p}$. They will be denoted as $\nabla J(\mathbf{X_0})$, $\nabla J(\mathbf{X_d})$, $\nabla J(\mathbf{Y_d})$, $\nabla J(\mathbf{A_p})$, $\nabla J(\mathbf{B_p})$, and $\nabla J(\mathbf{C_p})$ respectively. The sensitivity matrices, corresponding to **S** in Eq. 3.45, can be calculated using gradient operations (see, Appendix B), giving:

$$\begin{aligned}
\nabla J(\mathbf{X_0}) &= \mathbf{P} \\
\nabla J(\mathbf{X_d}) &= \mathbf{P} \\
\nabla J(\mathbf{Y_d}) &= \mathbf{K^T B^T Y_d B K} \\
\nabla J(\mathbf{A_p}) &= 2\mathbf{P\Lambda} \\
\nabla J(\mathbf{B_p}) &= 2\mathbf{\Lambda C^T K^T} + 2\mathbf{P B K Y_d K} \\
\nabla J(\mathbf{C_p}) &= 2\mathbf{K^T B^T \Lambda}.
\end{aligned} \tag{3.46}$$

Thus, using Eq. 3.45, the cost functions are of the form

$$\begin{aligned}
J_{X0} &= \mathrm{tr}\{\mathbf{PEP}\} \\
J_{Xd} &= \mathrm{tr}\{\mathbf{PEP}\} \\
J_{Yd} &= \mathrm{tr}\{\mathbf{K^T B^T Y_d B K E K^T B^T Y_d B K}\} \\
J_A &= \mathrm{tr}\{2\mathbf{P\Lambda E\Lambda P}\} \\
J_B &= \mathrm{tr}\{4(\mathbf{\Lambda C^T K^T} + \mathbf{P B K Y_d K})\mathbf{E}((\mathbf{\Lambda C^T K^T} + \mathbf{P B K Y_d}^T)\} \\
J_C &= \mathrm{tr}\{4\mathbf{K^T B^T \Lambda E \Lambda B K}\}
\end{aligned} \tag{3.47}$$

by augmenting these equations to the general problem formulation of Prob. 3.2 and using a further set of gradient operations, it is possible to generate the gradients with respect to **K**, i.e, $\nabla J(\mathbf{K})$ where $J = J + J_s$. . Alternatively, by forming a new Lagrangian, it is possible to find $\nabla J_s(\mathbf{K})$ explicitly for use, for instance, in a multi-objective problem formulation. It is also possible to derive sensitivity matrices with respect to the derivative control measures (Eq. 3.34) and the stochastic problem (Section 3.4.2). These are derived in a straightforward manner using gradient matrix operations (cf. [27-29] and Appendix B).

It should be noted that when a number of cost functions are being calculated, it is sometimes more computationally efficient to calculate gradients using a finite difference method. For poorly conditioned matrices, this approach may also be more accurate.

3.6 SERVOMECHANISMS

The purpose of this section is to show the proper formulation of a class of linear control problems known as tracking or servomechanism problems using integral quadratic measures of control.

Athans [30] has considered the design of PID controllers. Bernstein and Haddad(1987)[36] has considered tracking for constant gain output feedback controllers. We extend this here to the generalized two-degree-of-freedom feedforward/feedback controller (with or without integral action). Davison [32-34] has also considered the servomechanism problem with Full State Feedback but without consideration to limiting the control energy. We show here how control energy may be limited in order to avoid actuator saturation by the inclusion of special terms in the cost function.

Recently Arstein and Leizarowitz [35], and Choek *et al* [37] have also considered the servomechanism problem for the full state feedback case (or using state estimators). We assume here that not all the states are available for feedback.

The general problem can be described as follows:

Problem Definition: Given the plant description of Eq. 3.1 a reference input $\mathbf{y_{ref}}$ which is in the form of a set of step responses (or filtered steps) must be tracked by the outputs $\mathbf{y_p}$. The problem can be stated as follows:

LQ Servo Problem

$$\underset{\mathbf{u_p}\in\Re^m}{\text{minimize}} \quad J$$

$$\text{where } J = \int_0^\infty (\mathbf{y_{ref}}-\mathbf{y})^T\mathbf{Q}(\mathbf{y_{ref}}-\mathbf{y}) + (\mathbf{u_{ref}}-\mathbf{u_p})^T\mathbf{R}(\mathbf{u_{ref}}-\mathbf{u_p})) \, dt \tag{3.48}$$

where $\mathbf{y_{ref}}$ is the reference input signal to be tracked and $\mathbf{u_{ref}}$ is the corresponding input which is required to maintain tracking of $\mathbf{y_{ref}}$ in steady state.

We do not use the term $\mathbf{u_p}^T\mathbf{R_p}\mathbf{u_p}$ in the integral since this would tend to ∞ as $t\rightarrow\infty$. This is because for most systems it is necessary to apply a constant reference input to the control inputs in order to maintain tracking.

The approach we adopt here is to consider the problem in two parts: First to design a controller to maintain $\mathbf{u_{ref}}$ so that $\mathbf{y_{ref}}$ is tracked as $t\rightarrow\infty$. This requires the control structure to be either some form of integral action or for a constant-gain precompensator to be applied to the system so that the d.c. gain of the closed loop system is unity (or for multivariable systems the unit matrix **I**). Second, to minimize Eq. 48 with respect to the control structure being considered.

3.6.1 Two-Degree-of-Freedom (2DF) Control Structure

We define here a two-degree-of-freedom control structure or generalized feedforward/feedback controller (see, for instance, Hara and Sugie[38] and also [39-43]). This control structure consists of both feedforward elements and feedback elements and so is a generalization of all linear control structures, e.g. PID controllers.

The controller is described by the following equations:

$$\dot{x}_c = A_c x_c + B_{cfb} y_{ref} + B_{cff} y$$

$$u_p = C_c x_c + D_{cff} y_{ref} + D_{cfb} y \tag{3.49}$$

and is shown in Fig. 3.8.

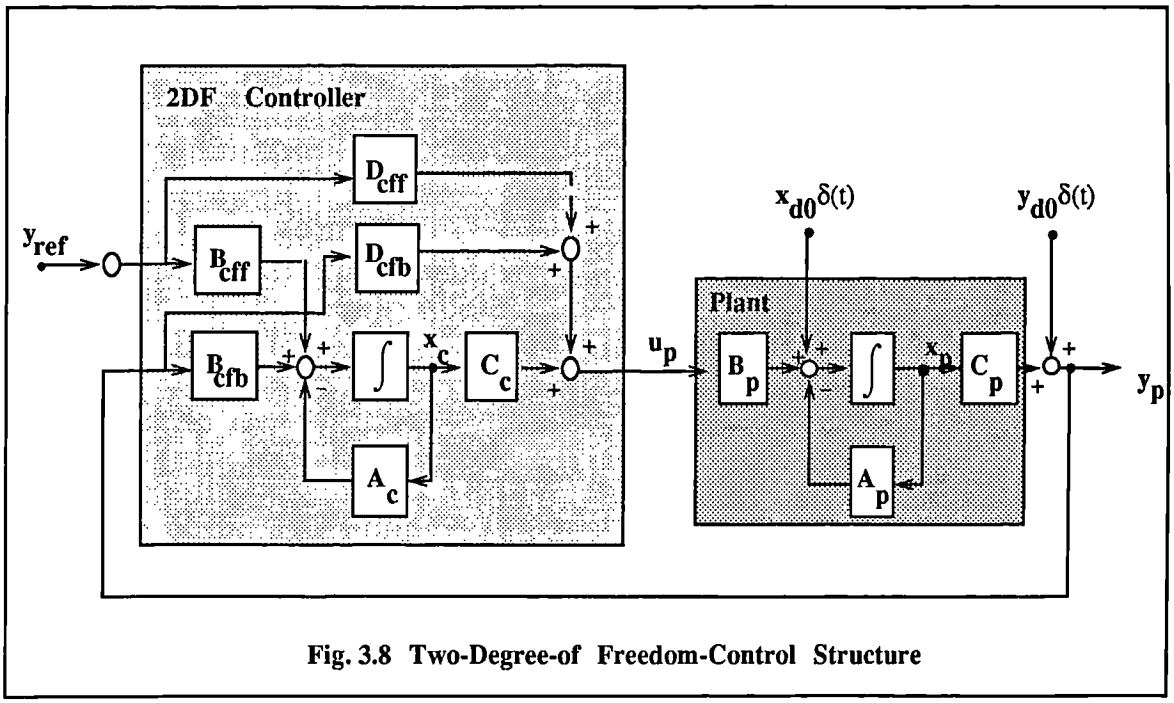

Fig. 3.8 Two-Degree-of Freedom-Control Structure

3.6.2 Design Cycle

The first phase of the design procedure is to constrain the compensator to be either a structure with integral action or to provide the closed loop system with a d.c. gain of unity by using, for instance, a constant gain precompensator.

Integral Action

The use of integrators in a control system has long been used as a means to ensure robust steady-state tracking. The addition of integral action to a system ensures that the system tracks any constant reference input provided that the closed-loop system remains stable. For the multivariable case, a set of m integrators (where m is the number of reference inputs) will eliminate steady state interaction between inputs.

The addition of integral action does have some disadvantages, however, since it may have a destabilizing effect on the plant and cause a degradation of time response or frequency characteristics.

There is generally a trade-off between the amount of integral action that is applied to the system and the transient performance that is required.

In order to incorporate integral action the control structure shown in Fig. 3.9 is used

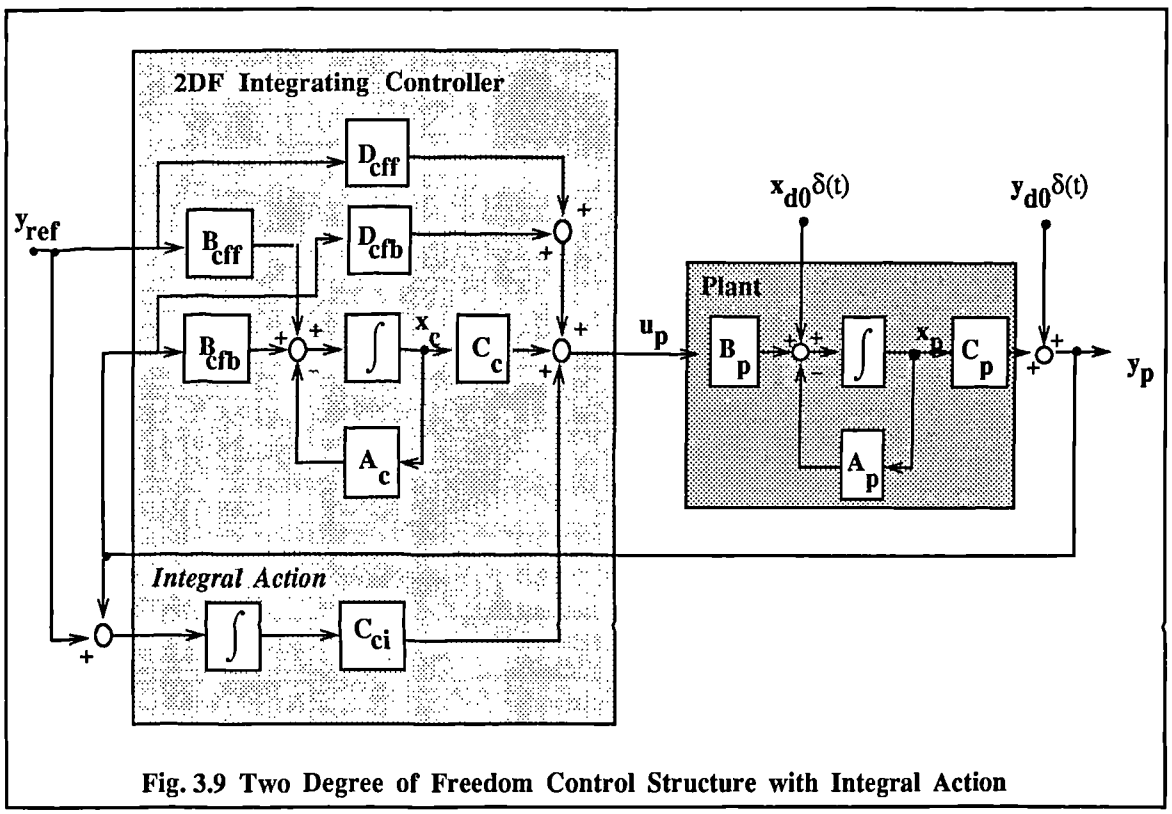

Fig. 3.9 Two Degree of Freedom Control Structure with Integral Action

To incorporate integral action into the generalized 2DF controller description of Eq. 3.49 the control matrices A_c, B_{cfb}, B_{cff} and C_c are augmented as follows:

$$A_c = \begin{bmatrix} A_c & 0 \\ 0 & 0_m \end{bmatrix} \quad B_{cfb} = \begin{bmatrix} B_{cfb} \\ -I_m \end{bmatrix} \quad B_{cfb} = \begin{bmatrix} B_{cff} \\ I_m \end{bmatrix}$$

$$C_c = \begin{bmatrix} C_c \\ C_{ci} \end{bmatrix}. \qquad (3.50)$$

When used as part of an optimization procedure the elements corresponding to the integral feedback are not free and must be fixed.

The amount of integral action is controlled by the matrix $\mathbf{C_{ci}}$ This matrix can be allowed to vary freely with respect to any performance criteria being used, e.g. Eq. 3.48. However $\mathbf{C_{ci}}$ should have bounds on the absolute value of its diagonal elements so that they do not go below a certain threshold value. This threshold value should be chosen to ensure adequate tracking is achieved even in the presence of modelling errors, non-linearities or aging effects.

Second Phase: Minimizing the Cost Function

Having ensured that the d.c. gain of the closed loop system is unity through the use of integral action, the next phase of the design procedure is to minimize (3.48). This can be achieved by converting the problem to a deterministic one by looking at the final values and initial values of the system, assuming step response inputs.

To see how this can be done, we first define the closed loop system which is subject to step responses at t =0:

$$\dot{\mathbf{x}} = \bar{\mathbf{A}}\mathbf{x} + \bar{\mathbf{B}}\mathbf{u} \begin{cases} \mathbf{u}=0 & t<0 \\ \mathbf{u}=\mathbf{y_{ref}} & t\geq 0 \end{cases}$$
$$\mathbf{y} = \bar{\mathbf{C}}\mathbf{x} \qquad (3.51)$$

where $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are the closed loop system matrices: $\bar{\mathbf{A}} = \mathbf{A+BKC}$, $\bar{\mathbf{B}} = \begin{bmatrix} \mathbf{B_p D_{cff}} \\ \mathbf{B_{cff}} \end{bmatrix}$,

and $\mathbf{y_{ref}}$ is a set of step responses applied to the input $\mathbf{u}$ at t=0.

In order to minimize the cost function of Eq. 3.48 an equivalent deterministic (LQR) regulator problem is found by considering final values and initial values of the system. Consider initially the first part of the integral:

$$J = \int_0^\infty (\mathbf{y_{ref}}-\mathbf{y})^T \mathbf{Q} (\mathbf{y_{ref}}-\mathbf{y})\, dt. \qquad (3.52)$$

The aim is to find the initial and final values of the closed loop system and to rewrite the problem in terms of an equivalent LQR problem. Assuming that the closed loop system is stable, then as $t\rightarrow\infty$, $\dot{\mathbf{x}}\rightarrow 0$. Therefore, the final value of $\mathbf{x}$ (as $t\rightarrow\infty$), $\mathbf{x_{fv}}$ can be defined as :

$$\mathbf{0} = \mathbf{A}\mathbf{x_{fv}} + \bar{\mathbf{B}}\mathbf{y_{ref}} \qquad (3.53)$$

giving $\mathbf{x_{fv}} = -(\bar{\mathbf{A}}^{-1})\bar{\mathbf{B}}\mathbf{y_{ref}}$.

Since $\mathbf{y}=\mathbf{C_p}\mathbf{x_p}$, Eq. 3.52 may be written as:

$$J = \int_{0_+}^\infty (-\bar{\mathbf{A}}^{-1}\bar{\mathbf{B}}\,\mathbf{y_{ref}} - \mathbf{x_p})^T \mathbf{C_p}^T \mathbf{Q}\, \mathbf{C_p}(-\bar{\mathbf{A}}^{-1}\bar{\mathbf{B}}\,\mathbf{y_{ref}} - \mathbf{x_p})\, dt. \qquad (3.54)$$

Proof

It is to proved here that the cost functions Eq. 3.52 and Eq. 3.54 are identical. An equivalent deterministic problem is also derived.

First define a state vector of the form:

$$x_s = -\bar{A}^{-1}\bar{B}\ y_{ref} - x_p \tag{3.55}$$

The cost function under consideration is:

$$J = \int_0^{\infty} x_s C_p{}^T Q\, C_p x_s\, dt, \tag{3.56}$$

where at t=0 $x_s = \bar{A}^{-1}\bar{B}\ y_{ref}$ (3.57)

and as $t\rightarrow\infty$, $x_s \rightarrow 0$.

Defining the state equation (from Eq. 3.55) gives:

$$\dot{x}_s = -\bar{A}\,(A^{-1}\bar{B}\,\dot{y}_{ref} + \dot{x}_p) + \bar{B}\,y_{ref} \tag{3.58}$$

Since y_{ref} is a step response then $\dot{y}_{ref} = 0$ for all $t > 0$, the state equation may thus be written as:

$$\dot{x}_s = \bar{A}x_p + \bar{B}\,y_{ref} \quad \text{defined for } t>0. \tag{3.59}$$

The cost function is therefore equivalent to a system which is released from rest and whose initial conditions are given by:

$$x_0 = -\bar{A}^{-1}\,\bar{B}\,y_{ref} + \bar{B}y_{ref} \tag{3.60}$$

A similar proof can be used for the cost function:

$$\int_0^{\infty} (u_{ref} - u_p)^T R(u_{ref} - u_p)\ dt \tag{3.61}$$

since u_p may be expressed as a function of x_p and x_c. It should be noted that u_{ref} is arbitrary here and is put into the cost function so that $(u_{ref} - u_p) \rightarrow 0$, as $t\rightarrow\infty$.

Summary

The evaluation of the cost function (Eq. 3.48) for the plant description (Eq. 3.1) is therefore equivalent to a deterministic problem in which the system is released from rest and whose initial conditions are given by:

$$x(0) = -\bar{A}^{-1}\bar{B}\, y_{ref} \tag{3.62}$$

Since the initial conditions are dependent on the augmented closed loop plant matrix, $\bar{A}$, where $\bar{A}= A + BKC$, then the cost functional, J= tr(PX), is dependent on **K**. Assuming that the step responses occur independently, i.e. $E\{y_{ref}\}=0$, and that there are independent plant disturbances, $E\{y_{d0}\}=0$, and output disturbances, $E\{x_{d0}\}=0$, then the matrix, **X** in the general problem formulation (Section 3.3.4), should be set according to

$$X = X_0 + X_d + KY_d + \bar{A}^{-1}\bar{B}Y_{ref}\bar{B}^T A^{-T} \tag{3.63}$$

where $X_0= E\{x_0 x_0^T\}$, $X_d= E\{x_d x_d^T\}$, $Y_d=E\{y_d y_d^T\}$, $Y_{ref}= E\{y_{ref} y_{ref}^T\}$,

and x_0, x_{d0} ,y_{d0},y_{ref} are statistically independent.

Using gradient matrices and the method of Lagrangian multipliers, the augmented gradient term is given by:

$$\nabla J(K) = 2(B^T P \Lambda C + RKC\Lambda C^T) + 2PK^T E\{y_0 y_0^T\} + 2(CA^{-1}B\, E(y_{ref}^T y_{ref}\}B^T(A^{-1})^T PA^{-1}B)^T. \tag{3.64}$$

Typically for a multivariable system the step response matrix, $Y_{ref}=E\{y_{ref}y_{ref}^T\}$, should be set with the identity matrix, **I**. This assumes that the step responses occur at intermittent intervals of time and after the transients from previous step responses have died away.

3.6.3 Servo Derivative Measures

As has been mentioned in Section 3.6.1 limiting the control derivative energy using the integral:

$$J_d = \int_0^\infty (\dot{u}_p^T S_p \dot{u}_p) \, dt \tag{3.65}$$

helps to minimize actuator rate saturation. When applying this to the servomechanism problem there is an additional term in the solution procedure caused by direct coupling of the input to the plant through the component $\mathbf{D_{cff}}$, since $\mathbf{u_p}$ is given by:

$$u_p = C_c x_c + D_{cff} y_{ref} + D_{cfb} y. \tag{3.66}$$

Assuming y_{ref} is defined in terms of the expected value of the norm of a set of step responses, i.e. $Y_d = E\{ y_{ref}^T y_{ref} \}$, then the contribution of $\mathbf{D_{cff}}$ to the cost function (3.65) is given by:

$$tr\{ Y_d D_{cff}^T S_p D_{cff} \} \tag{3.67}$$

since $tr\{ \int_0^\infty E(y_{ref}\delta(t))^T W(y_{ref}\delta(t)) \} \} = tr \{ E\{ y_{ref}^T y_{ref} \} W \}$.

Differentiating with respect to $\mathbf{D_{cff}}$ gives:

$$\nabla J_d(D_{cff}) = 2 \, Sp \, D_{cff} \, Y_{ref}. \tag{3.68}$$

3.7 OBSERVER DESIGN

The state observers may also be designed using quadratic measures in a similar way to the design of LQR or LQG controllers. Observers estimate the states of the plant given statistical information regarding disturbances acting on the system. It is possible to use an observer together with an adaptive control strategy which may have advantages over fixed gain control strategies. A 2DF observer of the form shown in Fig. 3.10 will be considered.

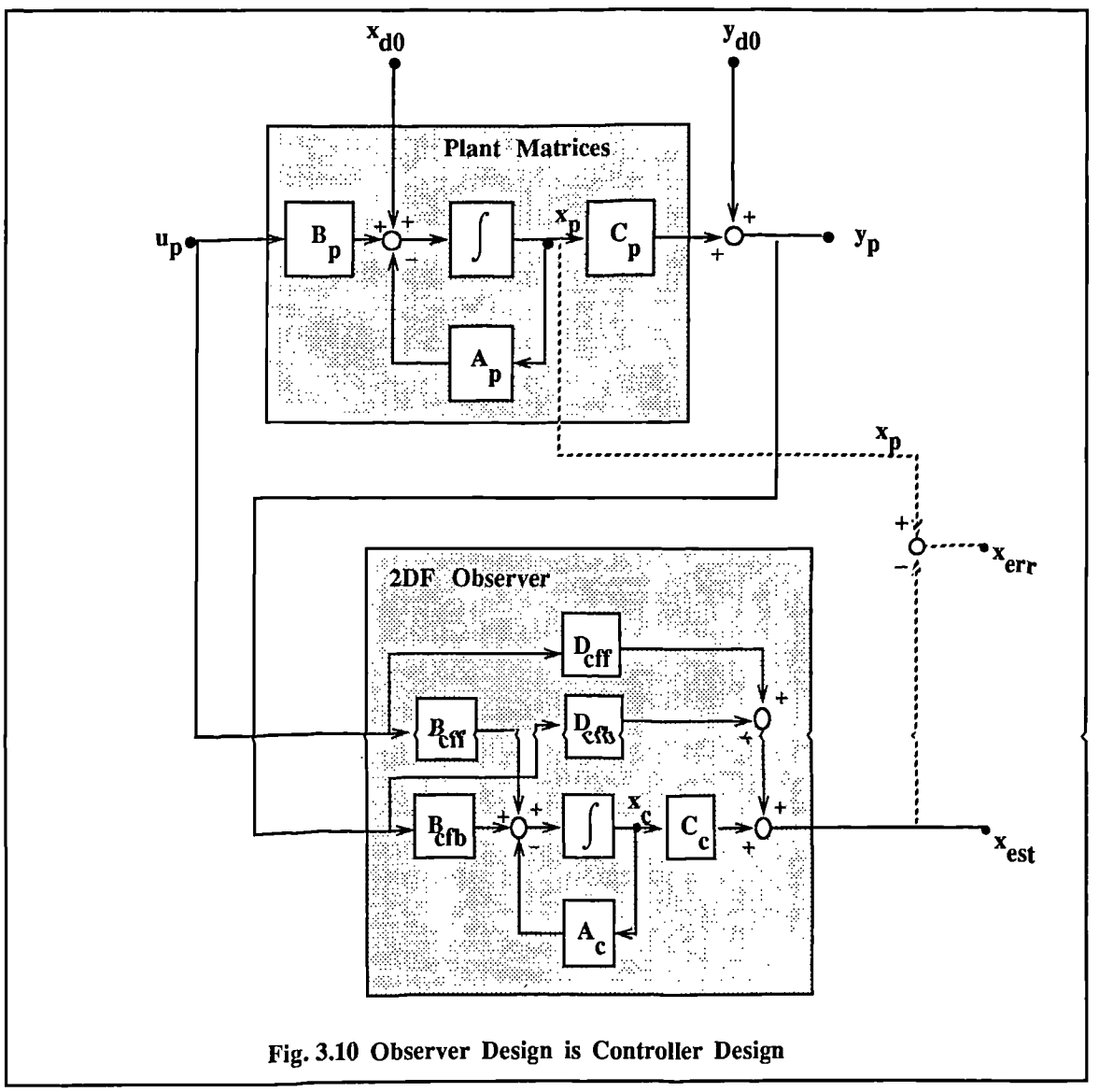

Fig. 3.10 Observer Design is Controller Design

The aim is to minimize x_{err}, given statistical information about the disturbances, x_{d0},y_{d0}. The disturbances x_{d0} and y_{d0} may be modelled as Gaussian or impulse type disturbances as for the LQR and LQG design approaches. If the disturbances are modelled as filtered disturbances then the matrix augmentations given in Section 3.5.3 can be used. The observer may also be designed with respect to statistical information regarding the input, u_p, and plant initial conditions, X_{p0}. The integral quadratic measure to be minimized is of the form:

$$J = \int_0^\infty (x_p - x_{est})\, dt = \int_0^\infty x_{err}\, dt. \qquad (3.69)$$

Using the principles developed for the design of output feedback controllers it possible to design the observers for a range of disturbances (e.g. Gaussian or impulse) and input types (e.g. impulse, Gaussian or step response models). Since, in the design of a steady-state Kalman-Bucy filter, no information regarding the characteristics of the input is used, the 2DF observer in Fig. 3.10 may give improved performance. This is because $\mathbf{B}_{cff}$ is generally set to $\mathbf{B}_p$ and $\mathbf{D}_{cff}$ is set to a matrix of zeros in the Kalman-Bucy filter which may not be the optimal settings, given information regarding the characteristics of $\mathbf{u}_p$.

Further research is necessary to develop the problem solution strategies and matrix augmentations for a range of disturbances and input characteristics.

3.8 MULTI-OBJECTIVE CONTROL SYSTEM DESIGN

Multi-Objective Optimization, and in particular the Goal Attainment method, is a powerful means of casting multiple and possibly conflicting design requirements into a numerically tractable optimization formulation (see Section 2.5).

A fundamental problem in CSD is concerned with making the necessary trade-offs between performance objectives, such as degree of stability, time and frequency response characteristics and how much control energy to apply in terms of control gains. If too much control energy is applied, excessive actuator saturation may occur which might lead to destabilization or a degradation of the performance characteristics. The problem is that, in general, disturbances to the system cannot be accurately predetermined and are generally stochastic in nature. If disturbance bounds cannot be estimated accurately then, assuming a linear feedback controller is used, actuator saturation is likely to occur at some threshold level of disturbance. In many circumstances, a level of saturation is desirable since it represents slewing of the system which may be appropriate for speed of response or stability. This is especially true in applications, such as servomechanisms where to achieve optimal tracking of changing reference input, the system slews initially before locking into the new reference input.

It is apparent that, while the amount of control gain that should be applied may be the result of a complicated decision making process, it is important for control energy in terms of magnitude limits and rate limits to be adjusted in a flexible and interactive manner. Although LQR and LQG methods can limit control energy through the use of weighting matrices, the weights are difficult to relate to true control energy constraints and performance measures. The design approach that is used here is therefore to allow control energy and performance measures to be adjusted independently using the Goal Attainment method. Integral measures are used to provide control energy restrictions and response output performance measures. At the end of each design cycle the designer is presented with time responses resulting from disturbance inputs he has defined in the design. The designer then decides whether further measures must be used to restrict the control energy or whether a trade-off can be made for better response characteristics. In this way the designer interacts with the design process giving preferences and weights to features of the design.

Programming Multi-Objective control problems can be a demanding process that requires the coding of often large numbers of varied performance criteria. The approach therefore used here is to provide a general Multi-Objective solution procedure based on integral quadratic measures to which the designer can add his own. The procedure requires no programming, instead the problem is entered as vectors of weighting matrices and disturbances which are grouped to form column-wise matrices. The number of columns in each of the matrices indicates the number of objectives to be applied in the design. The user may also enter a vectored matrix of different plant models which represent linear models taken at different operating points. Multi-objective design can then performed with respect to different operating conditions. Design examples illustrate this process.

3.9 DESIGN BY EVOLUTION

Practical control problems are often complex and of high-order. In many instances the control engineer in the early stages of a design may be bombarded with choices regarding control configuration, disturbance characteristics and design requirements. The control engineer, however, is unlikely to know the exact design requirements until he has some idea of what can possibly be achieved. Further, posing complex and large design problem using arbitrary design parameters is likely to result in problems with the optimization algorithm. This may cause long solution times and local optima instead of the global optimum.

To overcome these problems an evolutionary design process is used. As the design progresses, controller complexity, model order and the number of design objectives are increased. Fig. 3.11 shows how starting with simple objectives and a simple controller the problem is systematically evolved to incorporate a wider set of objectives and increasingly complex controllers. In a typical design the problem is posed initially using a low-order linear model, simple objectives such as stability requirements, and a fixed gain output feedback controller.

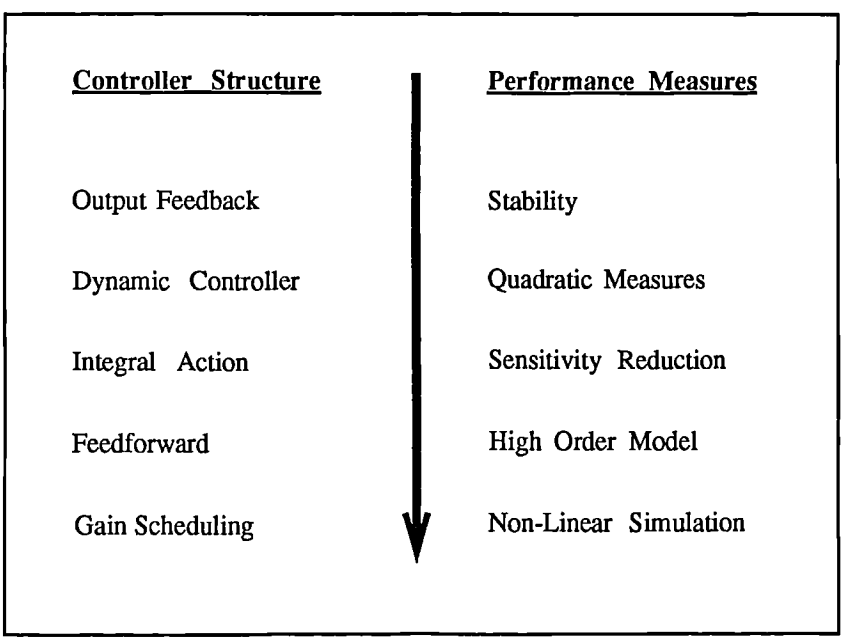

Fig. 3.11 Design by Evolution.

The optimization process is aided by using results from previous design phases as starting values for the next phase. To accommodate an increase in control order and to exploit the solution points from previous design phases, a method of pole/zero cancellation is used to map the low order controller onto a higher order controller (see Section 3.4.6). This is then used to restart the optimization cycle.

Through this evolutionary design process the control engineer gains insight and a growing appreciation of the systems best capabilities, the trade-offs associated with control order and competing design requirements. Further, since computationally expensive design goals are added later in the design cycle, this approach serves to reduce the computational burden. Using results from previous design phases in this way, as opposed to using arbitrary starting values, is likely to improve execution speed and reduce the likelihood of encountering local minima.

3.10 DESIGN EXAMPLES

A number of design examples will be used to illustrate the design techniques described in the previous sections.

3.10.1 Simple Tracking Example

We consider here a simple tracking problem taken from Nishikawa et al.(1984)[48] and also considered by Eitelburg(1987)[47]. The plant is given by the equations

$$G_P(s)= \frac{1}{(1+s)(3+s)^3} \tag{3.70}$$

We demonstrate here an evolutionary design procedure with a number of design phases. Fig. 3.12 shows how both the controller complexity and performance measures are systematically evolved in order to view the trade-offs between different performance measures and control structures The design approach which we will demonstrate here has 7 design phases. In the first phase a controller is designed using a proportional output feedback controller using only a scalar performance measure of control and output energy terms. In the later stages of the design a high order feedforward controller is designed using a multi-objective optimization strategy.

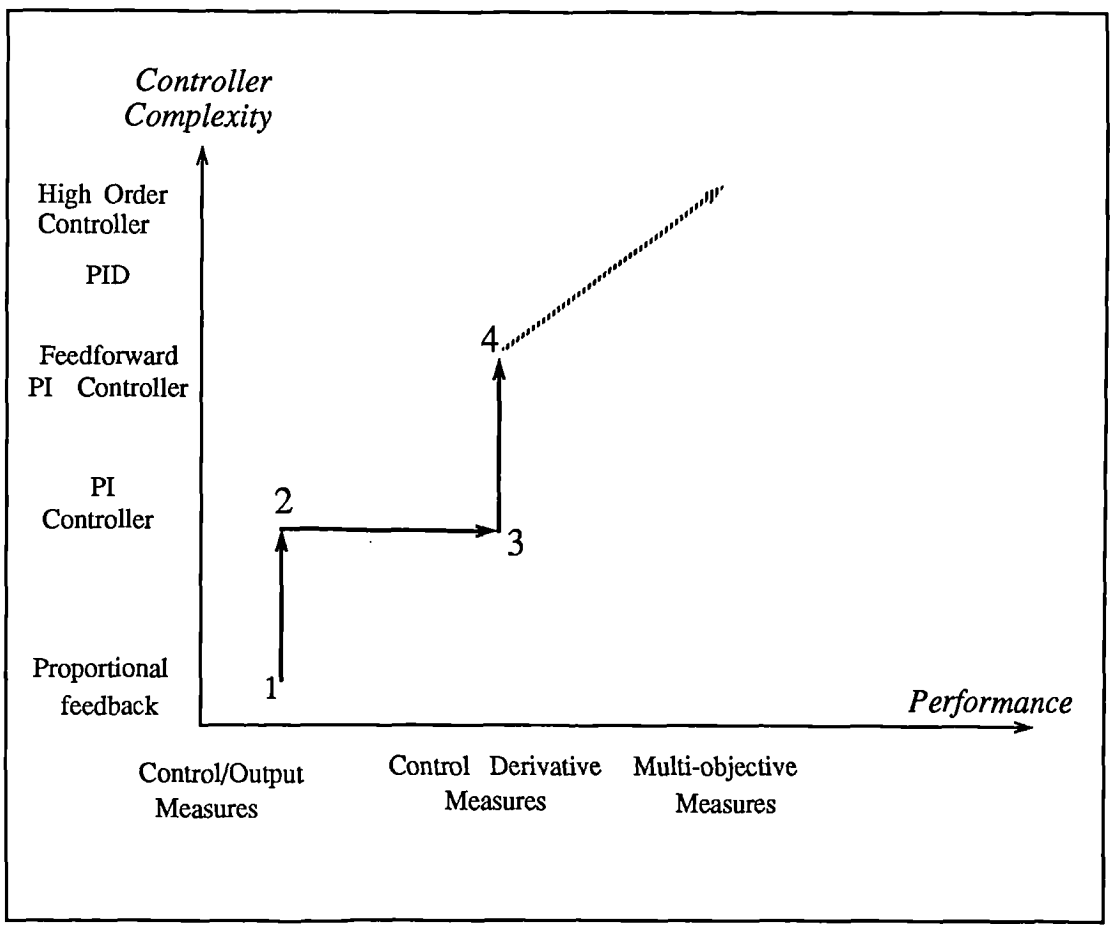

Fig. 3.12 Evolutionary Design Phases

Phase 1 Output Feedback Controller

We consider initially an output feedback controller shown in Fig. 3.13 below.

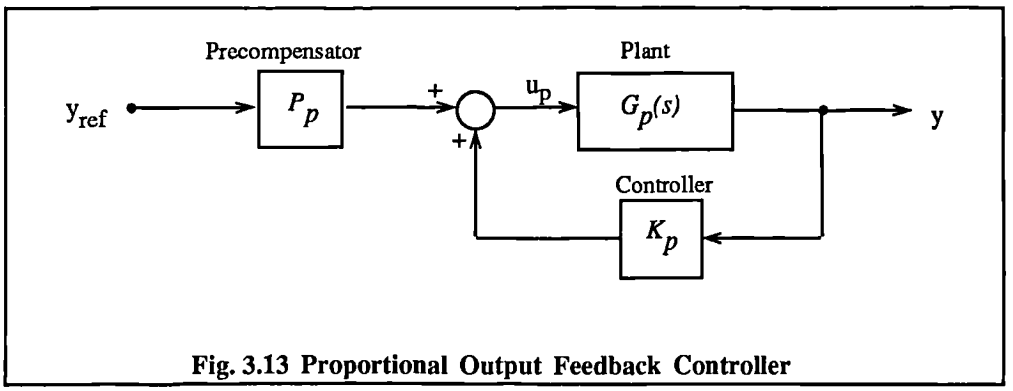

Fig. 3.13 Proportional Output Feedback Controller

The precompensator P_p *is* required in order to make the closed loop DC Gain equal to unity in order that the output tracks the input. Using a cost function of the form:

$$J = \int_0^\infty (y_{ref}-y)^2 + (u_{ref}-u_p)^2 dt, \tag{3.71}$$

where y_{ref} was taken to be a unit step response and u_{ref} is arbitrary to ensure the integral is finite. Starting at a value of K_p=0 P_p=1, a controller of K_p=-0.1973, P_p=1.1973 was found after 8 function evaluations and 3 gradient evaluations giving J=1.3001. The output and step responses are shown in Fig 3.14 below.

It should be noted that while this controller appears to give a good output response, it demands that the actuators reach a level of 1.1973 at t=0. Obviously in this situation such a demand would cause actuator saturation which might cause unwanted non-linear effects. Further, the tracking depends on the model of the plant being totally accurate since any deviation of the plant or controller parameters will result in a loss of tracking. A better control regime is one that uses integral action.

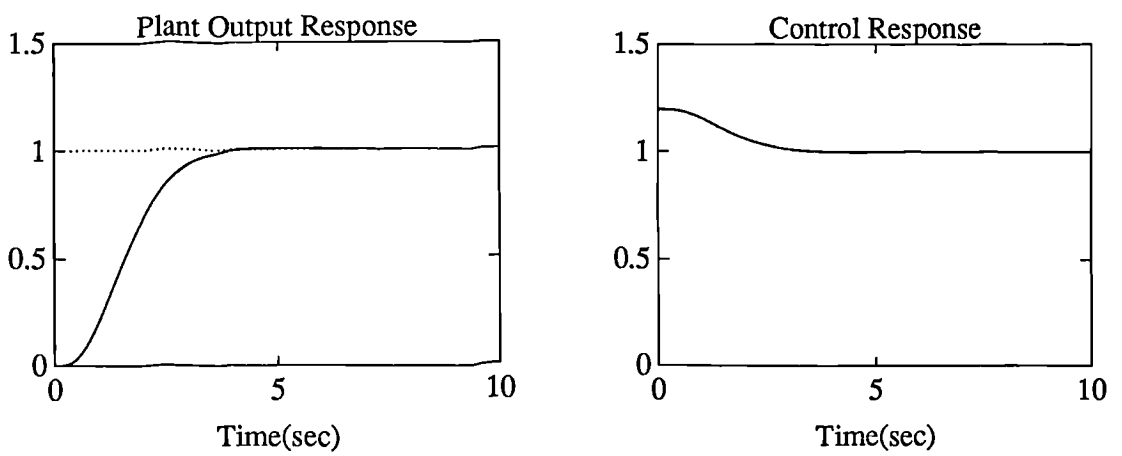

Fig. 3.14 Step Response for Output Feedback

Design 2 PI Controller

We now consider a PI controller shown in Fig. 3.15 below.

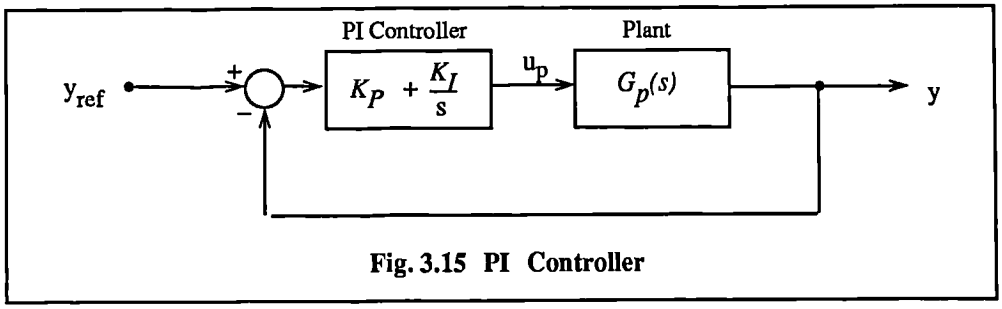

Fig. 3.15 PI Controller

The proportional and integral parameters K_P and K_I where found in order to minimize the cost function in Eq. 3.71. Since the integral action has the effect of adding an open loop pole at the origin, it was necessary to find a stabilizing controller before the optimization could commence. This was achieved by using a minimax optimization routine in which the eigenvalues of the closed system are moved to a sufficient distance into the left half plane. A stabilizing controller was found to be K_P=-0.2644, K_I=0.2807 giving J=4.747. The optimal controller was found after 13 function evaluations and 5 gradient evaluations to be K_P=-1.7746, K_I =1.1973 and J=2.175. The step response is shown as the solid line in Fig. 3.16.

Design 3 Derivative Measures

We can see how the effect of integral control puts less demands on the actuators than that of Design 1. A further measure which can be used to stop actuator rate saturation is to add derivative control measures to the cost function. Consider a cost function of the form

$$J = \int_0^\infty (y_{ref}-y)^2 + (u_{ref}-u_p)^2 + \dot{u}^2 \, dt \tag{3.72}$$

An optimal controller for this cost function was found after 14 function evaluations and 5 gradient evaluations to be K_P=-1.1290 and K_I =0.7804 with J=3.1645. The step response is shown as the dashed line in Fig. 3.16. We see how this response places less demand on the actuator rates although there is a trade-off in the speed of response.

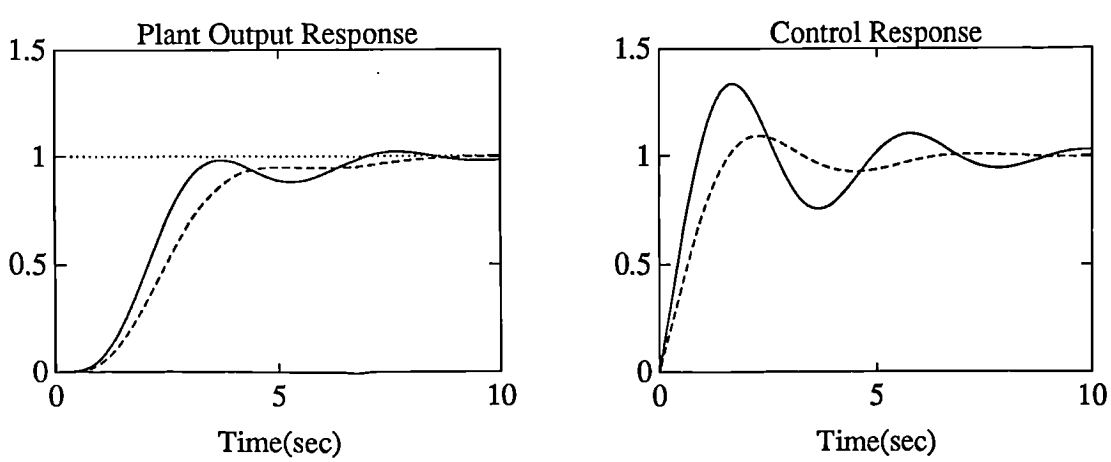

Fig. 3.16 Step Responses For Two PI Controllers

Design 4 Feedforward Controller

We now consider a feedforward PI controller of the form:

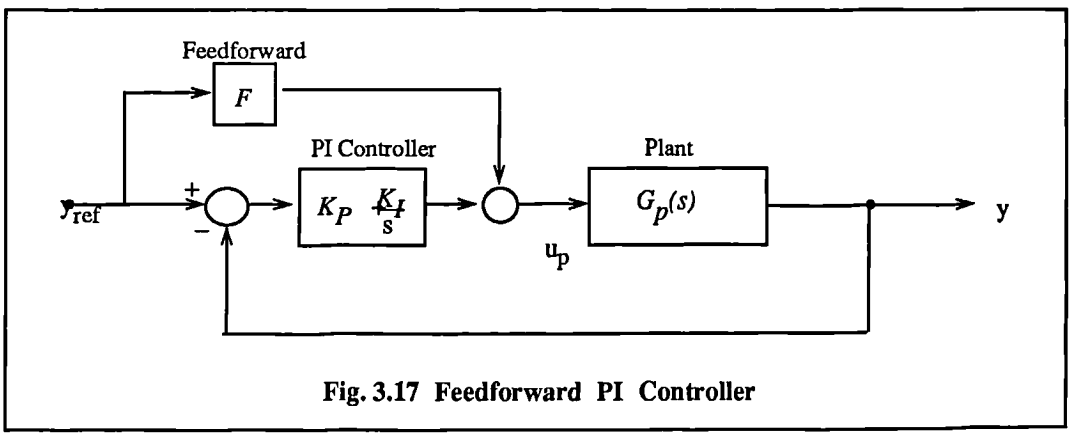

Fig. 3.17 Feedforward PI Controller

The optimal control values starting with F=0 and the K_P=-1.7746 and K_I =1.1973 for the two cost function were found to be :

Cost function 1 (Eq. 3.71): K_P=-0.24551, K_I =2.1516e-4, F=1.2452 J=1.2815 after 122 function evaluations and 60 gradient evaluations this is shown as the solid line in Fig. 3.18. Note, this design is approaching that of the output feedback design. This causes the slow convergence in the optimization routine as the pole corresponding to the integral action moves towards the origin, creating a marginally stable system. This stresses the importance of using derivative control measures or by ensuring lower bounds on the variable K_I.

Cost function 2 (Eq. 3.73): K_P=-0.53387, K_I =0.41699 F=0.66392 J=2.1006 after 17 function evaluations and 6 gradient evaluations this is shown as the dashed line in Fig. 3.18.

Although this controller demands that the actuators react quickly initially, it does not have the disadvantage of the output feedback controller in that y_{ref} is always tracked even in the presence of model uncertainties and non-linear effects.

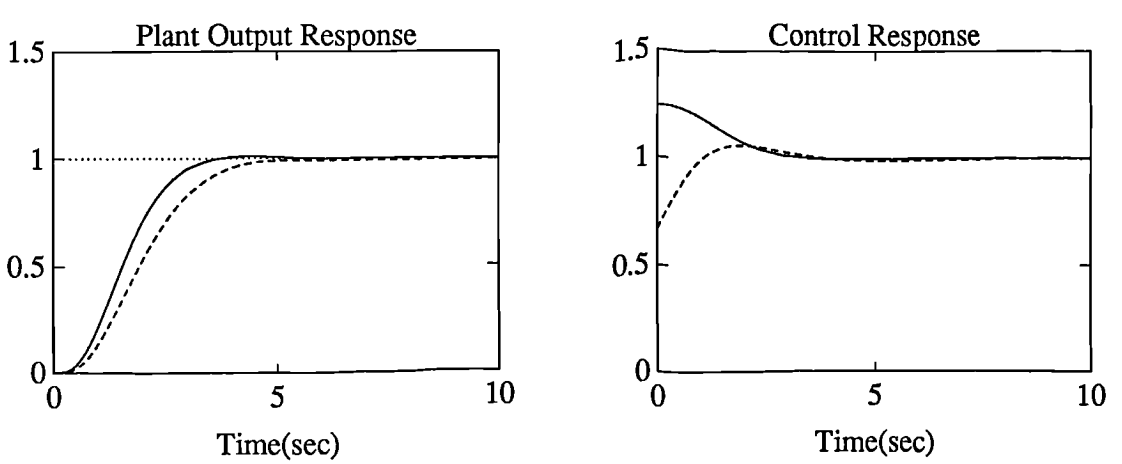

Fig. 3.18 Step Responses For 2DF PI Controllers

Designs 5-7 Two-Degree-of Freedom(2DF) Controllers using Multi-objective Optimization

In the case of the two-degree-of-freedom control structure the input reference, y_{ref} , is fed forward to both the controller as well as directly to the plant input. The control structure as depicted in Fig. 3.9 is used. A multi-objective design approach using a vector of objectives, f, is used. The objectives are follows:

$$f_1 = \int_0^\infty (y_{ref}-y)^2 dt \quad f_2 = \int_0^\infty (u_{ref}-u_p)^2 dt \quad f_3 = \int_0^\infty \dot{u}^2 \, dt \tag{3.73}$$

For higher-order controllers we consider relaxing the control energy terms ($f_2 f_3$) in order to achieve a faster speed of response. We therefore set the goals f_1*=1, f_2*=10, f_3*=10. In order to achieve the same percentage under or over achievement of the active objectives we set the weights w_1=1, w_2=10, w_3=10 (see eqn.(2.54) resulting in the following NP problem

Goal Attainment Formulation

$$\begin{aligned} &\underset{\gamma,K}{\text{minimize}} \ \gamma \\ \text{subject to:} \quad & f_1(K) - \gamma \le 1 \\ & f_2(K) - 10\gamma \le 10 \\ & f_3(K) - 10\gamma \le 10 \end{aligned} \tag{3.74}$$

In order to stop the integral term, C_i tending to zero it was bounded with the constraint $C_i \ge 0.1$. A 2DF PID controller gave a solution of f_1=0.8871, f_2=1.4691, f_3=8.8710 with γ=-0.1129 indicating that at least an 11.29% improvement of the original goals was achieved. The active constraints were reported to be f_1 and f_3 showing the restriction on faster control is the derivative control energy term. The solid line in Fig. 3.19 shows the time response for the 2DF PID controller.

Using the evolutionary mapping technique and canonical form described in Sections 3.5.5 and 3.5.6 higher-order controllers were designed. Only very small improvements were achieved for each increment in control order. For instance, a fourth order controller gave a result of f_1=0.8735, f_2=1.5528, f_3=8.7352 with γ=-0.1269 and superimposed as the dashed line in Fig. 3.19. Only 1.4% improvement was obtained using the fourth order controller over the PID controller, this is due to the restriction of control derivative energy and also because the plant is non-oscillatory and of relatively low-order. The controller matrices are given in Appendix B.

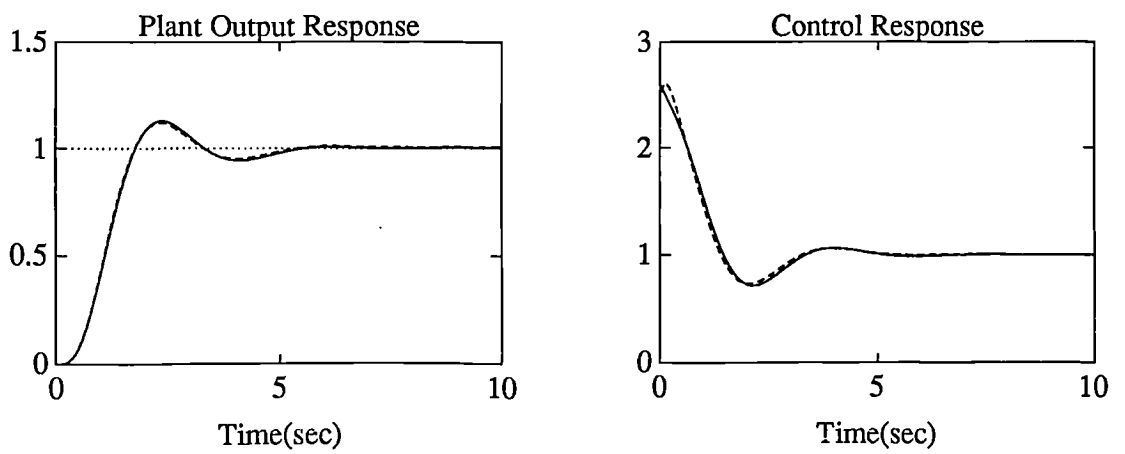

Fig. 3.19 Step Responses For 2DF PID and 2DF Fifth Order Controllers

3.10.2 Generic VSTOL Aircraft Model (GVAM) Tracking Example

In this example we consider a Generic Vertical Short Take-Off and Landing (VSTOL) Aircraft Model (GVAM) [49] supplied by RAE, Bedford. A linearized 10 state model is used which includes both the aircraft dynamics as well as the engine dynamics. A 200 knots flying operating point is considered in which the problem of tracking step demands in the upward velocity (VKD) and the airspeed (VTKT) is considered. The uncontrolled aircraft state space matrices are given in Appendix B.

Again we demonstrate an evolutionary design procedure with six design phases as shown in Fig. 3.20 Initially we consider the design of a PI controller with simple output and control measures. In later stages of this design a 2DF PID with partial state feedback is considered using multi-objective performance measures.

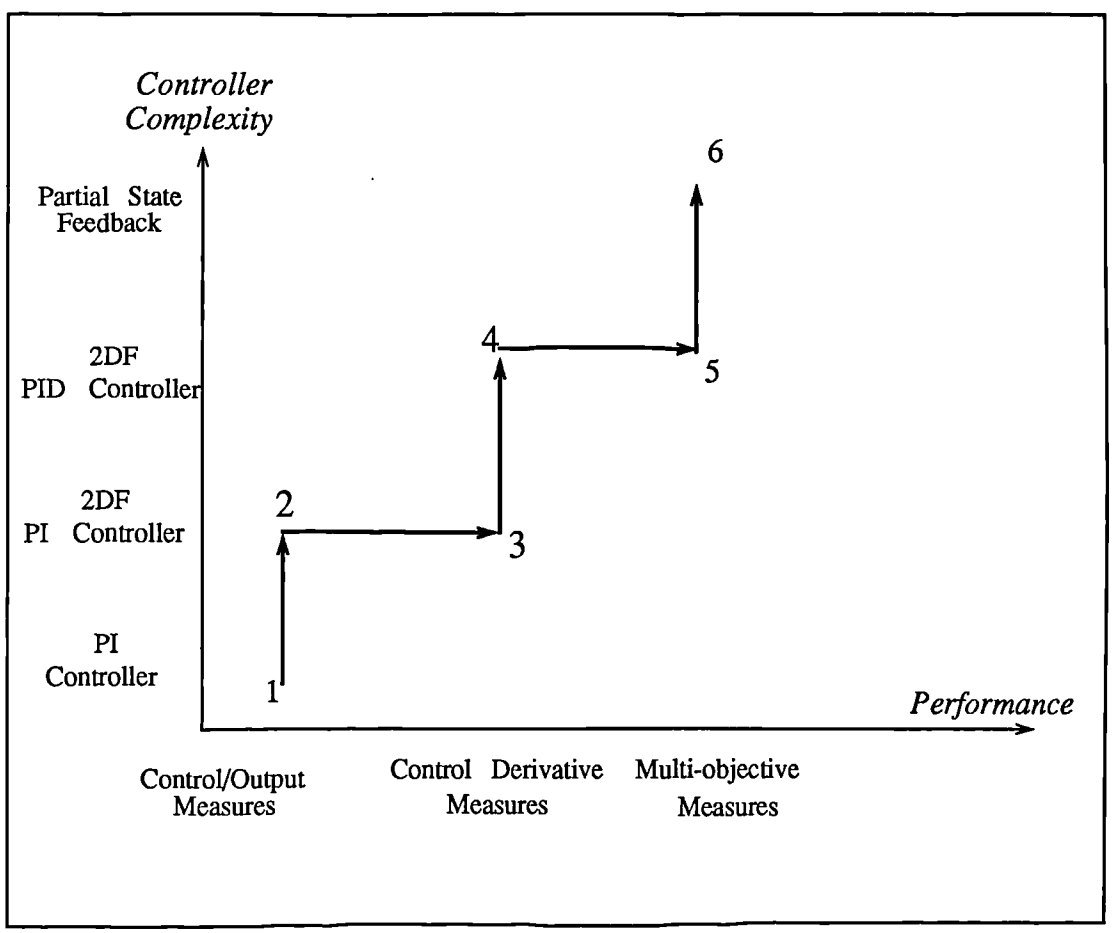

Fig. 3.20 Design Phases for GVAM Example

Design 1: PI Controller

In the first design phase a PI controller was designed using a cost function of the from J

$$J = J_1 + J_2 \tag{3.75}$$

where J_1 is the cost function for a step demand in VKD (y_1) and given by

$$J_1 = \int_0^\infty (1-y_1)^2 + (u_{ref1}-u_1)^2 + (y_2)^2 + (u_{ref2}-u_2)^2 \, dt \tag{3.76}$$

and J_2 is the cost function for a step demand in VTKT (y_2) and given by

$$J_2 = \int_0^\infty (y_2)^2 + (u_{ref1}-u_1)^2 + (1-y_2)^2 + (u_{ref2}-u_2)^2 \, dt \ , \tag{3.77}$$

where u_1 represents the longitudinal stick demand (ALONG) and u_2 represents the throttle demand (ATHROT).

A PI controller was designed giving J=3.2902 and the following control gains:

$$D_c = \begin{bmatrix} -0.4309 & -0.0326 \\ 0.0168 & 0.1286 \end{bmatrix} \qquad C_{ci} = \begin{bmatrix} 0.1904 & 0.0039 \\ -0.0162 & -0.0835 \end{bmatrix} \tag{3.78}$$

corresponding to the control structure in Fig. 3.9.

Fig. 3.21 shows the step responses for two step demands in VKD and VTKT. The left hand graphs are the output responses for a step demand in VKD, while the right hand graphs are for a step demand in VTKT. The solid line in the top two graphs represents the output being controlled, while the dashed line represents the output for which it is desired to remove interaction. For the control response the solid line in the left hand graph is ALONG while the dashed line is ATHROT (and vice-versa for the right hand graph).

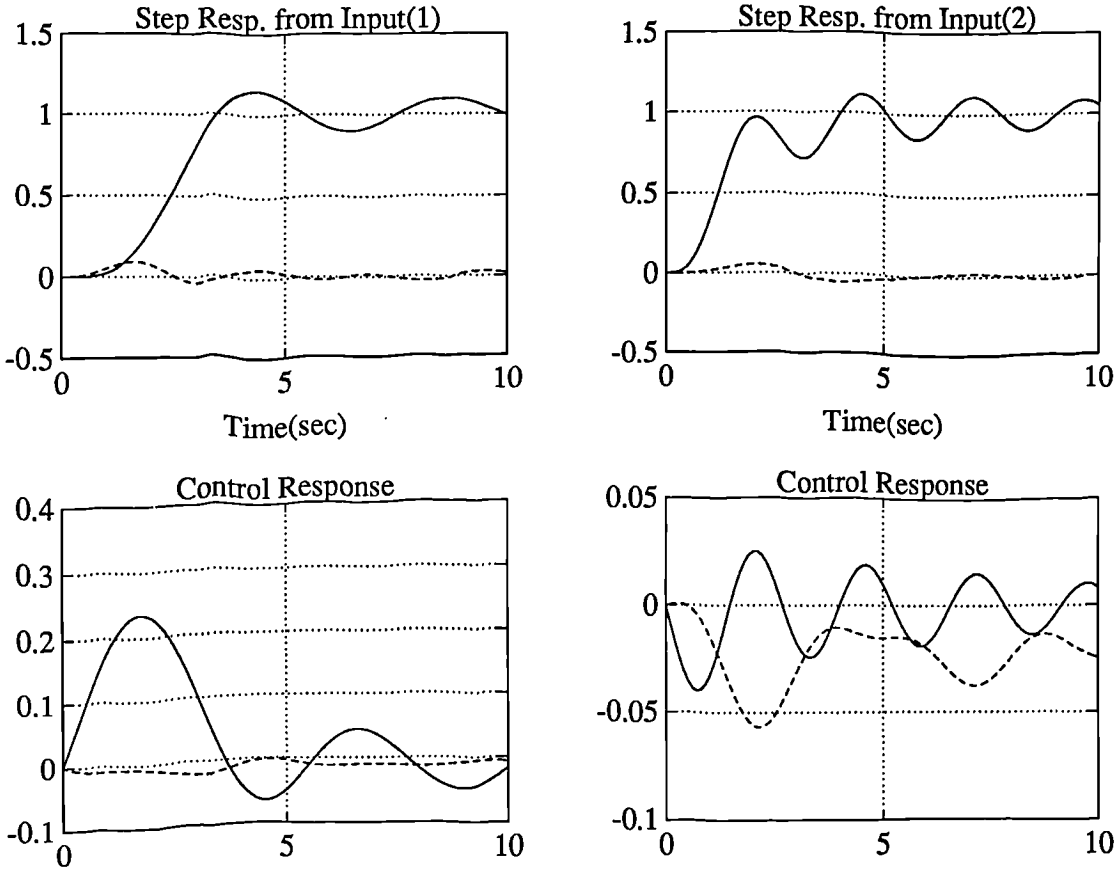

Fig. 3.21 GVAM Responses Using PI Controller

Design 2: Feedforward(2DF) PI Controller

In the second phase of the control design a feedforward(2DF) PI controller was designed. Using the control structure in Fig. 3.9 a controller was designed for the cost function (Eq. 3.75). A solution was found giving J=2.113 and the controller gain matrices equal to:

$$D_c=\begin{bmatrix} -0.3236 & 0.0020 \\ 0.0337 & 0.0890 \end{bmatrix} \quad C_{ci}=\begin{bmatrix} -0.0121 & -0.1021 \\ -0.0276 & -0.0039 \end{bmatrix} \quad D_{cff}=\begin{bmatrix} 0.3313 & 0.0069 \\ -0.0009 & -0.0892 \end{bmatrix} \tag{3.79}$$

It is clear that the addition of feedforward considerably reduces the value of the cost function. The step responses are plotted in Fig. 3.22. We see that the addition of feedforward has the effect of increasing the speed of response. The trade-off is in the actuator rate demands. In the next design we penalize the rate demands by adding a control derivative term to the cost function.

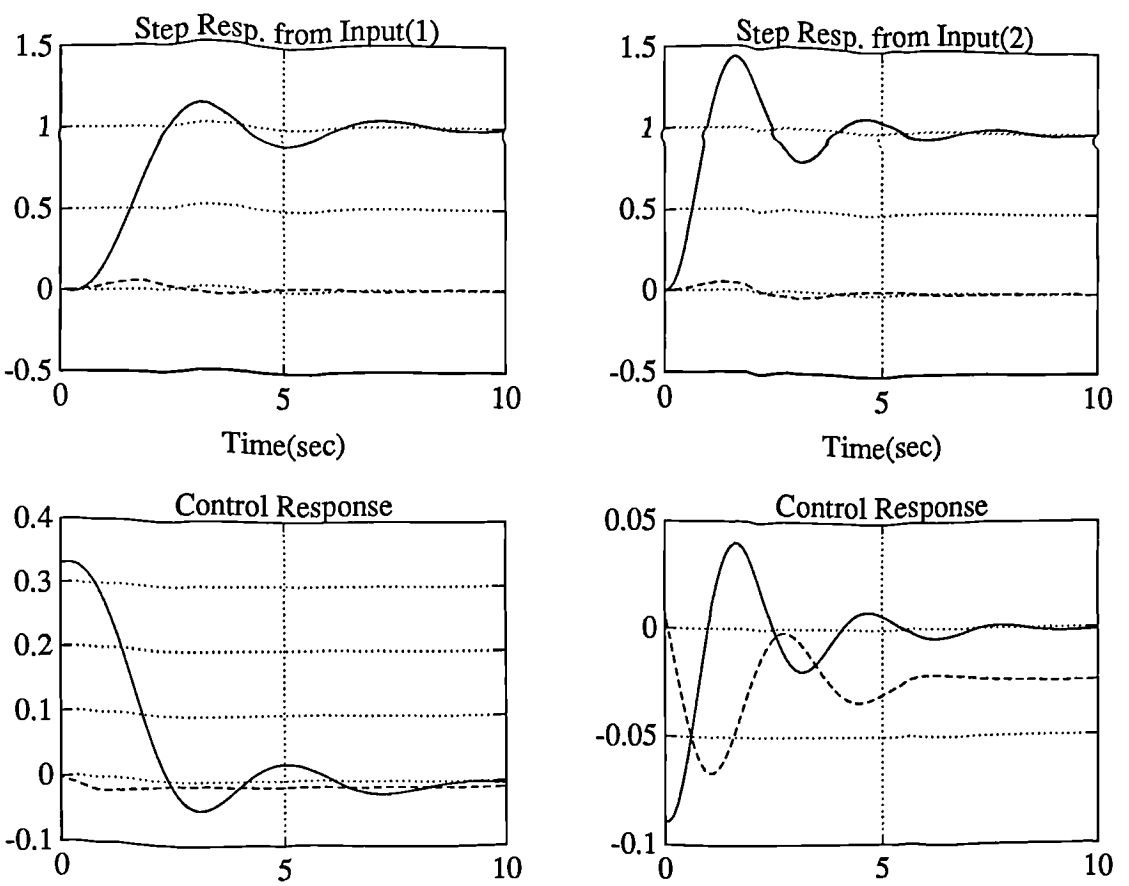

Fig. 3.22 GVAM Responses Using 2DF PI Controller

Design 3: Derivative Control Measures

In order to minimize actuator rate saturation we include the terms

$$J_d = \int_0^\infty (\dot{u}_1)^2 + (\dot{u}_2)^2 \, dt \tag{3.80}$$

to both J_1 in Eq. 3.75 and J_2 in Eq. 3.76. Using the same 2DF PI control structure as in Design 2 a controller was designed giving J=2.2692. The control gains were found to be as follows

$$D_c = \begin{bmatrix} -0.3244 & 0.0228 \\ 0.0300 & 0.0806 \end{bmatrix} \quad C_{ci} = \begin{bmatrix} 0.0272 & -0.0847 \\ -0.0209 & 0.0095 \end{bmatrix} \quad D_{cff} = \begin{bmatrix} 0.2668 & -0.0151 \\ -0.0024 & -0.0847 \end{bmatrix} \tag{3.81}$$

The step responses are plotted in Fig. 3.23. There is less actuator rate demand although only a marginal difference in the step response characteristics.

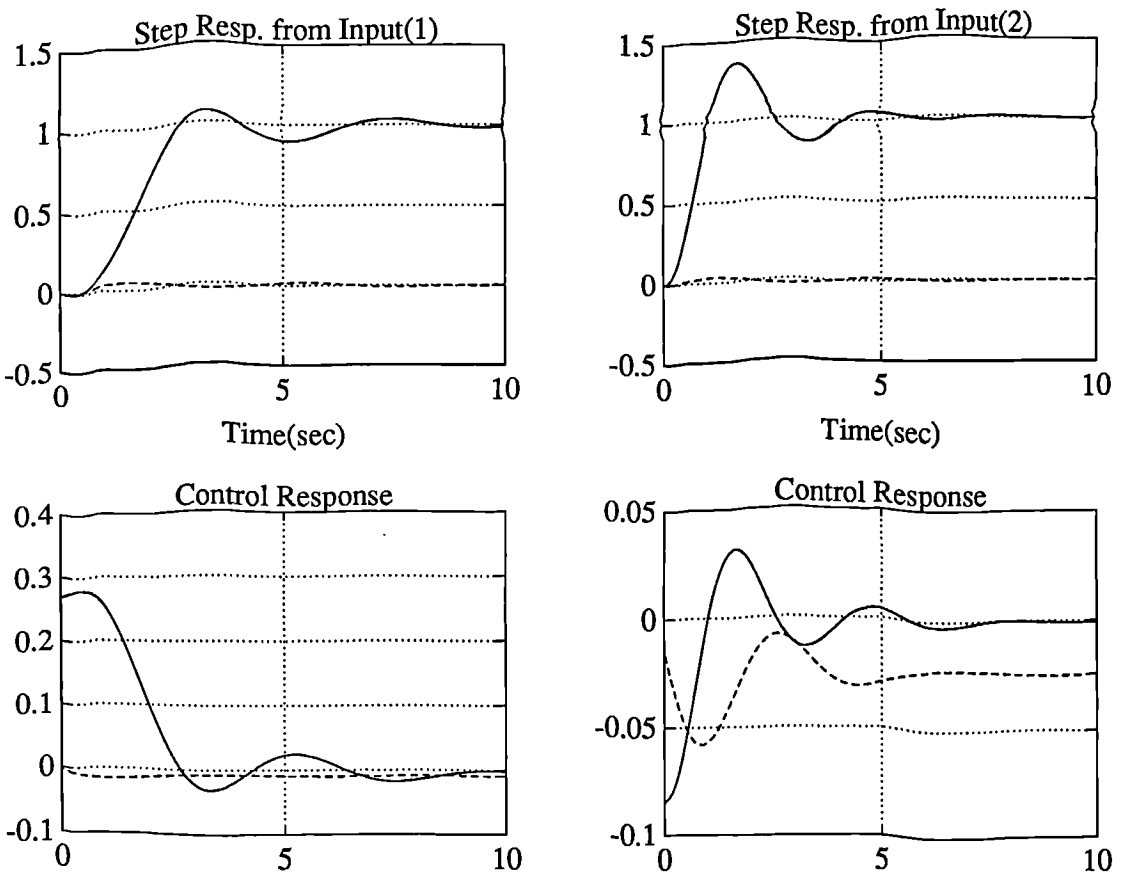

Fig. 3.23 GVAM Responses Using 2DF PI Controller and Derivative Control Measures

Design 4: Feedforward(2DF) PID Controller

By incrementing the control order using the mapping technique described in Section 3.5.6 a feedforward(2DF) PID controller was designed. The solution was found to occur giving J=2.253 and the controller gain matrices equal to:

$$D_c=\begin{bmatrix} -0.3149 & 0.0150 \\ 0.0297 & 0.0824 \end{bmatrix} \quad C_{ci}=\begin{bmatrix} 0.0345 & -0.0762 \\ -0.0236 & 0.0132 \end{bmatrix} \quad D_{cff}=\begin{bmatrix} 0.2572 & -0.0148 \\ -0.0005 & -0.0870 \end{bmatrix} \tag{3.82}$$

$$A_c=[\,-0.3249\,] \quad C_c=\begin{bmatrix} -0.0063 \\ 0.0006 \end{bmatrix} \quad B_{cff}=[\,-0.0039 \quad -0.0005\,] \quad B_{cff}=[\,-0.0039 \quad -0.0005\,]$$

The step responses are plotted in Fig. 3.24. The effect of increasing the order of the controller only marginally improves the cost function. This is because of the restriction in the control derivative measures. In the next design we consider relaxing the control constraints by considering a multi-objective formulation. In this design each of the eight responses in Fig. 3.24 are treated independently.

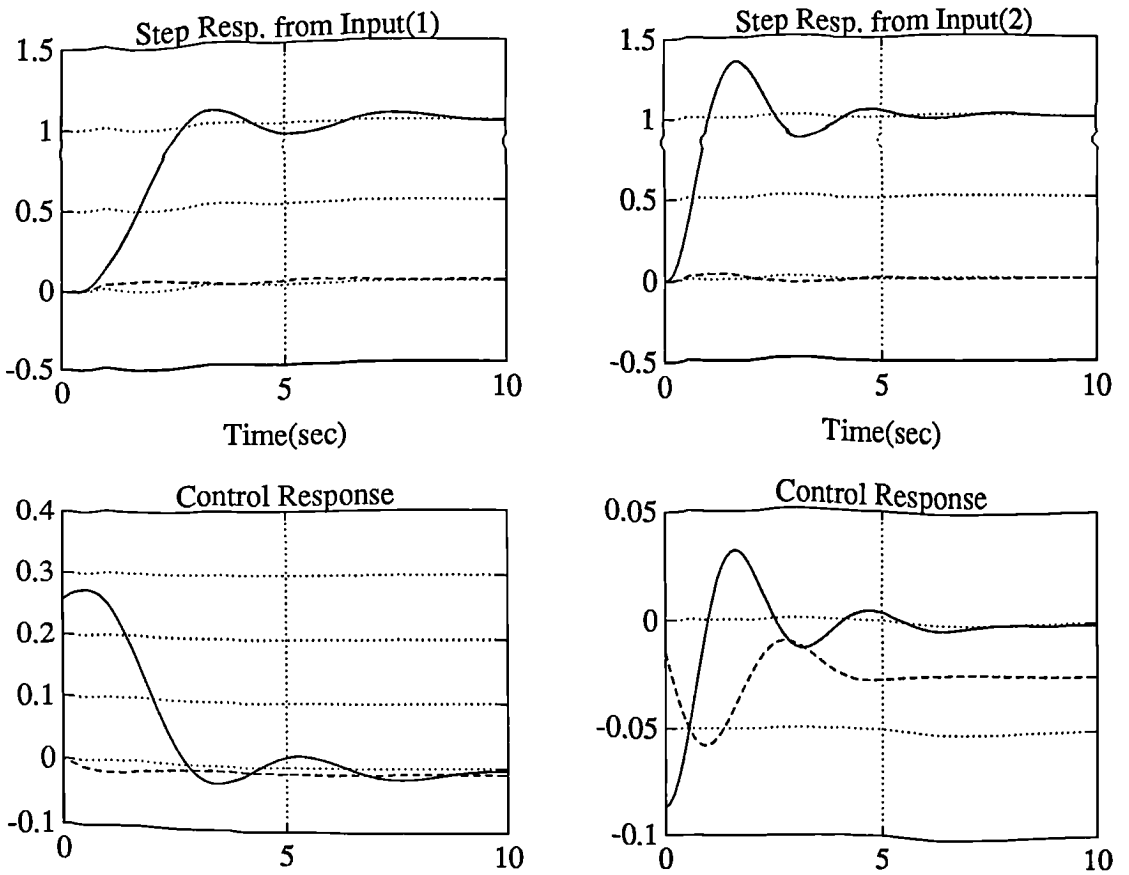

Fig. 3.24 GVAM Responses Using 2DF PID Controller

Design 5: Multi-objective Design using Feedforward(2DF) PI Controller

Each of the control objectives will now be considered independently using a Goal Attainment formulation. We use vector of objectives, f, consisting of:

$$f_1 = \int_0^\infty (y_{ref}-y)^2 dt \quad f_2 = \int_0^\infty (u_{ref}-u_p)^2 dt \quad f_3 = \int_0^\infty \dot{u}^2 \, dt \tag{3.83}$$

The objectives are used on the four input/output combinations, giving a vector of 12 objectives. For demonstration purposes we set the goal vector f^* all equal to ones and the weighing vector w all equal to ones. This is equivalent to a minimax problem. The problem formulation is shown in (3.84). The results are tabulated in Table 3.1 where the response numbers 1 and 2 are for step demands in VKD and VTKT respectively.

Goal Attainment Formulation

$$\begin{aligned} &\underset{\gamma,K}{\text{minimize}} \\ &\text{subject to: } f_1(K) - \gamma \le 1 \\ &\qquad f_2(K) - \gamma \le 1 \\ &\qquad \vdots \\ &\qquad f_{12}(K) - \gamma \le 1 \end{aligned} \tag{3.84}$$

Table 3.1

PI → / Resp. No. ↓	Output f_1	Control f_2	Der. Control f_3
1 VKD DEMAND	VKD **1.0600**	ALONG 0.2985	ALONG **1.0600**
	VTKT 0.9864	ATHROT 0.0217	ATHROT **1.0600**
2 VTKT DEMAND	VTKT **1.0600**	ATHROT 0.0223	ATHROT 0.5878
	VKD 0.4636	ALONG 0.0940	ALONG **1.0600**

The value of γ at the solution was γ=0.060 indicating a 6% under-achievement in the active objectives. The active objectives are emboldened in Table 3.1 and are the barriers to further improvement in the objectives. Step responses are plotted below in Fig. 3.25.

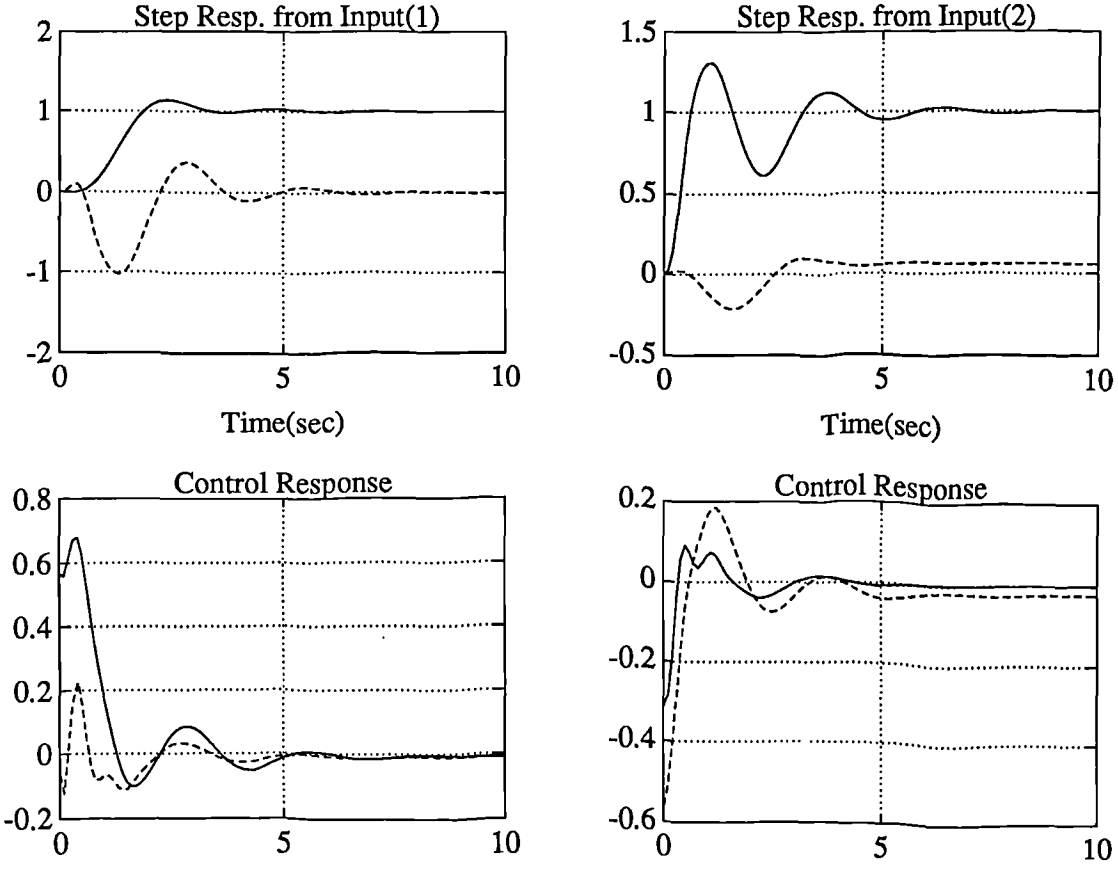

Fig. 3.25 GVAM Responses Using 2DF PID Controller

The 2DF PID controller was found to be

$$D_c=\begin{bmatrix} 65.5402 & 6.6777 \\ 243.879 & 23.403 \end{bmatrix} \quad C_{ci}=\begin{bmatrix} -0.0160 & -0.2362 \\ -0.1858 & -2.5823 \end{bmatrix} \quad D_{cff}=\begin{bmatrix} 0.5652 & -0.5679 \\ -0.0573 & -0.3145 \end{bmatrix}$$

$$A_c=[\,-138.7459\,] \quad C_c=\begin{bmatrix} -16.2992 \\ -59.7450 \end{bmatrix} \quad B_c=[\,561.1617 \quad 51.0555\,] \quad B_{cff}=[\,-0.1212 \quad -0.0051\,] \tag{3.85}$$

Design 6: Partial State Feedback

It is often the case that a number of states (although not all) are available for feedback. In this example we consider the possibility of feeding back pitch rate (state 2) and the engine fan speed (state 8). Using the same objectives as for Design 5 a controller was designed giving the objectives values shown in Table 3.2 and the control gain matrices (Eq. 3.86)

Table 3.2

PI → / Resp. No. ↓	Output f_1	Control f_2	Der. Control f_3
1 VKD DEMAND	VKD **1.0399**	ALONG 0.2655	ALONG **1.0399**
	VTKT 0.6144	ATHROT 0.0096	ATHROT 0.7087
2 VTKT DEMAND	VTKT 0.0676	ATHROT 0.0122	ATHROT 0.3059
	VKD 0.5830	ALONG 0.0101	ALONG 0.9992

$$D_c=\begin{bmatrix} -0.3149 & 0.0150 \\ 0.0297 & 0.0824 \end{bmatrix} \quad C_{ci}=\begin{bmatrix} 0.0345 & -0.0762 \\ -0.0236 & 0.0132 \end{bmatrix}$$

$$D_{cff}=\begin{bmatrix} 65.7097 & 7.0398 & -0.8281 & 1.0667 \\ 243.7259 & 24.1664 & -0.2819 & 0.6452 \end{bmatrix}$$

$$A_c=[\,-138.4329]$$

$$B_c=[\,561.6161 \quad 50.3773 \quad 0.2279 \quad -0.3759\,]$$

$$B_{cff}=[\,-0.1516 \quad -0.1053\,] \quad C_c=\begin{bmatrix} -0.0063 \\ 0.0006 \end{bmatrix} \tag{3.86}$$

The results indicate a 3.99% under-achievement for the active objectives (emboldened in Table 3.2) over the original goals. The step responses are shown in Fig. 2.26 showing a faster response speed of response and less actuator demands over using only output feedback.

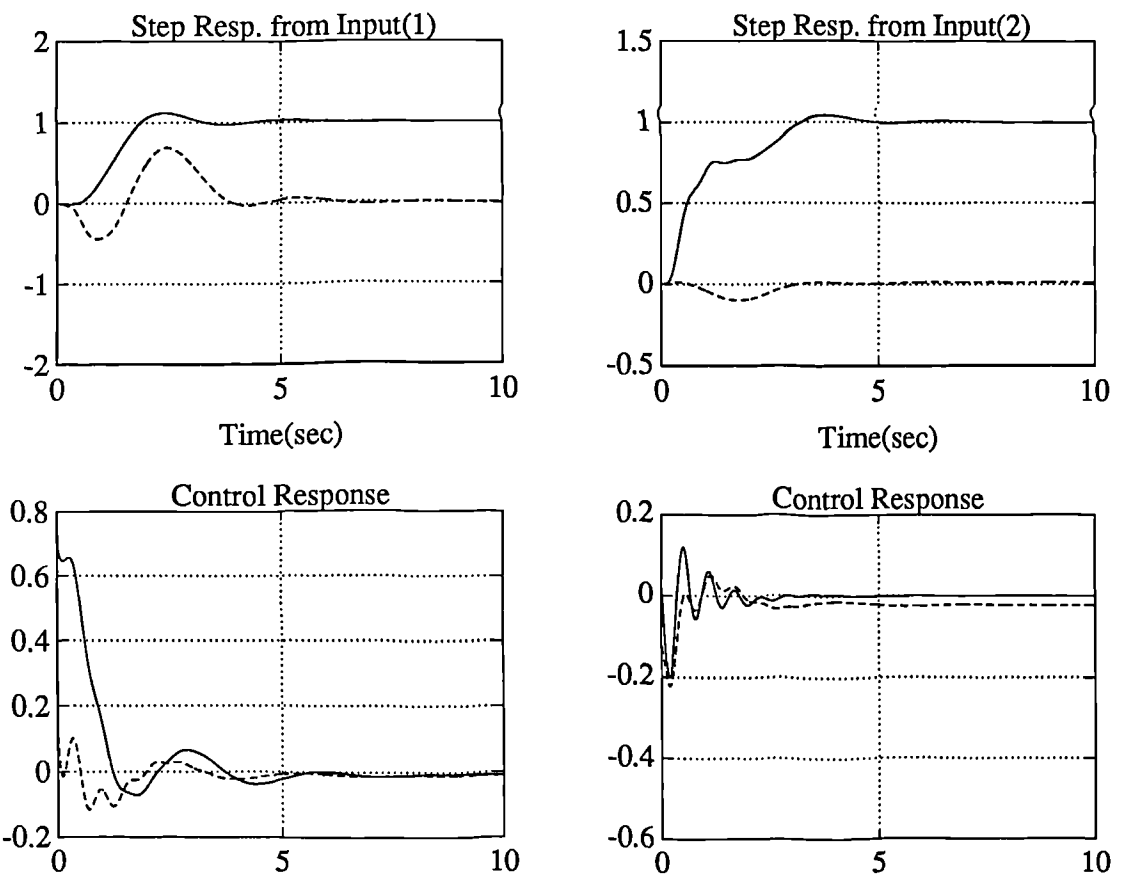

Fig. 3.26 GVAM Responses Using 2DF PID Controller

3.10.3 F4C Multi-Model Example

In this example we consider the control of the longitudinal motion of a McDonnell-Douglas F-4C fighter aircraft as described in Heffley and Jewel[50] and considered by Kreisselmeier and Steinhauser ([51] using a vector performance index. We consider designing a single fixed-gain controller for four extremal flight conditions. The longitudinal motion of the aircraft is modelled by a third order system of the form:

$$\frac{d}{dt}\begin{bmatrix}\dot{\Theta}\\ \alpha\\ \eta\end{bmatrix} = \begin{bmatrix}a_{11} & a_{12} & b_1\\ a_{21} & a_{22} & b_2\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\dot{\Theta}\\ \alpha\\ \eta\end{bmatrix} + \begin{bmatrix}0\\ 0\\ 20\end{bmatrix}\eta_c \tag{3.87}$$

where $\dot{\Theta}$ is the pitch rate, α is the (incremental) angle of attack,
η is the (incremental) elevator deflection
and the input variable is η_c (incremental elevator command).

The model represents a short period motion description of the aircraft plus a first order actuator system. The variable that we will consider controlling in this example is the pitch rate, $\dot{\Theta}$. Fig. 3.27 shows the flight envelope for the aircraft and the four operating points taken at different combinations of altitude and velocity. The aircraft data for a_{ij}, b_i, are given in Appendix B :

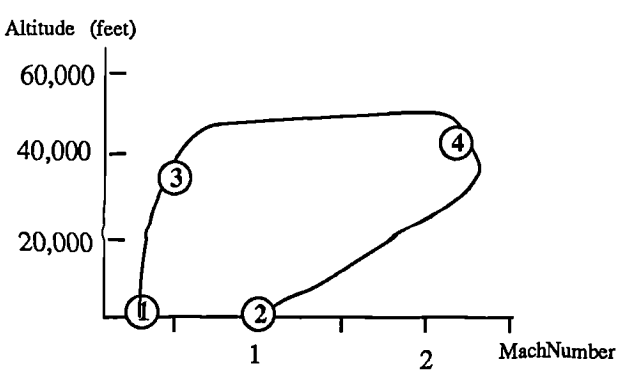

Fig. 3.27 Flight envelope of the McDonell-Douglas F-4C Phantom

We consider designing a 2DF controller with integral action as depicted in Fig. 3.9. Three performance objectives for each of the four flight conditions were used of the form

$$f_1 = \int_0^\infty (\dot{\Theta}_{ref} - \dot{\Theta})^2 dt \quad f_2 = \int_0^\infty (\eta_{c_{ref}} - \eta_c)^2 dt \quad f_3 = \int_0^\infty \dot{\eta}_c^{\,2}\, dt \tag{3.88}$$

where f_1 is a measure of the speed of response, f_2 is measure of actuator magnitude demands and f_3 is a measure of actuator rate demands. Initially we design a controller using a scalar weighted sum of the objectives for flight condition 1.

Design 1- Single Objective PI Control

Problem: minimize{ $J = f_1 + \frac{1}{40} f_2 + \frac{1}{30} f_3$ } for flight condition 1. The weights were chosen to reflect subsequent weights to be used in the multi-objective design.

Table 3.3

PI → Op. Point ↓	Output f_1	Control f_2	Der. Control f_3	Key to Fig.3.24
①	0.4220	0.2827	3.8154	———
②	0.2763	0.2446	0.7563	----------
③	0.3094	3.3487	1.7534	.....
④	0.5217	**91.2583**	5.2523	. - . - .. .. ..

A solution to this problem was found giving J==0.5563 and the controller gains:

$$D_c=2.1594 \qquad C_{ci}=-4.6988 \tag{3.89}$$

corresponding to the control configuration in Fig. 3.9. The cost functions are displayed for the other flight conditions in Table 3.3. The step responses are shown in Fig. 3.28 corresponding to the key given in Table 3.3. While this design appears to give reasonable output time responses, it requires excessive elevator demand for flight condition 4 (emboldened in Table 3.3). Such a demand will cause actuator saturation which will lead to non-linear effects, integral wind-up and possible instability. We therefore move to a multi-objective design approach in which the objectives (in Eq. 3.77 are treated as part of a Goal Attainment formulation. .

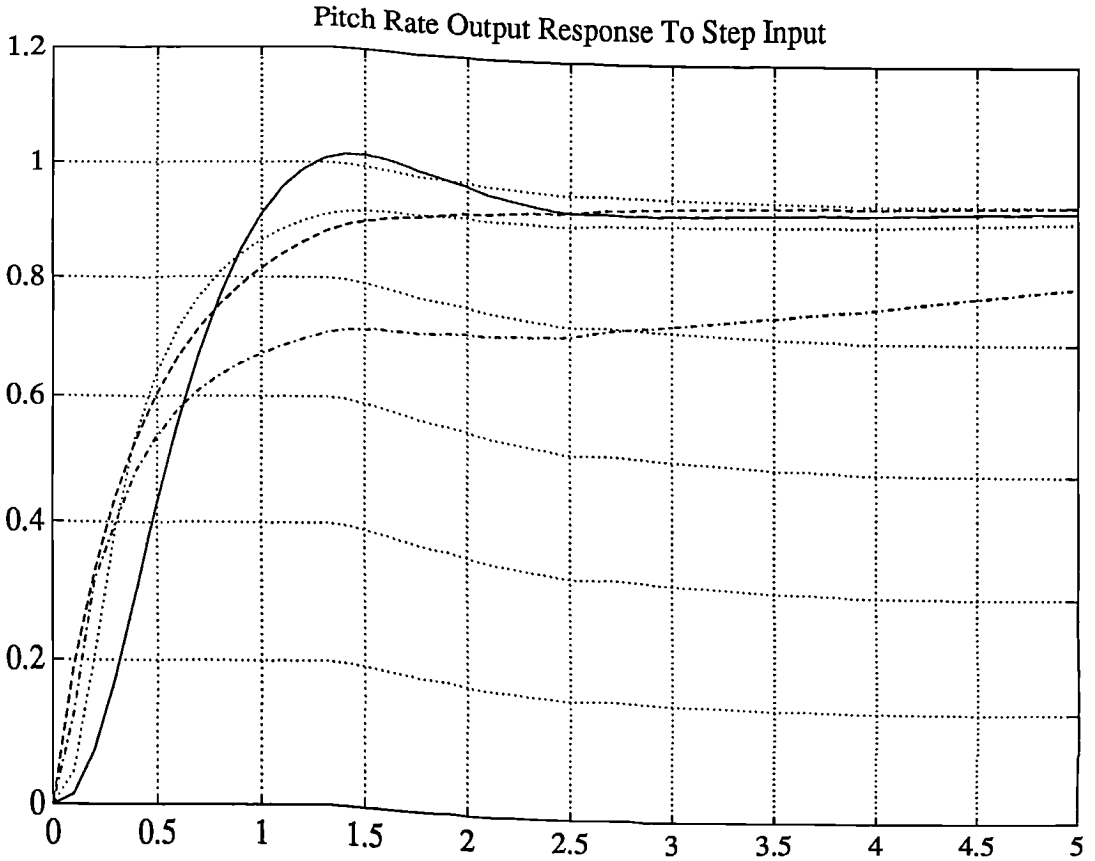

Fig. 3.28 Output Responses for PI Controller

Design 2- Multi-objective PI Design

For each of the four flight conditions we set the goals f_1*=1, f_2*=40, f_3*=30 giving a total of 12 constraints. We set the weights w_1=1, w_2=40, w_3=30 for each flight condition to achieve the same percentage under- or over-achievement in the active objectives. The control gains were found as follows for a PI controller

$$D_c=2.7389 \qquad C_{ci}=-16.3120 \tag{3.89}$$

The results are shown in Table 3.4 for the 12 objectives

Table 3.4

PI → Op. Point ↓	Output f_1	Control f_2	Der. Control f_3
①	0.2462	1.2457	**47.4413**
②	0.0961	0.1365	8.4780
③	0.1382	2.7818	21.5464
④	0.1330	**63.2550**	17.9042

The multi-objective design has reduced the magnitude actuator saturation for flight condition 4. The trade-off is a degradation in the output response as shown in Fig. 3.29 and an increase in actuator rate demands for Design 1. The active objectives are emboldened in Table 3.4 . These are the barriers to further improvement in the goals. The optimization routine reported a value of γ=0.5814 (58% under-achievement in the active objectives over the original goals). In order to gain improvement in the objectives higher-order 2DF controllers were designed using the evolutionary mapping technique described in Section 3.5.6.

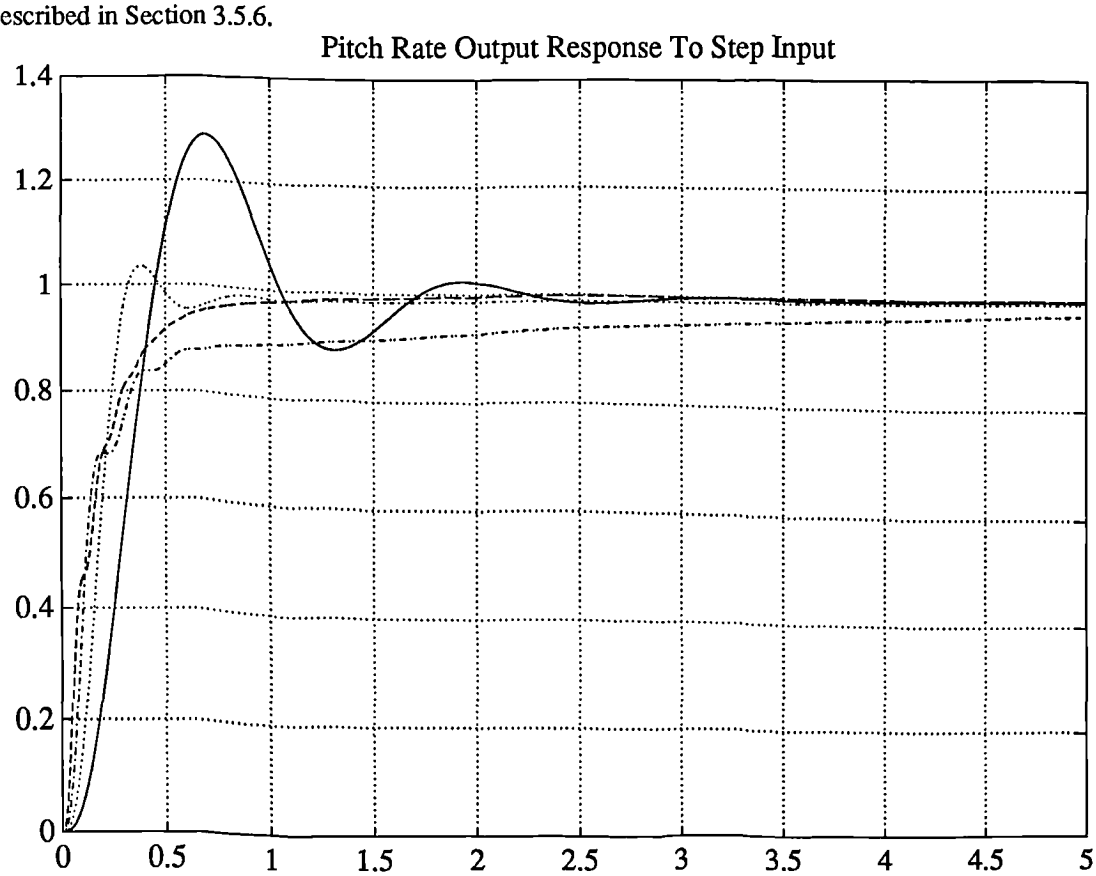

Fig. 3.29 Output Responses for MO Designed PI Controller

Design 1- Multi-objective 2DF High Order Controller Design

Using an evolutionary design procedure a third order controller with integral action was designed which gave the results shown in Table 3.5 and the output response as shown in Fig. 3.30

Table 3.5

PI → / Op. Point ↓	Output f_1	Control f_2	Der. Control f_3
①	**0.9452**	1.3913	**28.3572**
②	0.5818	0.2329	**28.3572**
③	0.6580	1.3241	5.4897
④	0.4082	**37.8096**	17.3380

The active objectives are emboldened in Table 3.5 indicating the barriers to further improvement in the objectives. The optimization gave a value of γ=-0.0548 showing that at least a 5.48% improvement has been achieved over the original goals. The controller gain matrices corresponding to Fig. 3.9 are:

$$A_c=\begin{bmatrix}0 & 0 & -11.4783\\ 1 & 0 & -107.3589\\ 0 & 1 & -20.2031\end{bmatrix} \quad B_c=\begin{bmatrix}1\\0\\0\end{bmatrix} \quad D_{cff}=-0.5469 \quad B_{cff}=\begin{bmatrix}-0.6844\\1.4632\\0.2469\end{bmatrix}$$

$$D_c=3.7731$$

$$C_{ci}=-59.5207 \qquad C_c=[-50.9844 \quad 11.8382 \quad -12.6933] \qquad (3.90)$$

Increasing the controller order again gave marginal improvement in the objectives (a 6% improvement over the original goals was achieved using a sixth order controller). The small improvement in performance in practice would not justify the use of such a high order controller.

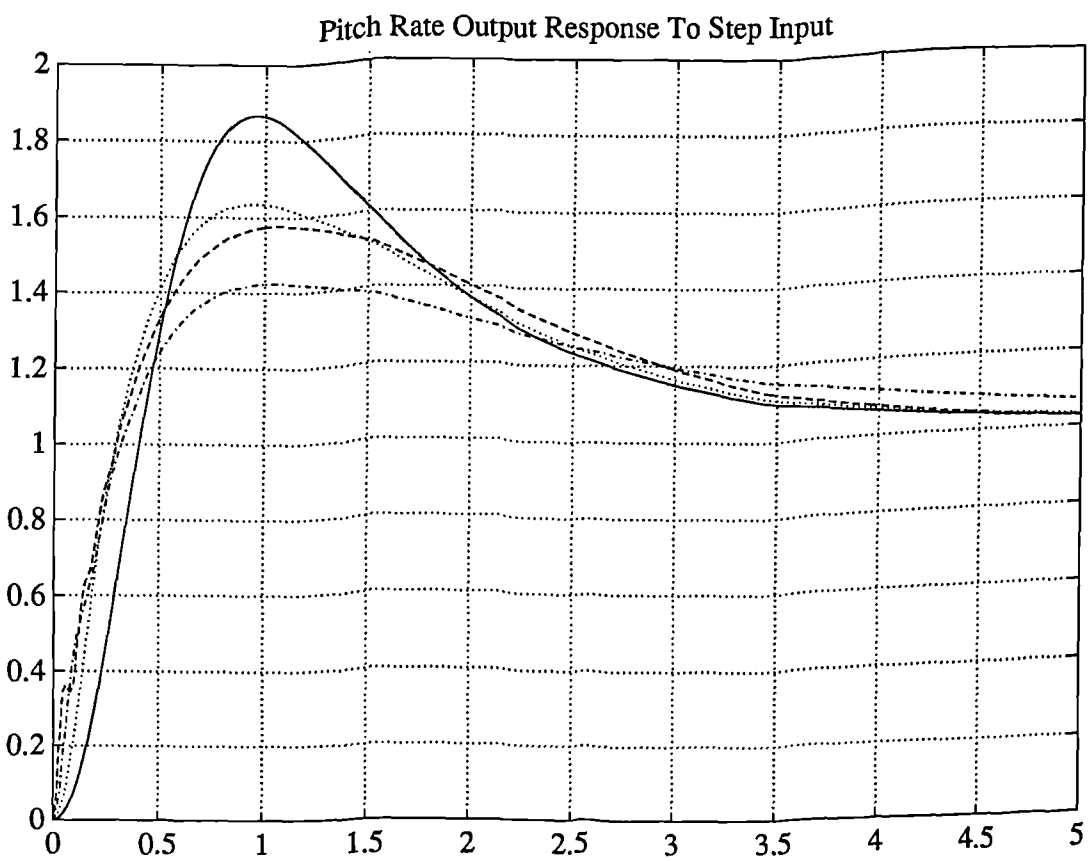

Fig. 3.30 Output Response for MO Designed 3rd Order 2DF I Controller

3.11 REVIEW

The aim of this part has been to present a control design methodology which can produce practical realizable controllers. In order to achieve this a theoretical basis has been established which incorporates a number of design options, disturbance types and controller configurations. The design of a 2DF controller with integral action has been focussed on. This has been used to design controllers for a number of different examples. For more efficient solution and better understanding of the problem an evolutionary design technique has been used. In the later stages of this process multi-objective optimization has been used to address problems such as reduction of interaction in multivariable systems, actuator rate limits and the design of fixed gain robust controllers for nonlinear systems using multiple operating points.

3.12 REFERENCES

General References

[1] Makila, P.M and Toivonen H.T. "Computational methods for parametric LQ problems - a survey," IEEE Trans. on Autom. Control, Vol.AC-32, No.8, 1987

[2] Anderson B.D.O and Moore J.B., "Linear optimal control," Prentice-Hall, 1971

[3] Patel R.V. and Munro N., "Multivariable system theory and design," Pergamon Press, International Seris on Control, Vol.4, 1982

[4] Friedland, B., "Control System Design: An introduction to state-space methods," McGraw-Hill, Series in Electrical Engineering, 1987

[5] Mac Farlane,A.G.J., "The calculation of the time and frequency response of a linear constant coefficient dynamical system," Quart. Journ. Mech. and Applied Math. , Vol.16, Pt.2, 1963

Output Feedback

[6] Levine, W.S. and Athans M.,"On the determination of the optimal constant output feedback gains for linear multivariable systems," IEEE Trans. on Autom. Control, Vol. AC-15, No.1, pp.44-48, 1970

[7] Kosut, R.L., "Suboptimal time-invariant systems subject to control structure constraints," IEEE Trans. on Autom. Control, Vol.AC-15, No.5, pp.557-563 1970

[8] Johnson, T.L. and Athans, M., "On the design of optimal constrained compensators for linear control systems," IEEE Trans. on Autom. Control, Vol. AC-15, No.1, pp.658-660, 1970

[9] Levine W.S., Johnson, T.L, and Athans, M., "Optimal limited state variable feedback for linear systems," IEEE Trans. on Autom. Control, Vol. AC-16, No.1, pp.785, 1971

[10] Basuthaker, S. and Knapp, C.H., "Optimal constant controllers for stochastic linear systems," IEEE Trans. on Autom. Control, Vol. AC-20, No.5, pp.664-666, 1975

[11] Wenk C.J. and Knapp C.H., "Parameter optimization in linear systems with arbitrarily constrained controller structure," IEEE Trans. on Autom. Control, Vol. AC-25, No.1, pp.44-48, 1980

[12] Horisberger H.P. and Belanger P.R., "Solution of the optimal constant output feedback problem by conjugate gradients," IEEE Trans. on Autom. Control, Vol. AC-19, pp.434-435, 1974

[13] Choi S.S and Sirisena H.R., "Computation of optimal output feedback gains for linear multivariable systems," IEEE Trans. on Autom. Control, Vol. AC-19, pp.257, 1974

[14] Newmann, M.H., "Specific optimal control of the linear regulator using a dynamical controller based on the minimal-order Luenberger observer," Int. J. Control, Vol.12, No.1, pp.33-48, 1970

[15] Weston J.E. and Bongiorno Jr., "Extension of analytical techniques to multivariable feedback control systems," IEEE Trans. on Autom. Control, Vol.AC-17, No.5, pp.613-620, 1972

[16] Sirisena, H.R. and Choi, S.S, "Design of optimal constrained dynamic compensators for linear stationary stochastic servomechanisms", Int J. Control, Vol.20, No.3, pp.363-369, 1974.

[17] Mendel J.M., "A concise derivation of optimal constant limited state feedback gains," IEEE Trans. on Autom. Control, Vol.AC-19, No.5, pp.447-448, 1974

[18] Fleming P.J., "A CAD Program For Suboptimal Control Linear Regulators," Proc. IFAC Symposium on "Computer-Aided Design of Control Systems", Zurich, 1979.

[19] Fleming P.J., "SUBOPT - A CAD Program for Suboptimal Regulators," Proc. Inst. Meas. Control Workshop on "Computer Aided Control System Design," 19-21 September, 1984, Sussex, U.K., pp.13-20.

[20] Kuhn. U, and Schmidt G., "Fresh look into the design and computation of optimal output feedback controls for linear multivariable systems," Int. J. Contiol., Vol. 46, No. , pp.75-95, 1987.

[21] Martin, G.D., and Bryson, A.E., "Attitude control of a flexible spacecraft," A.I.A.A. J. Guidance, Control and Dynamics, Vol. 3, pp. 37 1980.

[22] Fleming, P.J. "A non-linear programming approach to the computer-aided design of regulators using a linear-quadratic formulation," Int. J. Control, Vol.42, No.1,pp.257-268, 1985

[23] Kleinman, D.L. and Rao, P.K., "Extensions to the Bartels-Stewart Algorithm for Linear Matrix Equations," IEEE Trans. on Autom. Control, Vol AC-23,, pp.85, 1978

[24] Kwakernaak, H.and Sivan, R. "Linear Optimal Control Systems," John Wiley and Sons, 1972.

[25] Fleming, P.J., "Trajectory sensitivity reduction in the optimal linear regulator," PhD Thesis, Queen's University, Belfast, 1973.

[26] Kleinman, D.L. and Athans, M., "The design of suboptimal linear time-varying systems," IEEE Trans. on Autom. Control, Vol AC-13,, pp.150-159, 1968

Gradient Matrices

[27] Athans M. and Levine W.S., "Gradient matrices and matrix calculations," MIT Lincoln Labs., Lexington, Mass., Tech.Note 1965-53, 1965

[28] Athans M., "The matrix minimum principle," Inform. Contr., Vol.11, 1967

[29] Geering H.P., "On calculating gradient matrices," IEEE Trans. on Autom. Control, Vol AC-21, No. 1, pp.615-616, 1976

Servomechanisms

[30] Athans M., "On the design of PID controllers using optimal linear regulator theory," Automatica, Vol.7, pp.643-647, 1971

[31] Sandell N.Jr, and Athans M.,"Brief paper on 'Type L' multivariable linear systems,", Automatica, Vol.9 pp.131-136, Pergamon Press, 1973

[32] Davison E.J., 1976, "The robust control of a servomechanism problem for linear time-invariant multivariable systems," IEEE Trans. on Autom. Control, Vol AC-21, No. 1, pp.25-34., 1976

[33] Davison, E.J., and Ferguson, I.J., "The design of controllers for the multivariable robust servomechanism problem using parameter optimization methods," IEEE Trans. on Autom. Control, Vol AC-26, No. 1, pp.93-109., 1981

[34] Davison, E.J., and Scherzinger B.M., "Perfect control of the robust servomechanism problem," IEEE Trans. on Autom. Control, Vol AC-32, No. 8, pp.689-702., 1987

[35] Arstein, Z. and Leizarowitz A., "Tracking periodic signals with the overtaking criterion," IEEE Trans. on Autom. Control, Vol. AC-30, pp.1123-1126, 1985

[36] Bernstein D.S., and Haddad W.M, "Optimal output feedback for nonzero setpoint regulation," IEEE Trans. on Autom. Control, Vol. AC-32, No.7, pp.641-645, 1987

[37] Choek K.C., Loh N.K. and Ho J.B., "Continuous-time optimal robust servo-controller with internal model principle," Int. J.. Control Vol.48, No.5, pp.1993-2010, 1988

Two-Degree-of-Freedom Controllers and Feedforward Controllers

[38] Hara,S. and Sugie,T., "Independent parametrization of two-degree-of-freedom compensators in general robust tracking systems," IEEE Trans. on Autom. Control, Vol. AC-33, No.1, pp.59-67, 1988

[39] Grimble M.J., "Two-degrees of freedom feedback and feedforward optimal control of multivariable stochastic systems," Automatica, Vol.24, No.6, pp.809-817, 1988

[40] Seraji H., "Design of feedforward controllers for multivariable plants," Int. J. Control, Vol.46, No.5, pp.1633-1651, 1987

[41] Sebek M., Hunt K.J. and Grimble M.J., "LQG regulation with disturbance measurement feedforward," Int. J. Control, Vol.47, No.5,pp.1497-1505, 1988

[42] Hunt K.J., "General polynomial solution to the optimal feedback/feedforward stochastic tracking problem," Int. J. Control, Vol.48, No.3,pp.1057-1073, 1988

[43] Sobel K.M. and Shapiro E.Y., "A design methodology for pitch pointing flight control systems," J. Guidance, Vol.8,No.2,pp.181-187 1985

Sensitivity Reduction

[44] Fleming, P.J., "Desensitizing constant gain feedback linear regulators," IEEE Trans. on Autom. Control, AC-23, pp.933-936, 1978

[45] Subbayyan R., Sarma V.V.S an Vaithilingam M.C., "An approach for sensitivity-reduced design of linear regulators," Int. J. Systems, Vol.9, No.1, pp.65-74, 1978

[46] Yahagi, T., "Optimal output feedback control with reduced performance index sensitivity," Int. J. Control, Vol.25, No.5, pp.769-783, 1977

Design Examples

[47] Eitelberg E., "A regulating and tracking PI(D) controller", Int. J. Control, Vol. 45, No. 1, pp. 91-95, 1987.

[48] Nishikawa, Y., Sannomiya, N., Ohta, T. and Tanaka, H., Automatica, Vol.20, pp.321

[49] Muir, E.A.M., Kellett, M.G., "The RAF Generic VSTOL Aircraft Model: GVAM 87 User's Guide," RAE Technical Report in Preparation, RAE, Bedford, UK.

[50] Heffley, R.K., and Jewell, W.F., "Aircraft handling qualities data,", NASA CR-2144

[51] Kreisselmeier G., and Steinhauser R., "Application of vector performance optimization to a robust control loop design for a fighter aircraft," Int. J. Control, Vol.37, No.2, pp.251-284, 1983.

CONCLUSIONS

This thesis has presented a diverse yet unifying approach to Control System Design, ending with control examples which demonstrate how an effective control structure together with multi-objective design criteria is capable of producing practically realizable controllers covering a wide range of performance specifications. As this thesis has been fairly broad in nature, conclusions and suggestions for further research are given for each of the three main subjects.

CACSD

In Part 1 further evolution of a package such as MATLAB was considered. Various changes to the package were suggested including data structure aspects, inter-process communication facilities, compilation facilities, etc. It is apparent that MATLAB is a very powerful language for many types of mathematical problems and it has the potential to replace languages such as C and FORTRAN as a high-level language for numerical development. Already MATLAB is a proving a very useful tool for numerical algorithm development, however there is a trade-off in computational efficiency when compared to C or FORTRAN. If MATLAB is to become the next major numerical programming language, it is imperative that a compiler is written for it. The compiler will be in the form of a translation facility to C or FORTRAN code which will require a kernel database handler as well as a communication link to the MATLAB environment. Although a compilation facility is important, it requires a joint effort on the part of software companies and other workers to set standards and to undertake the necessary development.

Another very important concept which was discussed was the desirability of the integration of software. Ideally, all packages would be able to communicate freely to each other and on all machines. Much of current software development has been concerned with meeting this ideal with techniques such as cross-compilation, data transfer mechanisms and inter-process communication methods. Unfortunately in this domain, there is no panacea to resolve the fundamental problems of communication between programs and computers. Integration of software needs to be tackled using a pragmatic approach so that special requirements can be addressed.

One such area in which integration of software could be tackled would be for MATLAB to link easily and simply to other numerical libraries. A solution to this problem was suggested which required the development of a simple linking and compilation program. Such a project would provide MATLAB with a powerful interface to FORTRAN and C subroutines, and numerical libraries.

An example of a utility in which integration is troublesome is in the area of optimization software where the optimization program needs to communicate freely with the routine supplying objective functions and constraints. The problem here is one of speed since optimization programs often require large amounts of data to be transferred at high rates of transmission. This is due to the iterative nature in which the routines are called. It is therefore difficult to link optimization code to design environments such as MATLAB without directly inter-linking the packages via the source code. This motivated the direct linking of a FORTRAN version of MATLAB to a an optimization package, ADS. While this proved a useful and effective optimization tool it lacked the support of Pro- and PC-MATLAB versions in terms of graphics and other utilities. This prompted the development of a MATLAB Optimization Toolbox which could be directly integrated into the MATLAB environment.

OPTIMIZATION

In Part 2 optimization methods were discussed for a number of different types of problem formulation. Methods which are generally considered robust and iteratively efficient identified and implemented as a number of routines coded in the MATLAB command language. These routines form a MATLAB Toolbox which has possible wide ranging applications.

SQP was highlighted as a state-of-the-art method for Non-linear Programming. Further efficiencies for the SQP method could be achieved by using an active set strategy so that the gradients of all the constraints are not required at every major iteration (cf.[1]). Such a strategy would reduce the number of constraint gradient calculations for some problems significantly. This would make the possibility of considering semi-infinite problems using discretization strategies viable (cf.[2]).

A number of minor changes could also be made to the Quadratic Programming solution in order to improve the efficiency of the SQP method. One possible improvement would be to use updating factorizations of the projected Hessian matrix in the solution of the QP sub-problem in Section 2.7.2 as suggested in [3]. However, such procedures carry an overhead in terms of more coding which is likely to result in a beneficial trade-off only for larger problems.

Multi-objective optimization was discussed and the Goal Attainment method was introduced as a convenient method for solving problems with conflicting design requirements. Algorithm improvements were proposed to the SQP method and implemented as part of the Optimization Toolbox. Further research is necessary to develop other multi-objective methods and methods for statistical design in which the system operates with a level of uncertainty (cf.[4]).

CONTROL SYSTEM DESIGN

The Control System Design methodology described in Part 3 uses integral quadratic measures of control to design a wide range of performance requirements. Many design options and disturbance types were considered and cost functions were derived. The examples show that control design is possible using three basic measures of control, the output energy, the control energy and the derivative control energy. The designs were carried out using an evolutionary design procedure, which in the later stages used a multi-objective problem formulation to address the objectives independently. A two-degree-of-freedom (2DF) control structure was used to improve design characteristics over conventional feedback controllers. A mapping technique enabled good starting values to be found for higher order controllers reducing both the computation time and the likelihood of encountering local minima. Using the multi-objective approach it was demonstrated how to design controllers for plants with multiple operating conditions (i.e. nonlinear) permitting the design of robust fixed gain controllers.

The overall design approach has many far many reaching applications which could be further investigated. The design of optimal observers is also seen as an area of further research using this approach. Software for the design methods has been written as a general purpose code in the MATLAB command language. In order to incorporate a large number of design options the code is not particularly efficient for certain problem formulations. Further work is necessary to improve the efficiency of the algorithms by writing specific routines which exploit the characteristics of specific problem types. However, at present the routines remain a very flexible set of routines for general purpose Control System Design.

REVIEW

Overall this thesis has presented a viable and attractive way to perform Control System Design and other types of engineering optimization. The aim has been to provide a set of tools using the MATLAB environment which are accessible to a wide user group of control engineers and other workers. The methods employed have been chosen for their effectiveness and have been founded on well established mathematical theory. Control System Design methods have been used employing integral quadratic measures of control. The theory has been extended to include servomechanisms, output disturbances and a two-degree-of-freedom control structure.

Further areas of research have been suggested within the areas of CACSD, Optimization and Control System Design. These consist of both minor improvements to existing methods and new research areas which encompass broad aspects of the design approach.

REFERENCES

[1] Schittowski K., "NLQPL: A FORTRAN-subroutine solving constrained nonlinear programming problems", Annals of Operations Research, Vol. 5, 485-500, 1985.

[2] Polak, E. and Tits, A. "A recursive quadratic programming algorithm for semi-infinite optimization problems," J.Appl. Math. Optimiz., Appl. Math. Optim., Vol.8, pp.325-349, 1982

[3] Gill P.E., Murray W., and Wright M.H. "Practical Optimization", Academic Press, London, 1981.

[4] Brayton, R.K., Hachtel, G.D. and Sangiovanni-Vincentelli, A.L., "A survey of optimization techniques for integrated-circuit design," Proc. of IEEE, Vol.69, No.10, pp.1334-1363, 1981

Appendix A

Optimization Toolbox Users' Guide

Optimization Toolbox: Contents

OPTIMIZATION TOOLBOX: Overview

The Optimization Toolbox is a set of easy-to-use routines for solving optimization problems. It consists of a set of MATLAB m-files which implement a number of non-linear programming algorithms. The principle routines are as follows:

OPTIMIZATION TOOLBOX ROUTINES

TYPE	NOTATION	SYNTAX
Unconstrained Optimization:	$\min_X f(X)$:	[x,para]=unconstr(x,f,para)
Constrained Optimization:	$\min_X f(X)$ subject to : $G(X) \le 0$ } :	[x,para]=constr(x,f,g,para)
Goal Attainment:	$\min_{X,\gamma} \gamma$ s.t.: $F(X) - W.\gamma \le$ GOAL :	[x,para]=attaingoal(x,f,goal,w,para)
Minimax:	$\min_X \max F(X)$ subject to : $G(X) \le 0$:	[x,para]=minimax(x,f,g,para)
Non-Linear Least Squares:	$\min_X \sum F(X).*F(X)$:	[x,para]=leastsq(x,f,para)
Non-linear Equations:	$F(X) = 0$:	[x,para]=solve(x,f,para)
Linear Programming:	$\min_x \{ x^T Hx + cx \}$ subject to: $Ax \le B$:	[x]=lp(f,a,b)
Quadratic Programming:	$\min_x \{ f^T x \}$ subject to: $A.x \le B$:	[x,lambda]=qp(h,b,a,b)

The routines are designed to work with scalars, vectors and matrices. Matrices are indicated by upper-case bold letters, vectors by lower case bold letters and scalars by plain letters.

All the routines except lp and qp are called on an iterative basis by a user-defined function, a *script file* or by the user. The optimization routines do not call any user-supplied functions. Instead, the information in terms of functions evaluations and any available gradient information is supplied to the optimization functions at each iteration. This means that the optimization may be interactively interrupted in order to update or change the problem formulation or optimization parameters. It also gives the user freedom over program structure and helps to promote modularity.

Emphasis has been placed on ease-of-programming. The intention has been to provide a set of robust and iteratively efficient set of routines. The routines are ideal for complex problem solving and for design applications involving non-linear objectives. All the main routines are supported by graphics facilities which consist of graphical monitoring of both the design variable location and the function values. This is performed using contour plots and x-y graphs.

Default optimization parameters are used extensively, these may be changed by the user through the vector para. Gradients when needed are calculated using a finite difference approximation method unless they are supplied using the optional variable grad.

OPTIMIZATION TOOLBOX: A Tutorial

In this section the use of the Optimization Toolbox will be presented through examples. Although only the functions unconstr and constr will be considered, the features described below apply to all the optimization routines (attaingoal, minimax, minimaxabs, leastsq and solve). The only difference between the routines is in the problem formulation and the termination criteria.

Unconstrained Optimization

Consider initially the problem of finding a set of design variables $[x_1,x_2]$ to minimize the following function:

$$f(X) = e^{x1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)) \quad (1)$$

A solution to this problem can be found by typing in the following commands, entering them in a *script file* or as part of a user-defined function.

Unconstrained Example

```
PARA = 0;                                    % Initialize Optimization Parameters
X = [-1,1];                                  % Initialize Design Variables
while PARA(1) ~= 1                           % Check Termination Parameter
  F=exp(X(1)) * (4*X(1)^2 + 2*X(2)^2 + 4*X(1)*X(2) + 2*X(2) + 1);   % Evaluate F
  [X,PARA] = unconstr(X,F,PARA);             % Call Optimizer recursively
end
```

This routine is contained in the *script file* testunconstr2.m and may be run as a demonstration or used as a template for writing other optimization problems. Executing this *script file* gives the following solution after 49 iterations.

```
F =
  3.1158e-12
X =
  0.5000  -1.0000
```

At each iteration the function unconstr returns new values for the design variables $[x_1,x_2]$. Function evaluations for this new point must then be evaluated and passed back to unconstr. Notice that we have to make an initial guess on the design variables $[x_1,x_2]$ which may affect both the number of iterations and the value of the solution point should there exist a number of local minima. In the example above X has been initialized to [-1,1].

There is also a variable para which must be passed to unconstr. This is a vector of optimization parameters which may be used to change the characteristics of the optimization solution procedure. It contains values such as termination tolerances and algorithm choices. The first element of para is used for control flow. Initially this is set to zero, to ensure initialization of the optimization procedure. If no other values are specified for para a vector of default parameters is returned. When sufficient termination criteria have been met (see Reference Manual) the optimizer returns: para(1)=1. More about changing the default settings later.

Adding Constraints

Suppose now we wish to add inequality constraints to the problem in (1) giving a problem of the form:

$$\min_{x_1 x_2} (e^{x1}(4x_1^2+2x_2^2+4x_1x_2+2x_2+1))$$

subject to the constraints : $1.5 + x_1x_2 - x_1 - x_2 \leq 0$

$$x_1x_2 - 10 \leq 0$$

(2)

This can solved using the function constr in the following set of commands:

Constrained Example

```
PARA = 0;                                                    %Initialization
X = [-1,1];
while PARA(1)~=1                                             %Check Termination
  F = exp(X(1))*(4*X(1)^2 + 2*X(2)^2 + 4*X(1)*X(2) + 2*X(2) + 1); %Evaluate F
  G(1) = 1.5 + X(1)*X(2)  - X(1)  - X(2);                    %Evaluate Constraints
  G(2) = -X(1)*X(2) - 10;
  [X,PARA] = constr(X,F,G,PARA);                             %Call Optimizer recursively
end
```

This problem is found in the file testconstr2.m and gives the following solution after 37 iterations:

```
F =
  0.0236
G =
  1.0e-13 *
   0.0489  -0.5151
X =
  -9.5474  1.0474
```

User-Supplied Gradients

The above problem solution procedure uses a method to systematically perturb each of the design variables in order to estimate the function and constraint gradients. The problem can be solved more accurately and efficiently if we supply analytic partial derivatives of the function and constraints. This is done by introducing an additional argument, grad, in the function call to constr. The first column of grad contains the partial derivatives of the objective function, f(*X*), with respect to x. The next columns contain the partial derivatives of the constraints in order of location. When the constraints, *G*, are in the form of a matrix then the *i+1*th column of grad refers to the *i*th constraint of *G* when it is arranged as a column-wise vector using the command G(:). Gradients are only required when the optimizer returns para(1)=2. Thus problem (2) with analytic gradients can be programmed as:

User-Supplied Gradients

```
PARA = 0;                                                    %Initialization
X=[-1,1];
while PARA(1)~=1                                             %Check Termination
  if PARA(1) == 2                                %Calculate Analytic Gradients If Needed
    TEMP=exp(X(1))*(4*X(1)^2+2*X(2)^2+4*X(1)*X(2)+2*X(2)+1);
    grad= [TEMP + 4*exp(X(1)) * (2*X(1) + X(2)),  X(2)-1,   -X(2)
           4*exp(X(1))*(X(1)+X(2)+0.5),  X(1)-1,   -X(1)];
  else                                                       %Otherwise Calculate F and G
    G(1) =1.5 + X(1)*X(2)  - X(1)  - X(2);
    G(2) = -X(1)*X(2) - 10;
    F = exp(X(1)) * (4*X(1)^2 + 2*X(2)^2 + 4*X(1)*X(2) + 2*X(2) + 1);
  end
  [X,PARA] = constr(X,F,G,PARA,grad);                        %Call Optimizer
end
```

This is contained in the file testconstr3.m and gives the following result after 12 function evaluations and 12 gradient evaluations.

```
F =
  0.0236
G =
  1.0e-12 *
  -0.2558   0.2558
X =
  -9.5474   1.0474
```

Gradient Check

When user-supplied gradients are available, the user has the option of checking these, in the first few evaluations of the optimization process, with a set calculated using finite difference evaluation. This is particularly useful for detecting typing or other errors in either the objective function or the gradient calculation. It can also be used in other applications where the numerical evaluation of partial derivatives is required.

If such gradient checks are required then para(1) should be initialized with the value -1. The first cycle of the optimization is then concerned with checking the user-supplied gradients. If they do not match within a given tolerance the user is informed of the discrepancy and is given the option either to abort the optimization or to continue.

The routines may also be used to evaluate the gradient (partial derivatives) at a point without performing any optimization. If this is the case then para(1) should be initialized with -2. The gradient is then returned as a third argument in the input parameter list to the optimization function e.g. [X,PARA,GRAD] = unconstr(X,F,PARA).

Adding Bounds

Suppose we wish to restrict the variables to be within certain limits. This can be achieved by using the bounded syntax of the appropriate function. For constr the syntax is as follows

```
[X,PARA] = constr(X,F,G,PARA,VLB,VUB);
or [X,PARA] = constr(X,F,G,PARA,VLB,VUB,GRAD);
```

Where VLB and VUB contain lower and upper bounds on the variables. Thus the commands to restrict the variables in problem (2) to be greater than zero can be written as:

Bounded Example

```
PARA = 0;
X = [-1,1];
while PARA(1) ~= 1
  F=exp(X(1))*(4*X(1)^2+2*X(2)^2+4*X(1)*X(2)+2*X(2)+1);
  G(1)=1.5 + X(1)*X(2) -X(1) -X(2);
  G(2)=-X(1)*X(2)-10;
  [X,PARA]=constr(X,F,G,PARA,zeros(X),[ ]);          %Add Bounds
end
```

Generally VLB and VUB should be of the same size as X although the routines will also accept smaller sizes and will assume the undefined variables to be unbounded. Notice in this example that since there are no upper bounds we have passed down the empty matrix [ ]. This can also be done for lower bounds and for the constraint variable, G. Thus constr can also be used as an unconstrained optimization routine.

The above program is coded in testconstr4.m and gives the following solution after 7 iterations:

```
F =
  8.5000e+00
G =
         0 -1.0000e+01
X =
         0  1.5000e+00
```

Note that when we express lower bounds on the design variables then we must also express upper bounds on the variables although either may be set to the empty matrix as in the above example. This is so that constr can distinguish between the syntax for when user-supplied gradients are given, [X,PARA]=constr(X,F,G,PARA,GRAD), and when only bounds are supplied, [X,PARA]=constr(X,F,G,PARA,VLB,VUB). Alternatively we can express bounds using linear inequality constraints. This may be more appropriate when there are only a few bounded variables i.e.

Upper Bound: $x_i \leq U_B$ should be written as : $-x_i + U_B \leq 0$

Lower Bound: $x_i \geq L_B$ should be written as: $x_i - L_B \leq 0$

Notice, in the above problems, that the more constrained and bounded the problems have been, the less function iterations have been required. This is because the optimization can make better decisions regarding steplength and regions of feasibility than in the unconstrained case. It is therefore always wise to bound and constrain any problem whenever possible to promote a fast convergence to the solution.

Maximization

The optimization functions (uncosntr,constr,attaingoal,minimax, minimaxabs, leastsq) all perform minimization of the objective function(s), F. Maximization is achieved by supplying the routines with -F.

e.g. for $\max_X f(X)$ subject to: $G(X) \leq 0$ use: [X,PARA]=constr(X,-F,G,PARA)

Greater Than Zero Constraints

The Optimization Toolbox uses constraints of the form $g_i \leq 0$. Greater than zero constraints can be expressed as less than zero constraints by multiplying them by -1. I.e. if constraints of the form $g_i \geq 0$ are required then this is equivalent to the constraint $-g_i \leq 0$.

e.g. for $\min_X f(X)$ subject to: $G(X) \geq 0$ use :[X,PARA]=constr(X,F,-G,PARA)

Equality Constraints

Equality constraints are expressed in the first few elements of the matrix G. Para(13) must be initialized with the number of equality constraints. For example, a program which adds the constraint $x_1+x_2=0$ to problem (1) is as follows:

Equality Constrained Example

```
PARA(13)=1;                    %Set para(13) to the number of equality constraints.
X=[-1,1];
while PARA(1)~=1
   F=exp(X(1))*(4*X(1)^2+2*X(2)^2+4*X(1)*X(2)+2*X(2)+1);
   G(1)=X(1)+X(2);             %Evaluate Equality Constraint in first element of G
   G(2)=1.5 + X(1)*X(2) -X(1) -X(2);
   G(3)=-X(1)*X(2)-10;
   [X,PARA]=constr(X,F,G,PARA);                          %Call Optimizer
end
```

This is coded in the *script file* tesconstr5.m and produces the following solution after 13 iterations:

```
F =
   1.8951e+00
G =
  -4.2633e-14  8.5265e-14 -8.5000e+00
X =
  -1.2247e+00  1.2247e+00
```

Changing The Default Settings

The vector para contains a number of parameters which are used in the optimization routines. If on the first call to an optimization routine any of the elements of para contain a zero or if they are not defined (i.e. empty) then a set of default parameters is generated. The default parameters may be overridden by giving them any non-zero elements. This may be done either during the optimization cycle or on initialization. Some of the parameters are calculated using factors based on, for instance, problem size or convergence during the optimization (e.g. steplength). Many of the parameters are also dependent on the specific routine being used and are documented more fully in the reference manual, however, a general description of the parameters is as follows:

NO	FUNCTION	DEFAULT	DESCRIPTION
1	Control Flow	-	To initialize set para(1)= 0. Termination is indicated when para(1)=1 is returned from the optimization routine. When gradients are required then para(1)=2 will be returned, otherwise function and constraints evaluations should be performed.
2	Termination 1	1e-4	Termination criterion which is a measure of the worst case precision required of the design variables following convergence. N.B. The optimization will not terminate until all termination criteria have been met.
3	Termination 2	1e-4	Termination criterion which is a measure of the precision required of the objective function following convergence.
4	Termination 3	1e-7	Termination criterion whose function varies depending on routine being used. In constr it is a measure of the worst case constraint violation which can be accepted
5	Main Algorithm	0	Used to select the main optimization algorithm used in the routines.
6	Direction Algorithm	0	Changes the search direction algorithm to be used.
7	Search Algorithm	0	Changes the line search algorithm to be used.
8	Attainment Factor	-	Measure of performance. In the goal attainment and minimax routine it contains an attainment factor.
9	Display	2	Controls how much output is given during the optimization cycle. 1 displays nothing. 2 displays some results. 3 displays all results. 4 may be used as a Debugging Mode.
10	Function Counter	-	Function evaluation counter.
11	Gradient Counter	-	Number of function gradient evaluations or finite difference gradient calculations.
12	Constraint Counter	-	Total number of constraint gradient calculations or finite difference gradient calculations.
13	Equality Constraints	0	Number of equality constraints. Equality constraints should be put in the first few elements of the variable G.
14	Max Iterations	100n	Maximum number of iterations allowed. Termination is indicated by returning para(1)=1. The default is to set para(14) to 100 multiplied by the number of design variables.
15	Finite Difference1	1e-4	Factor used in working out finite difference gradients. Used to multiply last change in x to get the perturbation levels. Default 1e-4. For larger differences set to e.g. .0.1.
16	Finite Difference2	-	Minimum change in variables for finite difference gradients. Defaults: Constrained Optimization 1e-7. Unconstrained Optimization 1e-10.

17	Finite Difference3	0.1	Maximum change in variables for finite difference gradients.
18	Step length	-	Step length parameter. Generally on the first iteration this is set conservatively to a value of 1 or less depending on the gradients.
19	Graphics	0	Turns graphics facilities on(>0) or off(0). The value of para(19) Indicates the type of plot required. Set to 1 for a performance monitoring graph. Set to 2 for a contour plot. Set to 3 for an isometric plot. Set to 4 for a contour plot, an isometric plot and an on-line performance plot on the same screen.

The default parameters can be changed in a number of ways, either before the initialization or during the running of the optimization cycle. An example of how to change the termination criteria in problem (2) to 1e-8 might be

Changing the Default Settings

```
PARA(2)=1e-8; PARA(3)=1e-8;                    %Change Termination Parameters
X=[-1,1]; end
while PARA(1)~=1
 F=exp(X(1))*(4*X(1)^2+2*X(2)^2+4*X(1)*X(2)+2*X(2)+1);
 [X,PARA]=unconstr(X,F,PARA);
end
```

The above code is contained in the file testunconstr6.m and gives the following solution after 63 function evaluations:

```
F =
  2.1965e-14
x =
  0.5000 -1.0000
```

To get on-line help for the meanings of the parameters enter the command help setpara. To get a set of default parameters use the command: para=setpara([ ]).

Speeding Up The Optimization

The optimization routines use quite a lot of internal parameters in order to use information form previous iterations. This is temporarily stored to the file, tempoptim.mat, at each iteration. If file access is slow this may increase the total run-time of the optimization cycle. In order to avoid this the option is available to use global variables. This also has the advantage that at the end of the optimization the user may inspect internal parameters which may give useful information for educational or diagnostic purposes. In order to use global variables enter the command optimglob at the beginning of each session (or in your startup.m file). It is also necessary to use this command after every use of clear. Alternatively this command may be used as the first line in every optimization *script file* e.g.:

Adding Global Variables

```
optimglob                                          %Set Up Global Variables
PARA=0;
X=[-1,1];
while  PARA(1)~=1
    F=exp(X(1))*(4*X(1)^2+2*X(2)^2+4*X(1)*X(2)+2*X(2)+1);
    [X,PARA]=unconstr(F,X,PARA);
end
```

The use of global variables is highly recommended for reasons of execution efficiency. However, in certain cases it may be advantageous to store the variables to file at every iteration, for instance, it may be necessary to investigate how a background job is progressing in which case this can be done by loading the last saved version of tempoptim while the background job is still executing. The global variables have been named with capital letters and a G_ preceding each variable so that it is highly unlikely that this will coincide with other variable names.

The global variables are as follows:

G_MATL G_MATX G_PCNT G_STEPMIN G_SD G_GCNT G_OLDF G_GRAD G_HOW G_CHG G_LAMBDA G_GLOBFLAG G_LAMBDABEST G_XBEST G_FBEST

There are also a number of global variables associated with the graphics facilities they are follows:

G_MESH G_GPARA G_MDX G_MDY G_GXCNT G_GYCNT G_CONTOURS G_GSX G_GSY G_AXIS G_MESHC G_AXIS2

Of particular interest is the string variable G_HOW which contains a complete history of the optimization cycle. Another variable G_HESS contains an estimate of the Hessian matrix at the solution point.

Storing The Results

At the end of every optimization all relevant data is stored to a diary file called optimdata so that the results can be inspected at a later date. This allows the optimization to be run as a background process which may be more appropriate for large problems (e.g. when the number of design variables is greater than 20).

Graphics Facilities

All the main routines are supported by graphics facilities which are invoked simply be setting para(19) to an appropriate value on the first call to the optimization routine. The options for the graphs consist of a contour plot, an isometric (mesh) plot and/or on-line performance plots. The selection is achieved through para(19) which is set accordingly:

para(19)=1	gives performance monitoring plot(s).
para(19)=2	gives a position monitoring contour plot.
para(19)=3	gives an isometric(mesh) plot of the objective function and/or constraints.
para(19)=4	gives all of the above plots on the same screen.

When using the graphics facilitates global variables must be set up using optimglob. This is for reasons of execution efficiency. If they are not set the user will be informed and the program will abort. When plotting facilities are requested then the user is prompted for the necessary information to determine plotting parameters. This information is put in a global variable called G_GPARA which may be altered. Alternatively this information can be directly entered into G_GPARA by setting the elements with the following information.

Element of G_GPARA	Description
1	x-axis element variable for contour plot e.g. for x(3) set G_GPARA(1)=3.
2	y-axis element variable for contour plot e.g. for x(2) set G_GPARA(2)=2.
3	minimum value for x-axis variable on contour plot.
4	maximum value for x-axis variable on contour plot.
5	minimum value for y-axis variable on contour plot.
6	maximum value for y-axis variable on contour plot.
7	minimum value for objective function monitor plot.
8	maximum value for objective function monitor plot.
9	minimum value for constraint plot (log scale).
10	maximum value for constraint plot (log scale).
11	estimated maximum number of iterations to be displayed on plot.
12	number of subdivisions on contour plot for each axis.
13	position on screen for contour plot e.g. 111 is all of screen 221 is first quadrant.
14	position on screen for isometric plot.
15	position on screen for function performance plot.
16	position on screen for constraint performance plot.
17	when set to 1 the last generated mesh and inputted axis information is used for subsequent plots. When set to 2 a new mesh is generated but using the axis information in the elements of G_GPARA. These settings are useful when trying different starting values for the same optimization problem.
18-20	internal parameters. (18=used as indicator for termination of contour plotting, 19=number of constraints, 21=last iteration count for last plotted point)

G_CONTOURS is another global variable associated with the plotting which is either a scalar containing the number of contours or a vector containing precise values for the contours. G_GPARA(17)=1 allows the same contour and axis information to be used on subsequent cycles without recalculation of the contour values.

Example

Consider the doubly constrained problem below which is in the *script file* testconstr16.m:.

Graphics Example

```
optimglob                                          %Set up global variables
if ~exist('PARA'), PARA(19)=4; else PARA(1)=0; end        %Set up PARA
if ~exist('X') ,X=[2,4]; end
while PARA(1)~=1
   F=100*(X(2)-X(1)^2)^2+(1-X(1))^2;
   G(1)=(X(1))*(X(1))-0.3*(X(2)-2)*(X(2)-2)-0.01;
   G(2)=0.5*(X(1))*(X(1))+(X(2)-2)*(X(2)-2)-2;
   [X,PARA]=constr(X,F,G,PARA);
end
```

We have used the exist command in this routine in order to change the settings after a first running. After executing this code the user is prompted with information regarding the way the graphs are plotted. Here is a typical input:

Minimum value of x(1) = ? -2
Maximum value of x(1) = ? 2
Minimum value of x(2) = ? -1
Maximum value of x(2) = ? 4
Number of contour points to be taken for each axis(e.g. 10) ? 30
Enter the number of contours you want displayed
- alternatively enter a vector containing contour values: exp(2:2:20)
Maximum likely number of iterations (e.g. 100) ? 50
Minimum function value for plot ? 0
Maximum function value for plot ? 10
Give exponent (10^exp) for the minimum constraint value for plot ? - 10
Give exponent (10^exp) for the maximum constraint value for plot ? 2

Having entered this information the contour points are evaluated by the optimization routine. An isometric plot of the objective function is then displayed and the user is given the prompt:

Rotate ? (l=left,r=right, u=up, d=down, q=quit, 0=fun, 1,2..=constraint):

Options (l,r,u,d) have the effect of rotating the plot in increments of 10 degrees. When constraints exist entering a number greater than zero produces an isometric plot of the respective constraint which is plotted in place of the objective function, this may then be rotated. Entering 0 brings back the plot of the objective function. Typing the option q ends the isometric plotting phase. The remaining graphs are then displayed and the optimization begins. Points are then plotted as the optimization progresses. The above settings gave the following results and graphs.

Results after 51 iterations:
X = -7.2831e-01 6.8289e-01
G = 2.0841e-12 1.0418e-12
F = 5.3113e+00

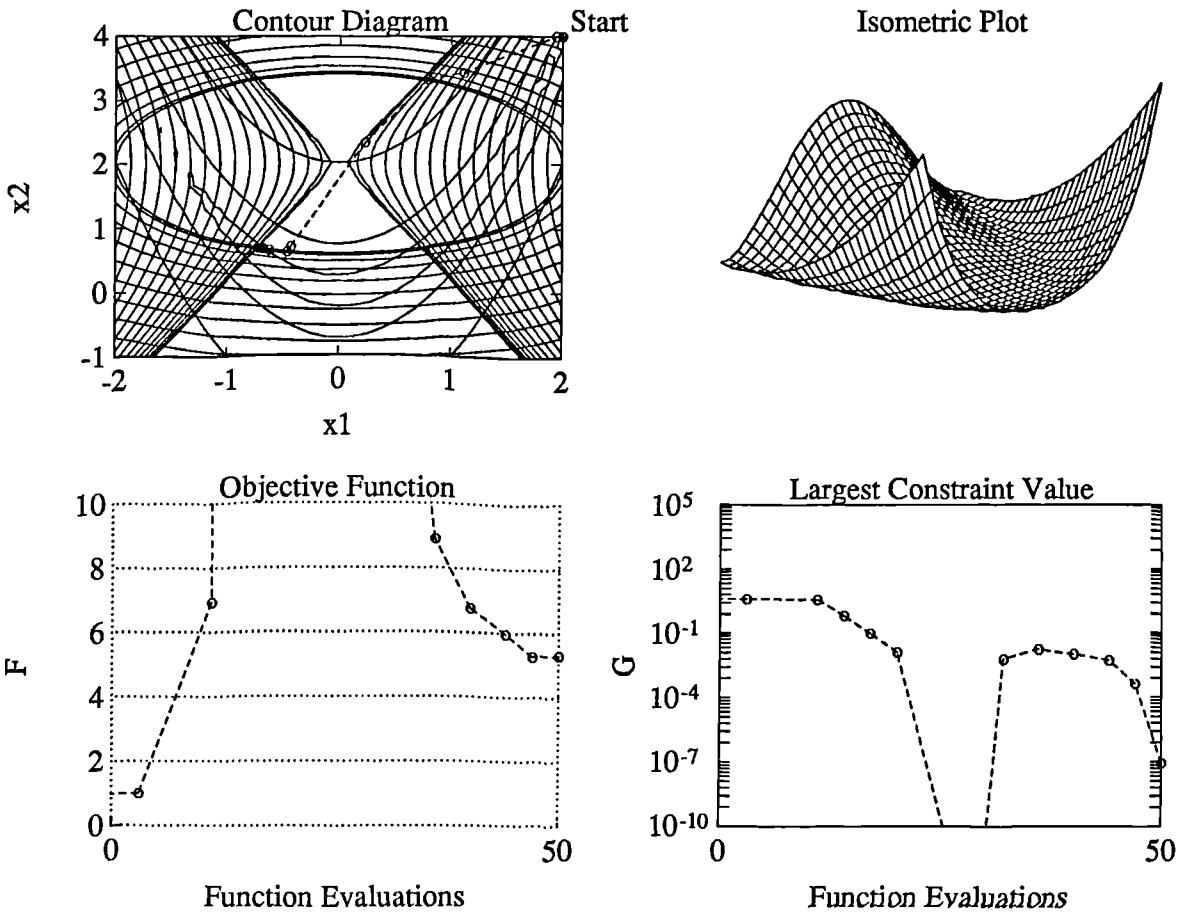

Fig 1: Graphics Output For Constrained Optimization Problem

The isometric plot is that of the objective function. The contour plot is that of the objective function and the constraints (dense contours). The constraints contours are only displayed for infeasible values and are displayed densely to indicate the area of infeasibility. The starting point of the Optimization cycle is at the point [2,4] at the top right-hand corner of the contour diagram. The path of the solution is represented by the dotted line which is plotted at the completion of each major iteration. The above example illustrates the ability of the optimizer to locate the feasible region despite the small gap between the constraints with which it must pass in order to enter the bottom part of the feasible region. However, it also illustrates a fundamental problem in optimization and that is the location of the global minimum in the face of a non-convex solution boundary. Consider what happens when we start the optimization problem from the opposing side of the contour diagram. We can use the contour points from the last plot and make the contour diagram fill the whole screen using the following commands:

```
PARA(19)=2                  %Set graph parameter to just plot contour diagram
X=[-2,4]                    %Reset X to new point.
G_GPARA(17)=1               %Tell graphics routine to use last evaluated set of contours
testconstr16                %Start optimization
```

Executing these instructions gives the following solution and graph:

Results after 52 iterations:
X = 7.2831e-01 6.8289e-01
G = 3.3983e-12 1.6982e-12
F = 2.3980e+00

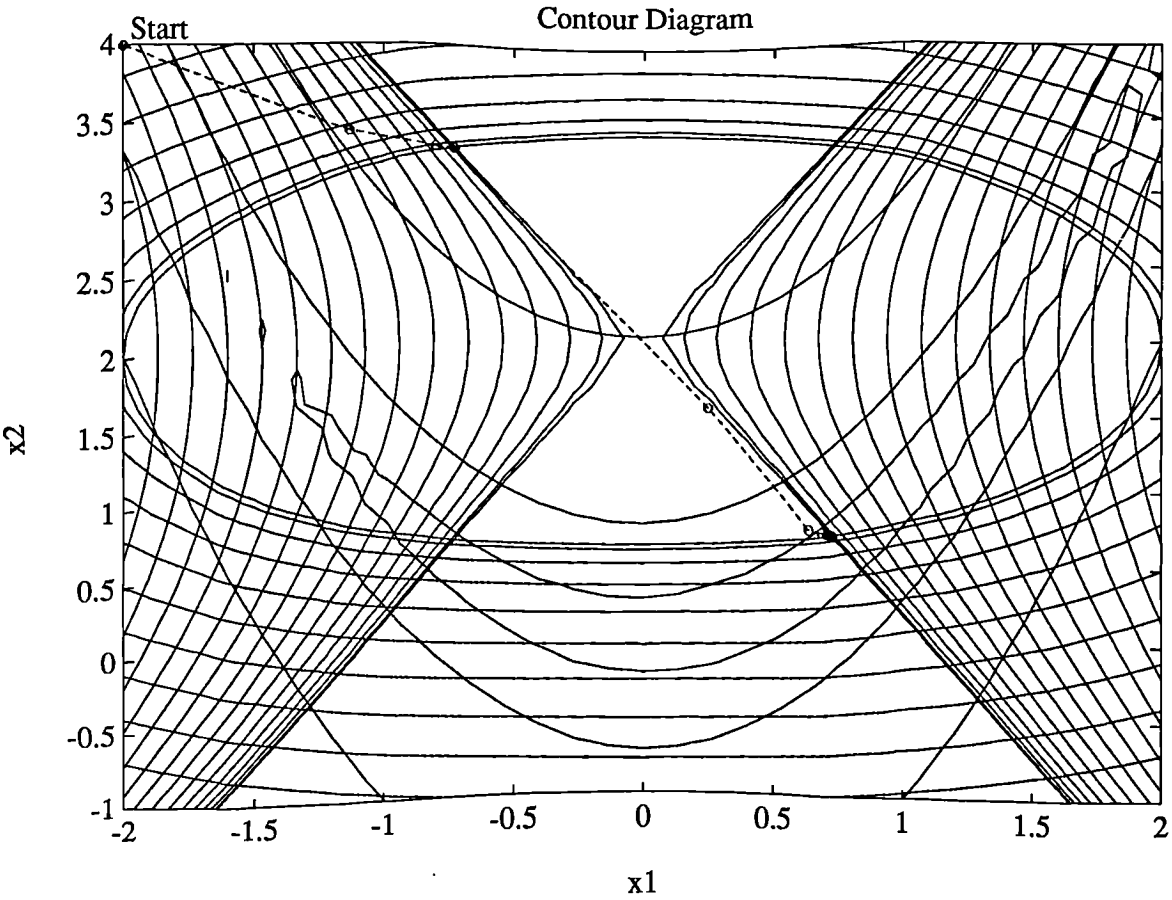

Fig 2: Global Minimum Location

The minimum located form this starting point has a function value which is less that achieved from the original starting point. This stresses the importance of trying different starting values when it is suspected that there may be a number of local minima.

Interrupting The Optimization

The optimization can be aborted at any time using the key sequence ctrl-C. All the optimization functions have the advantage that if you are running the optimization from a *script file* or through keyboard entered commands then no information will be lost on aborting. One option is therefore to set very high stopping tolerances in the optimization formulation and to abort the optimization when the results are satisfactory. In this way the user may interact with the optimization cycle and make changes to the optimization problem at will.

Common Problems

•The routines may only give local solutions, it is therefore necessary to try the optimization form a number of different starting points if global solutions are sought.

•If the routines do not converge check that you have not posed an infeasible problem.

•The routines make use of finite difference gradients if they are not user-supplied. It may therefore be necessary to interpolate all discrete functions such as time and frequency responses to avoid excessive errors in the gradient evaluation. This can be achieved using the spline function or any other interpolation method.

•Sometimes the optimization may give values for which it is impossible to evaluate F and G, such as the evaluation of a time response when the system is unstable. It is therefore necessary to properly bound the design variables or to give a large positive value to F and G when infeasibility is encountered.

•The function to be minimized must have continuous first and second derivatives. However, some success may be achieved for certain classes of discontinuities if the finite difference parameters are adjusted to appropriate values.

•The optimization routines do not provide for the case when the variable X can only take on discrete values, however, some success may be achieved by resetting the global vector CHG at each iteration. This variable corresponds to the finite difference gradient perturbation levels for the matrix X multiplied by para(15) i.e. For each variable indexed by i a partial derivative is calculated by perturbing X using the following formula : $x_{k+1} = x_k + \Delta x = x_k +$ CHG(i) *para(15) where para(16) $< \Delta x <$ para(17).

OPTIMIZATION TOOLBOX
Reference

This section contains detailed descriptions of the main OPTIMIZATION TOOLBOX functions. The routines contained in this section are as follows:

MAIN ROUTINES	
unconstr	unconstrained optimization
constr	constrained optimization
attaingoal	multi-objective goal attainment
minimax	minimax optimization.
leastsq	least squares optimization
solve	non-linear equation solver
lp	linear programming
qp	quadratic programming

UTILITY ROUTINES	
setpara	parameter settings and help
optimglob	sets up global variables

Purpose:

Solves unconstrained optimization problems.

Synopsis:

```
[x,para]=unconstr(x,f,para)
[x,para]=unconstr(x,f,para,grad)
```

Description:

Unconstr minimizes a scalar objective function of the form:

$$\min_{X} \quad f(X)$$

Values of the scalar function f(X) must be supplied to unconstr on an iterative basis. Values of X are returned at each iteration and a new value of f(X) must be evaluated. X may be a scalar, vector or a matrix.

Upon initialization para(1) should be set to 0. If other values for para are not supplied then unconstr returns default parameters (see Tutorial). Unconstr returns a value of para(1)=1 when the optimization has terminated following sufficient convergence or when the number of iterations exceeds para(14). The optimization will terminate successfully following convergence if the precision of X at a minimum is within the tolerance given by para(2) (default: 1e-4) and the objective function is estimated to be within the tolerance given by para(3) (default: 1e-4).

If the optional variable grad is not supplied gradients are calculated using a finite differences approximation.

Example:

Find values of X which minimize: $100*(x_2-x_1^2)^2 + (1-x_1))^2$ starting at the point [-1.2,1];

```
PARA=0;                                    %Reset Optimization Parameters
X=[-1.2,1];                                %Initialize Design Variables
while PARA(1)~=1                           %Check Termination Parameter
 F=100*(X(2)-X(1)^2)^2+(1-X(1))^2;         %Evaluate F
 [X,PARA]=unconstr(X,F,PARA);              %Call Optimizer
end
```

This program is contained in the *script file* testunconstr1.m and gives the following solution after 132 iterations:

```
F =
   8.8348e-11
x =
        1.0000 1.0000
```

Limitations:

The function to be minimized must be continuous. Unconstr may only give local solutions.

Algorithm:

The default algorithm is a quasi-Newton method. Setting para(5)=1 implements the simplex method of Nelder and Mead[2] and programmed by S.Hancock (gradients should not be supplied when using this method). If a quasi-Newton method is used then the default algorithm for updating the approximation of the Hessian matrix is the BFGS[3-6] formula. The DFP[7,8] formula which avoids direct calculation of the inverse Hessian, may also be selected by setting para(6)=1. A steepest descent is selected by setting para(6)=2 (although not recommended). The default line-search algorithm

(para(7)=0) is a safeguarded mixed quadratic and cubic polynomial interpolation and extrapolation method. Safeguarded cubic interpolation is the default line-search algorithm (para(7)=1) when gradients are supplied.

See Also:

setpara, optimglob

References:

[1]Nelder J.A. and Mead R., *A simplex method for function minimization*, Computer Journal Vol.7, pp. 308-313.

[2]Broydon C.G, *The convergence of a class of double-rank minimization algorithms*, J. of the Inst. of Mathematics and its Applic., Vol. 6, pp. 76-90, 1970.

[3]Fletcher R., *A new approach to variable metric algorithms*, Computer Journal, Vol. 13, pp. 317-322, 1970.

[4]Golfarb. D., *A family of variable metric updates derived by variational means*, Mathematics of Computing, Vol. 24, pp. 23-26, 1970.

[5]Shanno D.F., *Conditioning of quasi-Newton methods for function minimization*, Mathematics of Computation, Vol. 24 pp. 647-656, 1970.

[6]Davidon W.C., *Variable metric method for minimization*, A.E.C. Research and Development Report, ANL - 5990, 1959.

[7]Fletcher R., Powell M.J.D., *A rapidly convergent descent method for minimization*, Computer Journal, Vol. 6, pp. 163-168, 1963.

[8]Fletcher R., *Practical Methods of Optimization* Vol. 1, Unconstrained Optimization, John Wiley and Sons.

[9]Grace A.C.W., *Computer-Aided Control System Design using Optimization Techniques*, Ph.D. Thesis, University of Wales, Bangor. 1989

Purpose:

Solves constrained optimization problems.

Synopsis:

```
[x,para]=constr(x,f,g,para)
[x,para]=constr(x,f,g,para,grad)
[x,para]=constr(x,f,g,para,vlb,vub)
[x,para]=constr(x,f,g,para,vlb,vub,grad)
```

Description:

Solves the constrained problem of the form:

$$\min_X f(X)$$

$$g_{ij}(X) \le 0, \qquad i=1, ..., m_1 \;\; j=1, ..., m_2$$

$$\text{vlb}_{kl} \le x_{kl} \le \text{vub}_{kl} \qquad k=1, ..., n_1 \;\; l=1, ..., n_2$$

Values of the scalar objective function, f(X), and constraints, $G(X)$, must be supplied to constr on an iterative basis. Values of X are returned at each iteration and new values of f(X) and $G(X)$ must be evaluated. X and $G(X)$ may be a scalars, vectors or a matrices.

Upon initialization para(1) should be set to 0. If other values for para are not supplied then constr returns default parameters (see Tutorial). Constr returns a value of para(1)=1 when the optimization has terminated following convergence and when all the termination criteria (para(2:5)) have been met or when the number of iterations exceeds para(14). Para(2) is a measure of the precision required of X before the optimization will terminate (default: 1e-4). Para(3) is a measure of the precision required of the objective function at the solution (default: 1-4). Para(4) indicates the maximum constraint violation that can be tolerated before the optimization will terminate (default 1e-7).

Gradient information, if available, need only be supplied when para(1)=2. The first column of grad should contain the gradient of f(X) ; the remaining columns should contain the gradients of $G(X)$. To change default settings and for more information refer to Tutorial.

Lower and upper bounds on the design variables are set using the optional variables vlb, vub which may also be empty. Equality constraints should be put in the first few elements of g and para(13) should be set with the number of them (see Tutorial).

Example:

Find values of X which minimize: $-x_1x_2x_3$,

subject to the constraints: $-x_1 - 2x_2 - 2x_3 \le 0$,

$x_1 + 2x_2 + 2x_3 \le 7$ starting at the point X=[10,10,10]

```
PARA=0;                                          %Reset Optimization Parameters
X=[10,10,10];                                    %Initialize Design Variables
while PARA(1)~=1                                 %Check Termination Parameter
 F =-X(1)*X(2)*X(3);                             %Evaluate F
 G(1)=-X(1)-2*X(2)-2*X(3);                       %Evaluate Constraints
 G(2)= X(1)+2*X(2)+2*X(3)-72;
 [X,PARA]=constr(X,F,G,PARA);                    %Call Optimizer
end
```

This program is contained in the *script file* testconstr1.m and gives the following solution after 39 iterations:

```
F =
  -3.4560e+03
G =
  -7.2000e+01  -3.8654e-12
X =
   2.4000e+01   1.2000e+01   1.2000e+01
```

Algorithm:

Constr uses a Sequential Quadratic Programming(SQP) method. In this method a Quadratic Programming (QP) sub-problem is solved at each iteration. An estimate of the Hessian of the Lagrangian is updated at each iteration using the BFGS formula (see unconstr ref.[3-6]). A line search is performed using a merit function similar to that proposed by Han[1] and Powell[2,3]. The QP sub-problem is solved using an active set strategy similar to that described in Gill and Murray[4].

Limitations:

The function to be minimized and the constraints must be continuous. Constr may only give local solutions.

See Also:

setpara,optimglob

References:

[1] Han, S.P, *A Globally Convergent Method For Nonlinear Programming* J. of Optimization theory and Applications Vol 22. 1977 pp.297.

[2]Powell, M.J.D., *The Convergence Of Variable Metric Methods For Nonlineary Constrained Optimization Calculations,* in Nonlinear Programming 3, ed. O.L.Mangasarian, R.R. Meyer and S.M.Robinson (Academic Press) 1978.

[3]Powell, M.J.D. *A fast algorithm for nonlineary constrained optimization calculations,* Numerical Analysis, ed. G.A. Watson, Lecture Notes in Mathematics, Springer Varleg, Vol. 630, 1978.

[4]Gill, P.E., Murray, W., and Wright M.H. *Practical Optimization,* Academic Press, London, 1981.

Purpose:

Solves the Multi-Objective Goal Attainment[1] problem.

Synopsis:

```
[x,para]=attaingoal(x,f,goal,w,para)
[x,para]=attaingoal(x,f,goal,w,para,grad)
[x,para]=attaingoal(x,f,goal,w,para,vlb,vub)
[x,para]=attaingoal(x,f,goal,w,para,vlb,vub,grad)
```

Description:

Attempts to make a matrix of objectives, $F(X)$, attain a matrix of *goal values*, *GOAL*, by solving the problem:

$$\min_{X,\gamma} \gamma$$

$$f_{ij}(X) - w_{ij}\cdot\gamma \le goal_{ij} \qquad i=1,...,m_1 \; j=1,...,m_2$$

$$vlb_{kl} \le x_{kl} \le vub_{kl} \qquad k=1,...,n_1 \; l=1,...,n_2$$

The objectives, $F(X)$, may not reach the required goals in *GOAL* (under-attainment) or may be better than the goal values (over-attainment). The amount of under- or over-attainment can be controlled by setting the variable W. If the objectives are desired to be less than the objectives, then set W=abs(*GOAL*). If it is desired for the objectives to be greater than the goals, then set W=-abs(*GOAL*). This will ensure the same percentage under or over-attainment of the active objectives. For hard constraints set w_i=0.

If it is desired for a number of the objectives to be in the neighbourhood (or equal) to the goals then set PARA(13) with the number of objectives for which this is required. Such objectives should be partitioned into the first few element of F. W should be set to *GOAL* (or *-GOAL*) which will ensure the same percentage of over or under-attainment over the required values. The variables *GOAL*, W, $F(X)$, may be scalars, vectors or matrices of equal size, they should be supplied to attaingoal on an iterative basis. Para(8) contains the value of γ.

Attaingoal returns a value of para(1)=1 when the optimization has terminated following sufficient convergence or when the number of iterations exceeds para(14). Para(2) is a measure of the precision required of X before the optimization will terminate (default 1e-4). Para(3) is a measure of the precision required of the objective function (para(8)) at the solution. Para(4) indicates the maximum constraint violation that can be tolerated as a function of γ before the optimization will terminate (default: 1e-7).

If the optional variable grad is not supplied, gradients are calculated using a finite differences approximation. Set para(1)=0 on first iteration or to re-initialize. To change default settings, such as termination criteria, refer to the Tutorial. Gradient information, if available, need only be supplied when para(1)=2. The first columns of grad contain the gradients for the respective elements of $F(X)$ with respect to X. Lower and upper bounds on the design variables are set using the optional variables vlb, vub which may also be empty.

Example:

A system requires its eigenvalues to lie on the real axis in the complex plane to the left of the points [-1,-3,-3]. A proportional output feedback controller is to be designed with restrictions on the control gains not to exceed a value of 4 or be less than -4. The plant is a 2-input 2-output, open loop unstable system which is given in terms of a state space description (*A*,**B**,**C** matrices). A set of *goal values* for the closed loop eigenvalues are initialized as,*GOAL*=[-1,-2,-3]. To ensure the same percentage under or over attainment in the active objectives at the solution the weighting matrix, *W*, is set equal to abs(*GOAL*). Starting with a controller, *X*=[0,0;0,0], the problem is coded as follows.

```
PARA(1)=0;                                                    %Initialize
optimglob                                  %Use global variables for faster execution
A=[-0.5 0 0; 0 -2 10; 0 1 -2]; B=[1 0; -2, -2; 0 1]; C=[1 0 0; 0 0 1];   %Plant Matrices
X=zeros(2);                                          %Initialize controller matrix
GOAL=[-1,-2,-3]                                  %Set goal values for the eigenvalues
WEIGHT=abs(GOAL)          % Set W to give same percentage under or over-attainment
VLB=-2*ones(X); VUB=2*ones(X);                       %Set upper and lower bounds
while PARA(1)~=1                                  %Check Termination Parameter
 F=sort(eig(A+B*X*C))';                                  %Evaluate Objectives
 [X,PARA]=attaingoal(X,F,GOAL,W,PARA,VLB,VUB);           %Call Optimizer
end
```

This program is contained in the *script file* testattaingoal1.m and gives the following solution after 85 iterations.

```
The attainment factor PARA(8) =.-0.3865
F =
  -6.9313   -4.1588   -1.4099
x =
   -4.0000   -0.2564
   -4.0000   -4.0000
The set of active constraints is:
        1     2
```

Discussion

The attainment factor indicates that each of the objectives has been over-achieved by at least 38.63% over the original design goals. The set of active constraints indicates those objectives which are barriers to further improvement and for which the percentage over-attainment is met exactly.

In the above design the optimizer tries to make the objectives lees than the goals. For a worst case problem in which it is desired for the objectives to be as near as possible to the goals then PARA(13) should be set with the number of objectives for which this is required.

Consider the above problem where it is desired that the eigenvalues be equal to the goal values. A solution to this problem is found by adding a line to the beginning of the above program:

```
PARA(13)=3;
```

On execution of this program the following results were obtained after 49 iterations.

```
The attainment factor PARA(8) =4.0409e-23
F =
  -1.0000   -3.0000   -5.000
x =
   -1.5785    1.2185
   -0.4028   -2.9215
```

In this case the objectives have tried to match the goals. The attainment factor of 4.0409e-23 indicates that the goals have been matched exactly (within a tolerance of 4.40409e-21%).

Notes

These types of problem are often non-convex and the solution is dependent on the starting values given for the variable X. When the objectives and goals are complex then attaingoal tries to achieve the goals in a least squares sense.

Algorithm:

Attaingoal uses the same algorithms as constr with modifications similar to those described in [3]. The choice of merit function is set using PARA(7). The default is to use the merit function of Han[2] and Powell[3]. An exact merit function together with a modified Hessian (see [5] and [6]) can be used by setting PARA(7)=1. The exact merit function method tends to be more robust than the method proposed by Han and Powell but suffers from slower convergence in a number of examples.

Limitations:

The objectives must be continuous. Attaingoal may only give local solutions.

See Also:

setpara, optimglob.

References:

[1]Gembicki, F.W., *Vector Optimization for Control with Performance and Parameter Sensitivity Indices'*, Ph.D. Dissertation, Case Western Reserve Univ., Cleveland, Ohio, USA, 1974.

[2] Han, S.P, *A Globally Convergent Method For Nonlinear Programming* J. of Optimization theory and Applications Vol 22. 1977 pp.297.

[3]Powell M.J.D. *A fast algorithm for nonlineary constrained optimization calculations,* Numerical Analysis, ed. G.A. Watson, Lecture Notes in Mathematics, Springer Varleg, Vol. 630, 1978.

[4]Fleming, P.J., and Pashkevich, A.P, *Computer Aided Control System Design using a Multi-Objective Optimisation Approach'*, Control '85 conference, Cambridge UK, pp. 174-179.

[5]R.K.Brayton, S.W.Director, G.D.Hachtel, an L.Vidigal, *A new algorithm for statistical circuit design based on quasi-Newton methods and function splitting,* IEEE Trans. Circuits Syst., Vol. CAS-26, pp. 784-794, Sept. 1979.

[6]Grace A.C.W., *Computer-Aided Control System Design using Optimization Techniques,* Ph.D. Thesis, Univ. Of Wales, Bangor. 1989.

Purpose:

Solves minimax optimization problems.

Synopsis:

```
[x,para]=minimax(x,f,g,para)
[x,para]=minimax(x,f,g,para,grad)
[x,para]=minimax(x,f,g,para,vlb,vub)
[x,para]=minimax(x,f,g,para,vlb,vub,grad)
```

Description:

Attempts to minimize the worst case values of the matrix, $F(X)$, by varying X:

$$\min_X \{\max (f_{de}(X)\} \qquad d=1,...,p_1 \; e=1,...,p_2$$

$$g_{ij}(X) \le 0, \qquad i=1,...,m_1 \; j=1,...,m_2$$

$$\text{vlb}_{kl} \le x_{kl} \le \text{vub}_{kl} \qquad k=1,...,n_1 \; l=1,...,n_2$$

Values of the objective matrix, $F(X)$, and constraint matrix, $G(X)$, must be supplied to constr on an iterative basis. Values of X are returned at each iteration and new values of $F(X)$ and $G(X)$ must be evaluated. X, $F(X)$, $G(X)$ may be a scalars, vectors or matrices, $G(X)$ may be the empty matrix.

If it is required to minimze the worst case absolute value of F, {i.e. minimax abs{$F(X)$} }then set PARA(13) with the number of objectives for which this is required. Such objectives should be partitioned into the first few element of $F(X)$.

Upon initialization para(1) should be set to 0. If other values for para are not supplied then minimax returns default parameters (see Tutorial).Minimax returns a value of para(1)=1 when the optimization has terminated following sufficient convergence or when the number of iterations exceeds para(14). Para(2) is a measure of the precision required of X before the optimization will terminate (default: 1e-4). Para(3) is a measure of the precision required of the objective function at the solution. Para(4) indicates the maximum constraint violation that can be tolerated before the optimization will terminate (default: 1e-7).

Gradient information, if available, need only be supplied when para(1)=2. The first columns of grad should contain the gradients of $F(X)$; the remaining columns should contain the gradients of $G(X)$. To change default settings and for more information refer to the Tutorial. Lower and upper bounds on the design variables are set using the optional variables vlb, vub. Equality constraints should be put in the first few elements of g and para(13) should be set with the number of them (see Tutorial).

Example:

Find values of X which minimize the maximum value of $[f_1,f_2,f_3,f_4,f_5]$

where $f_1=2x_1^2+x_2^2-48x_1-40x_2,+304$, $f_2=-x_1-3x_2$, $f_3=3x_2+x_1-18$, $f_4=-x_1-x_2$, $f_5=x_1+x_2-8$

starting at the point x=[0,0]

```
PARA=0;                                          %Reset Optimization Parameters
X=[0.1,0.1],                                     %Initialize Design Variables
while PARA(1)~=1                                 %Check Termination Parameter
 F(1)=2*X(1)^2+X(2)^2-48*X(1)-40*X(2)+304;       %Evaluate Objectives
 F(2)=-X(1)-3*X(2);
 F(3)=X(1)+3*X(2)-18;
 F(4)=-X(1)-X(2);
 F(5)=-8+X(1)+X(2);
 [X,PARA]=minimax(X,F,[],PARA);                  %Call Optimizer
end
```

This program is contained in the *script file* testminimax1.m and gives the following solution after 29 iterations.

```
F =
    0.0000 -16.0000 -2.0000 -8.0000 0.0000
X =
    4.0000 4.0000
```

Notes

The worst case absolute values of the elements of F can be minimized by setting PARA(13) with the number of elements for which this is required. They should be partitioned into the first few elements of F.

For example consider the above problem in which it is required to find values of X which minimize the maximum absolute value of $[f_1,f_2,f_3,f_4,f_5]$. This is solved by adding as the first line

```
PARA(13)=5;
```

On execution of this program the following results were obtained after 39 iterations.

```
F =
  10.7609 -10.7609  -7.2391  -9.4382   1.4382
x =
   8.7769   0.6613
```

Algorithm:

Attaingoal uses a Sequential Quadratic Programming algorithm as for constr. The choice of merit function is changed by setting PARA(7). The default is to use the merit function of Han[2] and Powell[3]. An exact merit function together with a modified Hessian (see [1]) can be used by setting PARA(7)=1. The exact merit function method tends to be more robust than that proposed by Han and Powell but suffers from slower convergence in a number of examples.

Limitations:

The function to be minimized must be continuous. Minimax may only give local solutions. Minimax does not allow equality constraints to be expressed.

See Also:

setpara, optimglob.

References:

[1]R.K.Brayton, S.W.Director, G.D.Hachtel, an L.Vidigal, *A new algorithm for statistical circuit design based on quasi-Newton methods and function splitting*, IEEE Trans. Circuits Syst., Vol. CAS-26, pp. 784-794, Sept. 1979.

[2]Madsen K. and Schjaer-Jacobsen H., *Algorithms for worst case tolerance optimization*, IEEE Trans. Circuits and Systems, Vol. CAS-26, Sept 1979.

Purpose:

Solves Non-linear Equations.

Synopsis:

```
[x,para]=solve(x,f,para)
[x,para]=solve(x,f,para,grad)
```

Description:

Finds roots of algebraic non-linear equations of the form:

$$f_{ij}(X) = 0 \qquad i=1, ..., m_1 \;\; j=1, ..., m_2$$

Values of $F(X)$ must be supplied to solve on an iterative basis. Values of X are returned at each iteration and new values of $F(X)$ must be evaluated. X and $F(X)$ may be a scalars, vectors or matrices.

Upon initialization para(1) should be set to 0. If other values for para are not supplied then solve returns default parameters (see Tutorial).

Solve returns a value of para(1)=1 when a root has been found or when the number of iterations exceeds para(14). Para(3) indicates the precision with which a root is required (default: 1e-7).

Gradient information, if available, need only be supplied when para(1)=2. The columns of grad should contain the gradients (partial derivatives) of $F(X)$ for each element of X. If the optional variable grad is not supplied gradients are calculated using a finite differences approximation.

Example:

Find a matrix X which satisfies the equation; $X*X*X$= [1, 2 ; 3, 4]; starting at the point X=[1,1;1,1].

```
PARA=0;                                  %Reset Optimization Parameters
X=ones(2);;                              %Initialize Design Variables
while PARA(1)~=1                         %Check Termination Parameter
 F=X*X*X-[1,2;3,4];                      %Evaluate Non-linear Equation
 [X,PARA]=solve(X,F,PARA);               %Call Optimizer
end
```

This program is contained in the *script file* testsolve1.m and gives the following solution after 75 iterations:

```
A root has been found to the tolerance  = 4.4239e-10

X =
 -1.2915e-01   8.6022e-01
  1.2903e+00   1.1612e+00
```

Limitations:

The function to be solved must be continuous. Solve only gives one root if successful. Solve may converge to a non-zero point in which case other starting values should be tried.

Algorithm:

The choice of algorithm is made by setting para(5). The default algorithm (para(5)=0) is the Levenberg-Marquardt method. Other Least Squares methods can be chosen by setting para(5) as given in the leastsq reference section. Setting para(5)=5 implements a minimax method.

See Also:

setpara, optimglob

Purpose:

Solves Non-linear least squares optimization problems.

Synopsis:

```
[x,para]=leastsq(x,f,para)
[x,para]=leastsq(x,f,para,grad)
```

Description:

Minimizes a non-linear function composed of squared terms:

$$\min_X \{ \sum f_{ij}(x)^2 \} \qquad i=1,...,m_1 \; j=1,...,m_2$$

Values of $F(X)$ must be supplied to solve on an iterative basis. Values of X are returned at each iteration and new values of $F(X)$ must be evaluated. X and $F(X)$ may be a scalars, vectors or matrices. Upon initialization para(1) should be set to 0. If other values for para are not supplied then leastsq returns default parameters (see Tutorial).

Leastsq returns a value of para(1)=1 when the optimization has terminated following sufficient convergence or when the number of iterations exceeds para(14). The optimization will terminate successfully following convergence if the precision of x at a minimum is within the tolerance given by para(2) (default: 1e-4) and the objective function is estimated to be within the tolerance given by para(3) (default: 1e-4).

Gradient information, if available, need only be supplied when para(1)=2. The columns of grad should contain the gradients (partial derivatives) of $F(X)$ for each element of X. If the optional variable grad is not supplied gradients are calculated using a finite differences approximation.

Example:

Find values of X which minimize $\sum_{i=1}^{10} f(x)^2$: where:

$f(x) = \sum_{i=1}^{10} 2+2i - e^{ix_1} + e^{ix_2}$ starting at the point [0.3, 0.4];

```
PARA=0;                                          %Reset Optimization Parameters
X=[-1,1];                                        %Initialize Design Variables
while PARA(1)~=1                                 %Check Termination Parameter
 for i=1:10; F(i)=                               %Function Evaluations
  for i=1:10; F(i)=2+2*i-exp(X(1)*i)-exp(X(2)*i);end
 [X,PARA]=leastsq(X,F,PARA);                     %Call Optimizer
end
```

This program is contained in the *script file* testsolve1.m and gives the following solution after 33 iterations:

```
The sum of squares = 124.3622
X =
    0.25783  0.25783
```

Limitations:

The function to be minimized must be continuous. Leastsq may only give local solutions.

Algorithm:

The choice of algorithm is made by setting para(5). The default is the Levenberg-Marquardt method [1-3]. Setting para(5)=1 implements a Guass-Newton method (see for example[4]) . Setting para(5)=2 implements an unconstrained optimization method.

See Also:

setpara, optimglob

References:

[1]Levenberg K., *A method for the solution of certain problems in least squares*, Quart. Appl. Math. 2, pp. 164-168, 1944.

[2]Marquardt D., *An algorithm for least-squares estimation of nonlinear parameters*, SIAM J. Appl. Math. Vol 11, pp. 431-441, 1963.

[3]More J.J., *The Levenberg-Marquardt algorithm: implementation and theory*, Numerical Analysis, (G. A. Watson, ed.) Lecture Notes in Mathematics 630, Springer-Varleg, pp. 105-116, 1977.

[4]Dennis J.E., Jr. *Nonlinear Least Squares*, State of the Art in Numerical Analysis (D. Jacobs, ed.), Academic Press. pp. 269-312, 1977

Purpose:

Solves Linear Programming(LP) problems.

Synopsis:

```
[x,para]=lp(F,A,B)
```

Description:

Solves the linear programming problem:

$$\min_{\mathbf{x}} \{ \boldsymbol{f}^{\mathrm{T}}\boldsymbol{x} \}$$

$$\boldsymbol{A}.\boldsymbol{x} \le \boldsymbol{b} .$$

Where $\boldsymbol{f}$ is a the vector of coefficients of the linear objective function. The matrix, $\boldsymbol{A}$, and vector, $\boldsymbol{b}$, are the coefficients of the linear constraints. The vector, $\mathbf{x}$, is the set of design variables.

Example:

Find values of $\mathbf{x}$ which minimize: $-5x_1-4x_2-6x_3$

subject to: $x_1-x_2-x_3 \le 20$, $3x_1+2x_2+4x_4 \le 42$, $3x_1+2x_2+4x_2 \le 30$, $x_1,x_2,x_3 \ge 0$

Enetering the following commands

```
F=[-5,-4,-6]
A=[1 -1 1
  3  2 4
  3  2 0
 -1  0 0
  0 -1 0
  0  0 -1]
b=[20;42;30;0;0;0]
x=lp(F,A,B)
```

gives the solution:

```
X =
     0 15.0000 3.000
```

Algorithm:

lp uses a variation of the qp algorithm.

Purpose:

Solves Quadratic Programming(QP) problems.

Synopsis:

```
[x]=qp(H,c,A,B)
[x,lambda]=qp(H,c,A,B)
```

Description:

Solves the Quadratic Programming problem:

$$\min_{x} \{ x^T Hx + cx \}$$
$$Ax \leq b$$

Where the Hessian matrix, H, and vector, c, are the set of coefficients of the quadratic objective function. The matrix, A, and vector, b, are the coefficients of the linear constraints. The vector, x, is a set of design variables.

Example:

Find values of x which minimize: $f(X) = x_1\ x_2 \begin{bmatrix} -1 & 1 \\ 1 & -2 \end{bmatrix} \begin{matrix} x_1 \\ x_2 \end{matrix} - [\,2\ 6\,] \begin{matrix} x_1 \\ x_2 \end{matrix} + 10$

subject to: $x_1+x_2 \leq 2,\ -x_1+2x_2 \leq 2,\ 2x_1+x_1 \leq 3,\ x_1,x_2 \geq 0$

Entering the following commands

```
H=-[-1 1
    1 -2]
c=-[2;6]
A=[1 1
  -1 2
   2 1]
b=[2;2;3]
[X,lambda]=qp(H,c,A,b)
```

gives the solution:

```
x =
    0 15.0000 3.000
```

Algorithm:

Qp uses an active set method (which is also a projection method) similar to that described in Gill and Murray [1]. Another method is implemented in the routine qp2 (with the same arguments as qp) which uses Wolfe's method with a modified Simplex linear programming algorithm [2] (programmed by S.Hancock).

References:

[1]Gill P.E., Murray W., and Wright M.H. *Practical Optimization,* Academic Press, London, 1981.

[2]Wolfe P., *The simplex method for quadratic programming*, Econometrica, Vol. 27 pp.382-398, 1959

Purpose:

Gives help and returns default settings for optimization parameters..

Synopsis:

```
help setpara
para=setpara([])
para=setpara(para)
```

Description:

Setpara returns a vector of default parameters used in the optimization process. Typing help setpara gives details about the Optimization parameters used in the routines. For a fuller description refer of the parameters refer to the Tutorial Section.

Purpose:

Sets up Global Optimization parameters for faster execution and later inspection of the variables.

Synopsis:

optimglob

Description:

Optimglob is a *script file* which sets up a number of global variables used in the optimization routines. The advantage of this is that at each iteration it is no longer necessary to store the variables to an external file. This serves to improve efficiency and allows the inspection of the variables at the end of the optimization cycle.

In order to use global variables enter the command optimglob at the beginning of each session (or in your matlab.m or startup.m file). It is also necessary to use this command after every use of clear. Alternatively this command may be used as the first line in every optimization *script file*

The global variables are as follows:

G_MATL G_MATX G_PCNT G_STEPMIN G_SD G_GCNT G_OLDF G_GRAD G_HOW G_CHG G_LAMBDA G_GLOBFLAG G_LAMBDABEST G_XBEST G_FBEST

There are also a number of global variables associated with graphics facilities they are as follows:

G_MESH G_GPARA G_MDX G_MDY G_GXCNT G_GYCNT G_CONTOURS G_GSX G_GSY G_AXIS G_MESHC G_AXIS2

The variables have been given the prefix G_ to avoid naming confusions in other routines. Of particular interest is the string variable G_HOW which contains a complete history of the optimization cycle.

Example:

The file tesunconstr2.m can be made to use global variables by adding optimglob as the first line in the *script file*:

```
optimglob
PARA=0;
X=[-1,1];
while PARA(1)~=1
  F=exp(X(1))*(4*X(1)^2+2*X(2)^2+4*X(1)*X(2)+2*X(2)+1);
  [X,PARA]=unconstr(F,X,PARA);
end
```

On execution of this program the global variables may be inspected. For example, the variable G_HOW contains a complete history of the optimization cycle: The above example gives the following contents for G_HOW.

G_HOW =

```
ITERCNT   F        STEP     GRADIENT UPDATE STEP-CHANGE
1         1.839
6         1.962    1         0.6565  inter
9         1.724    0.3682    0.006    0.2513  update
12        1.692    0.3682   -0.099   -0.0073 incstep
15        1.565    1.363    -0.1558  -0.057  incstep
18        0.995    3.501    -0.2912  -0.3394 incstep
21        0.5117   7.966     0.7205   5.523   update  inter_st
24        1.254    5.227    -0.4566  red_step
27        2.016    2.613     0.253   inter
30        0.1167   0.6479    0.8461   2.448   update
33        0.05693  0.6479   -0.026    0.0836  update  incstep
36        0.04568  1         0.2182   0.4075  update  inter_st
39        0.0044   0.5579   -0.034    0.04653 update  incstep
42        9.7e-05  1        -0.0011   0.0066  update  incstep
45        1.7e-07  1.051     7.3e-06  0.00018 update  inter_st
48        3.1e-12  1.001     6.3e-10  3.4e-09 update  inter_st
 NO OF ITERATIONS=49
```

Where ITERCNT is the number of iterations; F is the value of the objective function , F(x); STEP is the step length used in the line-search; GRADIENT is the gradient of the new point in the direction of search; UPDATE is a measure of positive-definiteness of the Hessian update. The remaining columns give information regarding the procedures being performed at each stage, such as, updating of the Hessian(update), step-length increase (incstep) or decrease(red_step, inter_step) or cubic interpolation (inter)

Limitations

Optimglob must not be executed from a user-defined function, only form a *script file* or through keyboard entry.

Appendix B

Gradient Calculation and Matrix Values

B.1 CALCULATING GRADIENT MATRICES FOR TRACE FUNCTIONS

The calculation of gradient matrices was inroduced by Athans in [1] and can be alos be found in [2] and [3]. The mehthod is summarized below together with a Table of coomonly used funcuons.

Definition: Let $f(\mathbf{X})$ be a scalar valueed function of the elements x_{ij} of the $n \times m$ matrix $\mathbf{X}$. The gradient matrix, $\partial f(\mathbf{X})/\delta \mathbf{X}$, of $f(\mathbf{X})$ and denoted as $\nabla f(\mathbf{X})$ is the $n \times m$ matrix, the ijth element is defined by

$$\left| \frac{\partial f(\mathbf{X})}{\partial \mathbf{X}} \right|_{ij} = \frac{\partial f(\mathbf{X})}{\partial \mathbf{X}_{ij}} \tag{B.1}$$

Making use of the following idetnties:

$$\mathrm{tr}(\mathbf{A}^T\mathbf{X}) = \mathrm{tr}(\mathbf{X}\mathbf{A}^T) = \mathrm{tr}(\mathbf{A}\mathbf{X}^T) = \mathrm{tr}(\mathbf{X}^T\mathbf{A}) \tag{B.2}$$

and the techniques supplied in [1] to [3], the anumber of gradient matrices were calculated for the a funtion of the form $f(\mathbf{X})=\mathrm{tr}(\mathbf{F}(\mathbf{X}))$, where $\mathbf{F}(\mathbf{X})$ is a square matrix function of the elements x_{ij} of the $n \times m$ matrix $\mathbf{X}$. They are shown in Table B.1.

REFERENCES

[1] Athans M. and Levine W.S., "Gradient matrices and matrix calculatuons," MIT Lincoln Labs., Lexington, Mass., Tech.Note 1965-53, 1965

[2] Athans M., "The matrix minimum principle," Inform. Contr., Vol.11, 1967

[3] Geering H.P., "On calculating gradient matrices," IEEE Trans. on Autom. Control, Vol AC-21, No. 1, pp.615-616, 1976

Table B.1 Gradient Matrices For A Number of Trace Functions

F(X)	$\nabla(tr(F(X))$
AX	A^T
AX^T	A
AXB	A^TB^T
AX^TB	BA
XX	$2X^T$
XX^T	$2X$
$AXBX$	$A^TX^TB^T+B^TX^TA^T$
XAX^T	XA^T+XA
$AXBX^T$	A^TXB^T+AXB
$XABX^T$	$XB^TX^TA^T+A^TX^TXB^T+XAXB$
$AXBXCX^T$	$A^TXC^TX^TB^T+B^TX^TA^TXC^T+AXBXC$
$AXBXCXDX^T$	$A^TXD^TX^TC^TX^TB^T+B^TX^TA^TX^TX^TC^T$ $+C^TX^TB^TX^TA^TX^T+AXBXCXD$
$AXBX^TCX^T$	$A^TXC^TXB^T+CX^TAXB+AXBX^TC$
$AXBX^TCX^TDX^T$	$A^TXD^TXC^TXB^T+CX^TDX^TAXB$ $+DX^TAXBX^TC+AXBX^TCX^TD$
$AXBXCX^TDX^T$	$A^TXD^TXC^TX^TB^T+B^TX^TA^TXD^TXC^T$ $+DX^TAXBXC+AXBXCX^TD$
e^X	e^X
X^{-1}	$-(X^{-2})^T$
$AX^{-1}B$	$-(X^{-1}BAX^{-1})^T$
$(AX)^{-1}$	$-((AX)^{-2}A)^T$
$(AXB)^{-1}$	$-(B(AXB)^{-2}A)^T$
$(A+BXC)^{-1}$	$-(C(A+BXC)^{-2}B)^T$
$D(A+BXC)^{-1}E$	$-(C(A+BXC)^{-1}ED(A+BXC)^{-1}B)^T$
$D((A+BXC)^{-1})^TE$	$-(C(A+BXC)^{-1}D^TE^T(A+BXC)^{-1}B)^T$
$(D(A+BXC)^{-1}E)^{-1}$	$-(CYEZZDYB)^T$ where $Y=(A+BXC)^{-1}$, $Z=(D(A+BXC)^{-1}E)^{-1}$
$(E(A+BXC)^{-1}D(A+BXC)^{-1})^T$	$-B^TY^TEYDF^TC^T-B^TY^TE^TYD^TF^TC^T$ where $Y=(A+BXC)^{-1}$

B.2 CONTROLLER MATRICES (EX. 1, DESIGN NO. 4)

2DF PID Controller

D_{cff} = [2.6079] B_{cff} = [-1.4281] B_{cfb} =[1] A_c=[-0.9583] C_c=[0.7272]

C_{ci}=[0.1] D_c=[-1.3784] C_{ci}=[0.1]

2DF Fifth Order Controller

$$A_c = \begin{bmatrix} 0 & 0 & 0 & -1.4955 \\ 1 & 0 & 0 & -11.9179 \\ 0 & 1 & 0 & -20.4791 \\ 0 & 0 & 1 & -11.1962 \end{bmatrix} \quad B_{cff} = \begin{bmatrix} -3.5311 \\ -1.5529 \\ -0.8727 \\ -0.0929 \end{bmatrix} \quad B_{cfb} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

C_c= [1.5344 -7.3873 57.7516 -512.7458] D_{cff}= [2.4521] D_c= [-1.4094]

B.3 F4C LINEARIZED MODELS

$$A_{p1} = \begin{bmatrix} -4.6150e-01 & -3.6930e-01 & -1.4590e+00 \\ 9.7920e-01 & -4.5350e-01 & -2.9000e-02 \\ 0 & 0 & -2.0000e+01 \end{bmatrix}$$

$$A_{p2} = \begin{bmatrix} -3.1260e+00 & -7.2080e+01 & -6.3480e+01 \\ 1.0000e+00 & -2.1120e+00 & -2.0980e-01 \\ 0 & 0 & -2.0000e+01 \end{bmatrix}$$

$$A_{p3} = \begin{bmatrix} -4.4360e-01 & -1.8030e+00 & -4.9890e+00 \\ 9.8660e-01 & -2.9780e-01 & -4.1100e-02 \\ 0 & 0 & -2.0000e+01 \end{bmatrix}$$

$$A_{p4} = \begin{bmatrix} -3.7180e-01 & -4.2750e+01 & -1.7720e+01 \\ 9.9970e-01 & -4.8400e-01 & -4.1900e-02 \\ 0 & 0 & -2.0000e+01 \end{bmatrix}$$

B.4 GVAM LINEARIZED MODEL

$$A_p = \begin{bmatrix}
0 & 1.0000e+00 & 0 & 0 & 0 \\
-2.5690e+00 & -1.0420e+00 & 1.1860e-04 & -7.6040e-03 & -2.0820e-01 \\
-2.6730e+01 & 5.1230e-03 & -5.1610e-02 & 1.6030e-02 & -1.1730e-05 \\
-1.8370e+02 & -3.3090e+00 & -1.8640e-01 & -5.4360e-01 & -1.0310e+00 \\
0 & 0 & 0 & 0 & -2.0000e+01 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 5.3460e-05 & 0 & 0 \\
0 & 0 & 5.3460e-05 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 4.6160e-02 & 5.2490e-02 & -2.7490e-02 & -2.9040e-06 \\
0 & -2.3950e-02 & 2.2190e+01 & 1.6200e+01 & 3.3760e-03 \\
0 & -7.9150e-02 & -2.7580e+00 & -1.4910e+00 & -3.1070e-04 \\
0 & 0 & 0 & 0 & 0 \\
-1.0000e+01 & 0 & 0 & 0 & 0 \\
0 & -5.0000e+00 & 0 & 0 & 0 \\
0 & 0 & -2.8130e+00 & 2.6830e+00 & 9.2780e-04 \\
0 & 0 & 5.2970e-05 & -1.6420e+00 & 6.2720e-04 \\
3.1390e+04 & 0 & 0 & 0 & -1.3330e+01
\end{bmatrix}$$

$$B_p = \begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
-20 & 0 \\
0 & -10 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0
\end{bmatrix}$$

$$C_p = \begin{bmatrix}
0 & 0 & 0 & -1.6960e-01 & 0 \\
7.6290e-04 & 0 & 5.9220e-01 & 0 & 0
\end{bmatrix}$$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}$$