

Control System Design Applications With Hybrid Genetic Algorithms

By:

**Vito Dirita (BE, MEng, Electrical Engineering)
School of Engineering,
Department of Electrical and Electronic Engineering**

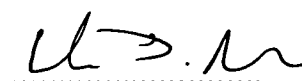
**Submitted in fulfillment of the requirements for the degree of:
Doctor of Philosophy.
University of Tasmania**

November 2002

Statement of Originality:

This thesis contains no material which has been accepted for a degree or diploma by the University or any other institution, except by way of background information and duly acknowledged in the Thesis, and to the best of my knowledge and belief no material previously published or written by another person except where due acknowledgment is made in the text of the Thesis.

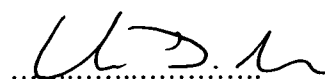
Vito Dirita.

A handwritten signature in black ink, appearing to read 'V.D. Dirita', written over a dotted line.

Authority of Access:

This Thesis may be made available for loan and limited copying in accordance with the Copyright Act 1968.

Vito Dirita.

A handwritten signature in black ink, appearing to read 'V.D. Dirita', written over a dotted line.

Abstract:

This thesis investigates the hybrid application of stochastic and heuristic algorithms, in particular genetic algorithms (GA), simulated annealing (SA) and Greedy search algorithms for the design of linear and nonlinear control systems. We compare the rate of convergence, computational effort required (FLOPS) and ease of implementation. Where possible, results are compared with the more traditional control system design methodologies. Two specific practical applications include aircraft flight control systems, and a nonlinear example of an industrial bioreactor fermentation process.

Stochastic algorithms (GA) and heuristic algorithms (SA, Greedy, Tabu search) are powerful search methods, capable of locating the global minimum or maximum (extremum) of multimodal functions. They operate without the need for function gradients and are robust to noisy data. The current research trend is directed towards the solution to constrained multiobjective optimization problems of multimodal functions which may result in a family of optimal solutions (i.e Pareto optimal set) and game theoretic approaches such as Nash and Stackelberg Equilibria.

Genetic algorithms suffer from one particular drawback, the rate of convergence can be unacceptably slow if accurate solutions are sought. To overcome this deficiency, hybridization of genetic algorithms with fast local search procedures are often used. Two heuristic based search procedures are: *greedy search* and fast *simulated annealing*.

We investigate three types of Hybrid algorithms: (i) genetic algorithms (GA), (ii) hybrid GA + simulated annealing (SA), and (iii) hybrid GA + greedy search. These methods are applied to solving off-line linear and nonlinear control problems which may otherwise have no direct analytical solution. In cases where solutions are obtainable using conventional methods, results are compared with hybrid algorithms. Robustness against modeling errors, nonlinearities, disturbances and parametric uncertainty will also be discussed.

We investigate five specific design applications, these include: training radial basis function (RBF) neural networks, robust eigenstructure assignment (ESA), model reference adaptive control (MRAC), robust mixed H_2/H_∞ design, and lastly fault detection and isolation (FDI).

We show that hybrid algorithms can perform better, can handle a broader class of problems, and have fewer restrictions than conventional methods. Furthermore, stochastic and heuristic methods can directly deal with constraints.

Acknowledgments:

I would like to thank my supervisor Dr. Zhihong Man for his kind assistance and guidance towards this thesis. I would also like to thank Prof. W. F. Budd for proof-reading this manuscript.

This thesis is dedicated to my dear wife Robina and my son Adrian.

Vito Dirita.

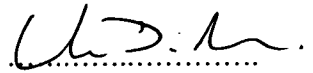
A handwritten signature in black ink, appearing to read 'Vito Dirita', written over a dotted line.

Table of Contents:

Abstract	iii
Acknowledgments	iv
Table of Contents	v
Preface	viii
Abbreviations	xi
 1. Introduction To Optimization:	
1.1 Introduction	p.1.2
1.1.1 Derivative Free Methods.	p.1.3
1.2 Conventional Optimization.	p.1.4
1.2.1 Derivative Free Methods.	p.1.4
1.2.2 First Derivative Methods	p.1.5
1.2.3 Second Derivative Methods	p.1.5
1.3 Stochastic and Heuristic Search Methods.	p.1.6
1.3.1 Evolutionary Programming.	p.1.6
1.3.2 Evolutionary Strategies	p.1.6
1.3.3 Genetic Programming.	p.1.7
1.3.4 Genetic Algorithms	p.1.7
1.3.5 Simulated Annealing.	p.1.8
1.3.6 Greedy Search	p.1.9
1.3.7 Tabu Search	p.1.9
1.4 Genetic Algorithms and Hybrid Methods	p.1.10
1.4.1 Conventional Genetic Algorithms	p.1.10
1.4.2 Hybrid Genetic Algorithms	p.1.15
1.5 Constrained and Multiobjective Optimization	p.1.20
1.5.1 Calculus Based Constrained Single Objective Optimization	p.1.20
1.5.2 Genetic Algorithm Single Objective Constrained Optimization.	p.1.22
1.5.3 Genetic Algorithm Multiobjective Optimization	p.1.22
1.6 Chapter Summary and Conclusion	p.1.26
1.7 References and Further Reading	p.1.27
 2. Training Radial Basis Functions with Hybrid Genetic Algorithms:	
2.1 Introduction	p.2.2
2.1.1 The Radial Basis Function Network	p.2.3
2.1.2 Training Radial Basis Function Networks	p.2.5
2.2 Training RBF Networks with Hybrid Genetic Algorithms	p.2.6
2.2.1 Bioreactor Mathematical Model..	p.2.7
2.2.2 Training with Conventional Methods	p.2.8
2.2.3 Training with Hybrid Genetic Algorithms	p.2.11
2.2.4 Comparison of Results	p.2.16
2.3 Chapter Summary and Conclusion	p.2.18
2.4 References and Further Reading	p.2.21

3. Eigenstructure Assignment Using Hybrid Genetic Algorithms:

3.1 Eigenstructure Assignment	p.3.2
3.1.1 Introduction	p.3.2
3.1.2 Full Eigenstructure Assignment and Moore's Method.	p.3.3
3.1.3 Partial Eigenstructure Assignment	p.3.5
3.1.4 Robust Eigenstructure Assignment.	p.3.6
3.1.5 Response of LTI Systems from Eigenstructure Information	p.3.9
3.2 Partial Eigenstructure Assignment for Static Compensators.	p.3.10
3.2.1 Theory	p.3.10
3.2.2 Simulation 3.1: Fixed Eigenvalues	p.3.13
3.2.3 Simulation 3.2: Domain Constrained Eigenvalues	p.3.15
3.3 Eigenstructure Assignment for Dynamic Compensators	p.3.19
3.3.1 Theory	p.3.19
3.3.2 Simulation 3.3: Static Output Feedback.	p.3.23
3.3.3 Simulation 3.4: Dynamic Control Output Feedback	p.3.25
3.4 Robust Eigenstructure Assignment	p.3.29
3.3.1 Theory	p.3.29
3.3.2 Simulation 3.5: Hybrid Genetic Algorithms.	p.3.30
3.5 Chapter Summary and Conclusion	p.3.34
3.6 References and Further Reading	p.3.36

4. Model Reference Adaptive Control With Hybrid Genetic Algorithms:

4.1 Model reference adaptive control	p.4.2
4.1.1 Introduction	p.4.2
4.2 Model Reference Adaptive Control: SISO Systems	p.4.5
4.2.1 Simulation-4.1: Lyapunov Stability method.	p.4.5
4.2.2 Simulation-4.2: MIT-rule Method	p.4.11
4.2.3 Simulation-4.3: Hybrid Genetic Algorithms	p.4.16
4.2.4 Comparison of Results.	p.4.22
4.3 Model Reference Adaptive Control: MIMO Systems	p.4.24
4.3.1 Simulation-4.4: Lyapunov Stability method.	p.4.24
4.3.2 Simulation-4.5: Gradient based (MIT-rule) Method	p.4.30
4.3.3 Simulation-4.6: Hybrid Genetic Algorithms	p.4.36
4.3.4 Convergence Rates.	p.4.43
4.4 Chapter Summary and Conclusion	p.4.45
4.5 References and Further Reading	p.4.48

5. Mixed H_2/H_∞ Controller Synthesis with Hybrid Genetic Algorithms:

5.1 Introduction	p.5.2
5.1.1 Robust Control Theory.	p.5.3
5.1.2 H_2 Control Theory And State Space Solutions..	p.5.4
5.1.3 H_∞ Control Theory And State Space Solutions.	p.5.7
5.1.4 Mixed H_2/H_∞ Control Theory	p.5.9

5.2 H_2 Controller Synthesis	p.5.11
5.2.1 Simulation Setup	p.5.11
5.2.2 Conventional State Space Solution.	p.5.13
5.2.3 Solution Using Genetic Algorithms	p.5.15
5.2.4 Convergence Rates for Hybrid Genetic Algorithms	p.5.19
5.3 H_∞ Controller Synthesis	p.5.21
5.3.1 Simulation Setup	p.5.21
5.3.2 Conventional State Space Solution.	p.5.22
5.3.3 Solution Using Genetic Algorithms	p.5.24
5.3.4 Convergence Rates for Hybrid Genetic Algorithms	p.5.32
5.4 Mixed H_2/H_∞ Controller Synthesis With Hybrid Genetic Algorithms	p.5.33
5.4.1 Simulation Setup	p.5.33
5.4.2 Solution Using Genetic Algorithms	p.5.35
5.5 Chapter Summary and Conclusion	p.5.41
5.6 References and Further Reading	p.5.42
 6. Fault Detection and Isolation Using Hybrid Genetic Algorithms:	
6.1 Fault Detection and Isolation	p.6.2
6.1.1 Introduction	p.6.2
6.1.2 Fault Detection and Isolation - Survey	p.6.4
6.1.3 Signal Based Methods	p.6.7
6.1.4 Model Based Methods	p.6.7
6.1.5 Observer Based Methods	p.6.9
6.1.6 Modeling Faults in Systems, Residual Generation.	p.6.12
6.1.7 Parity Space Methods - Theory	p.6.15
6.2 Detecting Faults: Hybrid Genetic Algorithms	p.6.17
6.2.1 Theory	p.6.17
6.2.2 Detecting Input/Output Faults in Linear Systems	p.6.20
6.2.3 Detecting Internal Faults:	p.6.29
6.3 Chapter Summary and Conclusion	p.6.31
6.4 References and Further Reading	p.6.32
 7. Summary and Conclusions.	
7.1 Conclusion	p.7.2
7.2 Challenges and Future Development	p.7.10
7.3 References and Further Reading	p.7.14
 8. Appendix.	
8.1 Aircraft Mathematical Model	p.8.2
8.2 Partial Eigenstructure Assignment	p.8.6
8.3 Bioreactor Mathematical Model	p.8.8
8.4 Hooke-Jeeves Search Flowchart.	p.8.10

Preface:

Ever since the inception of evolutionary programming and genetic algorithms by Holland in 1962, genetic algorithms have found wide acceptance in many fields such as combinatorial optimization, artificial intelligence, system identification and control. Genetic algorithms are a robust optimization technique capable of locating the global extremum of complex multimodal functions. Current research in genetic algorithms include constrained and unconstrained optimization, and multiobjective optimization. In many instances, single solutions to multiobjective optimization problems do not exist, and instead a family of solutions exists, this is known as a *Pareto optimal set*. Genetic algorithms do not require function gradients, but rather deal directly with the cost function to be optimized. This has the added advantage of being able to handle complex nonlinear cost functionals, or where the gradients are discontinuous or undefined, for instance: image classification problems.

Genetic algorithms can be applied to either *on-line* or *off-line* control problems. *Off-line* design of control systems can be applied to a wider range of optimization problems, for instance mixed H_2/H_∞ Multi Input Multi Output (MIMO) designs using reduced order compensators in which no direct design method currently exists, and other applications such as partial eigenstructure assignment with constraints. With genetic algorithms, there are no restrictions, the plant may be nonlinear, the controller may be linear, nonlinear, fuzzy or neural control based. Self tuning of controller parameters can be realized by a genetic algorithm which attempts to optimize some performance function (e.g. Linear Quadratic Regulator cost function) from the plant input and output measurements. This leads to several important issues of how to ensure internal plant stability and convergence of the genetic algorithm. There are two serious limitations which need to be resolved when dealing with genetic algorithms:

1. Because a genetic algorithm search is stochastic, there is no method currently available to guarantee their convergence. This is a serious limitation which needs to be addressed if genetic algorithms are to gain wider acceptance in *on-line* control applications.
2. Genetic Algorithms constitute a family of powerful global search and optimization algorithms which can deal with multimodal functions containing many local minima. Nevertheless, genetic algorithms can become excessively slow in their final stages of convergence, once a minimum has been found.

To obtain accurate solutions (with many decimal places), the genetic algorithm is inefficient. One way to overcome this problem would be to combine the genetic algorithm with a fast local search procedure. Once the minimum has been found by the GA search, the fast local search is then used to quickly converge the solution to the desired accuracy.

In this thesis, we address the second issue, combining genetic search with a fast local search to improve convergence properties of the hybrid algorithm. Fast local search procedures are also known as *hill-climbing* methods. Thus hybrid methods (also known as *genetic local search*) combine the reliability and robustness properties of the genetic algorithm and their original search heuristics with the accuracy and fast convergence of local search methods.

We investigate three types of Hybrid algorithms: (i) genetic algorithms (GA), (ii) hybrid GA + simulated annealing (SA), (iii) hybrid GA + greedy search. These methods are applied to solving off-line linear and nonlinear control problems which may otherwise have no direct analytical solution. In cases where solutions are obtainable using conventional methods, results are compared with hybrid algorithms.

The full potential of genetic algorithms is yet to be realized in the area of control, and in particular intelligent control and expert systems. In this thesis we investigate some applications of genetic algorithms in control. Each chapter deals with one specific area of control and where possible, comparison is made between conventional methods with solutions using genetic algorithms.

Organization of Thesis:

Chapter 1: Begins with an introduction to optimization, including evolutionary computation, genetic algorithms, simulated annealing, greedy algorithms, Tabu search, constrained optimization and multiobjective optimization using calculus based techniques as well as genetic algorithm based techniques.

Chapter 2: Discusses applications of genetic algorithms in training radial basis function networks. The example used is a model matching problem of a nonlinear system, often found in control system applications.

Chapter 3: Discusses applications of genetic algorithms in the design of robust eigenstructure controllers with partial eigenstructure specifications. Simulation results comparing conventional methods with GA are discussed. Simulation results including full state feedback and measurement feedback using dynamic compensators are given.

Chapter 4: We apply GA to solving model reference adaptive control problems, with constraints and multiple objectives. As seen by the results, genetic algorithms perform better than conventional MIT and Lyapunov based methods and require fewer assumptions to implement.

Chapter 5: We apply GA to solving mixed H_2 / H_∞ control problems, results are compared with conventional state space solutions. Full order dynamic compensators and reduced order compensators are described. The objective function is to minimize sensitivity norms (from disturbance to performance outputs) and maximize robustness against model uncertainties. Simulation results using the linear aircraft model is provided.

Chapter 6: This is a survey chapter on different types of fault detection and isolation. We show that fault detection based on GA outperforms conventional fault detection methods such as the widely accepted parity space technique. We also show that fault detection in linear and nonlinear systems is also possible with GA.

Chapter 7: Discussion and Conclusion.

Chapter 8: Appendix.

Each chapter is self contained, comprising of an introduction, theoretical background, simulation results, discussion, conclusion, and references. This individual chapter format should hopefully facilitate reading.

Abbreviations:

AI	Artificial Intelligence	RBF	Radial Basis Function
ANN	Artificial Neural Network	SA	Simulated Annealing
ART	Adaptive Resonance Theory	SISO	Single Input Single Output
BP	Back Propagation	SPRT	Sequential Probability Ratio Testing
BFGS	Broyden Fletcher Goldfarb Shanno	STR	Self Tuning Regulator
DFP	Davidon Fletcher Powell	TS	Tabu Search
EA	Evolutionary Algorithms	UIO	Unknown Input Observer
ECP	Equality Constrained Problem	VSO	Variable Structure Observer
EKF	Extended Kalman Filter		
EP	Evolutionary Programming		
ES	Evolutionary Strategies		
ESA	Eigen-Structure Assignment		
ETA	Event Tree Analysis		
FDF	Fault Detection Filter		
FDI	Fault Detection and Isolation		
FLC	Fuzzy Logic Controller		
FSM	Finite State Machine		
FTA	Fault Tree Analysis		
GA	Genetic Algorithms		
GP	Genetic Programming		
GS	Greedy Search		
ICP	Inequality Constrained Problem		
LMI	Linear Matrix Inequalities		
LQR	Linear Quadratic Regulator		
LTI	Linear Time Invariant		
MIMO	Multi Input Multi Output		
MLP	Multi Layer Perceptron		
MOP	Multiobjective Optimization Problem		
MRAC	Model Reference Adaptive Control		
MVIM	Multi Valued Influence Matrices		
PID	Proportional Integral Derivative		

1

Introduction To Optimization

Contents:

1.1	Introduction	p.1.2
1.1.1	Objectives	p.1.3
1.2	Conventional Optimization.	p.1.4
1.2.1	Derivative Free Methods.	p.1.4
1.2.2	First Derivative Methods	p.1.5
1.2.3	Second Derivative Methods	p.1.5
1.3	Stochastic and Heuristic Search Methods.	p.1.6
1.3.1	Evolutionary Programming.	p.1.6
1.3.2	Evolutionary Strategies	p.1.6
1.3.3	Genetic Programming	p.1.7
1.3.4	Genetic Algorithms	p.1.7
1.3.5	Simulated Annealing.	p.1.8
1.3.6	Greedy Search	p.1.9
1.3.7	Tabu Search	p.1.9
1.4	Genetic Algorithms and Hybrid Methods	p.1.10
1.4.1	Conventional Genetic Algorithms	p.1.10
1.4.2	Hybrid Genetic Algorithms	p.1.15
1.5	Constrained and Multiobjective Optimization	p.1.20
1.5.1	Calculus Based Constrained Single Objective Optimization	p.1.20
1.5.2	Genetic Algorithm Single Objective Constrained Optimization	p.1.22
1.5.3	Genetic Algorithm Multiobjective Optimization	p.1.22
1.6	Chapter Summary and Conclusion	p.1.26
1.7	References and Further Reading	p.1.27

1.1 Introduction:

This introductory chapter provides an initial background to the subject of optimization. Topics covered include: calculus based (or conventional optimization), heuristic and stochastic search, hybrid search methods, constrained and multiobjective optimization.

For the first part of the chapter, a brief discussion of conventional calculus based optimization is provided. Calculus based optimization falls into three main categories: (i) derivative free or pattern search methods such as the Nelder-Mead Simplex method [42], (ii) first derivative or gradient based methods such as gradient descent or conjugate gradient in which the gradient of the function must be known or estimated, and (iii) second derivative or variable metric (also known as Newton or Quasi-Newton) methods in which the Hessian matrix must be known or approximated. All three methods differ in complexity and convergence. Derivative free methods are the simplest to implement, but result in unacceptably slow rates of convergence. Second derivative methods have a greater rate of convergence, however at the expense of computational complexity. All these methods can only locate the local extremum of a function (local convergence characteristics).

The second part discusses stochastic and heuristic search techniques. Stochastic methods are probabilistic based search methods which include evolutionary computation [1]: evolutionary programming, evolutionary strategies, genetic programming, genetic algorithms and simulated annealing [72]. Heuristic search methods include: greedy algorithms [52] and Tabu search [56]. Stochastic search methods have global convergence characteristics, but can suffer from slow final convergence, while greedy algorithms and Tabu search have rapid final convergence.

In the third part, hybrid search methods are discussed. Hybrid methods generally combine two or more individual search techniques such that the resulting algorithm has superior convergence properties when compared to either individual methods. For instance, evolutionary computation has been combined with gradient based optimization to utilize the global search capability of evolutionary computation with the fast local convergence properties of the gradient based method.

Lastly, a discussion of multiobjective optimization and constrained optimization is briefly outlined. Multiobjective optimization using genetic algorithms, Pareto optimality, population Niching methods and Nash equilibria are discussed. Constrained optimization using penalty and repair functions are also described.

1.1.1 Objectives:

Before proceeding any further, a brief summary of the main objectives of this thesis is provided below:

1. To investigate potential applications of genetic and hybrid genetic algorithms to the design and synthesis of control systems, and to compare with the more traditional and conventional control system design methods. In particular, the following areas are investigated: (i) training neural networks to model nonlinear systems, (ii) robust eigenstructure assignment, (iii) model reference adaptive control, (iv) robust H_2 and H_∞ , and compensators with mixed H_2/H_∞ design objectives, and lastly (v) fault detection and isolation.
2. To show that genetic algorithms can converge rapidly, have fewer restrictions and can solve a wider range of control problems, including constrained and multiobjective problems, which may otherwise have no direct solution with conventional control design techniques.
3. Whilst genetic algorithms have powerful global search capability, they can sometimes suffer from slow final convergence once a solution is found. To overcome this problem, hybrid methods have been developed. To investigate hybrid GA methods by combining the global search capability of genetic algorithms with the convergence properties of a fast local search heuristic, without resorting to gradient or Hessian matrix computation. These methods can include: derivative free techniques, see chapter 1.2.1, or heuristic based such as Tabu search (section 1.3.7), Greedy search (section 1.3.6) or a fast Simulated Annealing (section 1.3.5). The three methods chosen are: (i) conventional genetic algorithms, (ii) genetic algorithms and simulated annealing, (iii) genetic algorithms and greedy search.
4. To show that hybrid genetic algorithms are more effective stochastic based search and optimization methods compared to conventional genetic algorithms.
5. To show that the use of floating point chromosomal codification can be readily and directly applied to control system applications.
6. To investigate adaptive control using hybrid genetic algorithms, and to compare results with traditional Lyapunov based stability and gradient based (MIT-rule) methods.

1.2 Conventional Optimization

Conventional optimization, also known as calculus based optimization, approximates the function to be minimized (or maximized) by a first or second order Taylor series expansion. Derivative free methods do not require a Taylor series approximation. Note that all these methods discussed are also known as *hill-climbing* methods.

1.2.1 Derivative Free Methods:

Derivative free methods, also known as direct search or pattern search techniques, do not require knowledge nor approximation of the function gradient. The most popular is the *Nelder-Mead Simplex* method [42] in which a simplex (tetrahedron) is defined consisting of $(n+1)$ vertices, where n is the number of dimensions of the function. At each iteration, the shape of the simplex changes according to the shape of the local landscape, gradually moving down towards into the valley of the function to be minimized. This adaptation process is achieved by three steps: *reflection*, *expansion* and *contraction*. Only several function evaluations are required for each iteration, however convergence is slow, requiring many iterations. This method is very robust and works well if the number of variables n does not exceed five or six. Convergence properties of the Nelder and Mead simplex have been described in [43]. Implementations in MATLAB® (optimization toolbox) and Numerical Recipes is also available.

Another effective method is *Powell's Method* [44, 45, 49]. Powell's method starts with a single initial point and search direction. At each iteration, n line minimizations must be performed, one for each direction, and a new search direction is obtained. A new (better) point is obtained by summing the old point and the search direction thus: $x_{k+1} = x_k + d_k$. Line minimization using a golden search or quadratic fit search is often used. Powell's method converges in fewer iterations compared to Nelder and Mead Simplex, works well with functions of up to twenty variables, but requires a line search minimization.

The last method known as the *Hooke and Jeeves* algorithm [46] starts with a single initial point and a search span range Δ_k . At each iteration, it operates in two steps or moves: *exploratory* and *pattern* moves, whereby the span range Δ_k is gradually reduced. A better point is then given by $x_{k+1} = x_k + \Delta_k$. The algorithm terminates when the magnitude of Δ_k is below a predefined value. Several other derivative free optimization methods exist including: Rosenbrock's algorithm [47], and Fletcher [48]. All these methods are limited to local search (local extrema) of a function, and convergence is generally slow and dependent on the initial starting point and shape of the function.

1.2.2 First Derivative Methods:

First derivative methods require the knowledge of the function gradient. The simplest, although generally not recommended, is the method of *steepest descent*, also known as *gradient descent*. Given an initial estimate x_k , the next iteration x_{k+1} gives a better estimate from: $x_{k+1} = x_k - \alpha \cdot \partial f / \partial x_k$, where α is a step size and $\partial f / \partial x_k$ is the gradient vector. The step size α can be a constant or can be found by a line minimization procedure by minimizing: $f(x_k - \alpha \cdot \partial f / \partial x_k)$ using a golden section or a quadratic fit search. The problem with steepest descent is that it will perform many small steps in going down a long, narrow valley even if the valley is a quadratic function. A more effective procedure however is to use the method of *conjugate gradients* [49]. This procedure also requires a line minimization and gradient calculation at each iteration. The method avoids the pitfalls of gradient descent by ensuring that at each new iteration, the next direction is conjugate to the previous. Thus for a quadratic function, only two steps are necessary to reach the minimum. There are two variants of the conjugate gradient: Polak-Ribiere and Fletcher-Reeves formula. These methods suffer from poor convergence where the gradient is near zero.

1.2.3 Second Derivative Methods:

Second derivative methods also known as *variable metric* or *Quasi-Newton* methods require a knowledge of the function's Hessian matrix. These methods attempt to approximate the function $f(x)$ as a quadratic by Taylor series expansion at the given point x_k . By minimizing the quadratic approximation, a better solution can be found x_{k+1} . This procedure is then repeated at the new point x_{k+1} . The iterative formula known as the *Newton-Raphson* method is: $x_{k+1} = x_k - H_k^{-1} \cdot g_k$, where g_k is the function gradient, H_k is the function Hessian matrix. The difficulty of such a method is in computing the inverse of the Hessian matrix, which may be numerically ill conditioned (poor condition number). This drawback leads to a new class of Quasi-Newton methods in which the matrix inverse H_k^{-1} is replaced (i.e. approximated) by a positive definite symmetric matrix G_k . At each iteration, the matrix G_k is updated such that as x_k approaches x^* (optimum), then G_k approaches H_k^{-1} . There are two main algorithms implementing this concept: *Davidon-Fletcher-Powell* (DFP) and *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) [3]. These methods are the preferred having very fast convergence properties, and are generally available in off-the-shelf numerical optimization software packages.

1.3 Stochastic and Heuristic Search Methods:

Stochastic search methods include Evolutionary Computation (EC) and Simulated Annealing (SA). Heuristic search methods include Greedy Algorithms and Tabu Search. Evolutionary computation (EC) is broadly classified into four categories: *evolutionary programming* (EP), *evolution strategies* (ES), *genetic programming* (GA) and *genetic algorithms* (GA). Whilst different, they all share one fundamental principle: reproduction, random variation, and selection. Of these four methods, genetic algorithms have found widest acceptance in the field of optimization, identification and control. Excellent sources of reference on evolutionary computation can be found in [1, 2, 4, 5, 6]. The four Evolutionary methods described above share the same characteristics and similarity in many respects. They all operate on a population of individuals, and have each individual represented by an encoded string (chromosome) using some alphabet such as binary, floating point etc. The definition of individual performance or fitness based on some objective function to be optimized, and the application of genetic operators (selection, crossover, mutation) recursively to arrive at the solution. A good introduction to evolutionary computation is provided by Fogel [1]. Greedy algorithms [50] and Tabu search [54] operate on a single individual (solution) and use rule-of-thumb heuristics to produce a better solution based on previous solutions. Solutions found using heuristic and metaheuristic methods are not necessarily globally optimal.

1.3.1 Evolutionary Programming (EP):

Evolutionary programming techniques work with a population of finite state machines (FSM). Each individual FSM (chromosome) represents a potential solution. The inputs are a sequence of symbols: a_1, a_2, \dots, a_n (belonging to a finite alphabet), and the fitness value is a measure of how accurately the individual is able to predict the next output a'_{n+1} , which is then compared with the next observed symbol a_{n+1} . Transition diagrams are used to represent the behavior for which nodes correspond to each state, and arrows, indicate transition from one state to another. Concepts of reproduction, mutation, crossover and selection are applied at each generation. Evolutionary programming is not suitable for numerical optimization problems.

1.3.2 Evolution Strategies (ES):

This technique has been developed to solve parameter optimization problems. Each *chromosome* consists of two float-vectors: $\{x, \sigma\}$, where the x vector represents a single point in the search space (potential solution) and σ represents a vector of standard deviations associated with x .

Only one genetic operator is used: mutation, the next population of offspring is generated by the expression: $\mathbf{x}_{t+1} = \mathbf{x}_t + N(0, \sigma)$, where $N(0, \sigma)$ is a vector of independent random gaussian numbers with zero mean and σ standard deviation. The offspring then replaces the parent if its fitness value is higher than that of the parent. This is in effect a random search, and convergence is slower when compared with genetic algorithms.

1.3.3 Genetic Programming (GP):

This is a relatively new approach in which the objective is to find the best algorithm to solve a particular problem rather than using an evolution program to solve a problem. In other words, each chromosome in a population represents a particular computer algorithm. The search space is then a hyperspace of all valid computer programs which can be viewed as a space or rooted trees. This in effect results in an evolving computer program. Genetic operators such as *crossover* and *mutation* swap and modify sub-branches of parent trees. These have applications in artificial intelligence, but are not suitable for continuous function optimization problems.

1.3.4 Genetic Algorithms (GA):

Genetic algorithms (GA), first proposed by John Holland [2], attempt to mimic the process of natural evolution and *survival-of-the-fittest* by processes of genetic operators and natural selection. It is this process of evolution (or natural adaptation) which enables a population to evolve and to solve complex optimization problems. There are four features which define the concept of GA: (1) *codification* of solution space by bit-strings also referred to as *chromosomal* representation, (2) *genetic operations* which include *crossover* and *mutation*, (3) *evaluation* and *selection*, and (4) a *population* solutions rather than a single solution. Genetic Algorithms operate on a bit-string representation of the solution variables rather than the variables themselves, furthermore, GA do not operate on a single solution but on a population of individuals (chromosomes), this concept is known as *intrinsic parallelism*. The average fitness of the population of individuals is improved with each iteration (or generation) by genetic operators of selection, crossover and mutation. The general workings of the original GA proposed by Holland [2] is as follows: an initial population N is created with random values which span the solution space or the search space. Two or more parents are chosen via a selection scheme, this selection is based on relative fitness of the individuals. The higher the fitness the more likely the individual is to be selected, this is known as *proportional selection*.

The parents are combined probabilistically using the genetic process of *crossover* to produce either a single or two offspring. *Mutation* is then applied with a small probability to the resulting offspring, which are then used to create a new population of individuals. This process is repeated usually N times, where N is the population size. Crossover is the main search operator, with mutation as a background operator which is applied with much lower probability. Whilst crossover allows the solution to work its way down to a minimum (or maximum), it can get stuck within a local minimum, and mutation overcomes this by enabling search to continue over a wide solution space. The basis of GA search is embedded within the concept of the *building block hypothesis*. This states that a better individual (offspring) can be created by combining substrings or blocks from two (or more) parent individuals. Holland's original work on the *schema theorem* [2] provides a formal analysis and convergence properties of the GA. The schema theorem was based on binary string codification, currently the trend however is towards floating point representation.

1.3.5 Simulated Annealing (SA):

Strictly speaking, simulated annealing (SA) is not an evolutionary programming method, however it owes its basis to natural phenomena and is also applied probabilistically as in GA. Simulated annealing, first proposed by Kirkpatrick [3] is an optimization technique analogous to the thermal process of annealing. The SA algorithm starts with a high temperature T_0 and initial states x (solution), a random perturbation δx is applied to the states with magnitude dependent on the temperature $\delta x = f(T)$, and new solutions are evaluated at $x + \delta x$. If the energy level (or fitness) is less than the energy level at x , then this solution is accepted. If it is greater however, it will only be accepted with a finite probability which decreases with temperature. In the next iteration, the temperature is reduced (*annealing schedule*) and the process is repeated again. This continues until equilibrium is reached or the temperature is below a specified value (termination criterion). This algorithm is also known as the *Metropolis algorithm*.

The key to achieving good performance with simulated annealing and global convergence is that a stationary distribution must be reached at each temperature and the cooling schedule must proceed very slowly. The SA algorithm is not as effective as the GA algorithm at finding global minimum, however it has very fast convergence properties near the solution. Note that SA operates on a single candidate solution rather than a population of solutions.

1.3.6 Greedy Search (GS):

A greedy algorithm is a heuristic search algorithm which looks for the best immediate solution without considering many other alternatives. In this sense, a greedy search generally quickly finds local rather than global optimal solutions. A typical greedy search algorithm would be as follows:

```
iterate
- look for adjacent solution(s) within a predefined search span/range.
- if adjacent solution is better, accept as the current solution.
- increase or decrease the search span/range accordingly.
end
```

Fig.1.1
Typical Greedy Search Algorithm

While there is no one single generic form of the greedy search algorithm, the above is typical and can be applied to both combinatorial optimization problems, discrete and continuous function optimization. Examples of greedy search can be found in [50] in discrete function optimization, continuous function optimization [51], combinatorial optimization [52], and applications to radial basis function networks [53]. Because greedy search algorithms have good local convergence properties, applications usually involve a hybrid approach with an algorithm having global convergence (e.g.: Genetic Algorithm) and a greedy local search algorithm.

1.3.7 Tabu Search:

Tabu search operates on the premise that some moves (from the current position) are forbidden or Tabu. Forbidden moves are those recently visited which did not yield an optimal solution. Tabu search requires a *Tabu list* which is a record of forbidden moves. At each iteration, Tabu search chooses a non-Tabu feasible move. After each step, a collection of moves that includes any returning immediately to the previous point is added to the Tabu list. This move is then forbidden for several iterations. After many iterations, the Tabu list is cleared and the procedure is repeated from the new current position. Tabu search is currently becoming an active area of research in many diverse fields. For instance Tabu search can be applied to optimization of functions in continuous domains [54], topological and combinatorial optimization [55, 61], introductory papers can be found in [56,57,58], applications to vehicle routing [59], comparison with simulated annealing and genetic algorithms [60]. The main strength of Tabu search is in combinatorial and topological optimization problems.

1.4 Genetic Algorithms and Hybrid Methods:

Conventional genetic algorithms and hybrid genetic algorithms comprise the core of all simulations contained within this dissertation. Genetic algorithms were introduced in section 1.3.4. In this next section, detailed aspects on genetic algorithms and hybrid genetic algorithms is presented.

1.4.1 Conventional Genetic Algorithms:

Genetic Algorithms, originally developed by John Holland [2], are based on the Darwinian biological evolutionary principle of *survival of the fittest* strategy. The concept is to mimic the mechanisms of biological evolution using mathematical abstractions of genetic operators. Genetic algorithms operate on a population of individuals (or chromosomes) in order to search for a solution. Each individual consists of a potential solution and its associated fitness value. This fitness value represents the individual's performance upon the solution of the problem. For example the fitness value could indicate the inverse of the RMS error between the simulated model output compared with actual plant output, or some optimization function to be minimized. Higher fitness values denote better solutions. Each individual in the population is represented by a bit string or *chromosome* (also known as codification) . Historically binary representation was used. This has the advantage of being more generalized, but has limited accuracy. Currently floating point representation is used [12]. Figure 1.2 below illustrates the traditional binary representation of a chromosome:

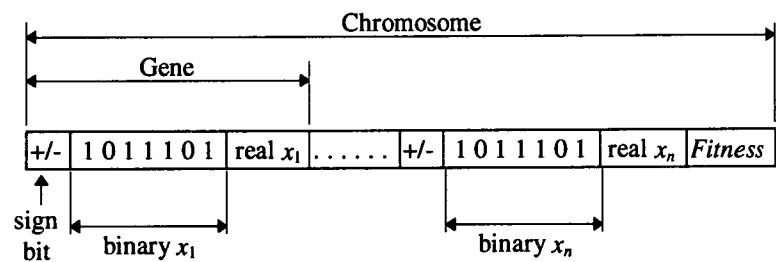


Fig.1.2
Original Binary Representation of a Chromosome

For instance, the above chromosomal representation can be used to encode the solution to the following unconstrained minimization problem: $\min \{f(x)\}$, where $x=[x_1, x_2, \dots, x_n]$, and the fitness value can be defined simply to be the inverse of the function thus: $fitness = 1 / f(x)$.

Referring to figure 1.2, the chromosome is subdivided into genes, and a gene encodes a particular function e.g.: node weights for a neural network. The complete string refers to a chromosome. The biological equivalent would be a DNA sequence.

Binary bit strings are no longer used and real number representation is more common, however binary representation is more domain independent, but is slow and factors of accuracy and finite length approximations result in problems with precision. The original *Schema* theorem developed by Holland [2] for the convergence analysis of GA used binary representation. Unfortunately it is difficult to see how the schema theorem is applicable to floating point representation. A comparison of floating point and binary representation is provided in [12]. The genetic algorithm in its simplest form is illustrated in the flowchart form below (Fig 1.3), noting that there are many other variations to this algorithm.

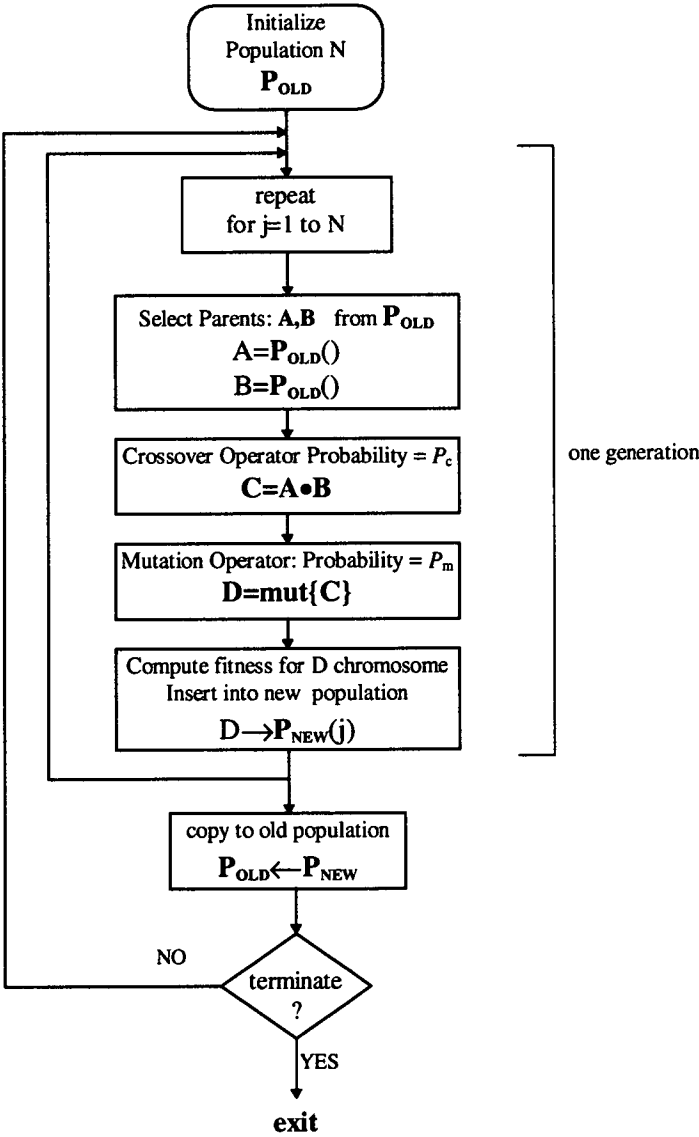


Fig. 1.3
The Genetic Algorithm

A new population is created with each generation, by using the genetic operators of selection, crossover and mutation, the average population fitness increases. Eventually the population converges whereby the majority of the population will have near-identical chromosomal values. Genetic algorithms have been applied successfully in training Multi Layer Perceptrons (MLP) and radial basis function neural networks [14]. Some excellent introductory textbooks on GA can be found in references [1, 2, 4, 5, 6]. Figure 1.3 illustrates a typical genetic algorithm. This is the traditional GA, sometimes also referred to as the *simple genetic algorithm*. The genetic algorithm uses no problem specific information, except when calculating the fitness value of a chromosome. The lack of gradient information however can result in slow convergence in regions where the objective function has nearly zero gradient. We next look at the four main genetic operators: selection, crossover, mutation and population inversion.

(i) The Selection Operator: The selection operator is used to choose parent individuals from the current population based on the individual's fitness. Holland's original work used the probability of selection proportional to the fitness value. This is known as the *roulette wheel* selection operator. With each generation step, the fitter individuals obtain more copies, thus producing a near identical population. This reduces the convergence rate, and selection becomes ineffective, the crossover operator also becomes ineffective due to lack of genetic diversity. Also, the possibility of creating a single *super-individual* which will quickly proliferate throughout the population and result in *premature convergence* possibly to a local minimum. Therefore the selection operator must be a careful balance between preventing premature convergence and maintaining adequate genetic diversity. There are two main groups of selection operators: *Fitness proportional selection* and *Rank based selection*.

Fitness Proportional Selection: This selection operator chooses parents with a probability directly proportional to the individual's fitness value. The most common is *roulette wheel selection*, similar in principle to a roulette wheel. Each member is represented as a slot of a roulette wheel, the width of the slot is proportional to its fitness. To select an individual, we simply spin the wheel (i.e. choose a uniform random number) and the slot where the random number ends up is the individual selected. This can result in premature convergence for super-fit individuals. To overcome this problem, fitness scaling can sometimes be applied to the population before selection. Many types of fitness scaling are available: linear static scaling, linear dynamic scaling, exponential scaling, logarithmic scaling, sigma truncation and Boltzman scaling.

Rank Selection: The individuals are ordered (sorted) by fitness values, only the relative fitness is important, and not absolute fitness. This method reduces the possibility of premature convergence, but ignores the actual fitness values of the individuals.

After sorting, several selection schemes may be applied including: tournament selection, stochastic universal sampling, and truncation selection. *Tournament Selection* selects m individuals randomly with uniform probability from the population, and the fittest (from m subpopulation) is then selected to be the parent. Generally m is two. A high value of m can produce premature convergence, a low number may result in a too slow convergence. Trial and error may be required in the choice of m . Variations of tournament selection can be found in reference [28] for multiobjective problems. *Stochastic Universal Sampling* is an optimal sampling algorithm with zero bias and minimal spread. It is also possible to scale and compute new fitness values according to the relative position of the individual in the rank, and then apply fitness proportional selection methods discussed above.

The selection operator can have a critical influence on the convergence properties of the GA. Tournament selection and stochastic universal sampling are currently the most popular, however some trial and error may be required in order to ascertain which selection operator works best for a particular application. An important quantity is the *selection pressure*, this is a measure of how strongly the fitter individuals are selected over the less fit individuals. For instance the ratio: **increase in average fitness/standard deviation of the population** can be used to quantify selection pressure. Fuzzy selection schemes have also been developed, for instance see [14B].

(ii) **The Crossover Operator:** The crossover operator (or recombination operator) takes two or more parents and recombines them to produce either one or more offspring. This is illustrated below in fig.1.4 for a binary string chromosome using single point crossover and producing a single offspring:

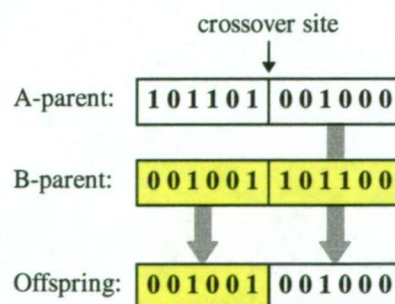


Fig. 1.4
The Single Point Crossover Operator

There are a number of variations of the crossover operator, these are: *two point crossover*, *multipoint crossover*, *uniform crossover*, *diagonal crossover*, and *weighted average crossover*. Two point crossover is more effective than single point. Uniform crossover simply swaps single bits chosen at random and not entire segments. Weighted average crossover only works with real numbers and simply averages the two parents thus: $offspring = \alpha \times ParentA + (1-\alpha) \times ParentB$, where α is a random number [0,1] chosen with uniform probability. We also found that sometimes a constant $\alpha=0.5$ can produce rapid convergence. For the crossover operator to be effective, a diverse population is required, because this is the main GA search operator. Once the population has converged, crossover becomes ineffective. When this occurs, the only search operator is mutation, at which point the genetic algorithm degenerates to a pure random search algorithm. This substantially reduces the rate of convergence. Crossover is applied statistically, with high probability values, typical probability: $P_c=0.7$ to 0.9 . Diagonal crossover is used in multi-parent (more than two parents) recombination.

(iii) The Mutation Operator: The mutation operator plays a secondary role to the genetic algorithm. Subsequently it is applied with low probability typically: $P_m=0.01$. Mutation changes bits of the chromosome at random. This is illustrated below for a binary string:

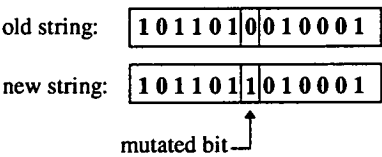


Fig. 1.5
The Mutation Operator

For floating point numbers, the mutation operator becomes: $x_j = x_j + k \times rand$, where x_j is an individual element of the chromosome, k =mutation intensity (or gain), and $rand$ has a uniform normal or gaussian distribution. The purpose of mutation is to prevent the GA from getting stuck in a local minima, to provide prolonged genetic diversity, and increased search space. The mutation intensity k or mutation gain, may be set with a value that should ideally decrement as the algorithm gradually converges. A high mutation intensity should be used near the start of the simulation, and gradual decrease with generation thus: $k=k(t)$. Another form of mutation is: $x_j = x_j \times (1 + k \times rand)$ which has a narrower search range, and the random function $rand$ is a gaussian distribution. As a rule of thumb, the probability of mutation P_m should be chosen to be the inverse of the dimension of the parameter space. Thus if 10 parameters are to be sought, then set the mutation probability to: $P_m = 0.1$.

(iv) The Population Inversion Operator: With each generation, a new population of offspring is created from the old parent population. Sometimes a new population is created by a combination of the best offspring and best parents. When generating a new population, it must be ensured that identical individuals are not duplicated reducing genetic diversity. Two methods which we have used are: (i) combine the N parents and N offspring into one $2N$ population, and choose the N fittest ones for the new population, or (ii) simply replace the old population with the new population. However, whichever method is chosen for generating a new population from the old, the concept of *elitism* in which the best individual from the old population is preserved into the new population unmodified, is found to be essential.

1.4.2 Hybrid Genetic Algorithms:

Genetic Algorithms constitute a family of powerful global search and optimization algorithms which can deal with multimodal functions containing many local minima. Nevertheless, genetic algorithms can become excessively slow in the final stages of convergence, once a global minimum has been found. To obtain accurate solutions (with many decimal places), the genetic algorithm is inefficient. This deficiency is in part due to population convergence, in which the crossover operator becomes ineffective. Also, the genetic algorithm does not exploit local landscape features such as function gradients. One way to overcome this problem would be to gradually reduce the mutation intensity or gain (see 1.4.1 part iii) once the population has reached steady state. The genetic algorithm then becomes a purely random search algorithm with an annealing schedule on the mutation operator. However, pure random searches are also unacceptably slow.

(i) Hybridization of Genetic Algorithms: Another method is to combine the genetic algorithm with a fast local search procedure. Once the minimum has been found by the genetic algorithm, the fast local search is used to quickly converge the solution to the desired accuracy. Fast local search procedures are also known as *hill-climbing* methods. Thus hybrid methods (also known as *genetic local search*) combine the reliability and robustness properties of the genetic algorithm and their original search heuristics with the accuracy and fast convergence of local search methods.

(ii) Examples: Examples of hybrid genetic algorithms which include nonlinear system identification [61] hybrid methods which combine genetic algorithms with Quasi-Newton (see 1.2.3) local search and Nelder-Mead Simplex (see 1.2.1) methods are discussed.

Again, in [62], variable metric methods using the BFGS (see 1.2.3) have been combined with genetic algorithms in multiobjective optimization applications. Hybrid genetic algorithms coupled with steepest descent methods can be found in [63, 65] in a seismic data imaging application. Combining genetic algorithms with heuristic local search methods can be found in [64] in which a greedy multi-start local search is used.

Applications to combinatorial optimization problems for the classical *traveling salesman problem* can be found in [66] using a simulated annealing local search procedure. Greedy local search algorithms have also been used to hybridize genetic algorithms, for instance [67, 70] describe an application in a continuous function domain.

Hybrid genetic algorithms using a fast simulated annealing local search procedure have been investigated, for instance in [68] where neural networks have been trained using hybrid GA+SA methods. In another application, genetic algorithms have been combined with Tabu search (see 1.3.7) to solve nonlinear continuous function optimization problems [69]. From the above list, many methods in diverse fields have been investigated.

Alternatively, it is also possible to hybridize the genetic operators, such that some local search is featured into either crossover or mutation operators. For instance, the pattern search method used by the *Hooke-Jeeves* algorithm described in section 1.2.1 can easily be incorporated into the crossover operator. This is described in more detail on the following section.

As a general rule however, it is impossible to accurately and reliably locate the global minimum of a multimodal function. This conflict is referred to as the *exploitation-exploration* trade-off, and must be borne in mind when attempting to hybridize or implement any optimization algorithm.

(iii) Combining genetic algorithms and metaheuristic searches: The aim of this thesis is to develop and compare hybrid methods with conventional genetic algorithms, and in particular apply these methods to a number of control system design problems. The objective is also to see how well hybrid genetic algorithms compare with conventional control system design methodologies. Some of the desirable properties required of the hybrid GA method are:

1. Use a fast local search procedure which does not require gradient computation. For instance, the methods discussed in 1.2.1 such as Nelder-Mead Simplex or Powell's method can be used.
2. We also wish to retain the stochastic and heuristic nature of the overall algorithm. Thus the genetic algorithm can be coupled to a fast greedy local search, fast simulated annealing, or a Tabu search. These methods also have some weak global search capability.

3. The hybrid genetic algorithm must also be able to deal with constrained optimization and multiobjective optimization problems.

The two hybrid genetic algorithms chosen are: (i) genetic algorithm coupled with a fast simulated annealing local search and (ii) genetic algorithm coupled with a fast greedy local search. These are chosen because constraints may be included and also have some global search capability. A multistart procedure is implemented for the local search algorithm. Note that Tabu search can also be used, however its application is more suited for combinatorial optimization problems than in continuous function domains. The two local search algorithms are detailed below.

(iv) Hybridization with greedy search algorithm: A greedy algorithm has no specific structure other than that illustrated by figure 1.1. However a typical greedy heuristic algorithm would use the following concepts: a variable search step size which contracts when convergence is slow, and expands when convergence is rapid. It must also keep track of the direction of recent success, so that the search is conducted over the direction of most rapid descent. This algorithm is outlined in figure 1.6 below. Referring to figure 1.6, the two vectors are *best_vec* and *best_sum*, where *best_vec* is the direction vector of most recent success, the magnitude of this vector expands and contracts according to rate of convergence. The vector: *best_sum* is a cumulative sum of *best_vec* and helps to search (i.e. exploratory move) in previous successful directions using long jumps. The function **random_vector()** simply returns a vector with the same magnitude (norm) as the input vector. A similar greedy local search algorithm can be found in reference [51]. Note that figure 1.6 is only a single iteration loop of the greedy algorithm, which must be repeated to obtain convergence.

(v) Hybridization with Fast simulated annealing algorithm: A faster variation of the classical conventional simulated annealing algorithm is used. Simulated annealing is comprised of three components: a *temperature annealing schedule*, a *gaussian-like function* for random state generation (*generating function*) and an *acceptance function* based on a boltzman probability distribution. Fast simulated annealing [71] is a semi-local search with occasional long jumps to overcome any local minimum. This version has a faster annealing schedule (exponential), while the generating function has a wider spread, and with a modified acceptance function. The algorithm is illustrated in figure 1.7.

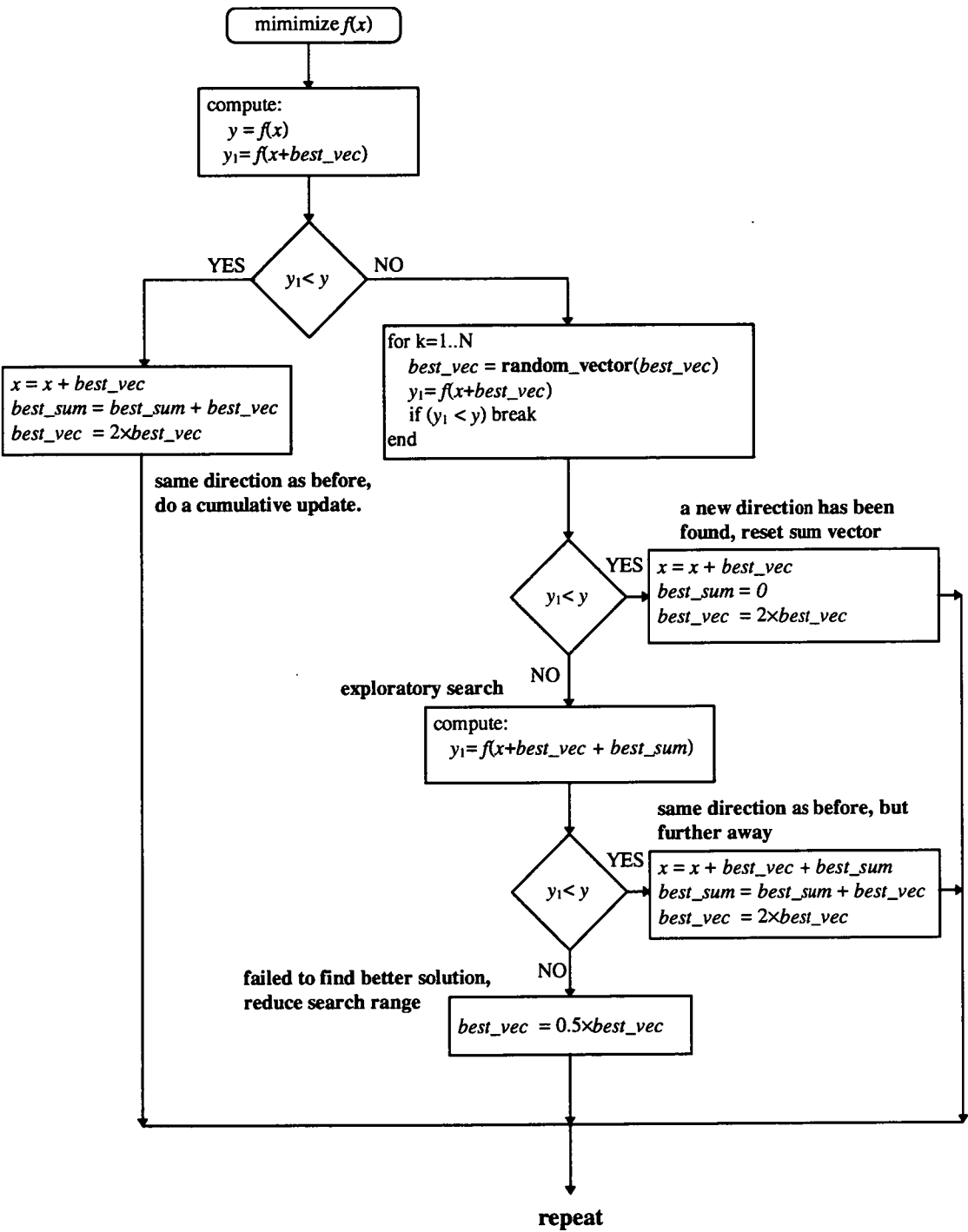


Fig.1.6
Typical Greedy Search Algorithm

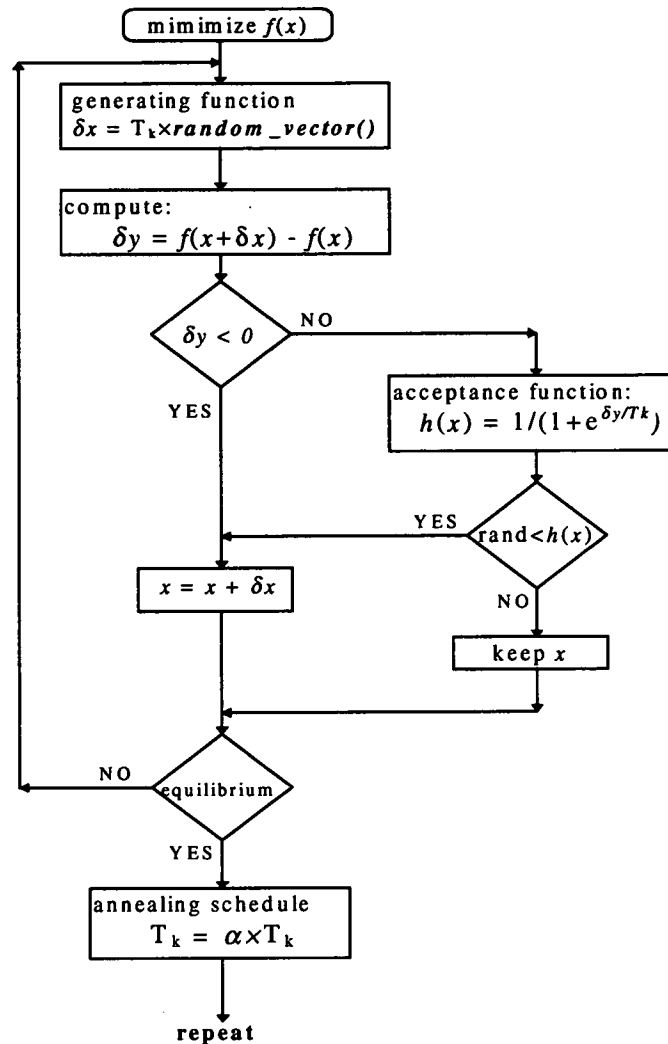


Fig.1.7
Typical Simulated Annealing Search Algorithm

(iv) **Pattern Search Crossover operator:** To further aid in convergence, we also hybridize the crossover operator by applying local search heuristics borrowed from the Hooke-Jeeves algorithm (see 1.2.1), and appendix. This heuristic is applied within the crossover operator with a finite probability. Given two parents, A and B, the *pattern-search* crossover operator is:

if (fitnessA > fitnessB) then
 offspring = $2 \cdot x_A - x_B$
 else
 offspring = $2 \cdot x_B - x_A$

where x_A and x_B represent the components of parent A and parent B chromosome. A flowchart of the Hooke-Jeeves algorithm is also provided in the appendix.

1.5 Constrained and Multiobjective Optimization:

Genetic Algorithms described earlier can be applied directly in solving unconstrained optimization problems. However in practice, most optimization problems are constrained, therefore the genetic algorithm must be modified to deal with such problems. Constrained optimization [15-28] problems can have linear or nonlinear constraints. The constrained optimization problem can be defined in several ways, for instance the **Equality Constrained Problem (ECP)**: is defined as: *minimize $f(x)$ subject to $h(x)=0$* , and the **Inequality Constrained Problem (ICP)**: *minimize $f(x)$ subject to $h(x)<0$* , where: $f: R^n \rightarrow R$, $h: R^n \rightarrow R^m$. It is possible to transform an ECP problem into an ICP and vice versa by the addition of *slack* variables. Thus the two problems are interchangeable and can be solved in the same fashion.

The field of constrained optimization using calculus based methods is well established, however with genetic algorithms this is a relatively new topic of research. Genetic algorithms can also be extended to these standard methods, or alternatively, we could modify the genetic rules to deal specifically with constrained optimization problems. An excellent survey of constrained optimization using evolutionary algorithms can be found in [18-23]. A brief summary of calculus based methods and genetic algorithm based methods is outlined below.

1.5.1 Calculus Based Constrained Single Objective Optimization:

(i) **Linear Programming Methods:** If we are dealing with only simple linear constrained problems, then there are techniques which are very effective, known as the *simplex* method. When the problem is of the form: *minimize: $f(x) = c.x$, subject to: $A.x = b$, $x \geq 0$* , it can be directly solved by matrix manipulation. Other forms include *two-phase simplex methods* and *duality methods*. A solution via the MATLAB® Optimization Toolbox is straightforward. We will not deal with these optimization problems.

(ii) **Penalty Function Methods:** The penalty function method transforms a constrained optimization problem into an unconstrained one. The minima of both the constrained and unconstrained functions is the same. There are many variations to the penalty function method, such as *nonquadratic penalty functions*, *Fletcher's method*, *Powell's method*, *quadratic penalty functions*.

For instance the following is a typical quadratic penalty function in which the penalty: ρ_k is progressively increased with each generation k thus: $L(x) = f(x) + \rho_k \cdot \sum |h_j(x)|^2$. Perhaps the simplest penalty function is the *Static Penalty Function* [27], this is given by:

$$L(x) = f(x) + \sum_{i=1}^m \rho_i \delta_i \quad \text{where } \delta_i = 1 \text{ if constraint } i \text{ is violated, else } \delta_i = 0 \text{ if constraint } i \text{ is not}$$

violated. This penalty function makes no use of a distance metric for the feasible region. The *dynamic penalty function* increases the severity of the penalty parameter with each generation. Adaptive penalty functions modify the penalty parameter ρ_i depending on the distance from the feasible solution (see reference [27]).

Penalty function methods have been successfully applied to constrained optimization with genetic algorithms. An excellent summary of the penalty function method is found in reference [23, 27], adaptive penalty methods in [19], and a more extensive discussion on static penalty, dynamic penalty, annealing penalties, and adaptive penalty methods can be found in reference [27].

(iii) **Lagrangian Function Methods:** All Lagrangian functions have the following general structure $L(x, \lambda) = f(x) + \sum_j \lambda_j \cdot h_j(x)$, where λ_j are the lagrange multiplier vectors or matrices

which also need to be solved for. The lagrangian function is similar to the penalty function method and has been successfully applied to constrained optimization problems in genetic algorithms. This is a very popular method, in which the solution can be found by computing partial derivatives: $\partial L / \partial x = 0$ and $\partial L / \partial \lambda = 0$ and solving a simultaneous set of equations. The above method only applies to equality constraints. This is also a very popular technique.

(iv) **Barrier Function Methods:** The barrier functions apply to inequality constraints and are also similar to the penalty function, a typical inverse barrier function and log barrier function is given by: $L(x) = f(x) + \rho_k \cdot \sum_j [h_j(x)]^{-1}$ and log barrier function: $L(x) = f(x) + \rho_k \cdot \sum_j -\ln(h_j(x))$ with

k increasing with time. Currently Barrier functions have had no application in genetic algorithms. Calculus based methods can also be implemented with genetic algorithms. However, genetic algorithms offer potentially new and novel possibilities for the solution to constrained optimization problems.

1.5.2 Genetic Algorithm - Single Objective Constrained Optimization:

Rather than transforming the constrained problem into an equivalent unconstrained one, we can modify the genetic operators of crossover, mutation and selection to directly deal with the constraint. Specialized GA methods exist when dealing with constrained optimization problems. There are essentially three methods of handling constraints with genetic algorithms (not counting the calculus based ones above). The methods are: *Decoders*, *Penalty functions*, and *repair algorithms*.

(i) **Decoders:** Decoders process instructions incorporated into the chromosome, which are used to construct a feasible solution. Essentially, a decoder is a mapping T from a representation space d (binary strings, vectors, integers) into a feasible part of the solution space s . Thus with a decoder, illegal chromosomes (infeasible solutions) cannot occur. The method is problem specific, can be computationally intensive to implement the transformation T . Further, there must be a unique mapping between the representation space d and solution space s . One criticism is that not all problems can be solved using this method.

(ii) **Penalty Functions:** Discussed above, a penalty function is used, with a gradually increasing penalty parameter, the penalty parameter is initially small, and gradually increases with each generation.

(iii) **Repair Algorithms:** A repair algorithm simply corrects an infeasible solution by mapping any infeasible individual into a feasible one. The repaired individual can be used for evaluation purposes or can be used to replace the original one (with some finite probability).

Repair algorithms are very popular in the area of evolutionary computation, due to their relative ease by which an infeasible individual can be repaired. This algorithm is problem dependent. A discussion can be found in reference [1]. We use repair algorithms and penalty functions in our simulations.

1.5.3 Genetic Algorithm Multiobjective Optimization:

Currently there are two methods of multiobjective optimization using genetic algorithms: Pareto dominance principle and Nash Equilibria [29]. A good review on multiobjective optimization (MOP) using genetic algorithms is found in [38]. A third less popular method known as *Stakelberg equilibria* exists, a brief discussion is given on these methods below.

(i) **Pareto Dominance Principle:** When dealing with multiple objectives, ie the function $F(x)$ is a vector function, then a single solution may not exist. Instead multiple solutions or a set of solutions may exist. In this case, the problem may be stated as follows, given the vector function: $\min \{F(x)\}$, where is defined as: $F(x)=[f_1(x), f_2(x), \dots, f_N(x)]$, there are $i=1..N$ functions to minimize, and $j=1..r$ constraints. In general the solution is not unique, and a family of solutions may exist. The pareto dominance principle provides an efficient means to find optimal solutions.

Defn.3.1: A solution x_1 is said to *dominate* x_2 if the following condition holds: $f_i(x_1) < f_i(x_2)$ for all values of $i=1,2,\dots,N$:

Defn.3.2: The Pareto Optimum is defined as follows: a solution $x^* \in X$ is Pareto optimal if and only if there exists no $x \in X$ such that $f_i(x) \leq f_i(x^*)$ for $i=1,2,\dots,N$ with $f_i(x) < f_i(x^*)$ for at least one i . Thus intuitively, the point x^* is optimal if no criterion can be improved without worsening at least one other criterion.

The group of *nondominated* solutions is called the *Pareto set*. This is illustrated graphically for two criteria $f_1(x)$ and $f_2(x)$ to clarify the concept:

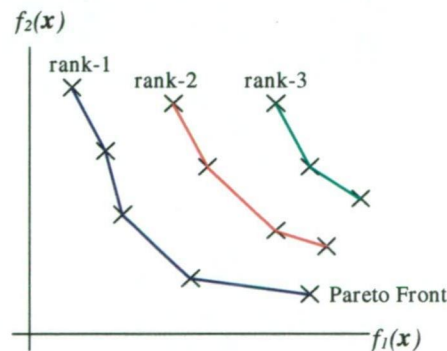


Fig.1.8
Pareto Front and Ranking Scheme

From figure 1.8, the Pareto front is the set of all nondominated solutions, this is assigned rank-1, this is then removed from the population, and the next front is determined and assigned rank-2. The procedure repeats until all individuals are accounted for. There are many references discussing Pareto optimality applications with genetic algorithms, see: [29 - 33].

Classical gradient based optimization algorithms are capable of finding the optimal value of only a single objective. Consequently the multiple objectives may be combined into one weighted sum:

$$U(x) = \sum_{i=1}^n W_i \cdot f_i(x) \quad \text{Eqn.1.1}$$

The function $U(x)$ is sometimes referred to as a *utility* or *composite function*.

(ii) **Nash Equilibria:** This is a relatively new concept of game theory in genetic algorithms, which is more robust and has faster convergence properties. Nash equilibria which originated in 1951 [34], is inspired from Games Theory and economics, and only produces a single solution rather than a family of solutions. Also referred to as *Non-Cooperative approaches*, the Nash strategy [29] consists of having N players, each optimizing its own criterion. However each player has to optimize his criterion given that all the other criteria are fixed by the rest of the players. When no player can further improve his criterion, the system has reached an equilibrium called the *Nash Equilibrium*. A good introduction to Nash equilibria is given by [35].

To understand Nash game theory, assume there are two players A, B , and there are two functions to minimize: $f_a(x, y)$ and $f_b(x, y)$. Player A minimizes the first function with respect to x while keeping y fixed by player B , conversely player B minimizes the second function $f_b(x, y)$ with respect to y while keeping x fixed by player A . This means that two populations are required, one for each player. Figure 1.9 below illustrates how Nash equilibria is applied with each generation to genetic algorithms.

Let x_{k-1} be the best value found by player-A at generation $k-1$, and y_{k-1} the best value found by player-B at generation $k-1$. Then at generation k , player-A optimizes x_k while using y_{k-1} , at the same time player-B optimizes y_k while using x_{k-1} . After this, player-A sends the best value x_k to player-B, and player-B sends the best value y_k to player-A. This is repeated until neither player-A or B can further improve their criteria, this is the Nash equilibrium.

Simulation studies [29] have shown that exchanges between player-A and B must be as frequent as possible, low exchange leads to low convergence rates.

From an evolutionary perspective, Nash equilibria can be viewed as an independent evolution of different species leading to the optimization or adaptation for each species to the natural environment. This can occur even when the behavior of one species has a direct influence on the others. A new genetic operator referred to as *exchange* is introduced to simulate the transfer of genetic material from one population to the other population.

$$U(x) = \sum_{i=1}^n W_i \cdot f_i(x) \quad \text{Eqn.1.1}$$

The function $U(x)$ is sometimes referred to as a *utility* or *composite function*.

(ii) Nash Equilibria: This is a relatively new concept of game theory in genetic algorithms, which is more robust and has faster convergence properties. Nash equilibria which originated in 1951 [34], is inspired from Games Theory and economics, and only produces a single solution rather than a family of solutions. Also referred to as *Non-Cooperative approaches*, the Nash strategy [29] consists of having N players, each optimizing its own criterion. However each player has to optimize his criterion given that all the other criteria are fixed by the rest of the players. When no player can further improve his criterion, the system has reached an equilibrium called the *Nash Equilibrium*. A good introduction to Nash equilibria is given by [35].

To understand Nash game theory, assume there are two players A, B , and there are two functions to minimize: $f_a(x, y)$ and $f_b(x, y)$. Player A minimizes the first function with respect to x while keeping y fixed by player B , conversely player B minimizes the second function $f_b(x, y)$ with respect to y while keeping x fixed by player A . This means that two populations are required, one for each player. Figure 1.9 below illustrates how Nash equilibria is applied with each generation to genetic algorithms.

Let x_{k-1} be the best value found by player-A at generation $k-1$, and y_{k-1} the best value found by player-B at generation $k-1$. Then at generation k , player-A optimizes x_k while using y_{k-1} , at the same time player-B optimizes y_k while using x_{k-1} . After this, player-A sends the best value x_k to player-B, and player-B sends the best value y_k to player-A. This is repeated until neither player-A or B can further improve their criteria, this is the Nash equilibrium.

Simulation studies [29] have shown that exchanges between player-A and B must be as frequent as possible, low exchange leads to low convergence rates.

From an evolutionary perspective, Nash equilibria can be viewed as an independent evolution of different species leading to the optimization or adaptation for each species to the natural environment. This can occur even when the behavior of one species has a direct influence on the others. A new genetic operator referred to as *exchange* is introduced to simulate the transfer of genetic material from one population to the other population.

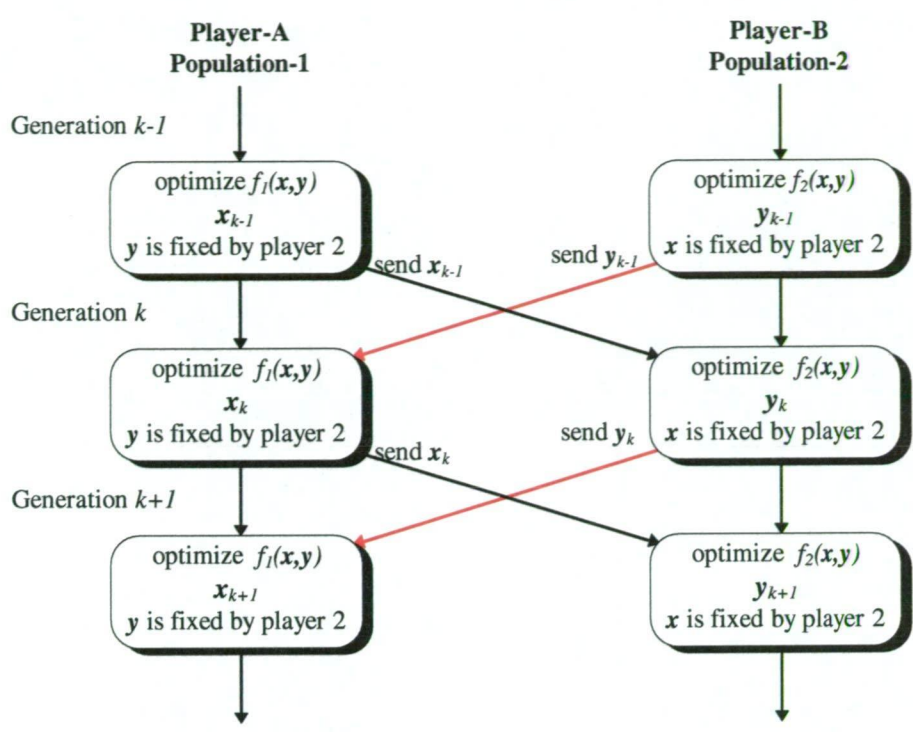


Fig.1.9
Nash Equilibria with two players applied to Genetic Algorithms

(iii) **Stackelberg Equilibria:** A similar strategy using *asynchronous* (less frequent) exchange of data exists, which is called the *Stackelberg Equilibria* [36] in which one player plays before the other, taking into account its reaction. All these techniques are part of *evolutionary game theory*, and offer new avenues of research in genetic algorithms. Refer to [35].

1.6 Chapter Summary and Conclusion:

This chapter has provided an introduction to concepts of evolutionary computation theory in which genetic algorithms are but just one area which have found wide acceptance in the control systems research community. Furthermore, a brief discussion on constrained optimization and multiobjective optimization was provided. It must be emphasized that the field of evolutionary computation is extensive and that many other concepts such as fuzzy-evolutionary computation [4,5], neuro-evolutionary and other hybrid approaches exist, too numerous to give adequate consideration. .

This thesis focuses primarily on the design and synthesis of control systems using conventional genetic algorithms and hybrid genetic algorithms. Genetic algorithms have recently been applied successfully to many control applications, in which conventional design methodologies are difficult to apply, or may not exist. Currently, the design trend is towards control systems which have a high level of autonomy, and are capable of dealing with plant changes, unknown environments, faults, nonlinearities, external disturbances, and systems capable of learning. In the field of control theory, such systems are generally termed robust, self tuning, adaptive, and reconfigurable control systems. Each one belonging to a particular area of control theory. Whilst self tuning and adaptive control can deal with a limited amount of plant changes, a broader class of autonomous control systems would generally embrace concepts of artificial intelligence, knowledge bases and expert systems, and are implemented using fuzzy and neural control. In such cases, the process of learning and adaptation can only be accomplished as a set of goal-oriented tasks rather than traditional control methodologies. In this instance, the objective may not necessarily be a single continuous mathematical function, but instead some abstract goal to be achieved. This goal can subsequently define the quality of the solution (i.e. fitness level).

In this thesis, we look at how genetic algorithms and hybrid genetic algorithms can be applied directly in a number of areas of control system design, and show that results are comparable and in some cases superior to the more traditional methods.

1.7 References and Further Reading:

Introductory References:

- [1] T.Back, D.B.Fogel, Z.Michalewicz
Handbook of Evolutionary Computation
Oxford University Press, 1997
- [2] J.H.Holland,
Adaption in Natural and Artificial Systems.
University of Michigan Press, Ann Arbor, 1975.
- [3] M.A.Bhatti
Practical Optimization Methods with Mathematica Applications
Springer-Verlag New York, Inc. 2000
- [4] W. Pedrycz
Fuzzy Evolutionary Computation
Kluwer Academic Publishers, 1997
- [5] E. Sanchez, T. Shibata, L. A. Zadeh
Genetic Algorithms and Fuzzy Logic Systems
World Scientific, 1997
- [6] David B. Fogel,
An Introduction to Simulated Evolutionary Optimization
IEEE Transactions on Neural Networks, Vol.5, No.1, pp.3-14, January 1994

Genetic Algorithms:

- [7] L. Davis
Handbook of Genetic Algorithms
Van Nostrand Reinhold, 1991
- [8] Charles L. Karr, L. Michael Freeman
Industrial Applications of Genetic Algorithms
CRC Press 1999
- [9] Mitchell, Melanie
Introduction to Genetic Algorithms
Cambridge, Mass. MIT Press 1996
- [10] David B. Fogel,
An Introduction to Simulated Evolutionary Optimization
IEEE Transactions on Neural Networks, Vol.5, No.1, pp.3-14, January 1994
- [11] Thomas Back, Frank Hoffmeister, Hans-Paul Schwefel
A Survey of Evolutionary Strategies
Proceedings of the Fouth International Conference on Genetic Algorithms, pp.2-9, 1991
- [12] C.Z.Janikow, Z. Michalewicz,
An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms
Proceedings fo the 4th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, California. pp.31-36, 1991.
- [13] R.Caponetto, L.Fortuna, S.Graziani, M.G.Xibilia
Genetic Algorithms and Applications to System Engineering: a Survey.
Transactionsof the Institute of Measurement and Control, Vol.15, No.3, 1993.

- [14] A.J.F. van Rooij, L.C.Jain, R.P.Johnson
Neural Network Training Using Genetic Algorithms.
World Scientific Publishing Co. 1996
- [14B] M.Mahfouf, D.A.Linkens, M.F.Abbod
Multi-objective Genetic Optimization of GPC and SOFLC Tuning Parameters Using a Fuzzy-Based Ranking Method
IEE Proceedings Control Theory Applications, Vo.47, No.3, pp344-354, May 2000

Conventiona and GA Constrained Optimization:

- [15] Edwin K.P.Chong, Stanislaw H. Zak
An Introduction to Optimization
John Wiley and Sons Inc. 1996
- [16] L.E. Scales
An Introduction to Non-Linear Optimization
Macmillan 1985
- [17] Dimitri P. Bertsekas
Constrained Optimization and Lagrange Multiplier Methods
Academic Press 1982
- [18] Carlos A Coello
A Survey of Constraint Handling Techniques used with Evolutionary Algorithms
WEB: <http://citeseer.nj.nec.com/did/202945>
- [19] David W. Coit, Alice E. Smith, David M. Tate
Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems
WEB: <http://citeseer.nj.nec.com/coit96adaptive.html>
- [20] P. Admidis, S.Kazarlis, V.Petridis
Advanced Methods for Evolutionary Optimization
WEB: <http://citeseer.nj.nec.com/did/15960>
- [21] Carlos A Coello
Evolutionary Algorithms for Constrained Parameter Optimization Problems
WEB: <http://citeseer.nj.nec.com/did/140843>
- [22] Z.Michalewicz, D.Dasgupta, R.G. Le Riche, M. Schoenauer
Evolutionary Algorithms for Constrained Engineering Problems
WEB: <http://citeseer.nj.nec.com/abib/0/324335/93248>
- [23] K. Sugihara
A Survey of GA Solutions for Optimization with Constraints
WEB: http://www.ics.hawaii.edu/~xiaochun/ics_691_ga/ga_survey.htm
- [24] H. Adeli, Nai-Tsang Cheng
Augmented Lagrangian Genetic Algorithm for Structural Optimization
Journal of Aerospace Engineering, Vol.7, No.1, January 1994, pp.104-118
- [25] H. Adeli, Nai-Tsang Cheng
Concurrent Genetic Algorithms for Optimization of Large Structures
Journal of Aerospace Engineering, Vol.7, No.3, July 1994, pp.276-296
- [26] Zbigniew Michalewicz, Cezary Z. Janikow
Handling Constraints in Genetic Algorithms
Proceedings of the Fouth International Conference on Genetic Algorithms, pp.151-157, 1991
- [27] Alice E. Smith, David W. Coit
Penalty Functions
WEB: <http://citeseer.nj.nec.com/did/255729>

- [28] W.A.Crossley, A.M.Cook, D.W.Fanjoy, V.B. Venkayya
Using the Two Branch Tournament Genetic Algorithm for Multiobjective Design
AIAA Journal, Vol.37, No.2, pp.261-267, Feb.1999

MultiObjective Optimization with GA:

- [29] D.Quagliarella, J.Periaux, C.Poloni, G.Winter
Genetic Algorithms and Evolution Strategies in Engineering and Computer Science
Recent Advances and Industrial Applications, John Wiley and Sons, 1998 (Book)
- [30] A.Osyczka, S.Kundu
A New method to Solve Generalized Multicriteria Optimization Problems Using the Simple Genetic Algorithm, Structural Optimization, Vol.10 pp.94-99, 1995
- [31] Carlos M. Fonseca, Peter J. Fleming
Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms- Part I: A Unified Formulation
IEEE Transactions on Systems Man and Cybernetics, Vol.28, No.1, pp.26-37, January 1998
- [32] Carlos M. Fonseca, Peter J. Fleming
Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms- Part II: Application Example
IEEE Transactions on Systems Man and Cybernetics, Vol.28, No.1, pp.28-47 January 1998
- [33] K.Fujita, N.Hirokawa, S.Akagi, S.Kitamura, H.Yokohata
Multiobjective Optimal Design of Automotive Engine Using Genetic Algorithms
Proceedings of DETC'98 1998 Design Engineering Technical Conference, Sept. 13-16, 1998 Atlanta, Georgia
- [34] J. Nash
Non-Cooperative Games
Annals of Mathematics, Vol.54, pp.286-295, 1951
- [35] Robert Gibbons
A Primer in Game Theory
Harvester Wheatsheaf, 1992 (Book)
- [36] Joao Pedro Pedroso
Numerical Solution of Nash and Stackelberg Equilibria: and Evolutionary Approach
- [37] John J. Grefenstette
Optimization of Control Parameters for Genetic Algorithms
IEEE Transactions on Systems Man and Cybernetics, Vol.SMC-16, No.1, pp.122-128, January-February 1986
- [38] H. Tamaki, H. Kita, S. Kobayashi
Multi-Objective Optimization by Genetic Algorithms: A Review
Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp.517-522, 1996
- [39] Eckart Zitzler, Lothar Thiele
Multiobjective Evolutionary Algorithms: a Comparative Case Study and The Strength Pareto Approach
IEEE Transactions on Evolutionary Computation, Vol.3, No.4, pp.257-271 Nov.1999
- [40] Jong-Hwan Kim, Hyun Myung
Evolutionary Programming Techniques for Constrained Optimization Problems
IEEE Transactions on Evolutionary Computation, Vol.1, No.2, pp.129-140 July 1997
- [41] Bruno Sareni, Laurent Krahenbuhl
Fitness Sharing and Nicheing Methods Revisited
IEEE Transactions on Evolutionary Computation, Vol.2, No.3, pp.97-106, September 1998

Derivative Free Optimization:

- [42] J.A.Nelder, R.Mead
A Simplex Method for Function Evaluation
The Computer Journal, Vol.7, pp.308-313, 1965
- [43] V. Torczon
On the Convergence of the Multidirectional Search Algorithm
SIAM Journal of Optimization, Vol.1, No.1, pp.123-145, February 1991
- [44] M.J.D. Powell
An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives.
The Computer Journal, Vol.7, pp.155-162, 1964
- [45] K.I.M. McKinnon
Convergence of the Nelder-Mead Simplex Method to a Nonstationary Point
SIAM Journal of Optimization, Vol.9, No.1, pp.148-158, February 1998
- [46] L.C.W. Dixon
Nonlinear Optimization, 1972
- [47] H.H.Rosenbrock
An Automatic Method for Finding the Greatest or Least Value of a Function
The Computer Journal, Vol.3, pp.175-184, 1960
- [48] R. Fletcher
Function Minimization Without Evaluating Derivatives - a Review
The Computer Journal, Vol.8, Issue.1, pp.33-41, April 1965
- [49] R. Fletcher, C.M. Reeves
Function Minimization By Conjugate Gradients
The Computer Journal, Vol.7, Issue.2, pp.149-154, July 1964

Greedy, Tabu Search:

- [50] M.Y. Wang
An Optimum Design for 3D Fixture Synthesis in a Point Set Domain
IEEE Transactions on Robotics and Automation, Vol.6 No.6, December 2000
- [51] R. Desai, R.Patil
SALO: Combining Simulated Annealing and Local Optimization for Efficient Global Optimization
Proceedings of the 9th Florida AI Research Symposium (FLAIRS-96), pp.233-237, June 1996
- [52] T.A.Feo, M.G.C.Resende
Greedy Randomized Adaptive Search Procedures
Journal of Global Optimization, Vol.6, No.2, pp.109-133, March 1995
- [53] R. Schaback, H.Wendland
Adaptive Greedy Techniques for Approximate Solution of Large RBF Systems
Numerical Algorithms, 2000
- [54] A.Fanni, A.Manunza, M.Marchesi, F.Pilo
Tabu Search Metaheuristics for Electromagnetic Problems Optimization in Continuous Domains
IEEE Transactions on Magnetics, Vol.35, No.3, pp.1694-1697, May 1999
- [55] C.Friden, A. Hertz, D. de Werra
An Exact Algorithm Based on Tabu Search for Finding a Maximum Independent set in a Graph
Computers and Operations Research, Vol.17, pp.437-445, 1990

- [56] F.Glover, E.Taillard, M.Laguna, D. de Werra
Tabu Search
Annals of Operations Research, Vol.41, 1993
- [57] F.Glover
Tabu Search, Part I
ORSA Journal on Computing 1, pp.190-206, 1989
- [58] F.Glover
Tabu Search, Part II
ORSA Journal on Computing 2, pp.4-32, 1990
- [59] M.Gendreau, A.Hertz, G.Laporte
A Tabu Search Heuristic for the Vehicle Routing Problem
Management Science, Vol.40, No.10, pp.1276-1290, 1994
- [60] B.L.Fox
Integrating and Accelerating Tabu Search, Simulated Annealing and Genetic Algorithms
Annals of Operations Research, Vol.41, pp.47-67, 1993

Hybrid GA Methods:

- [61] J.M.Renders, S.P.Flasse
Hybrid Methods Using Genetic Algorithms for Global Optimization
IEEE Transactions on Systems Man and Cybernetics, Vol.26, No.2, pp.243-258, April 1996
- [62] D.Quagliarella, A.Vicini
Coupling Genetic Algorithms and Gradient Based Optimization
Genetic Algorithms and Evolution Strategies in Engineering and Computer Science
Recent Advances and Industrial Applications, John Wiley and Sons, pp. 289-303, 1998 (Book)
- [63] K.E.Mathias, L.D.Whitley, C.Stork, T.Kusuma
Staged Hybrid Genetic Search for Seismic Data Processing
IEEE Conference on Evolutionary Computation, Vol.1, pp.356-361, 1994
- [64] M.Yagiura, T.Ibaraki
Genetic and Local Search Algorithms as Robust and Simple Optimization Tools
Meta-heuristics: Theory and Applications, pp.63-82, Kluwer Academic Publishers, Boston 1996
- [65] K.Krishna, M.N. Murty
Genetic K-Means Algorithm
IEEE Transactions on Systems Man and Cybernetics, Vol.29, No.3, pp.433-439, June 1999
- [66] A.Kolen, E.Pesch
Genetic Local Search in Combinatorial Optimization
Discrete Applied Mathematics, Vol.48, pp.273-284, 1994
- [67] A.Fanni, M.Marchesi, A.Serri, M.Usai
A Greedy Genetic Algorithm for Continuous Variables Electromagnetic Optimization Problems
IEEE Transactions on Magnetics, Vol.33, No.2, pp.1900-1903, March 1997
- [68] A.Abraham, B.Nath
Optimal Design of Neural Nets Using Hybrid Algorithms
Proceedings of the Sixth Pacific Rim International Conference on Artificial Intelligence pp.510-520, 2000
- [69] F.Glover
Tabu Search for Nonlinear and Parametric Optimization with Links to Genetic Algorithms
Discrete Applied Mathematics, Vol.49, pp.231-255, 1994

-
- [70] A.Fanni, M.Marchesi, A.Serri, M.Usai
Performance Improvement of a Hybrid Optimization Algorithm for Electromagnetic Device Design
IEEE Transactions on Magnetics, Vol.35, No.3, pp.1698-1701, May 1999

Simulated Annealing:

- [71] H.Szu, R.Hartley
Fast Simulated Annealing
Physics Letters A, Vol.122, No.3,4, pp.157-162, June 1987
- [72] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi
Optimization by Simulated Annealing
Science, Vol.220, pp-671-680, 1983

2

*Training Radial Basis Functions
With Hybrid Genetic Algorithms*

Contents:

2.1	Introduction	p.2.2
2.1.1	The Radial Basis Function Network	p.2.3
2.1.2	Training Radial Basis Function Networks.	p.2.5
2.2	Training RBF Networks with Hybrid Genetic Algorithms	p.2.6
2.2.1	Bioreactor Mathematical Model.	p.2.7
2.2.2	Training with Conventional Methods	p.2.8
2.2.3	Training with Hybrid Genetic Algorithms	p.2.11
2.2.4	Comparison of Results.	p.2.16
2.3	Chapter Summary and Conclusion	p.2.18
2.4	References and Further Reading	p.2.21

2.1 Introduction:

The purpose of this chapter is to apply hybrid genetic algorithm concepts developed in chapter 1 to the training of *radial basis function* (RBF) networks [8]. This is illustrated by way of an example of a model matching problem often found in control system applications. In this example, a radial basis function (RBF) network is trained to model a nonlinear bioreactor fermentation process.

Heuristic and stochastic search algorithms which include: genetic algorithms, simulated annealing, greedy and Tabu search are currently active areas of research in many diverse fields such as: combinatorial optimization, neural network training, industrial design, economics, image processing, system identification, machine learning, adaptive algorithms, pattern recognition, artificial intelligence, nonlinear and robust control system design. One of the main applications of stochastic search algorithms is in the area of optimization theory. When compared to traditional optimization methods based on calculus and enumerative strategies, these algorithms are found to be robust, globally converging, less influenced by noise and initial conditions, and relatively simple to apply to any problem domain. Additionally, stochastic algorithms do not require gradient or higher order derivative information for convergence, only a single cost functional (i.e. fitness function) is needed. Cost functionals need not necessarily be linear or continuous, for instance discontinuous cost functions can be used for pattern or classification problems when applied to neural network training.

The purpose of this chapter is to apply and compare the three hybrid genetic algorithms discussed in chapter 1 to training radial basis function networks, the algorithms are: Conventional Genetic Algorithms (GA), Genetic Algorithms with Fast Simulated Annealing (GA+SA) and Genetic Algorithms with Greedy Search (GA+GS). Rate of convergence, computational effort (FLOPS) and ease of implementation are compared. Results are also compared with more conventional RBF training algorithms.

Initially, a brief overview of radial basis function networks and current means of training is provided. Also included is a mathematical description on bioreactors. Simulation results follow in section 2.2. A good introduction to Genetic Algorithms is given by Davis [2], Mitchell [4], and practical industrial applications by Karr [3]. An introduction to simulated annealing can be found in [22], and greedy algorithms in [23] and [24], see also chapter 1 for many additional references.

2.1.1. The Radial Basis Function Network:

Radial basis function networks (RBF) are a class of feed-forward neural networks which are characterized by their topological simplicity and ease of training compared to other neural networks. Because of this, radial basis functions have been widely applied to signal processing applications, system identification, function interpolation and curve fitting. However, radial basis functions generally require an excessive number of nodes for accurate operation. The original RBF model required an equal number of hidden nodes as data points. This is clearly unacceptable because the number of data points is generally very large. It is possible however, to synthesize RBF networks with fewer nodes by applying globally converging training and optimization routines. This is the objective of this chapter.

The radial basis function network consists of three layers: the input layer is made up of source nodes, the second layer (hidden layer) performs some arbitrary basis for the input patterns, the output layer has adjustable weights and a single summation node. The hidden units (or nodes) consist of nonlinear elements which enable the RBF to perform nonlinear mappings and also enable effective separation of input vectors for pattern classification problems.

Training a radial basis function network in off-line system identification problems requires the selection of the gaussian function centers, variances and weights. The original paper by Broomhead and Lowe [11] suggested that the centers be selected randomly from the data. The variances (or spread of centers) can be estimated from a histogram plot of the data, and the weights calculated by least squares. This is by far the simplest and quickest method, but generally produces less than satisfactory results unless the number of hidden nodes is large.

Referring to figure 2.1 below, a radial basis function network comprises of three layers. The first layer is the input layer which is fully connected to the hidden layer, there are no (adjustable) connection weights between the input and hidden layers. The hidden layer consists of a non linear activation function or basis function. In each hidden layer node, the Euclidean distance between the centers and the input vector is calculated. The activation function uses the Euclidean distance in order to calculate the hidden node output. The output layer consists of a single output. The output layer is connected to the hidden layer via a set of adjustable synaptic weights. The topology is illustrated in figure 2.1 below:

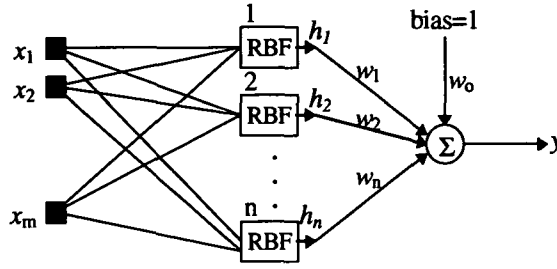


Fig. 2.1

The Radial Basis Function Network

Notation (for node-1 only):

w_1 : output weight associated with node 1, scalar.

$\underline{t}^{(1)}$: node 1 centers, this is a vector: $\underline{t}^{(1)} = \{t_1^{(1)} \dots t_m^{(1)}\}$

β_1 : node 1 spread of centers, also called standard deviations (sd), or widths, this is a scalar.

A RBF network implements the input-output mapping $\mathbf{R}^m \rightarrow \mathbf{R}^1$ according to:

$$y = w_o \times bias + \sum_{j=1}^n w_j \cdot \phi_j(\|\underline{x} - \underline{t}\|) \quad \text{Eqn.2.1}$$

Where: $\{w_o, w_1, \dots, w_n\}$ refers to the connection weights, for each hidden node a center \underline{t} is defined which is a vector \mathbf{R}^m , the Euclidean distance is given by the expression: $v^2 = \|\underline{x} - \underline{t}\|^2$ for $i=1 \dots m$. Several activation functions $\phi()$ are possible, the more common ones are:

Thin plate spline function	$\phi(v) = v^2 \cdot \log(v)$
Gaussian function	$\phi(v) = e^{-v^2/\beta^2}$
Multiquadric function	$\phi(v) = (v^2 + \beta^2)^{1/2}$
Inverse multiquadric function	$\phi(v) = (v^2 + \beta^2)^{-1/2}$

Table 2.1

Typical Basis Functions for RBF Network.

In this thesis, the *Gaussian function* will be used throughout all simulations. The β parameter which represents a standard deviation (width or spreading quantity) must also be determined during training. The RBF contains only a single output, for multi-output applications, the network is duplicated for each output, but the weights, centers and spread of centers must be determined for each individual network. To train a RBF network (or any other neural network), training data is required, the training data set is usually obtained from the actual response from the plant, using some known input function.

Referring to figure 2.1, the RBF has only a single output, therefore when dealing with multi-output systems, then multiple RBF networks are required. For instance, the bioreactor (section 2.2) has two outputs (x_o , s_o), and one input (s_i), therefore we require two individual RBF networks, see figure 2.2A,B. In this case, each RBF network may be trained separately.

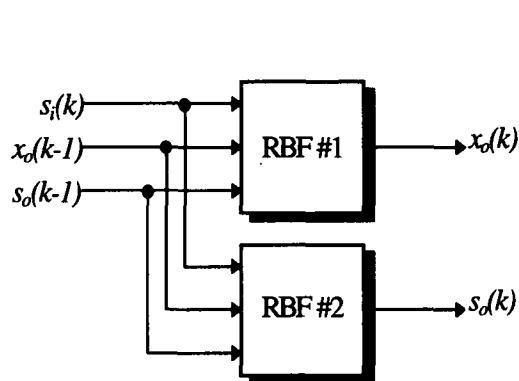


Figure 2.2.A
Training Setup for a RBF Network.

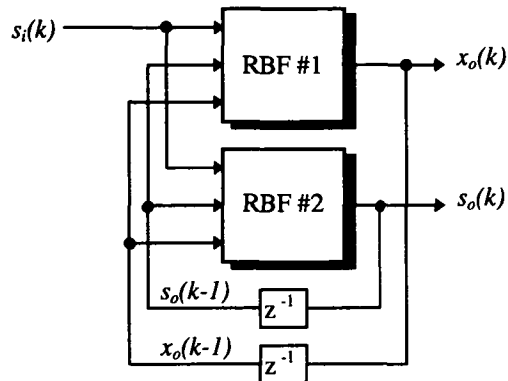


Figure 2.2.B
Operating Setup for a RBF Network

The output error is the difference between the RBF and actual bioreactor plant outputs, this is calculated for each sample j , and this is repeated for each individual RBF network thus:

$$Error = \left[\sum_{j=1}^N (RBF_j - bioreactor_j)^2 \right]^{1/2} \quad \text{Eqn.2.2}$$

Figure 2.2A illustrates the training configuration, and figure 2.2B illustrates the operating configuration in which the neural network simulates the bioreactor. In this setup, the outputs are fed back into the inputs via a z^{-1} delay operator. It is assumed that the forward propagation delay through the RBF network is negligible compared with the sampling time.

2.1.2. Training Radial Basis Function Networks:

The simplest method to train a RBF network is to choose the centers from the input data randomly [5]. The node widths can be estimated by analyzing the spread of centers from a histogram plot of Euclidean distances, the weights can then be computed by least squares. However, arbitrary selection of centers from the data often results in poor performance, requiring excessive number of hidden nodes. Since the performance of the RBF critically depends on the chosen centers, a better method is needed. A number of methods exist which address this problem, for example Chen [12, 13] uses a method of Orthogonal Least Squares to train a RBF, in another paper, Chen et al [15] uses a Hybrid Clustering algorithm for non linear system identification. These methods are more tailored for on line training and identification.

Another popular training method is the *k-means clustering algorithm* [25]. The k-means clustering algorithm first computes the node centers, it then estimates the node widths, and lastly the node weights. This is described in detail below.

(i) **Node Centers:** The node centers are determined by clustering or partitioning the training data set into n equal subclusters, where n is the number of hidden nodes of the RBF network. The average of each subcluster is then calculated. From this initial estimate, a better estimate can be obtained by computing the Euclidean distance between each training data point and the node centers for each node. The training data point is then placed into a bin (there are n bins) belonging to the node center closest to it. After all training points have been binned, the average in each bin is computed. This gives a better estimate of the centers for each particular node. The process is repeated until the centers have converged. From simulations, this can take 10-20 iterations, and is generally very fast.

(ii) **Node Widths:** The node widths are computed using a *p-nearest neighbor heuristic*, generally $p=2$ as suggested in [26]. Using only the node centers, each node width can be estimated by looking for 2 nearest node centers to it, and then computing:

$$\beta_j = \left[\frac{1}{p} \sum_{k=1}^p \|t_j - t_k\|^2 \right]^{1/2} \quad \text{Eqn.2.3}$$

where β_j is the width of the j^{th} node, t_j is its center, and t_k are the nearest centers to it.

(iii) **Node Weight:** The node weight is computed using least squares. All node weights are computed simultaneously. A regularization parameter is often introduced to prevent the weights from becoming too excessive and avoiding overtraining the network. Note also that the node weights are calculated in the same fashion when genetic algorithms are used to train the RBF network.

2.2 Training RBF Networks With Hybrid Genetic Algorithms

This simulation example involves training a Radial Basis Function (RBF) Network to model a bioreactor fermentation process, this is a model matching problem for a nonlinear system. We briefly describe the bioreactor nonlinear equations. We then compare training the RBF using conventional methods with hybrid genetic algorithms. References to bioreactors can be found in [5, 6, 7], and using RBF to model bioreactors [7B]. See also appendix section 8.3.

2.2.1. Bioreactor Mathematical Model:

The bioreactor consists of a tank containing water, nutrients (or substrate) and biomass (or cells). Nutrients and biomass are added to the tank (via the inlet), the nutrients are consumed by the biomass thereby increasing the overall biomass concentration in the tank. Furthermore, biomass is removed from the tank via an outlet, at the same flow rate as the inlet. The overall volume of the liquid in the tank remains constant. The bioreactor is illustrated in figure 2.3 below:

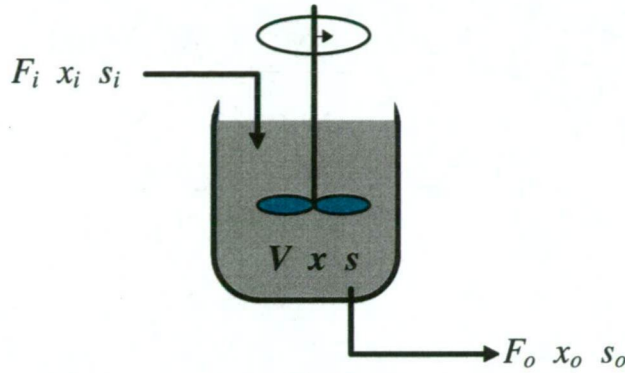


Fig.2.3
Schematic Diagram of a Bioreactor

Where:

- x_i : Input biomass concentration=0
- s_i : Input nutrient concentration.
- F_i : Input flowrate (constant).
- x : Biomass concentration inside the tank \rightarrow output biomass
- s : Nutrient concentration inside the tank \rightarrow output concentration
- x_o : Output biomass concentration
- s_o : Output nutrient concentration
- F_o : Output flowrate

Let $x_1=x$, $x_2=s$, $u=s_i$, then together with the above assumptions we can write in more conventional control system form, the dynamics is a second order nonlinear system. Referring to equation.2.4 below: x_1 =output biomass, x_2 =output nutrient concentration, u =input nutrient concentration:

$$\left. \begin{aligned} \dot{x}_1 &= \mu_m \cdot \left(\frac{x_2}{K_s + x_2} - \frac{F_i}{V} \right) \cdot x_1 \\ \dot{x}_2 &= -K_1 \cdot \left(\mu_m \frac{x_2}{K_s + x_2} \right) \cdot x_1 + \frac{F_i}{V} \cdot (u - x_2) \end{aligned} \right\} \quad \text{Eqn.2.4}$$

In continuous operation, the bioreactor runs at some steady state operating point, we assume that the flow rates are constant and identical i.e.: $F_i = F_o$, therefore the volume of liquid inside the tank is also constant. We assume that the output biomass and nutrient is the same as the biomass and nutrient within the tank i.e.: $x_o = x$, $s_o = s$, assume the input has no biomass $x_i = 0$.

Typical values for the saturation constant and growth rate coefficients are: $\mu_m = 0.3$ and $\kappa_s = 0.1$ to 0.4, $\kappa_1 = 1.25$, the initial conditions: $s(0) = 1.0$, $x(0) = 0.2$. The bioreactor open loop step response is illustrated below:

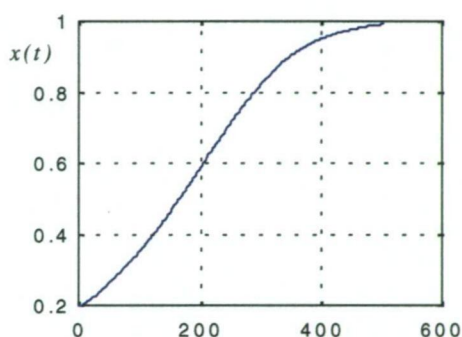


Fig.2.4.A
Open Loop Step Response: $x(t)$: Biomass Output

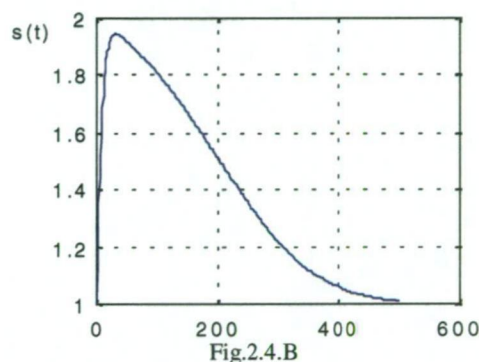


Fig.2.4.B
Open Loop Step Response: $s(t)$: Nutrient Output

When a step input (in nutrient) is added to the tank, assuming perfect and instantaneous mixing, the nutrient in the tank and hence output nutrient is initially high, but the nutrient is gradually consumed by the biomass (fig. 2.4.B) reducing with time. At the same time, the biomass concentration increases as a nonlinear function i.e. fig.2.4.A due to nutrient uptake. Because the bioreactor has two outputs, we require two separate radial basis function networks, this is illustrated in figure 2.2 above. In the next section, simulation results using conventional training methods is provided.

2.2.2. Training With Conventional Methods:

From the mathematical model of the bioreactor, three sets of responses are initially generated, the first response is used to train the RBF using a random input function, the other two responses are used to verify the network using a different random and step input function.

The dynamics of the bioreactor are intrinsically slow, a time step typically of 0.5 seconds is required in the simulation.

(i) **Results Using Conventional Training:** Simulation results with conventional training using MATLAB® running on a Pentium III/750MHz PC, with 300 training samples is given in following pages. Simulation results using 20 and 40 hidden nodes is shown in figures 2.5 and 2.6 respectively. Matlab includes a neural network toolbox which can be used to train the RBF networks. The Matlab functions are: *newrb()*, which is used for training, and *sim()* which is used for simulation and verification purposes. When using Matlab's *newrb()* function, the value of node widths (spread) must be specified. The value of node widths can be estimated from the training data set. Our training data set suggests that this value can be anywhere between 0.2 and 4.0. The choice of spread may require some trial and error before the optimum value can be found.

Table 2.2 below summarizes the results obtained using the conventional matlab neural network toolbox. Results are for 20 and 40 nodes in the first column, the value of spread in the second column, and computational effort (megaflops) in the third column. The last two columns give values for the training error (equation 2.2) for the configuration shown in figure 2.2A, and verification error for the configuration shown in figure 2.2B.

				Training Error		Verification Error	
Nodes	Spread	MFP	Figure	RBF#1	RBF#2	random	step
20	0.8	41	Fig. 2.5	0.01514	0.02343	0.9526	1.3504
40	1.4	105	Fig. 2.6	0.00460	0.00844	0.6998	0.9409

Table 2.2
Training results using conventional matlab neural network toolbox

The training error is shown for each individual network as the sum square difference between the RBF output and bioreactor output. The verification error however is the RMS sum of both networks, using random input and step input test data.

The choice of node spread β has a significant influence on the outcome of the training. The larger that *spread* is the smoother the function approximation will be. Too large a spread means a lot of neurons will be required to fit a fast changing function. Too small a spread means many neurons will be required to fit a smooth function, and the network may not generalize well. Figures 2.5 and 2.6 on the following page compare the RBF output (blue) with the actual bioreactor output (red) using random test data (first row) and step test data (second row) for 20 and 40 nodes.

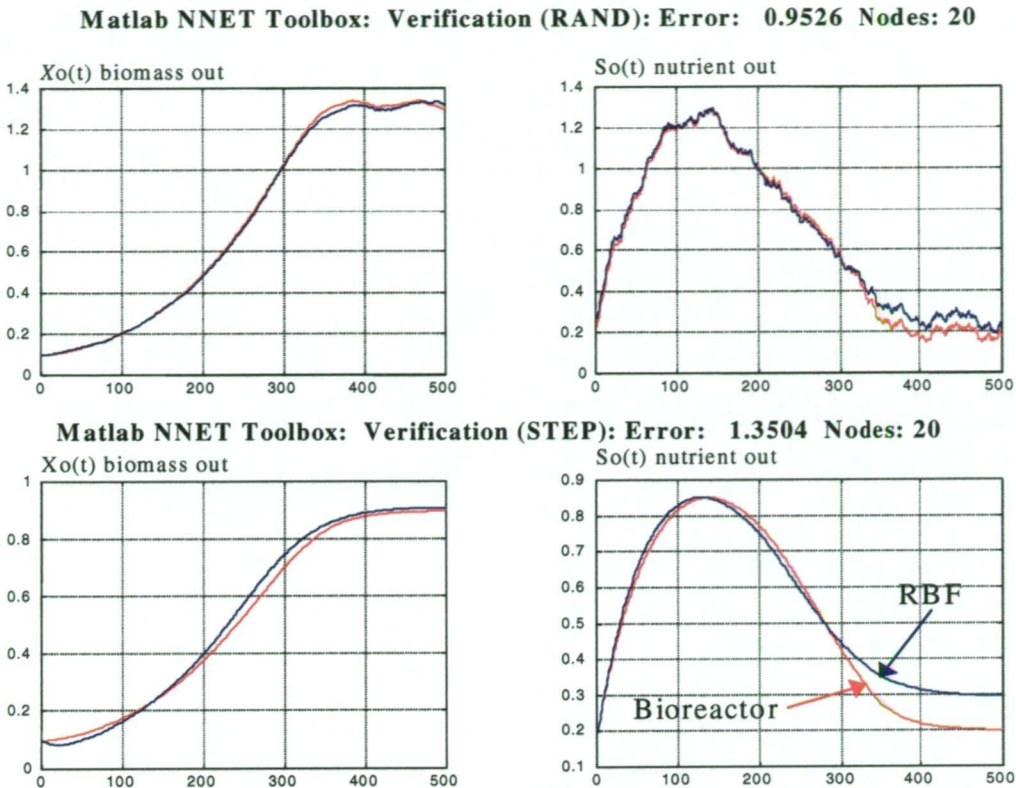


Fig. 2.5

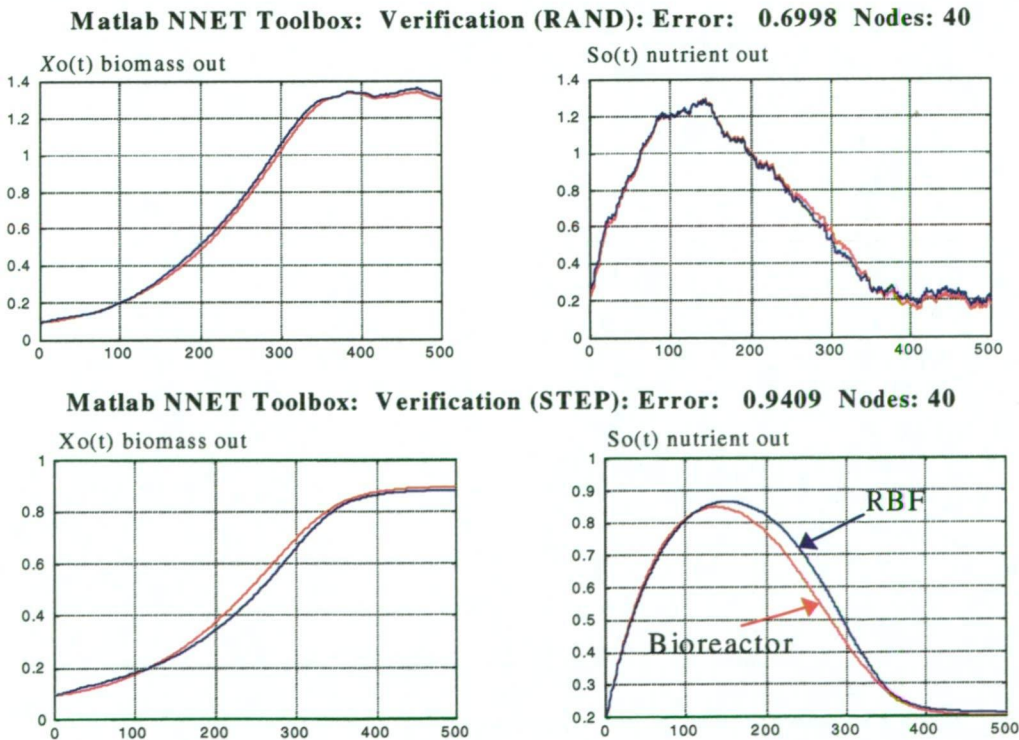


Fig. 2.6

2.2.3. Training With Hybrid Genetic Algorithms:

The RBF can also be trained using hybrid genetic algorithms. The three methods compared are: Conventional Genetic Algorithms (GA), Genetic algorithms + Simulated Annealing (GA+SA) and Genetic algorithms + Greedy Search (GA+GS).

(i) **Genetic Algorithms:** Before discussing the results, the chromosomal representation used for this simulation is illustrated in figure 2.7 below, where: w_o =bias weight, $[w_1 \ t_1^{(1)} \ t_2^{(1)} \ t_3^{(1)} \ sd_1]$ =node-1 weight, centers and standard deviation (widths) respectively. The same is repeated to the remaining nodes 2 to n. The *error* is the Euclidean norm of the difference of the RBF output and Bioreactor output (equation 2.2), the fitness is then computed as the inverse of the error thus: *fitness*=1/(*error*).

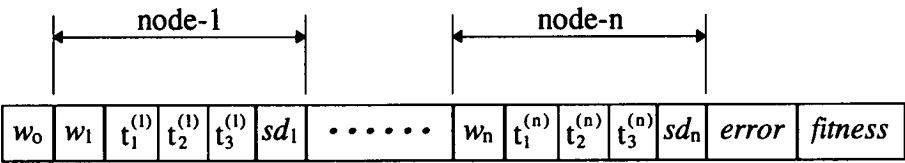


Fig. 2.7
Chromosomal Representation of RBF Network with GA

In this simulation, the GA maintains two separate populations, the first population is used to train the first radial basis function RBF#1, and the second is used to train RBF#2. Consequently both RBF networks can be trained simultaneously. The weights are computed using least squares. For this simulation we set: population=30, maximum generations=200, crossover probability=0.6 and mutation probability=0.1, binary tournament selection and floating point codification was used. Several simulation results are listed to illustrate the stochastic nature of the convergence. Results are tabulated using 20 and 40 nodes:

Nodes	Time	MFP	Gen	Training Error		Verification Error	
				RBF#1	RBF#2	random	step
20	10:43	10540	140	0.00233	0.00650	0.33975	0.59029
	10:43	10550	140	0.00269	0.00435	0.64191	0.79235
	10:56	10551	140	0.00426	0.00339	0.91685	0.26338
	10:49	10557	140	0.00192	0.00361	0.57244	0.81601
	10:55	10536	140	0.00214	0.00800	0.53976	0.83718
40	37:33	39917	160	0.00188	0.00176	0.33477	0.22392
	37:32	39908	160	0.00227	0.00263	0.32801	0.32676
	37:33	39866	160	0.00232	0.00252	0.34336	0.41723
	37:47	39832	160	0.00227	0.00276	0.32069	0.14259
	37:43	39836	160	0.00278	0.00367	0.34063	0.19805

Table 2.3
Training results using conventional genetic algorithms

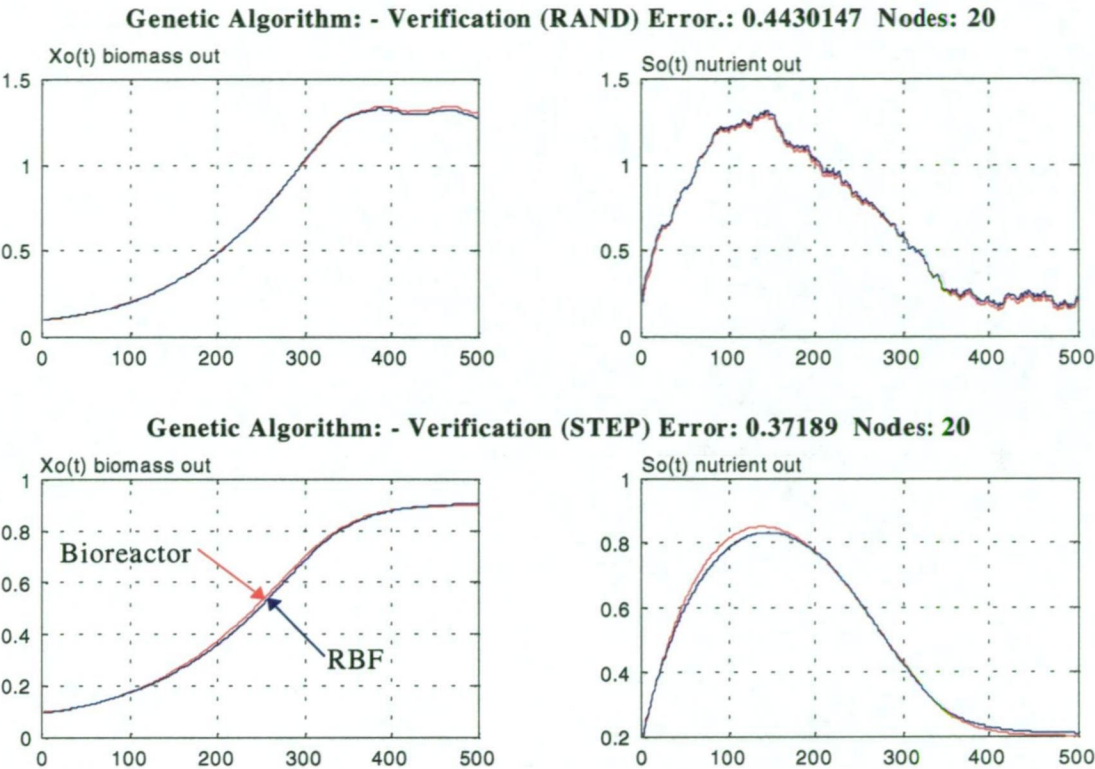


Fig. 2.8

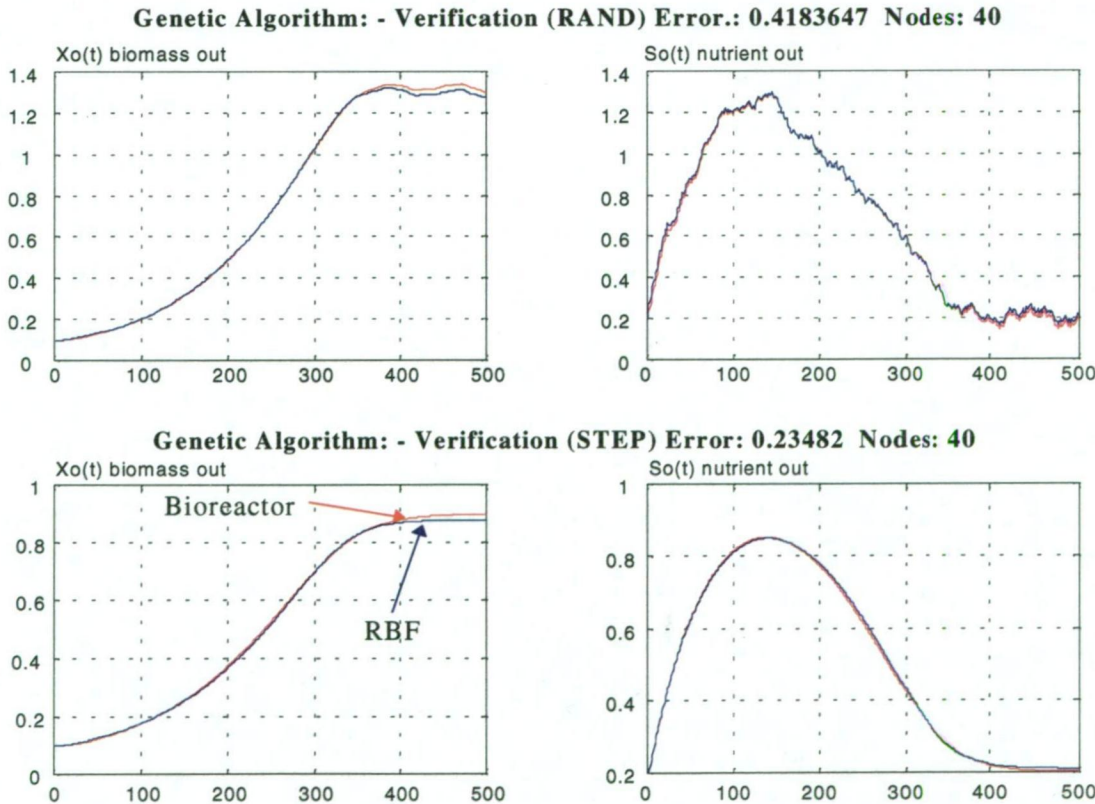


Fig. 2.9

(ii) **Genetic Algorithms+Simulated Annealing (GA+SA):** Simulated annealing requires a search vector and a temperature annealing schedule. The search vector is defined in a similar manner to that of genetic algorithms, and is illustrated in figure 2.10 below:

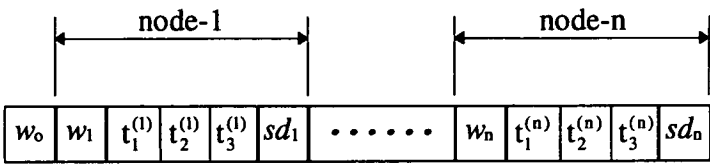


Fig. 2.10
Search vector for GA+SA algorithm

There are two search vectors, one for each RBF network. Again, both networks are trained simultaneously. The temperature annealing schedule is defined as: initial temperature: T_0 , final temperature: T_f , the temperature at the k^{th} iteration is given by: $T_k = \alpha \cdot T_{k-1}$, this is an exponential annealing schedule where alpha is computed from:

$$\alpha = 10^{\left(\frac{1}{N} \log(T_f/T_0)\right)} \tag{Eqn.2.5}$$

giving values of alpha typically between 0.9-0.98, N=number of iterations. Results for this simulation using 20 and 40 nodes is tabulated below, values are: $T_0=1$ (normalized), $T_f=0.001$, iterations=200 (20 nodes) and 240 (40 nodes)

Nodes	Time	MFP	Gen	Training Error		Verification Error	
				RBF#1	RBF#2	random	step
20	9:41	10409	200	0.00297	0.00464	0.41078	0.25718
	9:42	10406	200	0.00294	0.00362	0.43302	0.44228
	9:36	10409	200	0.00264	0.00342	0.67604	0.18233
	9:47	10408	200	0.00285	0.00365	0.48328	0.17402
	9:44	10407	200	0.00288	0.00367	0.41093	0.40195
40	35:35	41360	240	0.00178	0.00353	0.35444	0.21236
	35:28	41362	240	0.00182	0.00313	0.37063	0.19034
	35:39	41372	240	0.00212	0.00342	0.31717	0.28021
	35:17	41377	240	0.00189	0.00298	0.30040	0.24368
	35:24	41341	240	0.00201	0.00237	0.28266	0.33288

Table 2.4
Training results using genetic algorithms and simulated annealing

Typical plots for 20 and 40 nodes are illustrated on the following page Fig.2.11 and Fig.2.12.

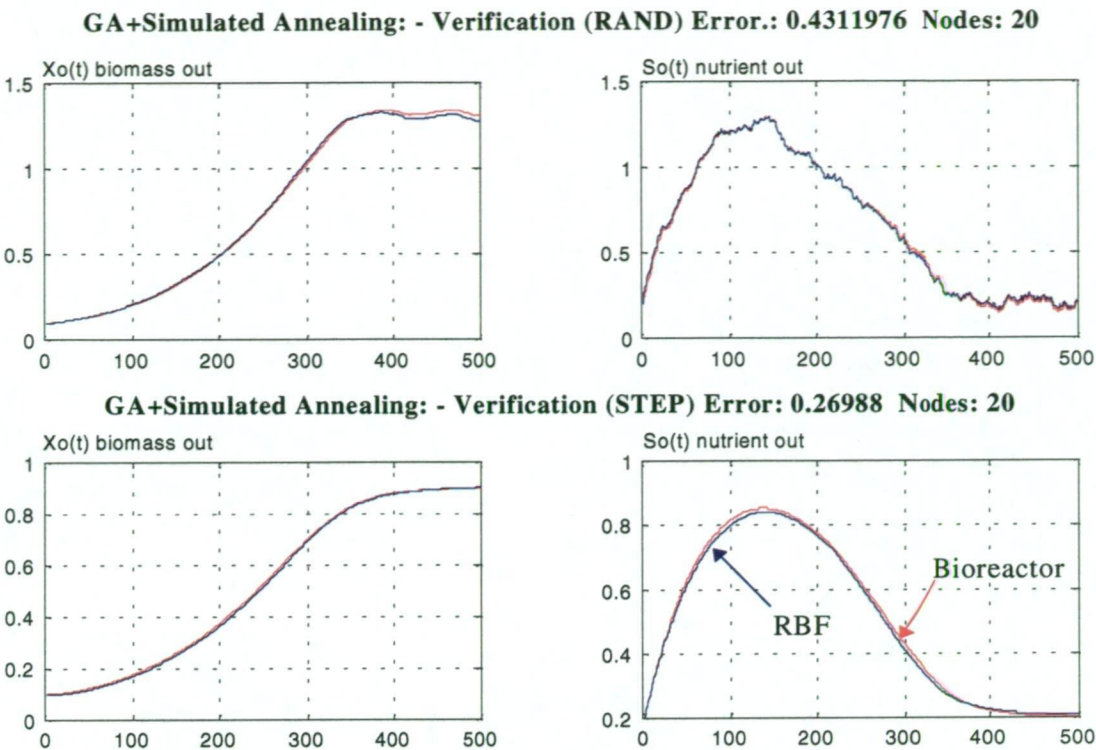


Fig. 2.11

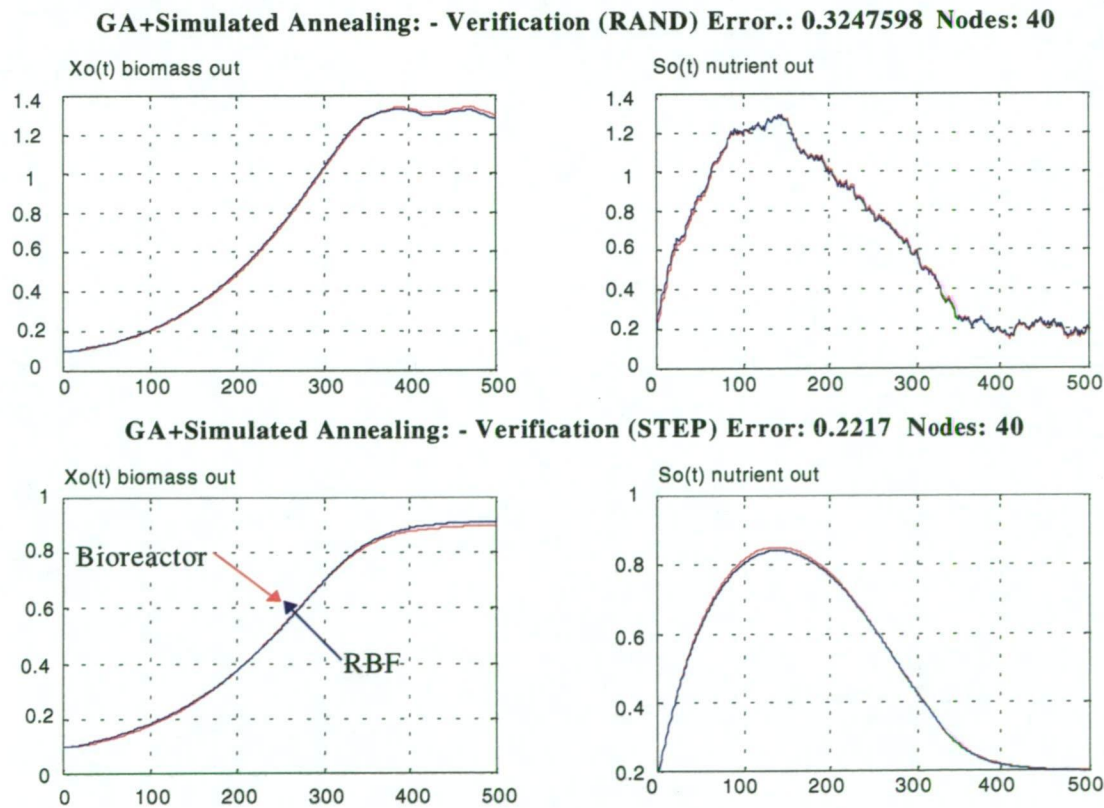


Fig. 2.12

(iii) **Genetic Algorithms+Greedy Search (GA+GS):** The greedy algorithm uses a search vector which is identical to that of simulated annealing (Fig.2.10). The performance of the greedy algorithm strongly depends upon the initial value. Typical results are tabulated below:

			Training Error		Verification Error	
Nodes	Time	MFP	RBF#1	RBF#2	random	step
20	4:50	4785	0.003160	0.005724	0.45574	0.42448
	4:52	4849	0.003453	0.004171	0.35973	0.44323
	5:46	5750	0.003258	0.005203	0.45351	0.38673
	5:25	5348	0.004603	0.003598	0.62629	0.26401
	6:35	6566	0.002091	0.005390	0.43774	0.45490
40	6:49	8158	0.002064	0.001980	0.31679	0.24220
	9:39	11830	0.001946	0.003705	0.26691	0.22193
	17:58	21789	0.001235	0.001946	0.28643	0.20710
	19:21	23347	0.001707	0.001740	0.44539	0.13030
	25:13	30777	0.001135	0.001781	0.28535	0.15008

Table 2.5
Training results using genetic algorithms and greedy search

The combined GA+GS converges in about half the time/flops when compared with the standard genetic algorithm. Results for 20 and 40 nodes are illustrated in figures 2.13 and 2.14 below.

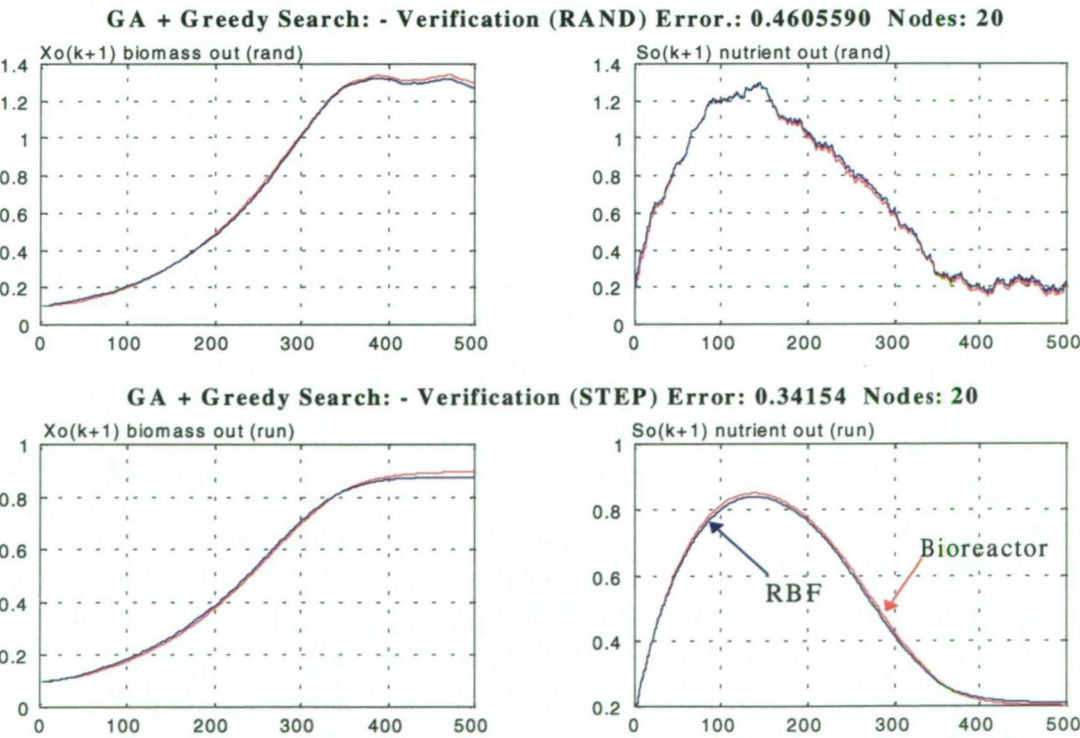


Fig. 2.13

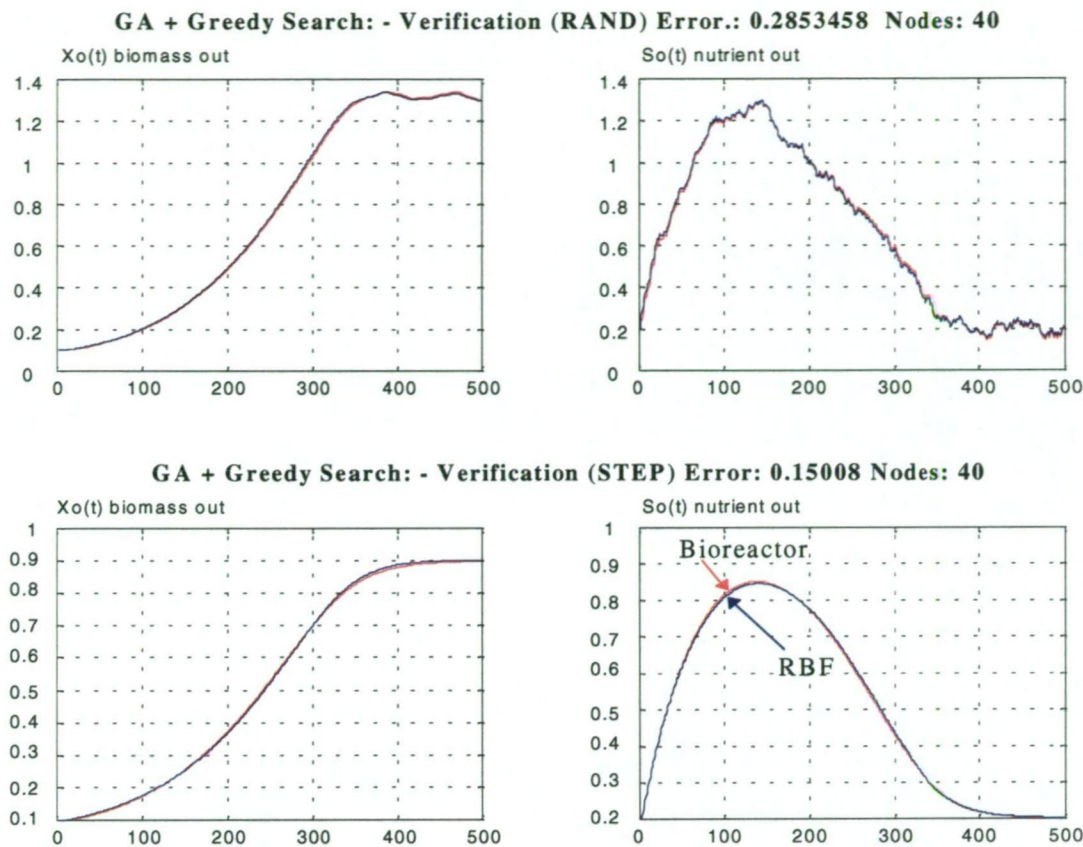


Fig. 2.14

2.2.4 Comparison of Results:

From the previous results, all three GA methods yield a network with superior performance when compared with a RBF network trained using conventional methods. However the results provide no indication of the actual convergence rate for each of the three GA methods. In figure 2.15 below, the network performance (i.e. training error) is plotted as a function of the training time (or FLOPS) for a network with 20 nodes. The conventional GA is plotted in red, hybrid GA+simulated annealing in green, and hybrid GA+greedy search in blue. Both hybrid methods converge slightly faster than the conventional GA. The same is repeated for the RBF network with 40 nodes, this is illustrated in figure 2.16 below.

Table 2.6 below summarizes the training and verification errors obtained after a fixed number of computations: 10,000 MFP for the 20 node RBF, and Table 2.7 for 40 nodes after 40,000 MFP computations.

Training and verification error comparison after 10,000 MFP computations, for 20 node RBF network:

	Training Errors:				Verification Errors	
METHOD:	RBF#1 Error	RBF#2 Error	Error1+Error2	Train time	RBF#1 Error	RBF#2 Error
MATLAB	0.015140	0.023430	0.038570	00:20	0.95300	1.35000
GA:	0.003535	0.007448	0.010984	11:10	0.44301	0.37189
GA+SA:	0.002625	0.003251	0.005877	9:50	0.43120	0.26988
GA+GREEDY:	0.003618	0.003011	0.006630	9:40	0.44312	0.11502

Table 2.6

Typical convergence rates for conventional genetic algorithms and hybrid genetic algorithms versus the computational effort for 20 node RBF network:

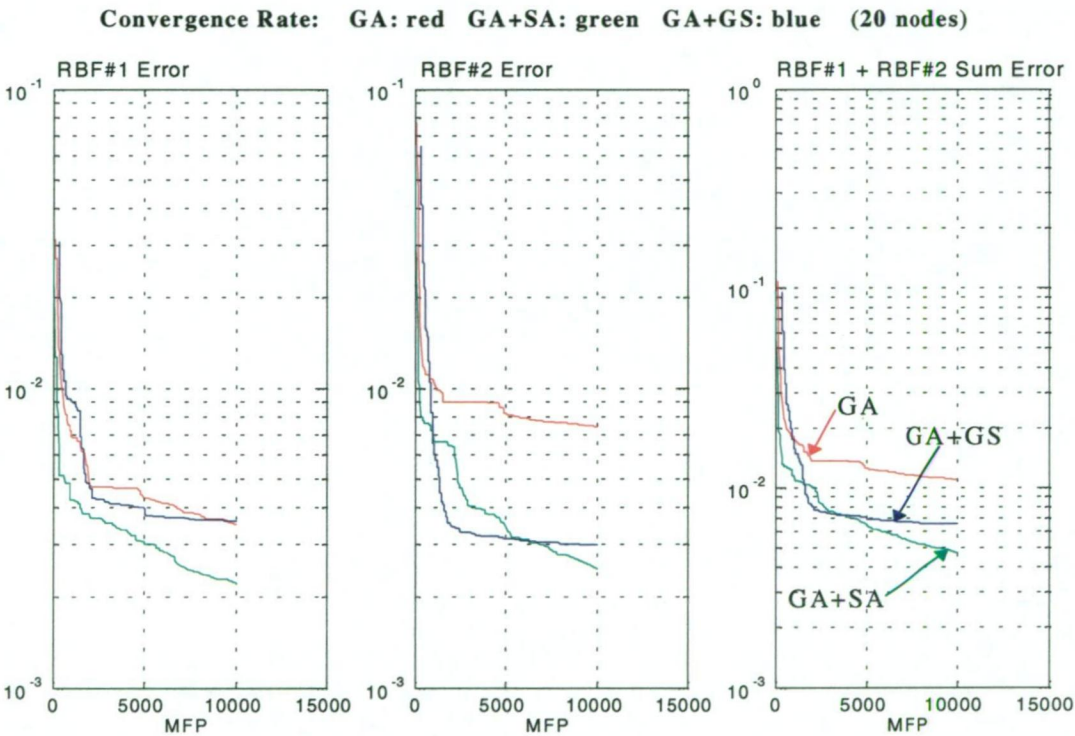


Fig. 2.15

Training and verification error comparison after 40,000 MFP computations, for 40 node RBF network:

	Training Errors:				Verification Errors	
METHOD:	RBF#1 Error	RBF#2 Error	Error1+Error2	Train time	RBF#1 Error	RBF#2 Error
MATLAB	0.004600	0.008440	0.013040	:41	0.9473	0.4495
GA:	0.002771	0.003924	0.006695	35:02	0.4183	0.2348
SA:	0.001572	0.001693	0.003265	32:05	0.3247	0.2217
GREEDY:	0.001269	0.002175	0.003444	30:47	0.2232	0.1478

Table 2.7

Typical convergence rates for conventional genetic algorithms and hybrid genetic algorithms versus the computational effort for 40 node RBF network:

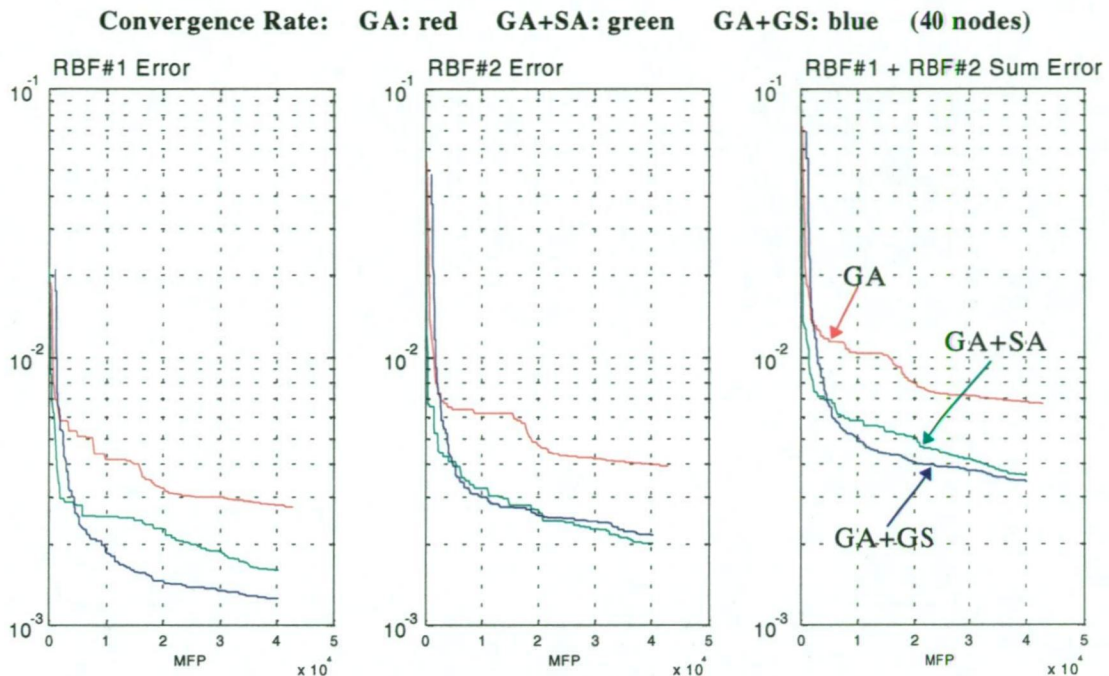


Fig. 2.15

2.3 Chapter Summary and Conclusion:

From the simulation results, it is clear that training RBF using genetic algorithms can produce a network with superior performance and fewer nodes when compared with conventional training schemes. However training times using GA are excessive. Even hybrid GA methods still require a high computational effort compared with the more traditional methods. Thus it is unlikely that applications requiring on-line training of RBF networks using these methods is appropriate. However, in applications where the smallest number of nodes is desirable, then off-line training using GA and hybrid GA may be more feasible.

We have investigated two different methods of crossover: swapping and weighted average. From simulation results, the averaging crossover converges quicker but loses genetic diversity more rapidly. The swapping crossover has slower convergence but retains diversity. The results above are for weighted average crossover only.

Some key points regarding GA are outlined next.

(i) Population Initialization:

The initialization of a population is an important factor. Two points to consider are: to ensure that the initial population spans the entire possible search space in which the solution is contained. And secondly, if the approximate solution is known, to initialize the population near the solution .

(ii) Mutation Operator:

Two forms of mutation operators are used:

$$x_j = x_j + k \times rand \quad \text{Eqn.2.10a}$$

and

$$x_j = x_j \times (1 + k \times rand) \quad \text{Eqn.2.10b}$$

the first (Eqn.2.10a) allows a wide search space to be analyzed, the second works well near the solution (narrower search space). The two methods are used with a probability of 0.5, and k is a mutation gain parameter which can be user selected or gradually decreases over time. Mutation is applied uniformly over the components of the chromosome. For instance, given the following chromosomal representation (Fig.2.16) with parameters $(x_1, x_2 \dots x_n)$ to solve for, the mutation operator is applied to each element of the chromosome in sequence, beginning from x_1 to x_n with probability P_m :

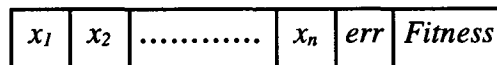


Figure 2.16

A typical mutation algorithm would be:

```
%BIASED MUTATION:
for j=1:n
    if (rand < Pm)
        r = fix(3*rand);
        gain = mutationGain*10^(-r);

        if (rand<0.5)
            %wide search space:
            GeneB(j) = GeneB(j) + gain*randn;
        else
            %narrower search space:
            GeneB(j) = GeneB(j)*(1 + gain*randn/10);
        end;
    end
end
```

Note that either one of the two mutation equations (Eqn.2.10a) and (Eqn.2.10b) is chosen randomly.

(iii) Crossover Operator:

A uniform crossover operator is used. This means that for each parameter x_j of the chromosome (Fig.2.16), the resulting offspring is the weighted average of the two parents. This is applied uniformly for $j=1..n$, and probability P_c to each parameter x_j , the crossover algorithm used is:

```
%SWAPPING CROSSOVER OPERATOR:
for j=1:n
    if (rand < Pc)
        eta = rand;
        GeneB(j) = eta*GeneA1(j) + (1-eta)*GeneA2(j);
    end;
end;

%HOOKE-JEEVES CROSSOVER OPERATOR:
if (rand<0.25)
    fit1 = GeneA1(cols);
    fit2 = GeneA2(cols);

    if (fit1>fit2)
        GeneB = 2*GeneA1 - GeneA2;
    else
        GeneB = 2*GeneA2 - GeneA1;
    end
end
```

If the swapping crossover method is used, the value of α is simply set to zero. A value of $\alpha=0.5$ can sometimes produce rapid convergence. Furthermore, the addition of the *Hooke-Jeeves* crossover operator discussed in chapter 1 is applied with a low probability of 0.25.

(iv) Population Inversion:

Two methods which we have used are: (i) combine the parents and offspring into one population, and then choose the fittest N chromosomes from this population, or (ii) simply replace the old population with the new population. We found that the first method can lead to premature convergence and loss of genetic diversity. The second method retains genetic diversity, but can also be inefficient because offsprings with very poor fitness can remain in the population. Trial and error may be required depending on the application.

(v) Future work:

1. As a topic of interest, compare genetic algorithms with orthogonal least squares in training radial basis function networks.
2. Use genetic algorithms and hybrid genetic algorithms to train multilayer perceptrons (MLP) neural networks, compare with backpropagation.
3. Hybridize genetic algorithms using Tabu local search, and compare with results using Greedy search and simulated annealing.

2.4 References and Further Reading:

Introductory References to Genetic Algorithms:

- [1] J.H.Holland,
Adaption in Natural and Artificial Systems.
University of Michigan Press, Ann Arbor, 1975.
- [2] L. Davis
Handbook of Genetic Algorithms
Van Nostrand Reinhold, 1991
- [3] Charles L. Karr, L. Michael Freeman
Industrial Applications of Genetic Algorithms
CRC Press 1999
- [4] Mitchell, Melanie
Introduction to Genetic Algorithms
Cambridge, Mass. MIT Press 1996

References on Bioreactors:

- [5] Y.Y. Yang, D.A. Linkens,
Modelling of Continuous Bioreactors via Neural Networks.
Transactions of the Institute of Measurement and Control, Vol.15, No.4, pp.158-169, 1993.
- [6] I.Queinnec, B.Dahhou, M.M'Saad
On Adaptive Control of Fedbatch Fermentation Processes.
International Journal of Adaptive Control and Signal Processing, Vol.6, pp.521-536, 1992.
- [7] J.D. Boskovic
Stable Adaptive Control of a Class of Nonlinearly-Parametrized Bioreactor Processes.
Proceedings of the American Control Conference, Seattle Washington, pp.1795-1799, June 1995.
- [7B] M.R.Warnes, J.Glassey, G.A.Montague, B.Kara
Application of Radial Basis Function and Feedforward Artificial Neural Networks to the Escherichia Coli
Fermentation Process.
Neurocomputing, vol.20, pp.67-82, 1998

References on Training Neural Networks and Genetic Algorithms:

- [8] A.J.F. van Rooij, L.C.Jain, R.P.Johnson
Neural Network Training Using Genetic Algorithms.
World Scientific Publishing Co. 1996
- [9] Hsi-Chin Hsin, Ching-Chung Li, M.Sun, R.J.Sclabassi,
An Adaptive Training Algorithm for Back- Propagation Neural Networks.
IEEE Transactions on Systems, Man, and Cybernetics, Vol.25, No.3, pp.512-514, March 1995.
- [10] C. Darken, J.Chang, J.Moody,
Learning Rate Schedules for Faster Stochastic Gradient Search.
Neural Networks for Signal Processing, 1992.
- [11] D.S.Broomhead, D.Lowe,
Multivariable Functional Interpolation and Adaptive Networks,
Complex Systems, Vol.2 pp.321-355, 1988.
- [12] S.Chen, C.F.N.Cowan, P.M.Grant
Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks
IEEE Transactions on Neural Networks, Vol.2, No.2, pp.302-309, March 1991

- [13] S.Chen, S.A.Billings, W.Luo
Orthogonal Least Squares Methods and their Application to Non-linear System Identification.
International Journal of Control, Vol.50, No.5, pp.1873-1896, 1989.
- [14] A.Sherstinsky, R.W.Picard
On the Efficiency of the Orthogonal Least Squares Training Method for Radial Basis Functions.
IEEE Transactions on Neural Networks, Vol.7, No.1, pp.195-200, January 1996.
- [15] S.Chen, S.A.Billings, P.M.Grant
Recursive Hybrid Algorithm for Non-linear System Identification using Radial Basis Function Networks.
International Journal of Control, Vol.55, No.5, pp.1051-1070, 1992.
- [16] S.A.Billings, G.L.Zheng
Radial Basis Function Network Configuration Using Genetic Algorithms.
Neural Networks, Vol.8, No.6, pp.877-890, 1995

Applications of Genetic Algorithms in Control Systems:

- [17] K.Krishnakumar, D.E.Goldberg
Control System Optimization Using Genetic Algorithms.
International Journal of Guidance Control and Dynamics, Vol.15, No.3, pp.735-740, May-June 1992
- [18] A. Varsek, T. Urbancic, B. Filipic
Genetic Algorithms in Controller Design and Tuning.
IEEE Transactions on Systems, Man and Cybernetics, Vol.23, No.5, pp.1330-1339, Sept-Oct. 1993
- [19] D.C.Dracopoulos, A.J. Jones
Neural Networks and Genetic Algorithms for the Attitude Control Problem.
From Natural to Artificial Computation, Lecture Notes in computer Science, Vol.930, Springer 1995
- [20] J.J. Grefenstette
Optimization of Control Parameters for Genetic Algorithms
IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-16, No.1, pp.122-128, Jan-Feb. 1986
- [21] T.Kumagai, M.Wada, R.Hashimoto, A.Utsugi
Dynamical Control by Recurrent Neural Networks Through Genetic Algorithms
International Journal of Adaptive Control and Signal Processing, Vol.13, pp.261-271, 1999

Simulated annealing and Greedy algorithms:

- [22] P.J.M van Laarhoven, E.H.L. Aarts
Simulated Annealing, Theory and Applications
Mathematics and Its Applications, Kluwer Academic Publishers, 1988
- [23] T.A.Feo, M.G.C. Resende
Greedy Randomized Adaptive Search Procedures
Journal of Global Optimization, Vol.6, No.2, pp.190-133, March 1995
- [24] A.Fanni, M.Marchesi, A.Serri, M.Usai
A Greedy Genetic Algorithm for Continuous Variables Electromagnetic Optimization Problems
IEEE Transactions on Magnetics, Vol.33, No.2, pp.1900-1903, March 1997
- [25] J.A.Leonard, M.A.Kramer
Radial Basis Function Networks for Classifying Process Faults
IEEE Control Systems, pp.31-38, April 1991
- [26] T.J.Moody, C.J.Darken
Fast Learning in Networks of Locally Tuned Processing Units.
Neural Computation, Vol.1, pp.151-160, 1989

3

Eigenstructure Assignment Using Hybrid Genetic Algorithms:

Contents:

3.1 Eigenstructure Assignment	p.3.2
3.1.1 Introduction	p.3.2
3.1.2 Full Eigenstructure Assignment and Moore's Method.	p.3.3
3.1.3 Partial Eigenstructure Assignment	p.3.5
3.1.4 Robust Eigenstructure Assignment.	p.3.6
3.1.5 Response of LTI Systems from Eigenstructure Information	p.3.9
3.2 Partial Eigenstructure Assignment for Static Compensators.	p.3.10
3.2.1 Theory	p.3.10
3.2.2 Simulation 3.1: Fixed Eigenvalues	p.3.13
3.2.3 Simulation 3.2: Domain Constrained Eigenvalues.	p.3.15
3.3 Eigenstructure Assignment for Dynamic Compensators	p.3.19
3.3.1 Theory	p.3.19
3.3.2 Simulation 3.3: Static Output Feedback.	p.3.23
3.3.3 Simulation 3.4: Dynamic Control Output Feedback	p.3.25
3.4 Robust Eigenstructure Assignment	p.3.29
3.4.1 Theory	p.3.29
3.4.2 Simulation 3.5: Hybrid Genetic Algorithms.	p.3.30
3.5 Chapter Summary and Conclusion	p.3.34
3.6 References and Further Reading	p.3.36

3.1 Eigenstructure Assignment:

3.1.1 Introduction:

The aim of this chapter is to apply hybrid genetic algorithms, and concepts of constrained optimization theory discussed in chapter 1, to the design of control systems based on eigenstructure assignment (ESA). Three different designs are considered: (i) Full state static feedback, (ii) Output feedback using a dynamic compensator, and (iii) Robust eigenstructure assignment. Results are verified with conventional eigenstructure assignment methods. An introduction to eigenstructure assignment is briefly outlined below.

Eigenstructure assignment is a powerful design technique which has developed over the last twenty years. The objective of eigenstructure assignment is to determine the feedback gain matrix K such that the closed loop eigenvalues and eigenvectors (eigenstructure) are as close as possible to some design specifications. This method allows the designer to directly satisfy damping, settling time and mode decoupling specifications by the proper choice of eigenvalues and eigenvectors. The behavior of a linear dynamic system can be completely characterized by its eigenstructure. The eigenvalues determine the stability of the system while the eigenvectors determine the contribution of each system mode to the overall system outputs or states. More specifically, the output for a linear discrete time system $x(k+1) = \Phi \cdot x(k)$ with zero input, is given by [1]:

$$x(k) = V \cdot \Lambda^k \cdot V^{-1} \cdot x(0) \quad \text{Eqn.3.1}$$

Where V =a matrix of eigenvectors of Φ , Λ^k =diagonal matrix of corresponding eigenvalues, and $x(0)$ initial condition.

There are essentially three types of feedback: full state feedback, output feedback, and constrained output feedback [1, 2]. Full state feedback [7] allows greater design freedom in the choice of eigenstructure placement, but may require an observer for state estimation. The more popular method is output feedback, this method has more restrictions on the placement of eigenvectors, but does not require a state observer. The third method of constrained output feedback sets some entries of the output feedback gain matrix to zero, reducing controller complexity and increasing reliability, however it is not always evident which entries should be zero. One obvious method [3, 4] would simply be to choose those entries which have the smallest influence upon the eigenvalues and eigenvectors of the closed loop system.

Note in particular that pole placement (i.e. Ackerman's formula) and optimal-LQR (matrix Riccati equation) controller designs are simply a special instance of eigenstructure assignment where only the eigenvalues are taken into consideration.

One popular method of computing the feedback gain matrix K for MIMO systems is by Moore's method [6], and is described in section 3.1.2 below. Other methods include parameterization of controllers [5] for full state feedback. Extensions to improve design freedom of parametric approaches include [7] in which all combinations of allowable subspaces is computed. Eigenstructure assignment has been used in reconfigurable control systems [8] in which the operating conditions of the plant change and new feedback gain matrix K is re-computed to maintain the eigenvalues and vectors as close as possible to the original design specifications. Applications to aircraft control using partial eigenstructure assignment in which not all eigenvalues are prescribed [9] uses minimum norm to ensure stability of the remaining unspecified eigenstructure. In [10], eigenstructure is used to achieve mode decoupling and desired damping/rise time for a high performance (F-15) aircraft using output feedback. Applications to a commercial transport (Boeing 767) using eigenstructure to design a lateral autopilot are discussed [12]. More recently, the area of robust eigenstructure assignment including reconfigurable control has received considerable attention. The task of reconfigurable control is twofold: first to guarantee performance and stability whenever possible, and secondly, to recover control effectiveness under changing or failed conditions. Reconfiguration is performed *on-line*, in the event of a failure the fault detection and isolation system (see chapter 6) should provide accurate isolation and identification of the fault. This chapter will investigate the application of hybrid genetic algorithms for solving general and robust eigenstructure problems.

3.1.2 Full Eigenstructure Assignment by Moore's Method

Eigenstructure assignment by Moore's method is presented below, this method can be later used as a comparison with solutions obtained using hybrid genetic algorithms. Moore's method [1] requires that all the eigenvalues and eigenvectors are specified at the design stage. In [9], partial eigenstructure assignment using Moore's method is discussed. Moore's method requires full state feedback, however the solution can be obtained without iteration. The procedure below is presented in algorithmic form rather than giving a complete derivation of Moore's method. Given a linear time invariant (LTI) system in state space and continuous time:

$$\dot{x} = A.x + B.u \quad \text{Eqn.3.2}$$

and a full state feedback control law:

$$u = -K.x \quad \text{Eqn.3.3}$$

for $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ we require that the closed loop eigenvalues and eigenvectors correspond as closely as possible to those specified, thus the eigenvalue problem becomes:

$$(A - B.K)v_i = \lambda_i.v_i \quad \text{Eqn.3.4}$$

where: $\{\lambda_i, v_i\}_{i=1,n}$, are the desired eigenvalues and eigenvectors respectively. The algorithm is given below, note the necessary condition: $n = \text{rank}(A)$. Moore's method requires full specification of all eigenvalues and eigenvectors:

Procedure for Moore's Method:

```

Repeat j=1 TO n (for each eigenvalue)
  setup the matrix:
     $S = [\lambda_j.I - A \mid B]$ 
  Compute the right nullspace of the above matrix  $\rightarrow M, N$ :
     $\begin{bmatrix} M \\ N \end{bmatrix} = \text{null}(S)$ 
  Compute the column vectors  $V$  and  $W$  by least squares solution
     $\alpha = (M^T.M)^{-1}.M^T.v_j$ 
     $v_j = M.\alpha$ 
     $w_j = N.\alpha$ 
  Construct matrices  $V, W$  from column vectors  $v_j, w_j$  thus:
     $V = [\dots v_j \dots]$ 
     $W = [\dots w_j \dots]$ 
end

```

Fig.3.1

The full state feedback gain can then be computed from the matrices thus: $K = -W.V^{-1}$. Note that Moore's method gives the best match (in the least squares sense) to the specified eigenvectors. In fact, the user specified eigenvectors may be unrealizable or unachievable, and Moore's method gives the closest best match to the specified eigenvectors. As we shall see later, the achievable eigenvectors must belong to the subspace spanned by the columns of $S_i = (\lambda_i.I - A)^{-1}.B$. If this is the case, Moore's method will then yield a precise match to the specified eigenstructure.

There have been many variations to this method with partial eigenstructure and output feedback instead of state feedback. When dealing with partial eigenstructure assignment where only some of the eigenvalues/eigenvectors have been specified, the question of how best to allocate the remaining ones is the subject of robust eigenstructure assignment.

Note also that Moore's method fails when one or more closed loop eigenvalues are required to be identical to the open loop eigenvalues. When dealing with partial eigenstructure assignment, this method can be modified to deal with eigenvalues/vectors which are not specified or are not critical in the design.

3.1.3 Partial Eigenstructure Assignment:

In many practical situations, the full specification of the eigenstructure is not known (or not necessarily required), but only certain elements of the eigenstructure are specified. Thus the problem is to find the best possible eigenstructure which matches the specified components of the required eigenstructure as closely as possible without regard to the other remaining unspecified components. This is the partial eigenstructure assignment problem. The conventional solution [1] is outlined below, for each single eigenvalue and desired eigenvector $v^{(d)}$

$$v^{(d)} = [v_1 \quad x \quad x \quad v_j \quad x \quad x \quad v_k \quad x]^T$$

where x =don't care (represents unspecified components) and v_i are the specified components. A simple re-ordering operation is used to rearrange the above vector into two subvectors:

$$reorder: \{v^{(d)}\} \rightarrow \bar{v}^{(d)} = \begin{bmatrix} n \\ d \end{bmatrix} \quad \text{Eqn.3.5}$$

where n =subvector of specified components, and d =subvector of unspecified components. The achievable eigenvectors must be selected from the subspace spanned by: $S = (\lambda.I - A)^{-1}B$. Thus all achievable eigenvectors are given by: $v^{(a)} = S.g$. The S matrix is also reordered in the same sequence as previously in equation 3.5:

$$reorder: \{S\} \rightarrow \bar{S} = \begin{bmatrix} N \\ D \end{bmatrix} \quad \text{Eqn.3.6}$$

In order to minimize the norm of the difference between the actual and desired eigenvectors the g vector can be estimated by least square thus:

$$g = (N^T \cdot N)^{-1} \cdot N^T \cdot n \quad \text{Eqn.3.7}$$

If however the dimension $\dim\{n\} < m$, where m =number of inputs ($u \in \mathcal{R}^m$), then the solution can be found given by:

$$g = N^T (N \cdot N^T)^{-1} \cdot n \quad \text{Eqn.3.8}$$

The feedback gain K can be computed from the g vector. A full detailed description of the partial eigenstructure algorithm we implemented is provided in the appendix (see 8.2). This will be used for comparison with solutions obtained using genetic algorithms. If only partial eigenstructure specification is given, then the question of how best to choose the remaining unspecified eigenvalues/eigenvectors becomes the next topic of discussion: *robust eigenstructure assignment*.

3.1.4 Robust Eigenstructure Assignment:

More recently, robust eigenstructure assignment has been a topic of research interest including areas of reconfigurable control systems [18]. The objective is to design a feedback control law in which the eigenstructure of the closed loop system is unaffected, or minimizing the effects caused by changes in the operating conditions (or failures) of the nominal system. Some examples of robust eigenstructure assignment include [15] in which the attempt is to minimize the difference (norm) between the desired and achievable eigenvalues/vectors.

This chapter outlines the general framework in which the robust eigenstructure problem can be defined and solved using hybrid genetic algorithms. Other methods of robust eigenstructure formulation include the minimization of sensitivity and complimentary sensitivity function norms [17], which also appear to be a popular techniques. Some other examples with genetic algorithms [16, 19] have recently emerged.

From the previous chapter, we discussed optimization problems in which hybrid genetic algorithms can be readily applied to. Genetic algorithms require that the robust eigenstructure problem first be formulated in a generalized multiobjective constrained optimization framework. This formulation is developed below.

The eigenstructure assignment problem starts with the definition of the desired closed loop eigenvalues and eigenvectors, and then computes the feedback gain matrix K to meet these requirements. In general, not all eigenvalues and eigenvectors are specified, the question is then of how best to choose the remaining (unspecified eigenvalues/eigenvectors) so that the system is stable, robust and the closed eigenvectors are as close to those specified. Consider the following linear time-invariant and completely controllable system:

$$\begin{aligned}\dot{x}(t) &= A.x(t) + B.u(t) \\ y(t) &= C.x(t)\end{aligned}\tag{Eqn.3.9}$$

Where $x \in \mathbb{R}^n$, is the state vector, and $u \in \mathbb{R}^m$ is the control vector, using full state feedback: $u(t) = -K.x(t)$ the closed loop system becomes:

$$\dot{x}(t) = (A - B.K).x(t)\tag{Eqn.3.10}$$

Where the closed loop eigenstructure of $(A - B.K)$ must match as closely as possible to those specified. It is assumed that the controllability condition is satisfied i.e. rank of the controllability matrix = n . Given this condition, all eigenvalues can be placed, and up to m entries in each eigenvector can be placed in specified locations. The robust eigenstructure assignment problem can be stated as follows: *determine the feedback gain matrix K such that:*

Robust Eigenstructure Problem Definition:

1. The eigenvectors of the closed loop gain $(A - B.K)$ are as close as possible to the specified eigenvectors $v = [v_1, v_2, \dots, v_q]$.
2. The eigenvalues of the closed loop system $(A - B.K)$ contain the specified eigenvalues $\lambda = [\lambda_1, \lambda_2, \dots, \lambda_q]$.
3. The remaining $(n-q)$ unspecified eigenvalues and eigenvectors are stable.
4. The stability margin is maximized to account for robustness against uncertainties in the state-space matrices.

The four requirements can be stated mathematically as a constrained multiobjective optimization problem. The eigenvector problem is defined as finding K such that for each eigenvalue/eigenvector: $(A - B.K)v_i = \lambda_i.v_i$.

Requirement-1: Match the eigenvectors as close as possible to the desired eigenvectors: let v_i be the desired eigenvectors, and v_i^a the achievable eigenvectors, then this is equivalent to minimizing the norm:

$$f_1 = \min \sum_{i=1}^q \|v_i^a - v_i\|^2 \quad \text{Eqn.3.11}$$

All achievable closed loop eigenvectors v_i^a must belong to the subspace spanned by the columns of $S_i = (\lambda_i I - A)^{-1} B$ see [1], in other words the vector v_i^a must correspond to the subspace: $v_i^a = S_i \cdot g_i$ where g_i is a vector to be solved for, the minimization now becomes (where H =complex conjugate transpose):

$$f_1 = \min \sum_{i=1}^q (S_i \cdot g_i - v_i)^H (S_i \cdot g_i - v_i) \quad \text{Eqn.3.12}$$

Requirement-2: Match the actual eigenvalues to the specified eigenvalues, from equation 3.4, the following condition must be zero:

$$h_1 = \sum_{i=1}^q (A + B \cdot K - \lambda_i I) \cdot S_i \cdot g_i = 0 \quad \text{Eqn.3.13}$$

This defines a first constraint. Since h_1 is a vector, we can minimize its trace.

Requirement-3: The remaining $(n-q)$ unspecified eigenvalues and eigenvectors must be stable. It is sufficient to satisfy the Lyapunov equation:

$$h_2 = A_c^T \cdot P + P \cdot A_c + Q = 0 \quad \text{Eqn.3.14}$$

where $A_c = (A + B \cdot K)$ is the closed loop gain, and Q is positive definite symmetric matrix. This defines a second constraint. Since h_2 is a matrix, we can minimize its trace.

Requirement-4: The stability margin is maximized to account for robustness against uncertainties in the state-space matrices. For unstructured perturbations, this translates to minimizing the quantity:

$$f_2 = \text{trace}(P^2) \quad \text{Eqn.3.15}$$

the smaller this value, the more robustly stable the closed loop system will be to unstructured perturbations.

The robust eigenstructure assignment problem can be formulated as a multiobjective optimization problem with two objectives and two constraints:

minimize:

$$f_1 = \min \sum_{i=1}^q (S_i \cdot g_i - v_i)^H (S_i \cdot g_i - v_i) \quad \text{Eqn.3.16}$$

$$f_2 = \min \{ \text{trace}(P^2) \} \quad \text{Eqn.3.17}$$

constraints:

$$h_1 = \sum_{i=1}^q (A_i + K - \lambda_i I) \cdot S_i \cdot g_i = 0 \quad \text{Eqn.3.18}$$

$$h_2 = A_c^T \cdot P + P \cdot A_c + Q = 0 \quad \text{Eqn.3.19}$$

Fig.3.2

This is the generalized framework for robust eigenstructure assignment. This can be solved by calculus based constrained optimization using Lagrange multiplier methods [15].

3.1.5 Response of LTI Systems from Eigenstructure Information:

The eigenvalues and eigenvectors of a matrix can be used to completely characterize the dynamic behavior of a LTI system. Refer to [1] and [22] (pp. 342-345). Given an unforced system with full state feedback K :

$$\dot{x}(t) = A \cdot x(t) \quad \text{Eqn.3.20}$$

with eigenvalues of the closed loop system $A_c = A - B \cdot K$ at: $\Lambda_A = \text{diag}[\lambda_1, \lambda_2, \dots, \lambda_q]$ and the eigenvectors $V_A = [v_1, v_2, \dots, v_q]$. The system is transformed to discrete time thus:

$$x(k+1) = \Phi \cdot x(k) \quad \text{Eqn.3.21}$$

where the matrix $\Phi = e^{AT} \approx T \cdot A$, and T = step size. Since the transformation involves only a scaling by T , then the eigenvectors of Φ are identical to those of $(A - B \cdot K)$ ie: $V_\Phi = V_A$, but the eigenvalues are scaled by T , thus: $\Lambda_\Phi = T \cdot \Lambda_A$. It can be shown that the response at time step k of this system is completely described by relation:

$$x(k) = V_{\Phi} \cdot \Lambda_{\Phi}^k \cdot V_{\Phi}^{-1} \cdot x(o) \quad \text{Eqn.3.22}$$

thus equating parts, we get:

$$A^k = V_{\Phi} \cdot \Lambda_{\Phi}^k \cdot V_{\Phi}^{-1} \quad \text{Eqn.3.23}$$

From the above relation, the eigenstructure of the system can be used to fully describe its dynamic response. Note that if a system has unique nonzero eigenvalues, then the eigenvectors will be linearly independent.

3.2 Partial Eigenstructure Assignment for Static Compensators

3.2.1 Theory:

We now look at how hybrid genetic algorithms can be used to design a full state feedback static compensator K for the partial eigenstructure assignment problem. In this simulation, all eigenvalues have been specified, but only partial specification is provided for the corresponding eigenvectors. This problem can be solved by conventional methods described earlier. We can compare the solution obtained using genetic algorithms with conventional methods (see appendix 8.2). Two individual simulations are considered:

- (a) In the first part, the eigenvalues have been fully specified, and partial specification is provided for the eigenvectors. We can verify the solution obtained by GA as this problem can also be solved by conventional eigenstructure assignment.
- (b) In the second part, the upper and lower range of the allowable eigenvalues is given, for instance: $\lambda_j^{(\text{lower})} \leq \lambda_j \leq \lambda_j^{(\text{upper})}$, and partial specification is provided for the eigenvectors as described above. This second method cannot be directly solved by conventional eigenstructure assignment. This is a constrained optimization problem.

The linearized lateral aircraft model is used for these two simulations (appendix 8.1). A description of the simulation setup is outlined next. Consider the following linearized dynamic system:

$$\left. \begin{aligned} \dot{x}(t) &= A \cdot x(t) + B \cdot u(t) \\ y(t) &= C \cdot x(t) \end{aligned} \right\} \quad \text{Eqn.3.24}$$

Where $x \in \mathbb{R}^n$, is the state vector, and $u \in \mathbb{R}^m$ is the control vector, assuming full state feedback: $u(t) = -K.x(t)$ the closed loop system becomes:

$$\dot{x}(t) = (A - B.K).x(t) \quad \text{Eqn.3.25}$$

Where the closed loop eigenvalues and eigenvectors of $(A - B.K)$ must match as closely as possible to those specified. It is assumed that the controllability condition is satisfied i.e.: rank of the controllability matrix= n . Given this condition, using full state feedback, up to n eigenvalues (i.e.: all) can be placed in specified locations.

(i) Assignability Conditions:

With full state feedback: $x \in \mathbb{R}^n$ is the state vector, and $u \in \mathbb{R}^m$ is the control vector, the maximum possible assignability of eigenvalues and eigenvectors are:

- (i). a maximum of n of closed loop eigenvalues can be assigned, i.e. all eigenvalues may be arbitrarily assigned.
- (ii). a maximum of $n \times m$ total eigenvector entries can be arbitrarily assigned,
- (iii) no more than m entries in any one eigenvector can be chosen arbitrarily, with n eigenvectors, gives a total of $n \times m$ entries.

For our system, $n=4$, $m=2$, giving a total of 4 maximum allowable eigenvalues which may be arbitrarily placed, and 4 eigenvectors, with only 2 entries in each eigenvector column arbitrarily assigned.

(ii) Objectives:

For this first simulation, the problem is to minimize the eigenvector assignment error given by the objective function 2.26 below:

$$f_i = \min \sum_{i=1}^q (S_i \cdot g_i - v_i)^H (S_i \cdot g_i - v_i) \quad \text{Eqn.3.26}$$

where $S_i = (\lambda_i.I - A)^{-1}.B$, the feedback gain can be calculated from the g_i vectors, i.e: $K = -G.V^{-1}$. V =achievable eigenvector matrix $n \times n$, and $G=[g_1, g_2, \dots, g_n]$ matrix of g_i column vectors $m \times n$. The above equation attempts to minimize the difference between the desired eigenvectors v_i with the achievable eigenvectors $S_i.g_i$.

Thus the achievable eigenvectors must belong to the subspace spanned by $S_i = (\lambda_i \cdot I - A)^{-1} \cdot B$. Since full state feedback is used, all eigenvalues are assignable.

(iii) Required eigenstructure:

Eigenstructure assignment is applied to the linearized aircraft lateral model, with roll mode and Dutch roll modes at: $\lambda=-2\pm1j$ and $\lambda=-1.5\pm1.5j$ respectively. This is illustrated in figure 3.3 below, and the eigenstructure specification is tabulated in figure 3.4. Note that the don't care states are denoted in red by an **x** symbol. The given eigenstructure provides partial decoupling between the roll and Dutch roll modes. The aircraft lateral dynamics, with full state feedback is illustrated in figure 3.3 below:

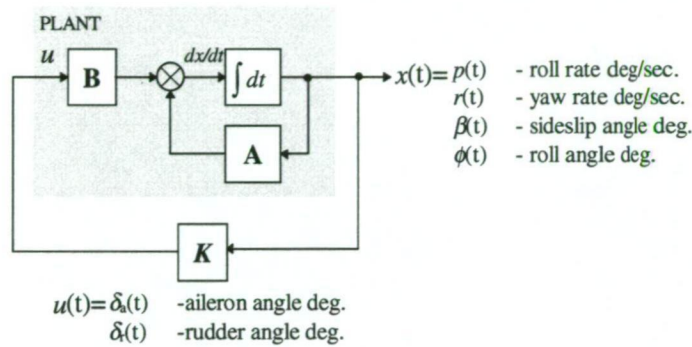


Fig. 3.3
Lateral Dynamics Used for Simulation

for the linearized lateral model, the required eigenstructure may be written in the form (see reference [39]), for each column, the first row is the eigenvalue and corresponding eigenvector below:

Roll Mode:		Dutch Roll Mode:	
$-2.0 + j1.0$	$-2.0 - j1.0$	$-1.5 + j1.5$	$-1.5 - j1.5$
$x1 + j1.0$	$x3 - j1.0$	$0.0 + j0.0$	$0.0 - j0.0$
$0.0 + j0.0$	$0.0 - j0.0$	$1.0 + jx6$	$1.0 - jx8$
$0.0 + j0.0$	$0.0 - j0.0$	$x5 + j1.0$	$x7 - j1.0$
$1.0 + jx2$	$1.0 - jx4$	$0.0 + j0.0$	$0.0 - j0.0$

Fig. 3.4
Eigenstructure Used in Lateral Aircraft Simulation

Where **x1, x2 ...x8** represent don't care values (unspecified values). Note that because complex conjugate pairs are present, then we have the condition: $x1=x3$, $x2=x4$, $x5=x7$, $x6=x8$, there are essentially only 4 parameters to solve for. Note that in this instance, the **g** vectors must also be complex conjugate pairs i.e.: $g_2 = \bar{g}_1$, $g_4 = \bar{g}_3$.

(iv) Chromosomal representation:

The chromosomal representation of this problem is illustrated in figure 3.5 below, where the error function (*error*) is given by equation 3.26 and the *fitness* is simply the inverse of the error function.

<i>n</i>	<i>m</i>	<i>X</i> ₁	<i>X</i> ₂	<i>X</i> ₃	<i>X</i> ₄	<i>X</i> ₅	<i>X</i> ₆	<i>X</i> ₇	<i>X</i> ₈	<i>error</i>	<i>Fitness</i>
----------	----------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	--------------	----------------

Fig. 3.5
Chromosomal Structure of Partial ESA Problem: Simulation-1

This is a generalized chromosomal representation which can be used for complex conjugates as well as purely real eigenvalues. There are several ways in which to encode the chromosome (Fig.3.5), one way would be to do a GA search on the *g_i* vectors (see equation 3.26) but this would require solving for 8 values, to see why, consider each *g* vector for each eigenvalue/eigenvector value consisting of 4 elements thus:

$$g_1 = \begin{bmatrix} a_1 + jb_1 \\ a_2 + jb_2 \end{bmatrix}$$

Since there are $2 \times g$ vectors to solve for (*g*₁, *g*₃) as the other two (*g*₂, *g*₄) are simply complex conjugates of the first two, this gives a total of 8 parameters to solve for.

The second method would simply be to do a GA search on the unspecified parameters: *x*₁, *x*₂ ...*x*₈, giving a total of only 4 parameters to solve for since: *x*₁=*x*₃, *x*₂=*x*₄, *x*₅=*x*₇, *x*₆=*x*₈. We can then estimate the *g_i* vectors from these values by least squares, and compute the fitness function 3.26. This second method is considerably more efficient and converges very rapidly. Results are given below.

3.2.2 Simulation 3.1: Fixed Eigenvalues:

(i) Objective: For this first simulation, we use the required eigenstructure described above, and compute the achievable eigenvectors. The aircraft lateral dynamics are given by the following matrices (refer to appendix 8.1)

$$\mathbf{A} = \begin{bmatrix} -3.9330 & 0.1260 & -9.9900 & 0 \\ 0.0020 & -0.2350 & 5.6700 & 0 \\ 0.0262 & -0.9997 & -0.1960 & 0.0345 \\ 1.0000 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -45.8300 & -7.6400 \\ -0.9210 & -6.5100 \\ 0.0071 & 0 \\ 0 & 0 \end{bmatrix}$$

The results from the first simulation are shown on the following page. Results using genetic algorithms and conventional eigenstructure assignment give identical results. Note the rapid convergence (within 40 generations) of the genetic algorithm.

Given the objective below, find the gain K such that the eigenstructure matches as closely as possible to the following specification, where **x**=don't care (can take any value):

Roll Mode:		Dutch Roll Mode:	
-2.0 + j1.0	-2.0 - j1.0	-1.5 + j1.5	-1.5 - j1.5
x 1 + j1.0	x 3 - j1.0	0.0 + j0.0	0.0 - j0.0
0.0 + j0.0	0.0 - j0.0	1.0 + j x 6	1.0 - j x 8
0.0 + j0.0	0.0 - j0.0	x 5 + j1.0	x 7 - j1.0
1.0 + j x 2	1.0 - j x 4	0.0 + j0.0	0.0 - j0.0

Fig.3.6

(ii) **Solution by Genetic Algorithms:** For this GA simulation, we use: Population: 60, Pc=0.6, Pm=0.1, max generations=200, binary tournament selection, objective: to match eigenvectors only (equation 3.26). The chromosomal representation as shown in figure 3.3. Results are given below:

match error: $f_1=0.0137$

Achievable Eigenvectors			
-1.9996 + 1.0000j	-1.9996 - 1.0000j	0.0000 - 0.0000j	0.0000 + 0.0000j
-0.0033 + 0.0050j	-0.0033 - 0.0050j	1.0000 + 1.8775j	1.0000 - 1.8775j
0.0109 - 0.0057j	0.0109 + 0.0057j	-0.3838 + 1.0000j	-0.3838 - 1.0000j
0.9998 - 0.0001j	0.9998 + 0.0001j	-0.0000 + 0.0000j	-0.0000 - 0.0000j

The feedback gain K is found as:

$$K = \begin{bmatrix} -0.002057691 & 0.064508221 & 0.268488544 & -0.111634509 \\ 0.003573822 & -0.403457113 & -0.302988201 & 0.015210675 \end{bmatrix}$$

Typical convergence rate of the genetic algorithm:

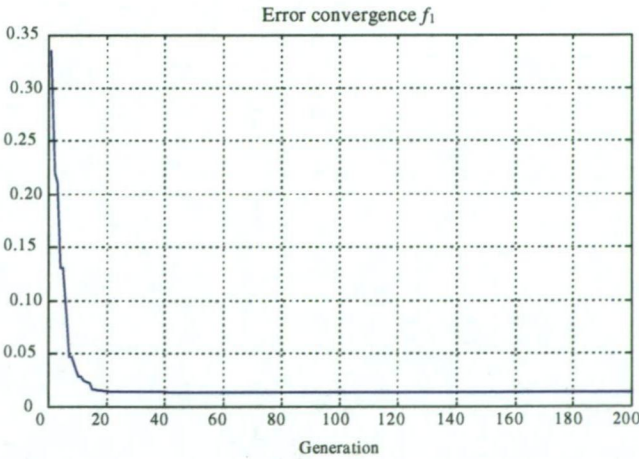


Fig. 3.7
Genetic Algorithm Error Convergence

(iii) **Solution by Conventional Methods:** (see appendix for algorithm): Using Moore's method discussed in section 3.1.2:

match error: $f_I = 0.0183$

Achievable Eigenvectors			
-1.9995 + 1.0000j	-1.9995 - 1.0000j	-0.0000 + 0.0000j	-0.0000 - 0.0000j
-0.0033 + 0.0050j	-0.0033 - 0.0050j	1.0000 + 1.8776j	1.0000 - 1.8776j
0.0109 - 0.0057j	0.0109 + 0.0057j	-0.3839 + 1.0000j	-0.3839 - 1.0000j
0.9998 - 0.0001j	0.9998 + 0.0001j	0 + 0.0000j	0 - 0.0000j

The feedback gain K is found:

$$K = \begin{bmatrix} -0.00205769 & 0.06450823 & 0.26848854 & -0.11163459 \\ 0.00357382 & -0.40345714 & -0.30298819 & 0.01521067 \end{bmatrix}$$

A comparison of the two methods is tabulated below:

Method:	$f_I()$	MFLOPS:	Time:
Genetic Algorithms:	0.0137	50	28 sec
Moore's Method:	0.0183	0.01	<1 sec

Fig.3.8
Comparing Genetic Algorithms with Conventional Partial Eigenstructure Assignment

Whilst the genetic algorithm gives a slightly better match, the solution is almost identical to the conventional method. The GA however requires almost 50 MFLOPS of computational effort compared with only 0.01 using conventional (Moore's) method. This simulation illustrates that while the GA converges rapidly, its computationally inefficient when compared with direct ESA design methods. The usefulness of the GA however can be demonstrated in the next ESA design application (simulation 3.2) in which no direct design method exists.

3.2.3 Simulation 3.2: Domain Constrained Eigenvalues:

(i) **Objective:** This second simulation is a constrained optimization problem, slightly more difficult to solve than the first. In this simulation, the upper and lower allowable range of the first eigenvalue is given, thus for the roll mode: λ_1 we allow the following valid range of eigenvalues:

$$\left. \begin{matrix} -2.5 \leq \text{real}(\lambda_1) \leq -1.5 \\ 0.5 \leq \text{imag}(\lambda_1) \leq 1.5 \end{matrix} \right\} \text{constraint} \tag{Eqn.3.27}$$

The eigenvalues of the roll mode are allowed to be in the specified range as above, the Dutch roll mode eigenvalues are fixed. Match the eigenstructure as closely as possible to:

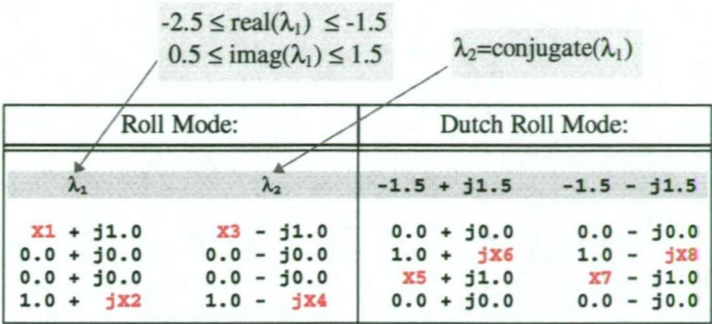


Fig.3.9

Find the gain K such that the closed loop eigenvalues and eigenvectors are as close to those above, where x =don't care (can take any value).

(ii) **GA solution:** This is a constrained (domain constraint) optimization problem which is solved using repair algorithms. Simulation results are given in the following pages. The chromosomal representation for this problem is shown below (Fig.3.10) again using floating point codification:

<i>n</i>	<i>m</i>	<i>X</i> ₁	<i>X</i> ₂	<i>X</i> ₃	<i>X</i> ₄	<i>X</i> ₅	<i>X</i> ₆	<i>X</i> ₇	<i>X</i> ₈	λ_1	λ_2	λ_3	λ_4	<i>error</i>	<i>Fitness</i>
----------	----------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-------------	-------------	-------------	-------------	--------------	----------------

Fig. 3.10

Chromosomal Representation of Partial Eigenstructure Assignment Problem: Simulation-3.2

For this GA simulation, we use: Population: 60, Pc=0.6, Pm=0.1, maximum generations=200, binary tournament selection, objective: to match eigenvectors only (equation 3.26). All eigenvalues are complex numbers, and the GA search only applies to λ_1 , the remaining eigenvalues are simply: $\lambda_2 = \overline{\lambda_1}$, and $\lambda_3 = -1.5 + j1.5$, $\lambda_4 = -1.5 - j1.5$ (fixed). Convergence is within 500 generations, this is illustrated in figure 3.11 below. The convergence of the eigenvalue λ_1 is also shown in figure 3.11. Convergence is initially very rapid for the first 50 generations. Similarity, convergence for the eigenvalue λ_1 is also initially rapid. The slow convergence is due to the fitness function being nearly flat near the optimum.

match error: $f_1=0.0084$

Achievable Eigenvectors

-1.3351 + 1.0000j	-1.3351 - 1.0000j	0.0000 - 0.0000j	0.0000 + 0.0000j
0.0000 + 0.0049j	0.0000 - 0.0049j	1.0000 + 1.8776j	1.0000 - 1.8776j
0.0024 - 0.0064j	0.0024 + 0.0064j	-0.3839 + 1.0000j	-0.3839 - 1.0000j
0.9999 - 0.3330j	0.9999 + 0.3330j	-0.0000 - 0.0000j	-0.0000 + 0.0000j

The eigenvalues are given by:

$$-1.5017 + 0.5000i \quad -1.5017 - 0.5000i \quad -1.5000 + 1.5000i \quad -1.5000 - 1.5000i$$

The feedback gain K is found:

$$K = \begin{bmatrix} 0.0195 & 0.0645 & 0.2685 & -0.0574 \\ 0.0046 & -0.4035 & -0.3030 & 0.0165 \end{bmatrix}$$

Typical convergence plots, including convergence of the eigenvalue λ_1 are illustrated below:

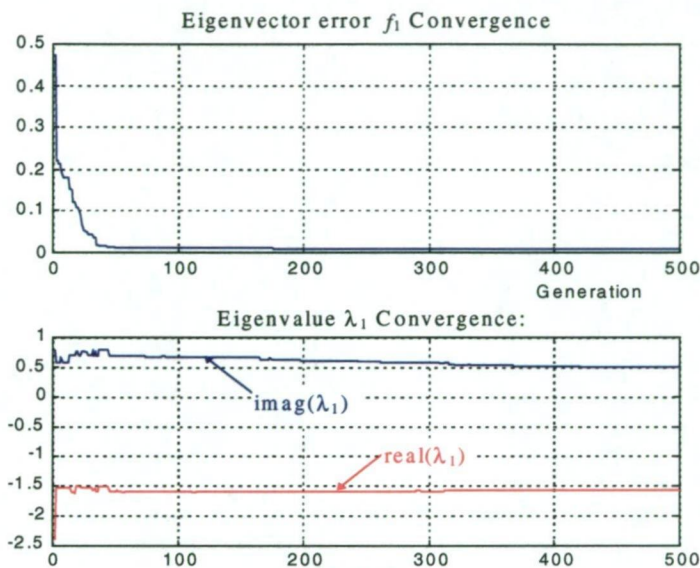


Fig. 3.11
Genetic Algorithm convergence (simulation 3.2)

(iii) **Conventional Solution:** There is no direct solution using conventional methods, however we can still check the validity of our results by simply plotting the value of f_1 for a whole range of λ_1 . i.e. given the range constraint:

$$\left. \begin{array}{l} -2.5 \leq \text{real}(\lambda_1) \leq -1.5 \\ 0.5 \leq \text{imag}(\lambda_1) \leq 1.5 \end{array} \right\} \text{constraint}$$

The algorithm would simply be:

```
for a=-1.5 TO -2.5
  for b=0.5 TO 1.5
     $\lambda_1 = a + jb$ 
     $\lambda_2 = \text{conjugate}(\lambda_1)$ 
     $f_1 = \text{solve by conventional eigenstructure assignment.}$ 
  end
end
```

Fig.3.12

This would result in a 3D surface plot of the value f_1 for a whole range of λ_1 along the two horizontal x axes, the results are illustrated below in figure 3.13. From this plot, we can in fact see that the minimum value of f_1 over the specified range of λ_1 given above occurs when $\lambda_1 = -1.5 + 0.5j$. This is the same result which we obtained previously with genetic algorithms. Comparing results from simulations 3.1 and 3.2, in both instances the genetic algorithm and conventional method give identical results. Note that whilst the GA takes longer to converge, it can be used to solve more complex eigenstructure assignment constrained optimization problems, whereas the conventional ESA method is restricted to solving only specific problems.

match error: $f_1=0.0117$

Achievable Eigenvectors			
-1.3332 + 1.0000j	-1.3332 - 1.0000j	-0.0000 + 0.0000j	-0.0000 - 0.0000j
0.0001 + 0.0049j	0.0001 - 0.0049j	1.0000 + 1.8776j	1.0000 - 1.8776j
0.0024 - 0.0064j	0.0024 + 0.0064j	-0.3839 + 1.0000j	-0.3839 - 1.0000j
0.9999 - 0.3333j	0.9999 + 0.3333j	0 + 0.0000j	0 - 0.0000j

The feedback gain K is found:

$$K = \begin{bmatrix} 0.019592379 & 0.064508233 & 0.268488541 & -0.057304768 \\ 0.004591785 & -0.403457114 & -0.302988199 & 0.016528473 \end{bmatrix}$$

The match error f_1 as a function of the eigenvalue λ_1 is plotted below in figure 3.11:

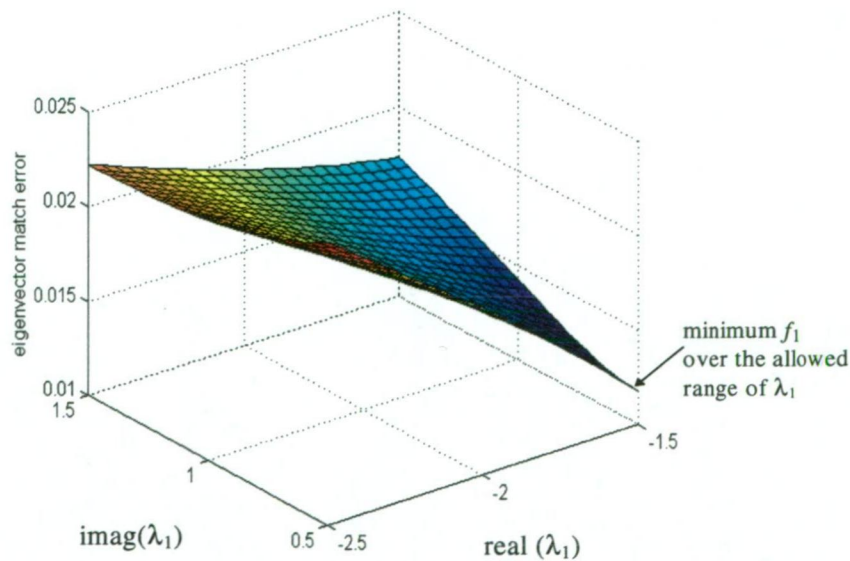


Fig. 3.13
Solution by conventional methods

Figure 3.11 above illustrates how the matching error f_1 is affected by the eigenvalue λ_1 as the real part of the eigenvalue spans the range $-2.5 \leq \text{real}(\lambda_1) \leq -1.5$, and the imaginary part spans the range: $0.5 \leq \text{imag}(\lambda_1) \leq 1.5$. The value of f_1 is calculated at each grid point on the surface using conventional eigenstructure assignment algorithm used previously in simulation 3.1.

Method:	$f_1()$	MFLOPS:	Time:
Genetic Algorithms:	0.0084	120	60 sec
Moore's Method:	0.0117	7	4 sec

Fig.3.14
Comparing Genetic Algorithms with Conventional Partial Eigenstructure Assignment

The computational effort required by the GA has increased from 50 to 120 MFP (factor of 2.5). However the computational effort by conventional methods requiring a search over the full range of lambda λ_1 has increased from 0.01 to 7 MFP, representing an increase of about 700. This simulation illustrates how the incorporation of constraints on the GA has only a small effect on the computational effort.

In these simulations, we have assumed the existence of full state feedback. In practice only measurement feedback may be available. In this case, a dynamic compensator is necessary. This is the topic of our next discussion.

3.3 Eigenstructure Assignment for Dynamic Compensators:

3.3.1 Theory:

When full state feedback is not available, then a dynamic compensator may be used to provide the additional design freedom. These simulations illustrate the design of dynamic compensators using genetic algorithms with only output feedback. The order of the compensator is generally chosen to be $p=n-r$ where n =number of states of system, and r =number of measured outputs. In these next set of simulations, genetic algorithms are applied to the design of dynamic output feedback compensators, the results are compared with conventional design methods. This simulation is divided into two parts:

- (i) The first part, eigenstructure assignment is used with only a fixed compensator, and output feedback to illustrate the limitations present.
- (ii) The second part, eigenstructure assignment is used with dynamic output feedback control to increase the number of degrees of freedom, and to overcome the limitations present in (i) above. We compare results obtained with both genetic algorithms and conventional methods.

The theory of dynamic output feedback control [2, 38] is outlined next. Given the linear time invariant system:

$$\left. \begin{aligned} \dot{x} &= A.x + B.u \\ y &= C.x \end{aligned} \right\} \quad \text{Eqn.3.28}$$

Where $x \in \mathbb{R}^n$, is the state vector, and $u \in \mathbb{R}^m$ is the control vector, $y \in \mathbb{R}^r$ is the measurement vector, it is assumed that $n > r$, using a dynamic compensator of the form:

$$\left. \begin{aligned} \dot{z} &= D.z + E.y \\ u &= F.z + G.y \end{aligned} \right\} \quad \text{Eqn.3.29}$$

The order p of the dynamic compensator $z \in \mathbb{R}^p$ should be $0 \leq p \leq n - r$, generally $p = n - r$. This is illustrated in figure 3.15 below. When $p=0$, this results in a static feedback gain matrix (section 3.2). The two equations can be combined into a composite system which can be solved in a similar fashion to the previous eigenstructure simulations. The composite system is given by: (see reference [2]) equation 3.30:

$$\left. \begin{aligned} \begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} &= \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} B & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \\ \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} &= \begin{bmatrix} C & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \\ \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} &= \begin{bmatrix} G & F \\ E & D \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \end{aligned} \right\} \quad \text{Eqn.3.30}$$

Which may be written more compactly in matrix form similar to the expression used in eigenstructure assignment for static compensators:

$$\begin{aligned}\dot{\bar{x}} &= \bar{A} \cdot \bar{x} + \bar{B} \cdot \bar{u} \\ \bar{y} &= \bar{C} \cdot \bar{x} \\ \bar{u} &= \bar{F} \cdot \bar{y}\end{aligned}\quad \text{Eqn.3.31}$$

This can now be solved in a similar manner as in section 3.2 above. The composite matrices for equation 3.31 are:

$$\bar{x} = \begin{bmatrix} x \\ z \end{bmatrix} \quad \bar{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \bar{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \bar{A} = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \quad \bar{B} = \begin{bmatrix} B & 0 \\ 0 & I \end{bmatrix} \quad \bar{C} = \begin{bmatrix} C & 0 \\ 0 & I \end{bmatrix} \quad \bar{F} = \begin{bmatrix} G & F \\ E & D \end{bmatrix}$$

In particular, note that: $y_1=y$ and $u_1=u$. Solving for the feedback gain matrix \bar{F} , the individual submatrices: G, F, E, D may be extracted. The plant and controller systems are illustrated in figure 3.15 below:

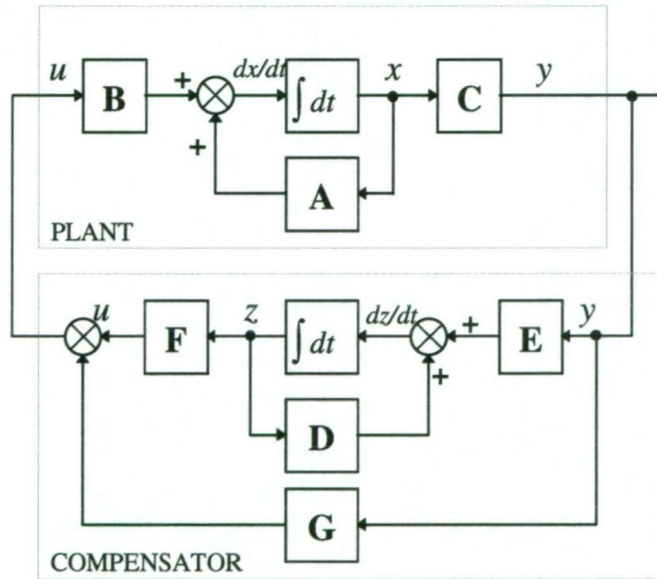


Fig. 3.15
Eigenstructure assignment using dynamic output feedback

The eigenstructure may be specified as before in the previous simulation with the addition of one eigenvalue (for the compensator: z) and one additional eigenvector entry for each, the don't care entries are shown in red below:

Roll Mode		Dutch Roll Mode		Compensator
-2.0 + j1.0	-2.0 - j1.0	-1.5 + j1.5	-1.5 - j1.5	λ_s
x + j1.0	x - j1.0	0.0 + j0.0	0.0 - j0.0	x
0.0 + j0.0	0.0 - j0.0	1.0 + jx	1.0 - jx	x
0.0 + j0.0	0.0 - j0.0	x + j1.0	x - j1.0	x
1.0 + jx	1.0 - jx	0.0 + j0.0	0.0 - j0.0	x
1.0 + j0	1.0 - j0	1.0 + j0	1.0 - j0	x

additional eigenvector components due to compensator dynamics

Fig. 3.16 Eigenstructure assignment using dynamic output feedback

Note the choice of additional eigenvector entry for the roll mode and Dutch roll mode (last row), they must be carefully selected to avoid the modal matrix from becoming numerically singular. The eigenstructure used above is taken from reference [2] and is commonly used for ESA in lateral dynamics for aircraft control studies.

As in the previous simulations, we note that the solution to the ESA problem with dynamic feedback requires the solution to:

$$(\overline{A} + \overline{B}\overline{F}\overline{C}).v_i = v_i.\lambda_i$$

Eqn.3.32

which can be rearranged thus:

$$v_i = (\lambda_i.I - \overline{A})^{-1} \overline{B}.\overline{F}.\overline{C}.v_i$$

Eqn.3.33

if we define the *g* vectors as:

$$g_i = \overline{F}.\overline{C}.v_i$$

Eqn.3.34

Then the eigenvectors must belong to the subspace spanned by:

$$v_i = (\lambda_i.I - \overline{A})^{-1} \overline{B}.g_i$$

Eqn.3.35

Solving for all the *g* vectors gives the matrix *G*=[*g*₁...*g*_{*n*}], and the achievable eigenvector matrix *V*=[*v*₁...*v*_{*n*}], then the dynamic compensator can be computed from:

$$\overline{F} = G.(\overline{C}.V)^{-1}$$

Eqn.3.36

In which the individual submatrices may then be extracted, i.e.

$$\bar{F} = \begin{bmatrix} G & F \\ E & D \end{bmatrix} \quad \text{Eqn.3.37}$$

Simulation results are provided in the following pages and compared with conventional solutions, using the previous values of the A and B matrices, with the addition of an output matrix C:

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} -3.9330 & 0.1260 & -9.9900 & 0 \\ 0.0020 & -0.2350 & 5.6700 & 0 \\ 0.0262 & -0.9997 & -0.1960 & 0.0345 \\ 1.0000 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{B} &= \begin{bmatrix} -45.8300 & -7.6400 \\ -0.9210 & -6.5100 \\ 0.0071 & 0 \\ 0 & 0 \end{bmatrix} \\ \mathbf{C} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{aligned}$$

A quick note about matrix dimensions used in this simulation:

$$\mathbf{A} \in \mathcal{R}^{4 \times 4} \quad \mathbf{B} \in \mathcal{R}^{4 \times 2} \quad \mathbf{C} \in \mathcal{R}^{3 \times 4} \quad \mathbf{D} \in \mathcal{R}^{1 \times 1} \quad \mathbf{E} \in \mathcal{R}^{1 \times 3} \quad \mathbf{F} \in \mathcal{R}^{2 \times 1} \quad \mathbf{G} \in \mathcal{R}^{2 \times 3} \quad \text{Eqn.3.38}$$

and for the composite system:

$$\bar{\mathbf{A}} \in \mathcal{R}^{5 \times 5} \quad \bar{\mathbf{B}} \in \mathcal{R}^{5 \times 3} \quad \bar{\mathbf{C}} \in \mathcal{R}^{4 \times 5} \quad \bar{\mathbf{F}} \in \mathcal{R}^{3 \times 4} \quad \text{Eqn.3.39}$$

3.3.2 Simulation 3.3: Static Output Feedback:

The first simulation illustrates the problems associated with eigenstructure assignability when only output feedback is available using only a static feedback compensator, i.e. constant feedback gain matrix K . This method can be used for later comparison with dynamic compensation. Thus given the system:

$$\left. \begin{aligned} \dot{x} &= \mathbf{A}x + \mathbf{B}u \\ y &= \mathbf{C}x \end{aligned} \right\} \quad \text{Eqn.3.40}$$

Where $x \in \mathcal{R}^n$, is the state vector, and $u \in \mathcal{R}^m$ is the control vector, $y \in \mathcal{R}^r$ is the measurement vector, find the feedback gain matrix K : $u = Ky$ such that the eigenstructure matches that specified in simulation-3.1. Eigenstructure assignability is now severely restricted due to the presence of the output matrix C. Conventional methods can be used to find the output matrix K , results are provided on the following pages (see simulation 3.4). See fig.3.17 below:

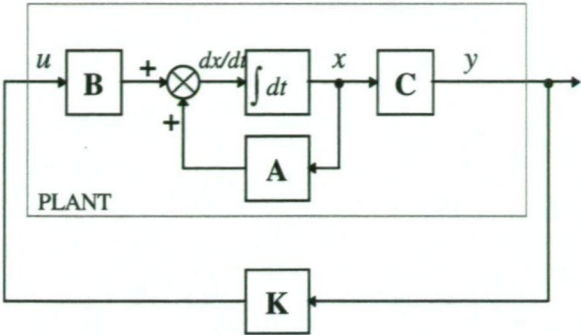


Fig. 3.17
Eigenstructure assignment using a static output feedback compensator

It can be shown that the maximum number of assignable eigenvalues is r , where $\mathbf{y} \in \mathbb{R}^r$, in this case $r=3$, assuming that the pair (A,B) is controllable, see [39]. In general however, it is assumed that: $m < r < n$.

(i) **Assignability Conditions:** With output feedback, the following restrictions apply:

- (i). a maximum of $\max(r,m)$ of closed loop eigenvalues may be assigned.
- (ii). a maximum of $\max(r,m)$ eigenvectors can be partially assigned with $\min(r,m)$ entries in each vector arbitrarily chosen.

For our system, $n=4$, $m=3$, $r=2$, giving a total of 3 maximum allowable eigenvalues which may be arbitrarily, and 3 eigenvectors, with only 2 entries in each eigenvector arbitrarily chosen. Simulation results are given on the following page. Results indicate that using output feedback fails to allocate all specified eigenvalues, and only the second conjugate pair is assigned. This is a limitation of output feedback with constant feedback gain K . A summary of the simulation results is given below.

(ii) **Objective:** Find the gain K such that the closed loop eigenvalues and eigenvectors are as close to those above, where x=don't care (can take any value) using only output measurement feedback and static compensator K :

Roll Mode:		Dutch Roll Mode:	
-2.0 + j1.0	-2.0 - j1.0	-1.5 + j1.5	-1.5 - j1.5
x1 + j1.0	x3 - j1.0	0.0 + j0.0	0.0 - j0.0
0.0 + j0.0	0.0 - j0.0	1.0 + jx6	1.0 - jx8
0.0 + j0.0	0.0 - j0.0	x5 + j1.0	x7 - j1.0
1.0 + jx2	1.0 - jx4	0.0 + j0.0	0.0 - j0.0

Fig.3.18

(iii) **Eigenstructure by Conventional Method:** The solution to the above problem using only static output feedback gives the following achievable eigenstructure :

Achievable eigenvalues:

-2.0000	0.0000	-1.5000 + j1.5000	-1.5000 - j1.5000
---------	--------	-------------------	-------------------

Achievable eigenvectors:

0.8944	-0.0000	0.0000 - 0.0000i	0.0000 + 0.0000i
-0.0038	-0.0302	1.0000 + 1.8776i	1.0000 - 1.8776i
-0.0064	-0.0215	-0.3839 + 1.0000i	-0.3839 - 1.0000i
-0.4472	-0.9993	0.0001 + 0.0001i	0.0001 - 0.0001i

Note that whilst the algorithm is able to match the last two of the eigenvalues and eigenvectors at -1.5+/-1.5j perfectly, it fails at assigning the first two. If however the first two eigenvalues are purely real, then the algorithm would allocate one of the two eigenvalues in addition to the second pair of complex conjugate.

We can clearly see the limitations of using output feedback when eigenstructure assignment is used as a design tool. Consequently dynamic feedback is required. This is the topic of discussion in the next two simulations which follow. Genetic algorithms are used to design the dynamic compensators. This will be compared to conventional eigenstructure assignment methods.

3.3.3 Simulation 3.4: Dynamic Control Output Feedback:

Looking at the previous results, output feedback presents serious limitations when all eigenvalues must be assigned. These problems may be overcome by using dynamic feedback or dynamic control. In this simulation, eigenstructure assignment for dynamic compensators is implemented using genetic algorithms, this is then compared with results obtained using conventional methods. Using dynamic feedback control, the composite equations may be written in the form:

$$\left. \begin{aligned} \dot{\bar{x}} &= \bar{A}.\bar{x} + \bar{B}.\bar{u} \\ \bar{y} &= \bar{C}.\bar{x} \\ \bar{u} &= K.\bar{y} \end{aligned} \right\}$$

Eqn.3.41

This is in the same form as previously required for eigenstructure assignment. Where the composite variables are:

$$\bar{x} = \begin{bmatrix} x \\ z \end{bmatrix} \quad \bar{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \bar{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \bar{A} = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \quad \bar{B} = \begin{bmatrix} B & 0 \\ 0 & I \end{bmatrix} \quad \bar{C} = \begin{bmatrix} C & 0 \\ 0 & I \end{bmatrix} \quad K = \begin{bmatrix} G & F \\ E & D \end{bmatrix} \quad \text{Eqn.3.42}$$

We note that the \bar{C} matrix has now $rank(\bar{C})=4$, in other words, we can use the augmented system to assign all four eigenvalues and eigenvectors. The composite system matrices now become:

$$\bar{A} = \begin{array}{cccc|c} -3.9330 & 0.1260 & -9.9900 & 0 & 0 \\ 0.0020 & -0.2350 & 5.6700 & 0 & 0 \\ 0.0262 & -0.9997 & -0.1960 & 0.0345 & 0 \\ 1.0000 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}$$

$$\bar{B} = \begin{array}{cc|c} -45.8300 & -7.6400 & 0 \\ -0.9210 & -6.5100 & 0 \\ 0.0071 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 1 \end{array}$$

$$\bar{C} = \begin{array}{cccc|c} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}$$

Since only up to four eigenvalues may be assigned, then the compensator λ_5 eigenvalue cannot be arbitrarily assigned, however it must be stable. For our system, $n=5$, $m=4$, $r=3$, giving a total of 4 maximum allowable eigenvalues which may be placed, and 4 eigenvectors, with only 3 entries in each eigenvector arbitrarily chosen. Simulation results are given on the following page. Compensator eigenvector specification is not required.

(i) **Objectives:** Find the dynamic compensator gain K such that the closed loop eigenvalues and eigenvectors are as close to those above, where x=don't care (can take any value) using output feedback:

Roll Mode:		Dutch Roll Mode:		Compensator
-2.0 + j1.0	-2.0 - j1.0	-1.5 + j1.5	-1.5 - j1.5	λ_5
x1 + j1.0	x3 - j1.0	0.0 + j0.0	0.0 - j0.0	x
0.0 + j0.0	0.0 - j0.0	1.0 + jx6	1.0 - jx8	x
0.0 + j0.0	0.0 - j0.0	x5 + j1.0	x7 - j1.0	x
1.0 + jx2	1.0 - jx4	0.0 + j0.0	0.0 - j0.0	x
1.0 + j0	1.0 - j0	1.0 + j0	1.0 - j0	x

Fig.3.20

(ii) **Simulation Results using Conventional Methods:**

Achievable Eigenvalues:

-2.0 + j1.0	-2.0 - j1.0	-1.5 + j1.5	-1.5 - j1.5	-0.0378
-------------	-------------	-------------	-------------	---------

Achievable Eigenvectors:

-1.9995 + 1.0000i	-1.9995 - 1.0000i	0.0000 + 0.0000i	0.0000 - 0.0000i	0.0377
-0.0033 + 0.0050i	-0.0033 - 0.0050i	1.0000 + 1.8776i	1.0000 - 1.8776i	-0.0301
0.0109 - 0.0057i	0.0109 + 0.0057i	-0.3839 + 1.0000i	-0.3839 - 1.0000i	-0.0213
0.9998 - 0.0001i	0.9998 + 0.0001i	-0.0000 - 0.0000i	-0.0000 + 0.0000i	-0.9983
1.0000	1.0000	1.0000	1.0000	-0.0243

Dynamic Compensator Matrices: G, F, E, D can be found from the terms of the K matrix:

$$\mathbf{K} = \begin{array}{c|c} \mathbf{G} & \mathbf{F} \\ \hline \mathbf{E} & \mathbf{D} \end{array}$$

Results from the simulation gives:

$\mathbf{K} =$	0.003074418	-0.129655514	-0.146167328	0.112103407
	-0.003712355	0.412333707	0.286321415	-0.015274564
	1.017115088	-0.539781790	2.513499919	0.004653040

(ii) Simulation Results using GA:

The simulation is repeated using genetic algorithms, the chromosomal structure is illustrated below, note that the dynamic compensator is not included as part of the search. Thus any compensator eigenvalue/eigenvector is acceptable as long as $\lambda_5 < 0$. Once a solution is obtained, the compensator eigenvalue and eigenvector is simply obtained by using the eig() matlab function thus: $\text{eig}(\overline{A} + \overline{B}.\overline{F}.\overline{C})$ of the closed loop system.

X ₁	X ₂	X ₃	X ₄	X ₅	X ₆	X ₇	X ₈	error	Fitness
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	-------	---------

Fig.3.21

Achievable Eigenvalues:

-2.0000 + j1.0000	-2.0000 - j1.0000	-1.5000 + j1.5000	-1.5000 - j1.5000	-0.0378
-------------------	-------------------	-------------------	-------------------	---------

Achievable Eigenvectors:

-1.9995 + 1.0000i	-1.9995 - 1.0000i	0.0000 + 0.0000i	0.0000 - 0.0000i	0.0377
-0.0033 + 0.0050i	-0.0033 - 0.0050i	1.0000 + 1.8776i	1.0000 - 1.8776i	-0.0301
0.0109 - 0.0057i	0.0109 + 0.0057i	-0.3839 + 1.0000i	-0.3839 - 1.0000i	-0.0213
0.9998 - 0.0001i	0.9998 + 0.0001i	-0.0000 - 0.0000i	-0.0000 + 0.0000i	-0.9983
1.0000	1.0000	1.0000	1.0000	-0.0243

Dynamic Compensator Matrices: G, F, E, D can be found from the terms of the K matrix:

$$\mathbf{K} = \begin{array}{c|c} \mathbf{G} & \mathbf{F} \\ \hline \mathbf{E} & \mathbf{D} \end{array}$$

Results from the simulation gives:

$$K = \begin{array}{ccc|c} 0.003074193 & -0.129655002 & -0.146168284 & 0.112102533 \\ -0.003712325 & 0.412333637 & 0.286321545 & -0.015274445 \\ \hline 1.017114982 & -0.539776031 & 2.513489097 & 0.004643255 \end{array}$$

Convergence properties of the genetic algorithm are illustrated below Fig.3.22, the genetic algorithm converges very rapidly, within the first 40 generations. We can also see that all eigenvalues and eigenvectors are assigned as required.

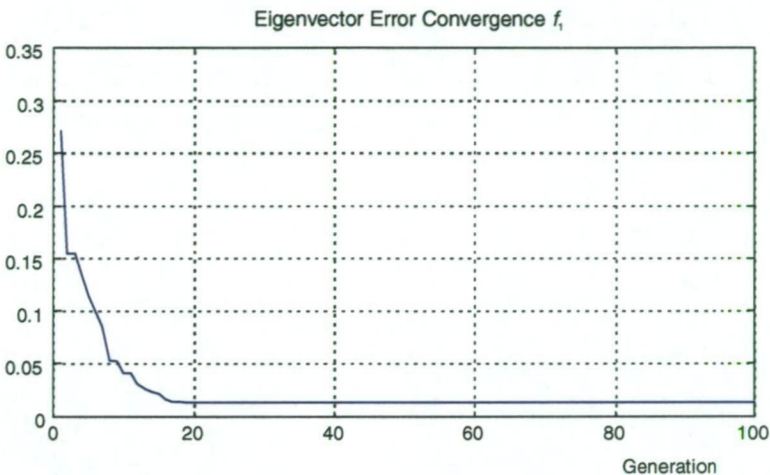


Fig. 3.22
Convergence properties of genetic algorithm:

(iii) Notes on Simulation:

1. This simulation is essentially identical to the first (simulation 3.1), with the exception that we use the composite system matrices instead of the original plant matrices. When the feedback gain matrix K is found, the dynamic compensator terms: D, E, F, G , can be extracted using equation 3.42.
2. Note that with dynamic feedback, all four eigenvalues (roll mode, Dutch roll mode) can be assigned arbitrarily. Note also that the compensator eigenvalue cannot be assigned, however any stable compensator eigenvalue is allowed. Using conventional ESA methods, the stability of the compensator eigenvalue cannot be guaranteed and difficult to impose constraints to ensure its stable. With genetic algorithms however, the stability of the compensator eigenvalue can be guaranteed by using constrained optimization such as penalty functions or repair algorithms.
3. Both the conventional method and genetic algorithm method produce identical results. The convergence of the GA is shown in figure 3.16.

3.4 Robust Eigenstructure Assignment:

3.4.1 Theory:

This last simulation involves solving the robust eigenstructure assignment problem defined earlier in section 3.1.4. Full state feedback using a static compensator is assumed. There is no conventional design method presently available to solve this type of eigenstructure assignment problem. Our results can be verified by comparing three methods: (i) Conventional Genetic Algorithms (GA), (ii) Genetic Algorithms and Simulated Annealing (GA+SA), and (iii) Genetic Algorithms and Greedy Search (GA+GS). Convergence rates and computational effort are compared. From section 3.1.4, the robust eigenstructure assignment problem is defined as a multiobjective optimization problem, in which the functions to minimize f_1 and f_2 and constraints $h_1=0$ and $h_2=0$ are given by:

minimize:

$$f_1 = \min \sum_{i=1}^q (S_i \cdot g_i - v_i)^H (S_i \cdot g_i - v_i)$$

$$f_2 = \min \{ \text{trace}(P^2) \}$$

constraints:

$$h_1 = \sum_{i=1}^q (A_i + K - \lambda_i I) \cdot S_i \cdot g_i = 0$$

$$h_2 = A_c^T \cdot P + P \cdot A_c + Q = 0$$

Fig.3.23

A simpler problem would be to minimize a composite cost functional defined as a weighted sum of the two cost functionals f_1 and f_2 and thus:

$$f(x) = f_1(x) + \beta \cdot f_2(x) \quad \text{Eqn.3.43}$$

where β is varied from 0.1, 1, 10 to show the effects of adding more emphasis on one function against the other function. The second constraint $h_2=0$ is dealt with by defining the Q matrix (positive definite symmetric) as the identity matrix, and solving this constraint by solving the lyapunov function directly. The matlab *lyap()* function can be used to solve for P . Note that the solution P must also be a positive definite symmetric matrix.

Since h_1 and h_2 both represent matrices, we can compute the matrix norm or trace for each constraint. The matrix: $A_c=(A+B.K)$ is the closed loop gain. This problem is effectively identical to section 3.2, with the addition of two constraints and a composite functional to minimize.

3.4.2 Simulation 3.5: Hybrid Genetic Algorithms:

(i) **Objective:** The required eigenstructure is defined as follows Fig.3.24 with the roll mode identical to the originally defined eigenstructure in section 3.2. However, the Dutch roll mode is replaced by two decoupled modes: the first has a pole at -1 , the second pole is at λ_4 and is unspecified. Further, we constrain the unspecified pole to be within the range: $-2.5 < \lambda_4 < -0.5$.

Roll Mode:		Decoupled Modes	
$-2.0 + j1.0$	$-2.0 - j1.0$	-1	λ_4
$x_1 + j1.0$	$x_3 - j1.0$	x_5	0
$0.0 + j0.0$	$0.0 - j0.0$	x_6	1
$0.0 + j0.0$	$0.0 - j0.0$	1	x_7
$1.0 + jx_2$	$1.0 - jx_4$	0	x_8

Fig. 3.24

The values of A,B matrices remain the same. The search is over the parameters: $\{X_1, X_2, X_5, X_6, X_7, X_8, \lambda_4\}$, with $X_3=X_1$ and $X_4=X_2$.

(ii) **Genetic algorithms:** the chromosomal representation for this problem is illustrated in figure 3.25 below, again floating point codification is used:

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	λ_4	f_1	f_2	h_1	h_2	Fitness
-------	-------	-------	-------	-------	-------	-------	-------	-------------	-------	-------	-------	-------	---------

Fig. 3.25

Chromosomal Representation of Partial Eigenstructure Assignment Problem: Simulation-4

The fitness is the inverse of the composite cost functional Eqn.3.43. Additionally, a penalty is introduced such that if λ_4 is not within the desired range : $-2.5 < \lambda_4 < -0.5$ then the fitness is set to zero. Similarly, if constraints $h_1 \neq 0$ or $h_2 \neq 0$ then the fitness is also set to zero. The solution to the lyapunov equation (Eqn.3.19) must also exist, and the P matrix must be positive symmetric, if a solution does not exist then the fitness is set to zero. A chromosome with zero fitness is said to be *infeasible* and produces zero offspring. Simulation results using: Population=30, crossover probability $P_c=0.6$, mutation probability $P_m=0.02$, Generations=200 and binary tournament selection are tabulated below in Fig.3.26 for values of beta: 0.1, 1, 10:

	gen:	X1:	X2:	X5:	X6:	X7:	X8:	λ_4 :	f1():	f2():	Time:	MFP:
0.1	200	-2.00	-0.00	0.00	0.80	0.43	0.00	-2.50	0.0004	2.3846	0:31	75
	Closed loop eigenvalues (A-B.K): -2.0000 + 1.0000i -2.0000 - 1.0000i -1.0000 -2.5000 Feedback Compensator gain K: -0.0018 0.0776 0.3235 -0.1117 0.0021 -0.4821 -0.6334 0.0155											
1	200	-2.00	-0.00	0.01	0.80	0.43	0.00	-2.50	0.0004	2.3844	0:35	75
	Closed loop eigenvalues (A-B.K): -2.0000 + 1.0000i -2.0000 - 1.0000i -1.0000 -2.5000 Feedback Compensator gain K: -0.0018 0.0778 0.3232 -0.1117 0.0021 -0.4821 -0.6334 0.0155											
10	200	-2.00	-0.00	0.04	0.80	0.44	0.00	-2.50	0.0012	2.3842	0:34	76
	Closed loop eigenvalues (A-B.K): -2.0000 + 1.0000i -2.0000 - 1.0000i -1.0000 -2.5000 Feedback Compensator gain K: -0.0018 0.0782 0.3222 -0.1117 0.0021 -0.4821 -0.6333 0.0155											

Fig. 3.26
Simulation results using conventional genetic algorithms

In all instances, the GA converges within 200 generations, and the solution is independent of the value of beta. The closed loop eigenvalues of (A-B.K) can be verified using Matlab's *eig()* function. In all cases, the unspecified pole λ_4 is at -2.5.

(ii) **GA+Simulated Annealing:** The search vector used is identical to genetic algorithms without the function and fitness entries, this is illustrated below:

X ₁	X ₂	X ₃	X ₄	X ₅	X ₆	X ₇	X ₈	λ_4
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	-------------

Fig. 3.27
Chromosomal Representation of Partial Eigenstructure Assignment Problem: Simulation-4

The temperature annealing schedule is given by: $T(k+1)=\alpha.T(k)$ where alpha is given by:

$$\alpha = 10^{(1/N \log(T_f/T_o))} \quad \text{Eqn.3.44}$$

T_o =initial temperature normalized to 1.0, and T_f is the final temperature 0.001, N is the number of iterations set to 200. The value of alpha is generally ≈ 0.8 -0.95.

Simulation results using three values of beta: 0.1, 1, 10 are summarized below, the SA algorithm converges within 200 iterations:

	gen:	X1:	X2:	X5:	X6:	X7:	X8:	λ_4 :	Time:	MFP:
0.1	200	-2.00	-0.00	-0.00	0.80	0.44	-0.00	-2.50	00:17	51
	Closed loop eigenvalues (A-B.K): -2.0000 + 1.0000i -2.0000 - 1.0000i -1.0000 -2.5000									
	Feedback Compensator gain K: -0.0018 0.0776 0.3235 -0.1117 0.0021 -0.4821 -0.6334 0.0155									
1	200	-1.99	-0.00	0.02	0.80	0.44	-0.00	-2.50	00:17	51
	Closed loop eigenvalues (A-B.K): -2.0000 + 1.0000i -2.0000 - 1.0000i -1.0000 -2.5000									
	Feedback Compensator gain K: -0.0018 0.0778 0.3231 -0.1117 0.0021 -0.4821 -0.6334 0.0155									
10	200	-1.99	0.00	0.04	0.80	0.44	0.01	-2.50	00:17	51
	Closed loop eigenvalues (A-B.K): -2.0000 + 1.0000i -2.0000 - 1.0000i -1.0000 -2.5000									
	Feedback Compensator gain K: -0.0018 0.0782 0.3223 -0.1117 0.0021 -0.4821 -0.6333 0.0155									

Fig. 3.28
Simulation results using Hybrid genetic algorithms + simulated annealing

The hybrid GA+SA converges faster than conventional genetic algorithms, and yielding an identical compensator and eigenvalue λ_4 is at -2.5 .

(iii) **GA+Greedy Search** : The search vector used is identical to simulated annealing, results for greedy search are summarized below in figure 3.29. Results show that the hybrid GA+greedy search converge more rapidly than either conventional genetic algorithms or hybrid GA+simulated annealing. Computation time is approximately 7 seconds for the hybrid GA+greedy search, 17 seconds for the hybrid GA+simulated annealing, and 35 seconds for the conventional genetic algorithm.

A true indication of convergence rates for the three methods can be obtained by plotting the error (inverse of equation 3.43) as a function of computation time. This is illustrated in figure 3.30 on the following page. Results obtained using hybrid GA + greedy search:

	gen:	X1:	X2:	X5:	X6:	X7:	X8:	λ_4 :	Time:	MFP:
0.1	500	-2.00	0.00	0.00	0.80	0.43	-0.00	-2.50	7	19
	Closed loop eigenvalues (A-B.K): -2.0000 + 1.0000i -2.0000 - 1.0000i -1.0000 -2.5000									
	Feedback Compensator gain K: -0.0018 0.0776 0.3235 -0.1117 0.0021 -0.4821 -0.6334 0.0155									
1	500	-2.00	0.00	0.01	0.79	0.43	-0.00	-2.50	7	19
	Closed loop eigenvalues (A-B.K): -2.0000 + 1.0000i -2.0000 - 1.0000i -1.0000 -2.5000									
	Feedback Compensator gain K: -0.0018 0.0779 0.3230 -0.1117 0.0021 -0.4821 -0.6334 0.0155									
10	500	-2.02	0.00	0.02	0.80	0.41	0.00	-2.50	7	18
	Closed loop eigenvalues (A-B.K): -2.0000 + 1.0000i -2.0000 - 1.0000i -1.0000 -2.5000									
	Feedback Compensator gain K: -0.0018 0.0782 0.3223 -0.1117 0.0021 -0.4821 -0.6333 0.0155									

Fig. 3.29
Simulation results using hybrid genetic algorithms + greedy search

Figure 3.30 below is a typical convergence plot comparing the three methods. Conventional GA is plotted in red, hybrid GA+SA is plotted in green and hybrid GA+GS is plotted in blue. All methods converge to the same value, however the hybrid methods converge more rapidly when compared with conventional genetic algorithms. In particular, the hybrid GA+greedy search has superior convergence compared to the other two methods. All methods yield identical compensators.

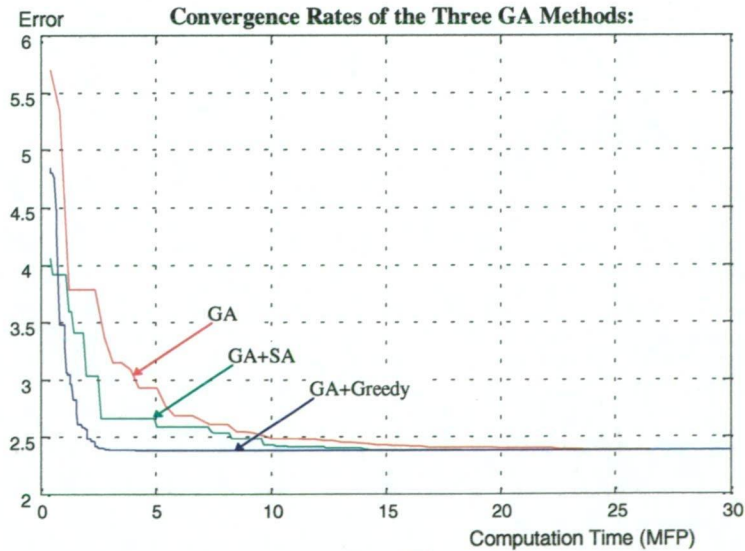


Fig. 3.30
Simulation results comparing convergence rates for the three GA methods

3.4 Chapter Summary and Conclusion:

From simulation results, clearly genetic algorithms can be used to synthesize controllers both static and dynamic for a variety of different eigenstructure assignment applications. Convergence is generally very rapid. Genetic algorithms have fewer restrictions and can directly deal with constraints. The simulations have been kept relatively simple for the purpose of verification with conventional partial eigenstructure assignment methods. The robust eigenstructure assignment problem can also be directly solved with conventional and hybrid genetic algorithms.

Simulation results show that hybrid genetic algorithms converge more rapidly than conventional genetic algorithms, in particular the greedy search converges by a factor of four compared with conventional GA.

(i) Additional Notes:

1. When dealing with a control distribution matrix B which has a very small minimum singular value, then the application of pseudocontrol may be required. In our simulations, the singular values of B are: [46.5053, 6.2642], this is not an issue in the design.
2. Most of the examples chosen for the simulations have been relatively simple for the purpose of being able to verify the results with conventional eigenstructure assignment methods.

With genetic algorithms, there are fewer restrictions and thus a wider range of problems can be solved, including nonlinear and reconfigurable control.

3. Doing a search on the X_j (don't care) components results in a faster convergence compared to searching the g_j vectors directly.

(ii) Future Work:

1. The robust eigenstructure problem can be formulated using lagrange multiplier (calculus based) methods. However this method has local rather than global search characteristics. Genetic algorithms can be used to solve the robust eigenstructure assignment problem by combining a GA start to find the global minimum, then using lagrange multipliers to quickly find the minimum. This method requires the calculation of gradients. It also requires the calculation of additional auxiliary variables: i.e. the lagrange multiplier matrix.

-
2. Other conventional methods are available for solving robust eigenstructure assignment problems in which the performance indices are given in terms of sensitivity and complementary sensitivity functions [16], [19]. Eigenstructure for gain suppression in which selected entries in the output feedback gain matrix are removed [4]. Robust eigenstructure assignment for systems dealing with state space uncertainty [21]. Modeling errors and uncertainty can be included as part of the GA optimization search.
 3. Eigenstructure assignment using radial basis function networks as a feedback control for nonlinear systems. Training using conventional and genetic algorithms.
 4. Areas of reconfigurable control have been investigated [18] in which the objective is to implement a compensator that results in a closed loop eigenstructure invariant under plant changes (A, B matrices). Genetic algorithms can be potentially used for such off-line compensator design.

3.5 References and Further Reading:

References on Eigenstructure Assignment:

- [1] A.N.Andry, E.Y.Shapiro, J.C.Chung
Eigenstructure Assignment for Linear Systems
IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-19, No.5, pp.711-729, December.1983
- [2] K.M.Sobel, E.Y.Shapiro, A.N.Andry
Eigenstructure Assignment
The Control Handbook, CRC Press 1996, Chapter-38
- [3] J.R. Calvo-Ramon
Eigenstructure Assignment by Output Feedback and Residue Analysis
IEEE Transactions on Automatic Control, Vol.31, No.3, pp.247-249, 1986
- [4] K.M.Sobel, W.Yu, F.J.Lallman
Eigenstructure Assignment with Gain Suppression Using Eigenvalue and Eigenvector Derivatives
International Journal of Guidance Control and Dynamics, Vol.13, No.6, pp.1008-1013, 1990
- [5] S.M.Karbassi, D.J.Bell
New Method of Parametric Eigenvalue Assignment in State Feedback Control
IEE Proceedings Control Theory Applications, Vol.141, No.4, pp.223-225, July 1994
- [6] B.C. Moore
On the Flexibility Offered by State Feedback in Multivariable Systems Beyond Closed Loop Eigenvalue Assignment.
IEEE Transactions on Automatic Control, Vol. AC-21, No.3, pp.689-692, 1976
- [7] S.Askarpour, T.J.Owens
Integrated Approach to Eigenstructure Assignment by State Feedback
IEE Proceedings Control Theory Applications , Vol.146, No.2, pp.113-118, March 1999
- [8] J. Jiang
Design of Reconfigurable Control Systems Using Eigenstructure Assignments
International Journal of Control, Vol.59, No.2, pp.395-410, 1994
- [9] Hee-Seob Kim, Youdan Kim
Partial Eigenstructure Assignment Algorithm in Flight Control System Design
IEEE Transactions on Aerospace and Electronic Systems, Vol. 35, No.4, pp.1403-1408, October 1999
- [10] George M. Siouris, Jang Gyu Lee, Jae Weon Choi
Design of a Modern Pitch Pointing Control System
IEEE Transactions on Aerospace and Electronic Systems, Vol. 31, No.2, pp.730-737, April 1995
- [11] Howard Kaufman, Paul Berry
Adaptive Flight Control Using Optimal Linear Regulator Techniques
Automatica, Vol. 12, pp.565-576, 1976
- [12] D.Gangsaas, K.R.Bruce, J.D.Blight, Uy-Loi Ly
Application of Modern Synthesis to Aircraft Control: Three Case Studies
IEEE Transactions on Automatic Control, Vol. AC-31, No.11, pp.995-1013, November 1986
- [13] R.J.Patton, G.P.Liu, J.Chen
Multiobjective Controller Design Using Eigenstructure Assignment and the Method of Inequalities
International Journal of Guidance Control and Dynamics, Vol.17, No.4, pp.862-864, 1993

- [14] K.M. Sobel, E.Y. Shapiro
Application of Eigenstructure Assignment to Flight Control Design: Some Extensions
International Journal of Guidance Control and Dynamics, Vol.10, No.1, pp.73-81, 1987

References on Robust Eigenstructure Assignment:

- [15] I.K.Konstantopoulos, P.J.Antsaklis
Optimisation Approach to Robust Eigenstructure Assignment
IEE Proceedings Control Theory Applications, Vol.146, No.6, pp.561-565, November 1999
- [16] G.P.Liu, R.J.Patton
Robust Control Design Using Eigenstructure Assignment and Multi-Objective Optimization
International Journal of System Science, Vol.27, No.9, pp.871-879, 1996
- [17] J.Kautsky, N.K.Nichols, P.Van Dooren
Robust Pole Assignment in Linear State Feedback
International Journal of Control, Vol.41, No.5, pp.1129-1155, 1984
- [18] Jin Jiang
Design of Reconfigurable Control Systems Using Eigenstructure Assignment
International Journal of Control, Vol 59, No.2, pp.395-410, 1994
- [19] G.P.Liu, R.J.Patton
Robust Control Design via Eigenstructure Assignment, Genetic Algorithms and Gradient Optimization
IEE Proceedings Control Theory Applications, Vol.141, No.3, pp.202-208, May 1994
- [20] L.F. Faleiro, R.W. Pratt
Multi-Objective Eigenstructure Assignment with Dynamic Flight Control Augmentation Systems
AIAA
WWW:
- [21] W.Yu, K.M. Sobel
Robust Eigenstructure Assignment with Structured State Space Uncertainty
International Journal of Guidance Control and Dynamics, Vol.14, No.3, pp.621-628, June 1991
- [22] Raymond G. Jacquot
Modern Digital Control Systems
1981

References on MultiObjective Optimization with GA:

- [23] D.Quagliarella, J.Periaux, C.Poloni, G.Winter
Genetic Algorithms and Evolution Strategies in Engineering and Computer Science
Recent Advances and Industrial Applications
John Wiley and Sons, 1998 (Book)
- [24] A.Osyczka, S.Kundu
A New method to Solve Generalized Multicriteria Optimization Problems Using the Simple Genetic Algorithm
Structural Optimization, Vol.10 pp.94-99, 1995
- [25] Carlos M. Fonseca, Peter J. Fleming
Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms- Part I: A Unified Formulation
IEEE Transactions on Systems Man and Cybernetics, Vol.28, No.1, pp.26-37, January 1998
- [26] Carlos M. Fonseca, Peter J. Fleming
Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms- Part II: Application Example
IEEE Transactions on Systems Man and Cybernetics, Vol.28, No.1, pp.28-47 January 1998

-
- [27] K.Fujita, N.Hirokawa, S.Akagi, S.Kitamura, H.Yokohata
Multiobjective Optimal Design of Automotive Engine Using Genetic Algorithms
Proceedings of DETC'98 1998 Design Engineering Technical Conference, Sept. 13-16, 1998 Atlanta, Georgia
 - [28] J. Nash
Non-Cooperative Games
Annals of Mathematics, Vol.54, pp.286-295, 1951
 - [29] Robert Gibbons
A Primer in Game Theory
Harvester Wheatsheaf, 1992 (Book)
 - [30] Joao Pedro Pedroso
Numerical Solution of Nash and Stackelberg Equilibria: and Evolutionary Approach
WEB: <http/>
 - [31] John J. Grefenstette
Optimization of Control Parameters for Genetic Algorithms
IEEE Transactions on Systems Man and Cybernetics, Vol.SMC-16, No.1, pp.122-128, January-February 1986
 - [32] H. Tamaki, H. Kita, S. Kobayashi
Multi-Objective Optimization by Genetic Algorithms: A Review
Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp.517-522, 1996
 - [33] Eckart Zitzler, Lothar Thiele
Multiobjective Evolutionary Algorithms: a Comparative Case Study and The Strength Pareto Approach
IEEE Transactions on Evolutionary Computation, Vol.3, No.4, pp.257-271 Nov.1999
 - [34] Jong-Hwan Kim, Hyun Myung
Evolutionary Programming Techniques for Constrained Optimization Problems
IEEE Transactions on Evolutionary Computation, Vol.1, No.2, pp.129-140 July 1997
 - [35] Bruno Sareni, Laurent Krahenbuhl
Fitness Sharing and Niching Methods Revisited
IEEE Transactions on Evolutionary Computation, Vol.2, No.3, pp.97-106, September 1998
 - [38] T.L. Johnson, M.Athans
On the Design of Optimal Constrained Dynamic Compensators for Linear Constant Systems
IEEE Transactions on Automatic Control, Vol.15, pp.658-669, December 1970
 - [39] K.M.Sobel, W.Yu, F.J.Lallman
Eigenstructure Assignment for the Control of a Highly Augmented Aircraft
International Journal of Guidance Control and Dynamics, Vol.12, No.3, pp.318-324, May-June 1989

4

*Model Reference Adaptive Control
With Hybyrid Genetic Algorithms*

Contents:

4.1	Model Reference Adaptive Control	p.4.2
4.1.1	Introduction	p.4.2
4.2	Model Reference Adaptive Control: SISO Systems	p.4.5
4.2.1	Simulation-4.1: Lyapunov Stability method.	p.4.5
4.2.2	Simulation-4.2: MIT-rule Method	p.4.11
4.2.3	Simulation-4.3: Hybrid Genetic Algorithms	p.4.16
4.2.4	Comparison of Results.	p.4.22
4.3	Model Reference Adaptive Control: MIMO Systems	p.4.24
4.3.1	Simulation-4.4: Lyapunov Stability method.	p.4.24
4.3.2	Simulation-4.5: Gradient based (MIT-rule) Method	p.4.30
4.3.3	Simulation-4.6: Hybrid Genetic Algorithms	p.4.36
4.3.4	Convergence Rates.	p.4.43
4.4	Chapter Summary and Conclusion	p.4.45
4.5	References and Further Reading	p.4.48

4.1 Model Reference Adaptive Control:

4.1.1 Introduction

The objective of this chapter is to apply hybrid genetic algorithms optimization to the implementation of model reference adaptive control systems (MRAC). Results from conventional MRAC methods (i.e. MIT-gradient rule, and Lyapunov stability theory) are compared with genetic algorithms. In this chapter, we investigate the following two applications:

- (i) Model reference adaptive control applied to simple linear Single Input Single Output (SISO) systems. Simulation results compare: MIT-rule, Lyapunov stability methods, and hybrid genetic algorithms.
- (ii) Model reference adaptive control extended to more complex linear Multi Input Multi Output (MIMO) second order systems. Simulation using the lateral aircraft dynamics is used. Again, we compare: MIT-rule, Lyapunov methods, and hybrid genetic algorithms.

A brief introduction to MRAC is outlined below.

Model Reference Adaptive Control (MRAC) theory is well established, originally developed to deal with aircraft adaptive control in a changing environment and changing operating conditions. Unfortunately, early applications in the 1950's failed to stimulate interest due to lack of hardware and nonexistent stability theory [2]. Renewed interest in adaptive control emerged in the 1960's following the development of state space techniques and Lyapunov based stability theory were introduced. The availability of rudimentary computer hardware made physical realizability possible.

Presently, theory on MRAC control has matured, providing a systematic procedure to control linear systems with partially known or changing parameters. Currently, MRAC has been extended to include variables structure control with only input-output measurements [5, 6], nonlinear systems [1, 8], and adaptive sliding mode [9], and fuzzy logic control [12].

A typical MRAC system is illustrated below in figure 4.1. The goal of MRAC is to modify the controller (parameters θ_c) such that the closed loop input/output response of the plant and controller is the same as the reference model. Thus the error between the plant and model outputs must be made to approach zero asymptotically.

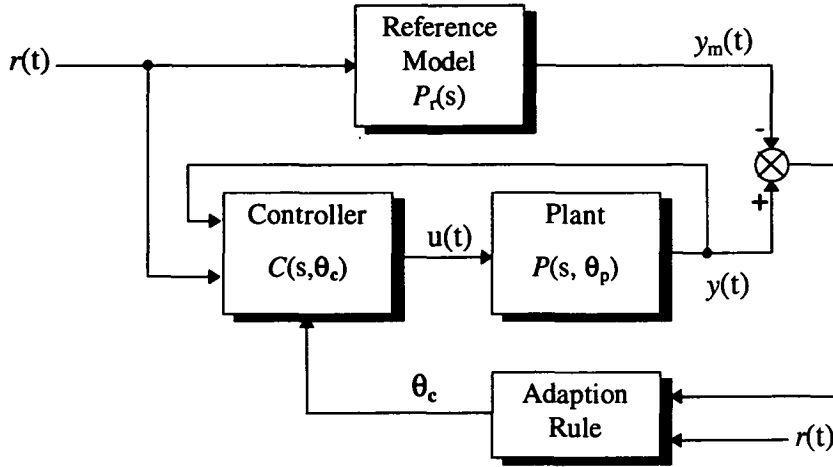


Fig.4.1
Typical (Direct) MRAC Scheme Configuration

From figure 4.1, notice the two loops: the regular feedback control loop and the parameter adjustment loop. The dynamics of this closed loop system combined with the parameter adjustment law actually constitute a nonlinear system. Additionally, the feedback controller $C(s, \theta_c)$ must be designed such that all signals are bounded. The problem is generally simplified by defining a fixed controller structure. The controller incorporates some adjustable parameters θ_c , these are modified by some *adaptive rule* such that global asymptotic stability of the error equation $e(t)$ is guaranteed. Thus the objective is to determine the adaptive law which is a function of the error $e(t) = y(t) - y_m(t)$, whilst ensuring stability, thus:

$$\theta_c(t) = \text{adaption_law}(e(t), r(t)): \text{ such that: } e(t) \rightarrow 0 \text{ asymptotically}$$

Almost all literature on MRAC control involves specifying the controller structure and the manner by which the controller parameters are to be adjusted (adaptive law). The two commonly used parameter adjustment methods are: (i) gradient based (or MIT rule), sometimes commonly referred to as the *sensitivity function*, and (ii) Lyapunov stability methods. Both methods are described in detail in this chapter, with specific reference to state-space solutions. Other techniques such as small gain theorem and passivation theory have also been used in conjunction with nonlinear adaptive control. Note that the controller may be a simple PID with adaptable gain $\{K_p, K_i, K_d\}$ parameters [10], or a more complex neural network such as a radial basis function for nonlinear systems [11].

Model reference adaptive control systems fall under two main categories: (i) *Direct MRAC* and (ii) *Indirect MRAC*:

- (i) *Direct MRAC*: Consists of only one step, the error $e(t)$ and the input $r(t)$, these feed into the adaptive rule block which is used to directly estimate the controller parameters θ_c such that the error reduces to zero asymptotically. There is no system identification block. This is illustrated in figure 4.1.
- (ii) *Indirect MRAC*: There are two steps to indirect MRAC, the first is to estimate online the plant unknown (or changing parameters) θ_p from the error $e(t)$ and the input $r(t)$, this is also known as *system identification*. This information is then used to compute or adjust the controller parameters θ_c using some relationship or function: $\theta_c = F(\theta_p)$. This is illustrated in fig.4.2 below:

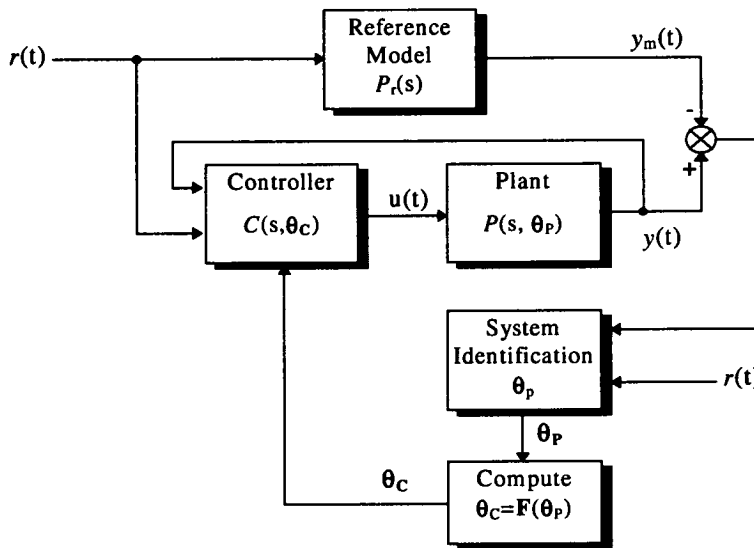


Fig.4.2
Indirect MRAC Scheme Configuration

This second method is generally more involved, but has the added advantage of estimating some useful plant parameters, for instance payload mass of a robotic manipulator. Adaptive control schemes also fall under other categories of self tuning regulators (STR), self organizing control (SOC), gain scheduling (also known as open loop adaptive control), dual control, stochastic self tuning control, and embody concepts of system identification, real time parameter estimation, such as recursive least squares.

Applications include: robotic manipulators [9], disk drive control, aircraft stability augmentation, aircraft reconfigurable control [4,7]. Industrial applications are also emerging, due to availability of software and hardware. The subject of adaptive control is extensive, as seen in [3], and this chapter will only attempt to deal with a small subset i.e.: *direct MRAC*.

4.2 Model Reference Adaptive Control: SISO Systems:

For this first set of simulations, a simple SISO system to illustrate and compare hybrid genetic algorithms with conventional model reference adaptive control schemes. Three different methods are compared:

- (i) Generation of parameter update rules using Lyapunov stability theory, section 4.2.1.
- (ii) Generation of parameter update rules using the MIT-rule, section 4.2.2.
- (iii) Generation of parameter update rules using hybrid genetic algorithms, section 4.2.3

A summary and discussion on relative performance and merit of each method is found at the end of this section. Most of the theory presented below is based on reference [11], with greater detail for simulation implementation purposes. Mathematical derivations are provided in each section. In all simulations, it is assumed that full state feedback is available.

4.2.1 Simulation-4.1: Lyapunov Stability Method:

(i) Theory:

In this section, parameter update rules are derived for a simple SISO system using the Lyapunov stability method. The final configuration is illustrated in figure 4.3 below. Consider a simple SISO system in state variable form:

-Plant:

$$\dot{x} = a.x + b.u \quad \text{Eqn.4.1}$$

in which the scalar coefficients a and b are time varying or unknown, a feedback controller is sought with the general structure:

-Controller:

$$u = d.u_c - g.x \quad \text{Eqn.4.2}$$

where the controller parameters to be solved for are defined by the vector: $\theta_c = \{d, g\}$. Substituting equation 4.2 into 4.1 gives the following closed loop system:

-Closed loop system:

$$\dot{x} = (a - b.g).x + b.d.u_c \quad \text{Eqn.4.3}$$

The above closed loop system is required to follow the reference model given by the expression:

-Reference Model:

$$\dot{x}_m = a_m.x_m + b_m.u_c \quad \text{Eqn.4.4}$$

Comparing equation 4.3 with 4.4, the models are perfectly matched when the controller coefficients are the same as the model coefficients:

$$g^o = b^{-1} \cdot (a - a_m) \quad \text{and} \quad d^o = b^{-1} \cdot b_m \quad \text{Eqn.4.5}$$

Perfect model following occurs when the coefficients in equation 4.3 are: $g=g^o$ and $d=d^o$. Obviously, we cannot simply substitute equations 4.5 into the controller 4.2 because the plant coefficients a and b are unknown or time varying. The error (time derivative) equation between the closed loop plant and the reference model is:

$$\dot{e} = \dot{x} - \dot{x}_m \quad \text{Eqn.4.6}$$

Substituting equations 4.3 and 4.4 into equation 4.6, we get:

$$\dot{e} = ((a - b \cdot g) - a_m) \cdot x + (b \cdot d - b_m) \cdot u_c + a_m \cdot e \quad \text{Eqn.4.7}$$

Substituting the equations for perfect model following equation 4.5 into equation 4.7, we get a simplified form:

$$\dot{e} = \psi_1 \cdot (g - g^o) + \psi_2 (d - d^o) + a_m \cdot e \quad \text{Eqn.4.8}$$

where the terms are: $\psi_1 = -b \cdot x$, and: $\psi_2 = b \cdot u_c$. Note that this equation requires that the coefficient b is known. As we will see later, the coefficient b can simply be replaced by b_m under some mild conditions. To get the adaptive rules for g and d , define a Lyapunov function of the form:

$$V(e, g, d) = \frac{1}{2} \gamma \cdot e^T \cdot P \cdot e + \frac{1}{2} (g - g^o)^T (g - g^o) + \frac{1}{2} (d - d^o)^T (d - d^o) \quad \text{Eqn.4.9}$$

The Lyapunov function has been written in vector/matrix form for generalization to MIMO systems. The two conditions required to be satisfied by a Lyapunov function are: $V>0$ for all values of e, g, d ; which is clearly satisfied by equation 4.9, and $V=0$ when: $e=0, g=g^o, d=d^o$. The matrix P must be positive symmetric, and the coefficient γ can be used to control the parameter update rate. Differentiating equation 4.9 with respect to time, and noting that $e(t), g(t), d(t)$ are also functions of time, we get:

$$\dot{V} = \frac{1}{2} \gamma \cdot \dot{e}^T P e + \frac{1}{2} \gamma \cdot e^T P \dot{e} + (g - g^o)^T \dot{g} + (d - d^o)^T \dot{d} \quad \text{Eqn.4.10}$$

substituting the error function given by equation 4.8 into 4.10, and simplifying to a scalar system, the equation becomes:

$$\dot{V} = \gamma.P.a_m e^2 + (g - g^o).(\gamma.P.e.\psi_1 + \dot{g}) + (d - d^o).(\gamma.P.e.\psi_2 + \dot{d}) \quad \text{Eqn.4.11}$$

Clearly, the necessary conditions for convergence require that $\dot{V} < 0$, which can easily be satisfied by choosing:

$$\left. \begin{aligned} P.a_m &= -q \\ \dot{g} &= -\gamma.P.e.\psi_1 \\ \dot{d} &= -\gamma.P.e.\psi_2 \end{aligned} \right\} \quad \text{Eqn.4.12}$$

where $\gamma > 0$, substituting $\psi_1 = -b.x$, and: $\psi_2 = b.u_c$ into equation 4.12 gives the adaptive equations thus:

$$\boxed{\frac{dd}{dt} = -\gamma.P.e.b.u_c \quad \text{and} \quad \frac{dg}{dt} = \gamma.P.e.b.x} \quad \text{Eqn.4.13}$$

Converting to discrete time for computer simulation, the values for g and d at each time step can be updated with the simple numerical integration rule: $g_{n+1} = g_n + \Delta T.dg/dt$, and: $d_{n+1} = d_n + \Delta T.dd/dt$. Note also that whilst the update equation is independent of the coefficient a , it depends on b . The general approach is to replace b by b_m . This approach is valid as long as both b and b_m have the same sign, i.e.: $\text{sign}(b) = \text{sign}(b_m)$. If this condition is not satisfied, then the adaptive equation fail to converge (as seen later from simulations). The complete adaptive system is illustrated in figure 4.3 below:

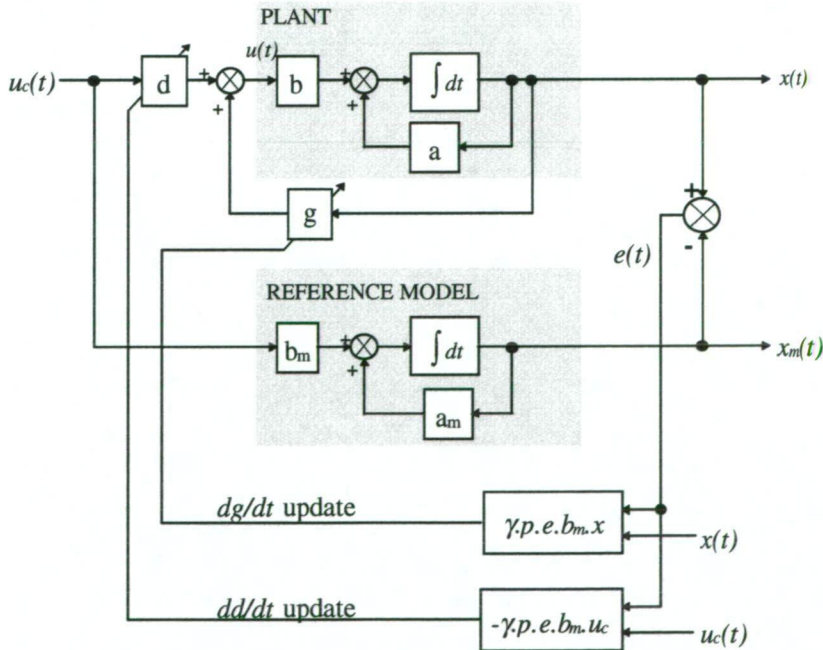


Fig.4.3
MRAC Scheme Using Lyapunov stability (SISO system)

(ii) Simulation Setup:

Given the following plant and reference model coefficients: $a=-1.0$, $b=2.0$, $a_m=-0.2$, $b_m=0.5$.

Plant:	$\dot{x} = a \cdot x + b \cdot u$
Controller:	$u = d \cdot u_c - g \cdot x$
Closed Loop System:	$\dot{x} = (a - b \cdot g) \cdot x + b \cdot d \cdot u_c$
Reference Model:	$\dot{x}_m = a_m \cdot x_m + b_m \cdot u_c$

If the values of a and b are known in advance, then to compute the feedback gains: $g=(a-a_m)/b = -0.4$ and $d=b_m/b = 0.25$. However since a and b are not known or can change, we cannot calculate the d and g values, the following simulation shows convergence of the d and g values to their correct values of $g=-0.4$ and $d=0.25$.

(iii) Simulation Results:

Simulation results for the setup illustrated in figure 4.3 are given in simulation 4.1 below. The coefficients used in the simulation are: $a=-1.0$, $b=2.0$, $a_m=-0.2$, $b_m=0.5$. The controller gains should converge to: $g=(a-a_m)/b = -0.4$, and $d=b_m/b = 0.25$. Results are illustrated in figure 4.4 below.

1. The first (topmost) plot illustrates the convergence of the d parameter, this correctly converges to 0.25.
2. The second plot illustrates the convergence of the g parameter, this also correctly converges to -0.4.
3. The third graph illustrates the convergence of the $(a-b \cdot g) \rightarrow a_m$, this also correctly converges to the value of a_m , i.e.: -0.2.
4. The fourth graph illustrates the convergence of the $b \cdot d \rightarrow b_m$, this also correctly converges to the value of b_m , ie: 0.5.

The rate of convergence strongly depends upon the value of γ . Table 4.1 below summarizes the results obtained after 3000 iterations with different values of γ . With a high value of γ , such as 1, the convergence is very rapid, within the first 1000 iterations. Note also that increasing the value of γ increases the oscillatory behavior of the convergence. Further increases in γ will result in instability.

γ	Iterations	$d \rightarrow 0.25$	$g \rightarrow -0.4$	error	Flops
0.01	3000	0.2492	-0.4111	0.0506	104817
0.05	3000	0.2500	-0.4000	-0.0000	104817
0.20	3000	0.2500	-0.4001	0.0001	104817

Table 4.1
Convergence as a function of γ

Since there are two parameter update equations, we could also use a different value of γ in each equation, thus:

$$\frac{dg}{dt} = \gamma_1 \cdot P.e.b.x \quad \text{and} \quad \frac{dd}{dt} = -\gamma_2 \cdot P.e.b.u_c$$

in order to get the best convergence for each parameter. Figure 4.5, illustrates the problem associated when $sign(b) \neq sign(b_m)$, in this simulation the value of $b=2.0$ and $b_m=-0.1$, and the adaptive algorithm actually fails to converge.

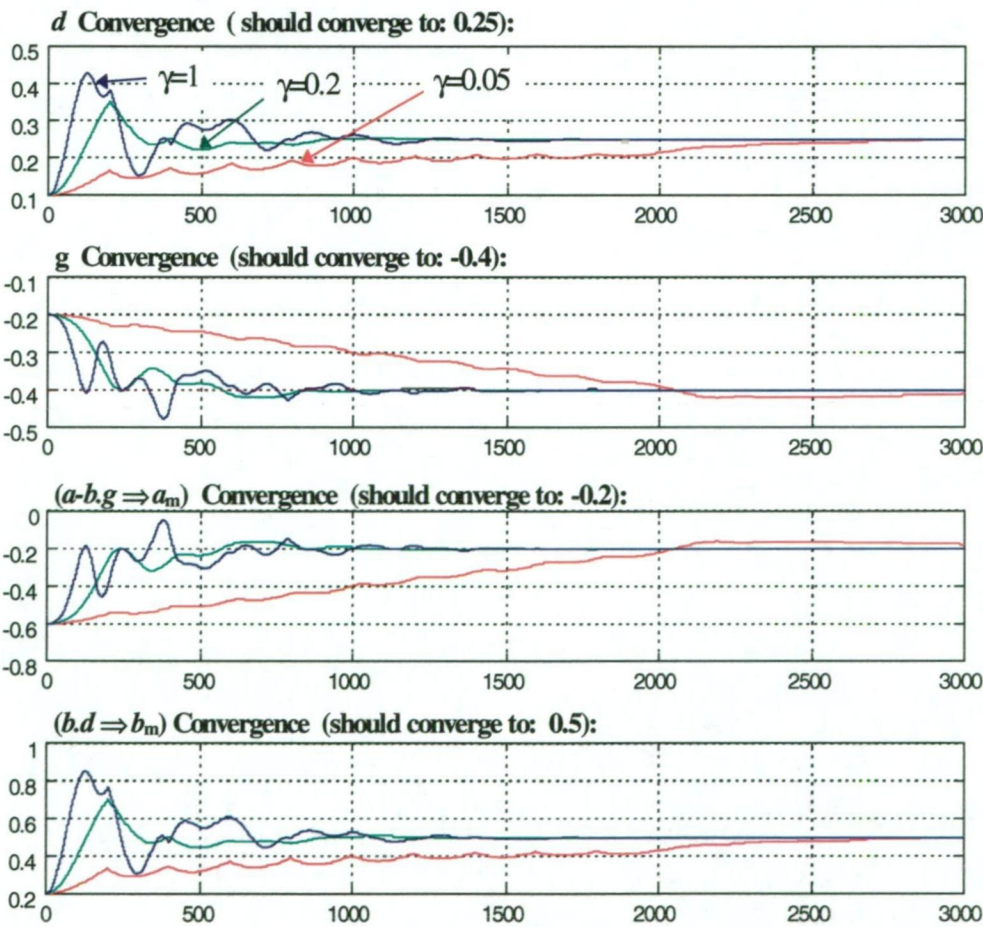


Fig.4.4
MRAC Scheme Using Lyapunov stability (SISO system) Convergence

The simulation shows convergence of the d and g values to their correct values of $g=-0.4$ and $d=0.25$, using different values of gamma. Note that the estimated value of a_m given by: $(a-b.g)$ must remain negative (i.e. eigenvalue) for the closed loop system to remain stable.

(iv) Failure To Converge:

The above simulation shows that convergence is possible. However the assumption is made that $\text{sign}(b)=\text{sign}(b_m)$ in the derivation of the update rule for the adaptive control scheme. In some instances this may not be valid, and the algorithm fails to converge. For instance consider the following situation: $b_m=-0.1$, the results now illustrate the adaptive algorithm failing to converge.

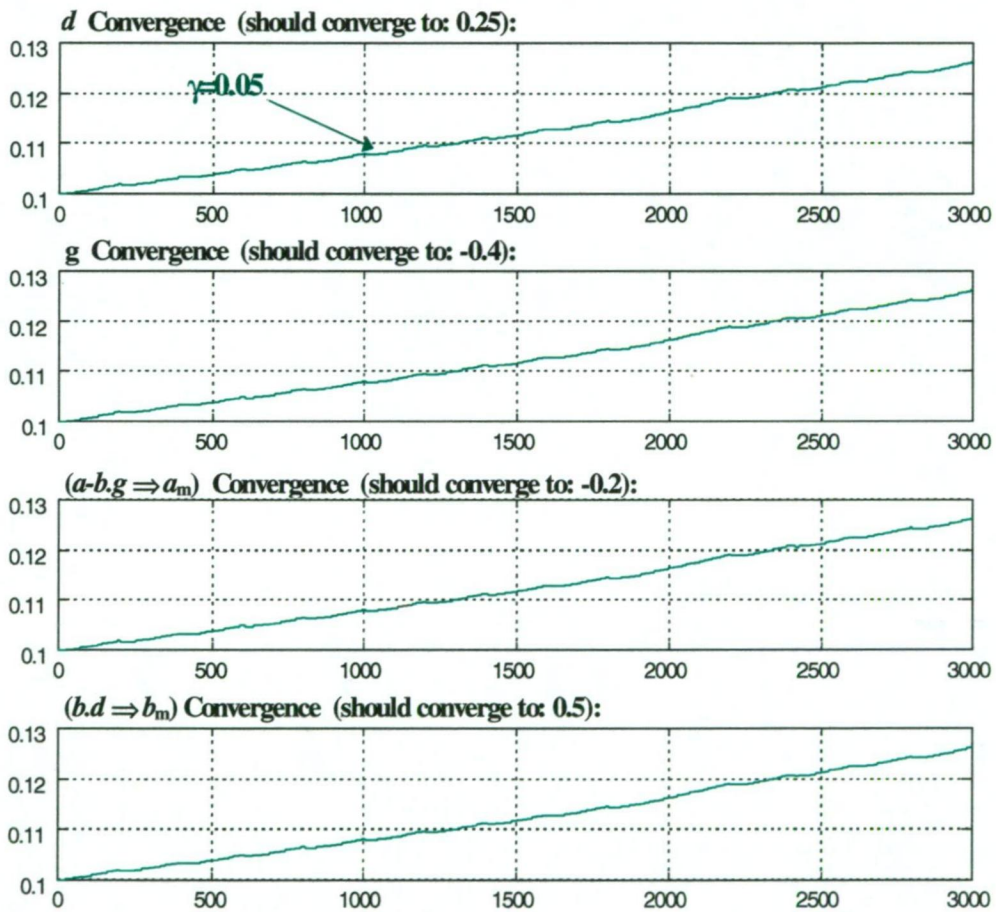


Fig.4.5
MRAC Scheme Using Lyapunov stability (SISO system) Failure to Convergence

Another cause for failure to converge is the choice of gamma, if chosen too small, convergence can be slow, if chosen too large, instability may result.

4.2.2 Simulation-4.2: MIT-Gradient Based Method:

(i) Theory:

In this section, the parameter update equations are derived for the same SISO system using the MIT-rule method. This method is also known as *gradient based*, *steepest descent* or *sensitivity based* method. The final configuration is illustrated in figure 4.6 on the following page. Consider the SISO system previously defined:

Plant:	$\dot{x} = a \cdot x + b \cdot u$
Controller:	$u = d \cdot u_c - g \cdot x$
Closed Loop System:	$\dot{x} = (a - b \cdot g) \cdot x + b \cdot d \cdot u_c$
Reference Model:	$\dot{x}_m = a_m \cdot x_m + b_m \cdot u_c$

Eqn.4.14

The MIT rule is defined as the negative of the cost function gradient, similar to gradient based optimization, an example of this method is: *backpropagation* when training multi-layer-perceptrons neural networks. The MIT rule is given by the general expression:

$$\frac{d\theta}{dt} = -\gamma \cdot \frac{\partial J}{\partial \theta} \quad \text{Eqn.4.15}$$

where θ =parameters to update, for the SISO problem this is simply $\theta=\{d,g\}$, and J is some error related cost function to minimize. Typically, J is given by the output error function:

$$J(\theta) = \frac{1}{2} e^2(\theta) \quad \text{Eqn.4.16}$$

Computing partial derivatives:

$$\frac{\partial J}{\partial \theta} = e \cdot \frac{\partial e}{\partial \theta} \quad \text{Eqn.4.17}$$

Substituting equation 4.17 into equation 4.15 gives the parameter update rule:

$$\frac{d\theta}{dt} = -\gamma \cdot e \cdot \frac{\partial e}{\partial \theta} \quad \text{Eqn.4.18}$$

Thus for the SISO system given above, for each component, the update rule is:

$$\begin{aligned} \frac{dd}{dt} &= -\gamma \cdot e \cdot \frac{\partial e}{\partial d} \\ \frac{dg}{dt} &= -\gamma \cdot e \cdot \frac{\partial e}{\partial g} \end{aligned} \quad \text{Eqn.4.19}$$

Note that any cost function can be used instead of equation 4.15, for instance another choice would be to use $J(\theta) = |e(\theta)|$. The choice of cost function affects the final outcome of the parameter update equations and their rate of convergence. Referring to equation 4.19, the parameter update equations require both the error and the error derivative functions. To obtain the error function, as in section 4.3.1, write:

$$e = x - x_m \quad \text{Eqn.4.20}$$

where x and x_m are obtained by taking Laplace transform of equations 4.14 thus:

$$\begin{aligned} x &= (s - (a - b.g))^{-1} . b.d.u_c \\ x_m &= (s - a_m)^{-1} . b_m.u_c \end{aligned} \quad \text{Eqn.4.21}$$

note the slight abuse in notation used in equation 4.21 for the new variables x and x_m . The error is then given by:

$$e = (s - (a - b.g))^{-1} . b.d.u_c - (s - a_m)^{-1} . b_m.u_c \quad \text{Eqn.4.22}$$

where s =Laplace operator. Computing the error gradients with respect to each parameter gives:

$$\left. \begin{aligned} \frac{\partial e}{\partial d} &= (s - (a - b.g))^{-1} . b.u_c \\ \frac{\partial e}{\partial g} &= -(s - (a - b.g))^{-2} . b^2 . d.u_c \end{aligned} \right\} \quad \text{Eqn.4.23}$$

Substituting equations 4.23 into equations 4.19, and after some simple manipulation giving the following parameter update equations:

$$\begin{aligned} \frac{dd}{dt} &= -\gamma . e . \left(\frac{b}{s - (a - b.g)} \right) . u_c \\ \frac{dg}{dt} &= \gamma . e . \left(\frac{b}{s - (a - b.g)} \right) . x \end{aligned} \quad \text{Eqn.4.24}$$

Note the similarity of these update equations with those derived using Lyapunov stability theory in the previous section (equations 4.13). Note also that since both a and b are unknown, the following approximations are often made: $b \approx b_m$ and: $(a - b.g) \approx a_m$. The update equations are:

$$\begin{aligned} \frac{dd}{dt} &= -\gamma . e . \left(\frac{b_m}{s - a_m} \right) . u_c \\ \frac{dg}{dt} &= \gamma . e . \left(\frac{b_m}{s - a_m} \right) . x \end{aligned}$$

Eqn.4.25

As a further observation, when comparing with the Lyapunov derived update equations (Eqn.4.13), is that the above expressions for dd/dt and dg/dt define a first order dynamical system instead of a constant (Eqn.4.13). The rate of convergence is dependent on γ , which may be chosen differently for each parameter update equation in 4.25 above. The entire MRAC setup is illustrated in figure 4.6 below.

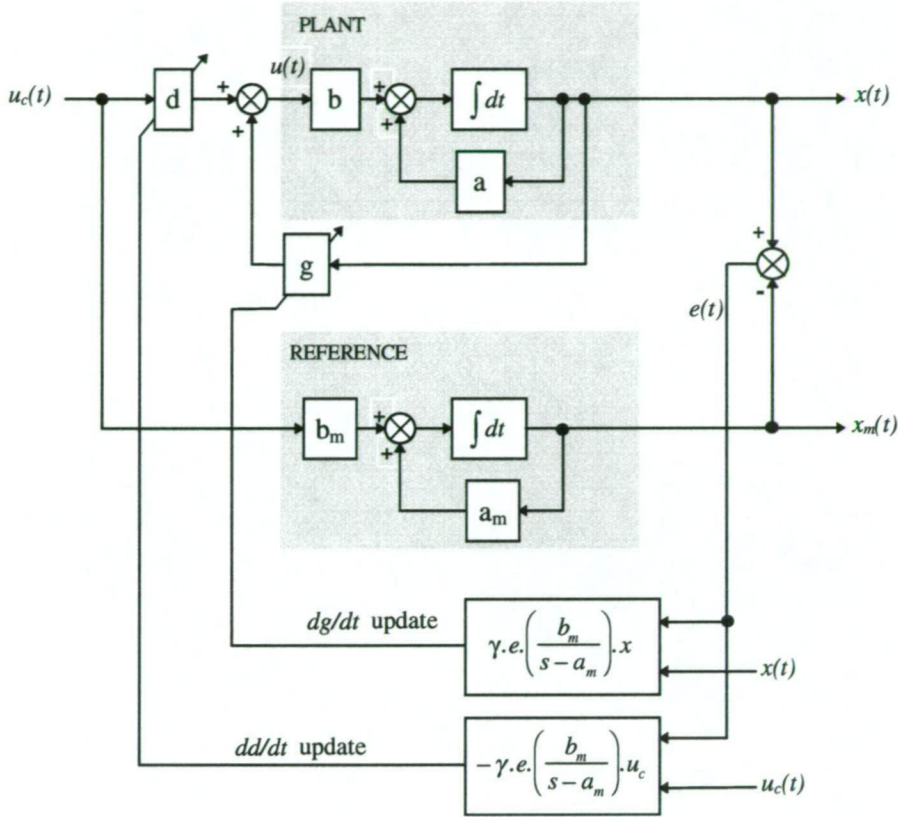


Fig.4.6
MRAC Scheme Using the MIT rule (SISO system)

Implementation issues: for MATLAB simulation purposes, equations 4.25 are better handled in state space form, thus if we define $d_1 = d$ and $d_2 = \dot{d}_1$, then it can be re-written as:

$$\begin{pmatrix} \dot{d}_1 \\ \dot{d}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & a_m \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} + \begin{pmatrix} 0 \\ -\gamma \cdot e \cdot b_m \end{pmatrix} \cdot u_c \quad \text{Eqn.4.26a}$$

This can then be treated the same way as a state-space problem. The same applies to the g parameter update equation, if we define $g_1 = g$ and $g_2 = \dot{g}_1$, then:

$$\begin{pmatrix} \dot{g}_1 \\ \dot{g}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & a_m \end{pmatrix} \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \gamma \cdot e \cdot b_m \end{pmatrix} \cdot x \quad \text{Eqn.4.26b}$$

(ii) Simulation Setup:

Identical to 4.2.1 above.

(iii) Simulation Results:

Simulation results for the setup illustrated in figure 4.6 are given in simulation 4.2 below. The coefficients used in the simulation are as before: $a=-1.0$, $b=2.0$, $a_m=-0.2$, $b_m=0.5$. Therefore the controller gains should converge to: $g=(a-a_m)/b = -0.4$, and $d=b_m/b = 0.25$. Simulation results are illustrated in figure 4.7 below:

1. The first (top figure) plot illustrates the convergence of the d parameter, this correctly converges to 0.25.
2. The second plot illustrates the convergence of the g parameter, this also correctly converges to -0.4.
3. The third graph illustrates the convergence of the $(a-b.g) \Rightarrow a_m$, this also correctly converges to the value of a_m , i.e.: -0.2.
4. The fourth graph (bottom) illustrates the convergence of the $b.d \Rightarrow b_m$, this also correctly converges to the value of b_m , i.e.: 0.5.

The rate of convergence depends on the value of γ , the table 4.2 below summarizes the results obtained after 6000 iterations with different values of γ :

γ	Iterations	$d \rightarrow 0.2500$	$g \rightarrow -0.4000$	error	Flops
0.01	6000	0.2525	-0.4007	0.0124	336201
0.02	6000	0.2514	-0.3996	-0.0044	336201
0.04	6000	0.2803	-0.4041	0.0719	336201

Table 4.2
Convergence as a function of γ

From table 4.2, convergence is slower when compared with the Lyapunov method in simulation 4.1. Large values (0.04) produce an oscillatory behavior which actually reduces convergence. Thus a small value of gamma results in a too-slow convergence due to low update, a high value of gamma can also result in a too-slow convergence due to oscillation. Hence the proper selection of gamma is essential.

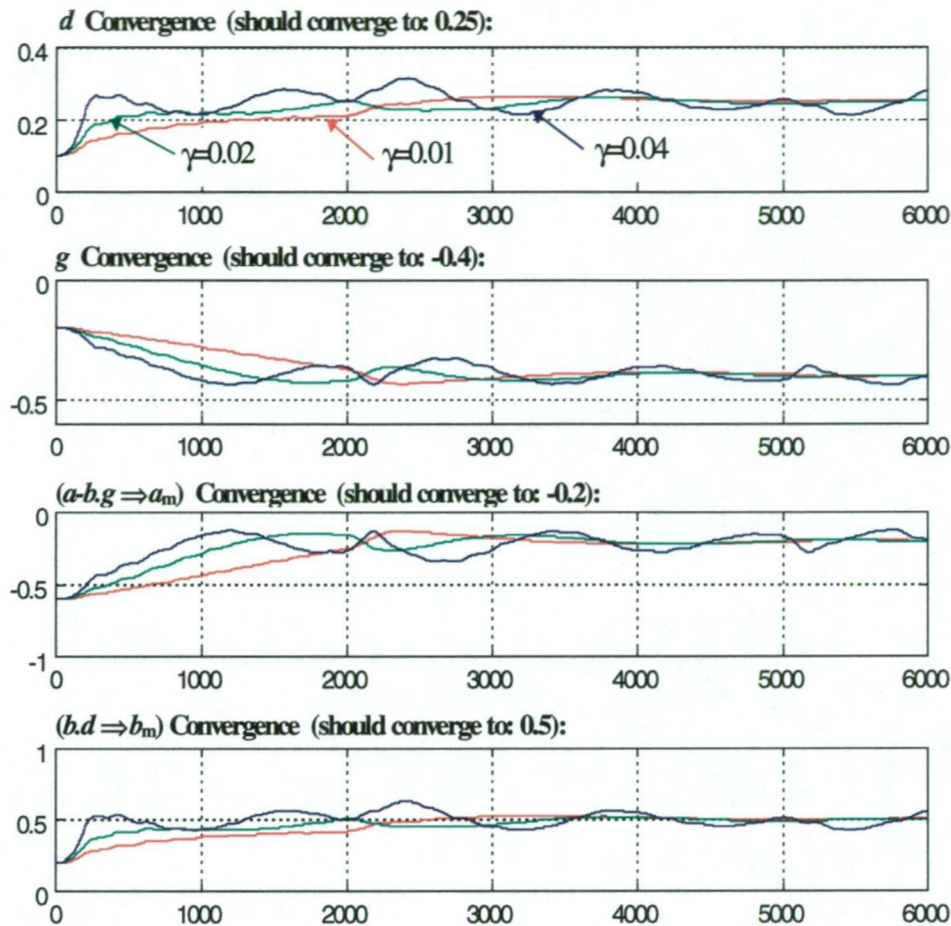


Fig.4.7
MRAC Scheme Using the MIT-rule (SISO system) Convergence

Note that the solution $(a-b.g)$ is always negative indicating that the closed loop system remains stable during the adaptive process. Convergence is slower than the first method, refer to table 4.2. In general the MIT (or gradient based) methods require a small value of gamma to avoid instability.

(iv) Failure To Converge:

The above simulation shows that while convergence is possible, the assumption is made that $b=b_m$, and the approximation: $a-b.g=a_m$ in the derivation of the update rule for the adaptive control scheme. In some instances this is not always valid and the algorithm fails to converge. Thus, this scheme also suffers from the same problem as the first simulation, when $sign(b) \neq sign(b_m)$, it fails to converge. A further drawback, the MIT method produces a second order update law when compared with the Lyapunov method which produces only a first order update law.

4.2.3 Simulation 4.3: Hybrid Genetic Algorithms:

In this section, the parameter update rules are derived for the same SISO system using hybrid genetic algorithms. The three methods used are: (i) conventional genetic algorithms (GA), (ii) GA+simulated annealing, and (iii) GA+greedy search. Computational effort and convergence rates are compared. The final configuration is illustrated in figure 4.9 below. The same SISO system is used in the simulations:

Plant:	$\dot{x} = a \cdot x + b \cdot u$
Controller:	$u = d \cdot u_c - g \cdot x$
Closed Loop System:	$\dot{x} = (a - b \cdot g) \cdot x + b \cdot d \cdot u_c$
Reference Model:	$\dot{x}_m = a_m \cdot x_m + b_m \cdot u_c$

Eqn.4.27

(i) **Conventional Genetic Algorithms:** The fitness function is defined to be the inverse of the error function. The error is computed as the RMS value of the output difference $x(t) - x_m(t)$ sum:

$$e^2 = \sum_{j=1}^n (x_j(t) - x_{m_j}(t))^2 \quad \text{Eqn.4.28}$$

Where n =number of sample points, the chromosomal representation for this problem is illustrated in figure 4.8 below. The controller parameters to solve for are: d , and g . The error *err* is defined by equation 4.28 above, and the fitness is the inverse of the error: *fitness* = 1/*error*.

d	g	<i>err</i>	<i>fitness</i>
-----	-----	------------	----------------

Fig.4.8

MRAC Scheme Using the Genetic Algorithm (SISO system)

This problem represents an unconstrained optimization problem which is relatively straightforward to solve with genetic algorithms. The problem is solved essentially off-line, thus from past n measurements of x and x_m , the genetic algorithm computes the d , and g values which minimize the error function 4.28.

Simulation results for the setup illustrated in figure 4.9 are given in simulation 4.3 below. The coefficients used in the simulation are: $a=-1.0$, $b=2.0$, $a_m=-0.2$, $b_m=0.5$. Therefore the controller gains should converge to: $g=(a-a_m)/b = -0.4$, and $d=b_m/b = 0.25$.

For the genetic algorithm, the following setup was used: mutation and crossover probability tested over a range of values, selection type: binary tournament, population size: 20, sample points $n=200$, convergence results are summarized in table 4.3 below:

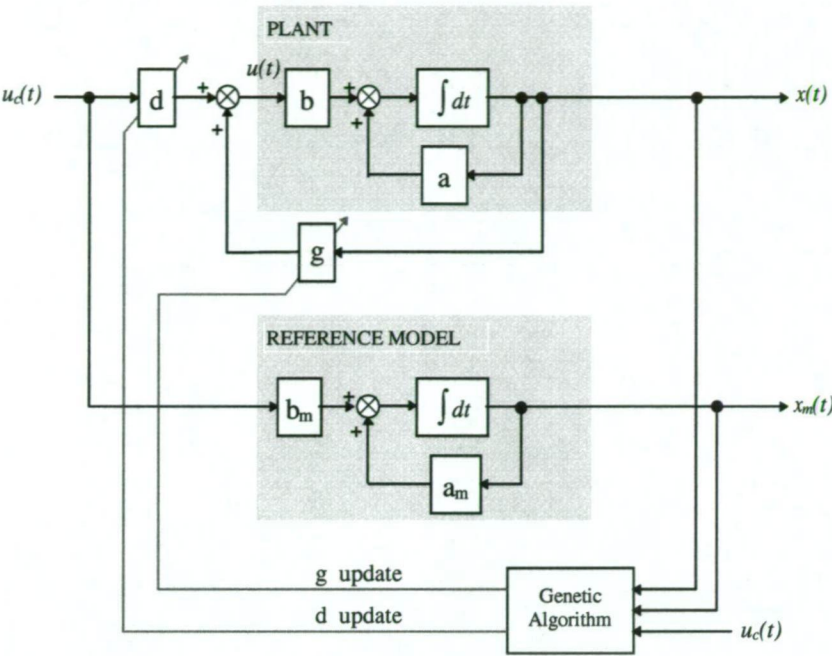


Fig.4.9
MRAC Scheme Using the Genetic Algorithm (SISO system)

Table 4.3 below summarizes the convergence results obtained using the genetic algorithm. Because genetic algorithms are a stochastic search based algorithms, the simulation was conducted ten times with different values of mutation probability P_m and crossover probability P_c to observe the variation in convergence. In all cases, the GA converges within 60 generations (3.9 Mflops) irrespective of crossover or mutation probability. The fast convergence rate is due to the fact that this is a SISO system and only two parameters are searched for: $\{d, g\}$. Consequently this is a relatively simple problem to solve using a GA. Convergence plots are illustrated in fig. 3.10.

Sim:	Generations	Pc:	Pm:	d→0.25	g→-0.4	Error:	MFP:
1	60	0.40	0.02	0.2553	-0.3961	0.06495	3.9
2	60	0.50	0.02	0.2500	-0.4000	0.00001	3.9
3	60	0.60	0.02	0.2500	-0.4000	0.00000	3.9
4	60	0.70	0.02	0.2500	-0.4000	0.00000	3.9
5	60	0.80	0.02	0.2500	-0.4000	0.00002	3.9
6	60	0.40	0.20	0.2500	-0.4000	0.00016	3.9
7	60	0.50	0.20	0.2515	-0.3989	0.01722	3.9
8	60	0.60	0.20	0.2500	-0.4000	0.00000	3.9
9	60	0.70	0.20	0.2500	-0.4000	0.00001	3.9
10	60	0.80	0.20	0.2500	-0.4000	0.00000	3.9

Table 4.3
Conventional Genetic Algorithm

Typical convergence of the d and g parameters are plotted below. The first (topmost) plot is the error convergence, the middle graph shows the d parameter convergence, and the last (bottom) graph is the g parameter convergence. After only 60 generations, the GA has fully converged.

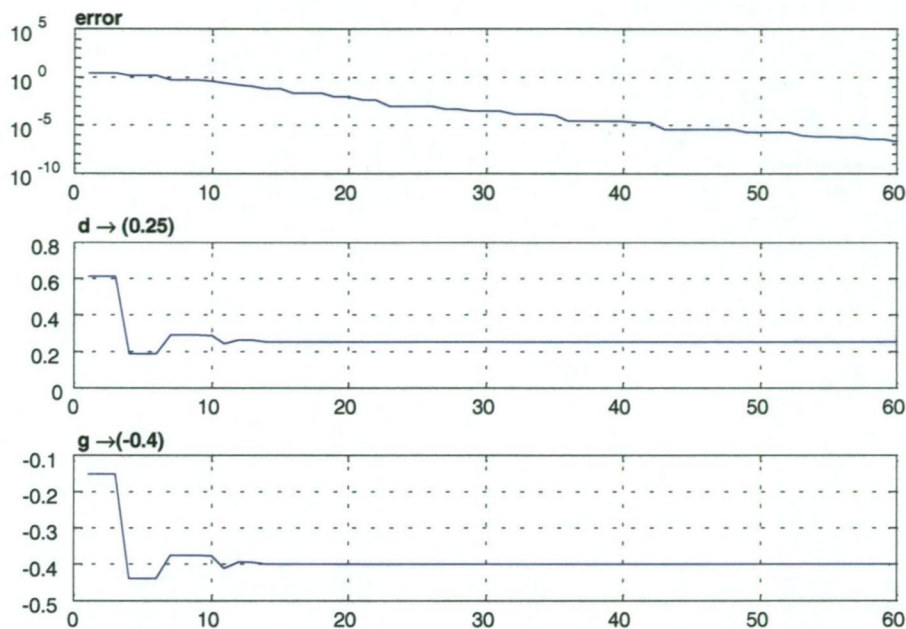


Fig.4.10
MRAC Scheme Using Genetic Algorithms (SISO system) Convergence

Note that unlike the previous two simulations (Lyapunov and MIT methods), the genetic algorithm does not update the values of d and g progressively i.e. at each time step, but the update is made after the genetic algorithm has converged, or after a finite number of generations. Note also that whilst both Lyapunov and MIT methods converge, the actual convergence is gradient based and may be only local. Furthermore, unlike the other two methods, the genetic algorithm uses all past output data of $x_m(t)$ simultaneously when running a search. Instead both the Lyapunov and gradient based methods, greater weight is placed upon current measurements.

Failure To Converge Issues:

Genetic algorithms can take unusually long time to converge or even fail to converge altogether when loss of genetic diversity occurs. Because the search is primarily dependent on crossover, to maintain effective search, variation is necessary. Premature convergence can lead to local minima and convergence thereafter is significantly reduced. Advantages of genetic algorithms is that no assumption is required as seen previously in the Lyapunov method (in which b is replaced by b_m) and the MIT method in which $(a-g.b)$ is replaced by a_m .

In the following simulation, where both the MIT and Lyapunov methods failed to converge when the value of $b_m=-0.1$ we see that the genetic algorithm has successfully converged after 100 generations: convergence is: $d=-0.0498$, $g=-0.4003$, the actual values should be: $d=-0.05$ and $g=-0.4$, refer to figure 4.11 below:

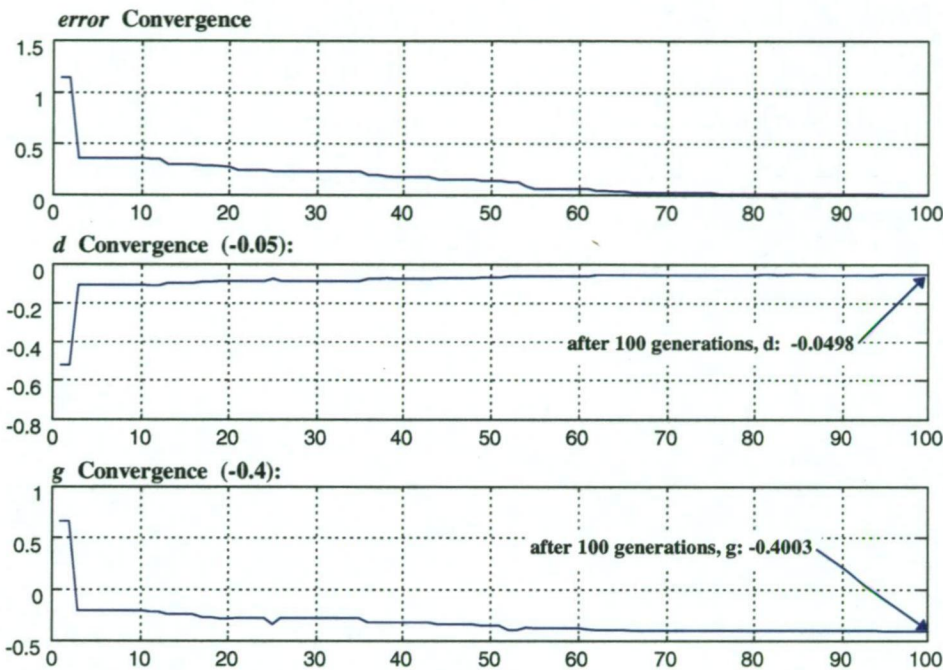


Fig.4.11
MRAC Scheme Using Genetic Algorithms (SISO system) Convergence

(ii) **Genetic Algorithms+Simulated annealing:** The fitness function is defined to be the inverse of the error function Eqn.4.28. The search vector is shown below:

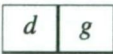


Fig.4.12
Search Vector for Genetic Algorithm+SA (SISO system) scheme

The temperature annealing schedule is given by: $T(k+1)=\alpha.T(k)$, where alpha is given by:

$$\alpha = 10^{(1/N*\log(T_f/T_o))} \tag{Eqn.4.29}$$

Where: T_o =initial temperature normalized to 1.0, and T_f is the final temperature 0.001, N is the number of iterations set to 100. The value of alpha is generally ≈ 0.8 -0.95. Simulation results are illustrated below for 10 simulations, running time is approximately 23 seconds, using 200 data samples.

Sim:	d→0.25	g→-0.4	Error:	MFP:
1	0.2503	-0.3997	0.00662	3.2
2	0.2500	-0.4001	0.00325	3.2
3	0.2499	-0.4001	0.00234	3.2
4	0.2500	-0.4000	0.00096	3.2
5	0.2500	-0.4000	0.00447	3.2
6	0.2504	-0.3997	0.00632	3.2
7	0.2494	-0.4005	0.00693	3.2
8	0.2501	-0.3999	0.00092	3.2
9	0.2501	-0.3999	0.00210	3.2
10	0.2501	-0.4000	0.00222	3.2

Table 4.4
Hybrid Genetic Algorithms + Simulated Annealing

The hybrid GA+simulated annealing gives marginally better convergence results compared with the conventional genetic algorithm. Figure 4.13 below illustrates the {d,g} parameter and error convergence of the hybrid genetic algorithm.

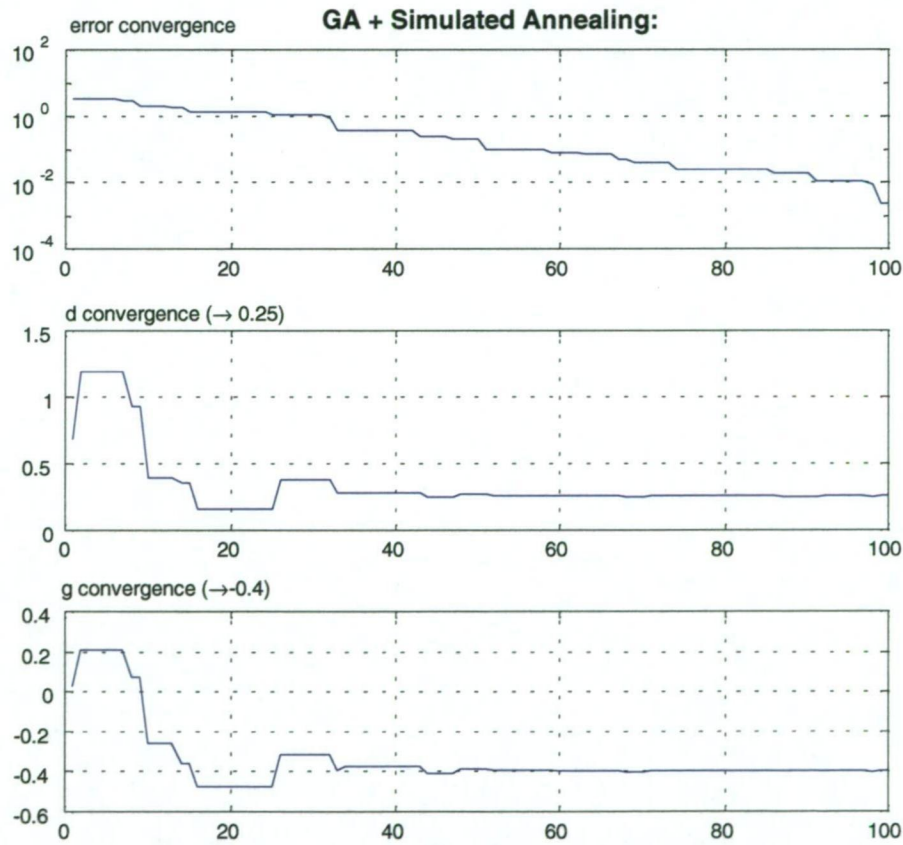


Fig.4.13
Typical convergence of the hybrid genetic algorithm + simulated annealing

(iii) **Genetic Algorithms+Greedy Search:** The search vector is identical to simulated annealing (fig.4.12). Results from the hybrid GA and greedy search are summarized below in table 4.5 for 10 simulation runs. Convergence time is approximately 7 seconds, samples=200.

Sim:	d→0.25	g→-0.4	Error:	MFP:
1	0.2500	-0.4000	0.00060	1.2
2	0.2501	-0.3999	0.00098	1.2
3	0.2500	-0.4000	0.00081	1.2
4	0.2500	-0.4000	0.00005	1.2
5	0.2500	-0.4000	0.00034	1.2
6	0.2500	-0.4000	0.00002	1.2
7	0.2487	-0.4008	0.01768	1.2
8	0.2501	-0.3999	0.00205	1.2
9	0.2499	-0.4001	0.00113	1.2
10	0.2500	-0.4000	0.00053	1.2

Table 4.5
Hybrid Genetic Algorithms + Greedy Search

Typical convergence plots for the hybrid GA and greedy search are illustrated below, convergence is better than either conventional genetic algorithms or hybrid genetic algorithms + simulated annealing.

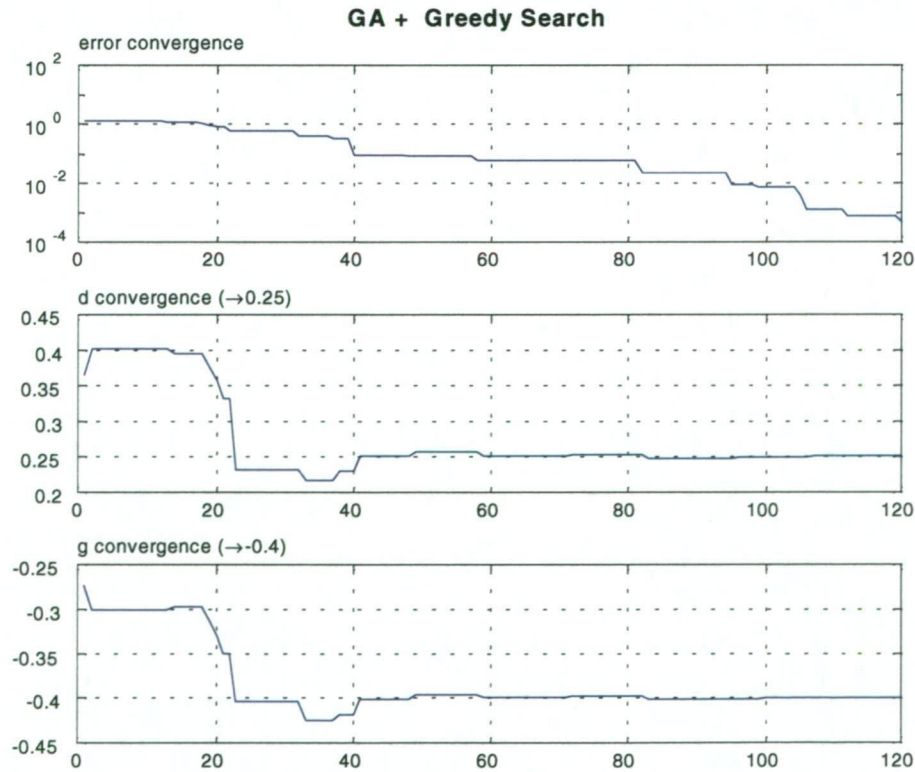


Fig.4.14
Typical convergence of the hybrid genetic algorithm + greedy search

4.2.4 Comparison of Results:

(i) **Summary:** A summary of the three simulations is tabulated below, this is by no means comprehensive, as there are many variations to the conventional Lyapunov and MIT rule to overcome some of the problems and improve convergence rates.

Lyapunov's Method:

Advantages	Disadvantages
<ul style="list-style-type: none">- Easy and simple to implement,- Fast convergence,- Many possible adaptive algorithms possible by choice of Lyapunov functions. Theory well developed- Derivations are relatively straightforward, can be easily extended to MIMO systems.- Produces simple first order update equations.	<ul style="list-style-type: none">- Requires the assumption: $sign(b)=sign(b_m)$- In deriving the update rules, the assumption is made: $b=b_m$- Difficult to apply to nonlinear systems.- Convergence is local.

MIT rule

Advantages	Disadvantages
<ul style="list-style-type: none">- Easy and simple to implement,- Derivations are relatively straightforward, can be easily extended to MIMO systems.- Theory well developed.	<ul style="list-style-type: none">- Requires the assumption: $sign(b)=sign(b_m)$- Slow Convergence, can become easily unstable even at low values of γ, critically depends on γ.- Produces second order update equation.- In deriving the update rules, the assumption is made: $b=b_m$, and $(a-b.g)=a_m$.- Difficult to apply to nonlinear systems.

Hybrid Genetic Algorithms

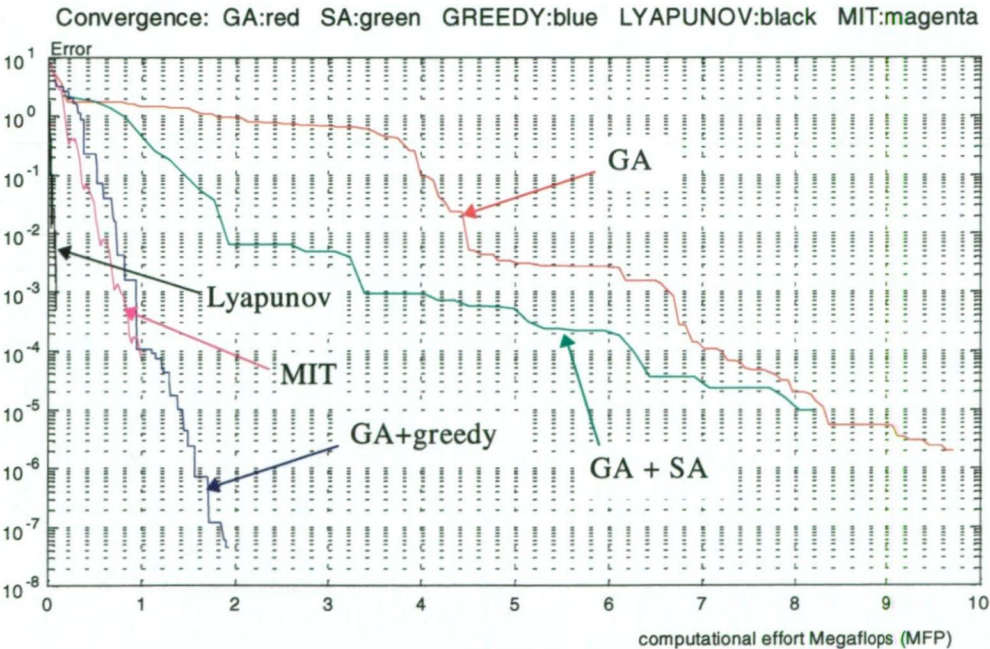
Advantages	Disadvantages
<ul style="list-style-type: none">- Easy and simple to implement.- No mathematical derivations necessary.- Easily extended to nonlinear systems.- Can deal with constraints.- Does not require γ parameter.- No assumptions necessary such as: $sign(b)=sign(b_m)$.- Search is global.	<ul style="list-style-type: none">- Slow convergence, not really a on-line adaptive system.- Equal weight is assigned to all sample points.- No proof of stability, rate of convergence not guaranteed.

(ii) **Convergence Rates:** Table 4.6 below summarizes the convergence properties of the 5 methods used. Lyapunov method is the most efficient in terms of convergence time, the GA is the least efficient. Simulated annealing gives similar convergence results. Greedy algorithms converge faster than both GA and hybrid GA+SA, but slower than the MIT rule based method, and retains the global search feature of a full heuristic algorithm. Whilst the hybrid genetic algorithm require a computational effort greater than either the Lyapunov or MIT rule algorithms, this is attributed to the calculations being conducted with 200 samples per iteration compared with only one sample per iteration of the Lyapunov and MIT-rule methods. If we however compare the computational effort *in flops per sample per iteration*, then a more accurate comparison emerges. The hybrid GA+greedy search method is now comparable with both Lyapunov and MIT-rule methods. This can be seen from the plot shown in figure 4.15 below.

Method	Iterations:	FLOPS/sample/iteration:	Total FLOPS:	Factor:
Lyapunov	3000	35	104,000	1.0
MIT rule	6000	56	336,000	3.2
GA (pop=20)	60	326	3,920,000	37.7
GA+SA	100	163	3,277,000	31.0
GA+Greedy	120	50	1,200,000	11.0

Table 4.6
Comparison of Convergence Rates and computational effort

Figure 4.15 below compares the error convergence as a function of computational effort for the five methods investigated.



4.3 Model Reference Adaptive Control: MIMO Systems:

In these next set of simulations, we repeat the procedures developed in section 4.2 and apply model reference adaptive control to multivariable multi-input multi-output (MIMO) systems. This section consists of the following simulations:

- (i) Generation of parameter update rules using Lyapunov theory for MIMO systems, section 4.3.1
- (ii) Generation of parameter update rules using the MIT-rule for MIMO systems, section 4.3.2.
- (iii) Generation of parameter update rules using hybrid GA for MIMO systems, section 4.3.3

Systems are in state variable form, and full state feedback is assumed to be available. Parameter update equations are derived for each simulation. While the methodology essentially follows on from section 4.2, the equations are more slightly more involved.

4.3.1 Simulation 4.4: Lyapunov Stability Method:

(i) Theory:

The parameter update rules for the MIMO system using Lyapunov stability theory are derived below. The final configuration is illustrated in figure 4.16. Consider a MIMO system in state variable form:

$$\text{Process Dynamics:} \quad \dot{x} = A.x + B.u \quad \text{Eqn.4.29.a}$$

$$\text{Controller:} \quad u = D.u_c - G.x \quad \text{Eqn.4.29.b}$$

$$\text{Closed Loop System:} \quad \dot{x} = (A - B.G).x + B.D.u_c \quad \text{Eqn.4.29.c}$$

$$\text{Reference Model:} \quad \dot{x}_m = A_m.x_m + B_m.u_c \quad \text{Eqn.4.29.d}$$

Where: $x \in \mathbb{R}^n$, $x_m \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $u_c \in \mathbb{R}^m$. comparing equations 4.29.c and 4.29.d, the plant and reference models are perfectly matched when:

$$\begin{aligned} A_m = (A - B.G) &\Rightarrow G^o = B^{-1}.(A - A_m) \\ B_m = B.D &\Rightarrow D^o = B^{-1}.B_m \end{aligned} \quad \text{Eqn.4.30}$$

Note that while the inverse B^{-1} may not exist, a solution to the matrices G and D can still be found. The state error and its time derivative is given by:

$$\begin{aligned} e &= x - x_m \\ \dot{e} &= \dot{x} - \dot{x}_m \end{aligned} \quad \text{Eqn.4.31}$$

Substituting the two equations for the plant (4.29.c) and the reference model (4.29.d) into the above error equation, and after some manipulation gives:

$$\dot{e} = ((A - B.G) - A_m).x + (B.D - B_m).u_c + A_m.e \quad \text{Eqn.4.32}$$

Substituting into equation 4.32 the equations for perfect model following (equation 4.30), we get a simplified form:

$$\dot{e} = A_m.e + B.(D - D^o).u_c - B.(G - G^o).x \quad \text{Eqn.4.33}$$

The above error expression must be written in the same format as equation 4.8 (SISO system) so that the Lyapunov function (Eqn.4.9) can be applied in the same fashion. Additionally the matrices $(D - D^o)$ and $(G - G^o)$ must be vectorized (i.e. convert to single column vector) because when multiplied, it must result in a scalar Lyapunov function, refer to equation 4.9. Therefore, we can re-write the term:

$$B.(D - D^o).u_c = B.\Delta D.u_c = B. \begin{bmatrix} u_c^T & 0 & . & 0 \\ 0 & u_c^T & . & 0 \\ . & . & . & . \\ 0 & 0 & . & u_c^T \end{bmatrix} \begin{bmatrix} \Delta d_{11} \\ . \\ . \\ \Delta d_{mn} \end{bmatrix} \quad \text{Eqn.4.34.a}$$

The Δd vector is simply a *vectorized* version of the ΔD matrix, i.e. the columns of ΔD stacked on top of one another with the first column of ΔD placed at the top, and $\Delta D = D - D^o$. The same applies to the third term of Eqn.4.33:

$$-B.(G - G^o).x = -B.\Delta G.x = -B. \begin{bmatrix} x^T & 0 & . & 0 \\ 0 & x^T & . & 0 \\ . & . & . & . \\ 0 & 0 & . & x^T \end{bmatrix} \begin{bmatrix} \Delta g_{11} \\ . \\ . \\ \Delta g_{mn} \end{bmatrix} \quad \text{Eqn.4.34.b}$$

We can now write the error function in a simplified and more compact format:

$$\dot{e} = A_m.e + \psi_1.\Delta d + \psi_2.\Delta g \quad \text{Eqn.4.35}$$

where:

$$\psi_1 = B. \begin{bmatrix} u_c^T & 0 & . & 0 \\ 0 & u_c^T & . & 0 \\ . & . & . & . \\ 0 & 0 & . & u_c^T \end{bmatrix} \quad \psi_2 = -B. \begin{bmatrix} x^T & 0 & . & 0 \\ 0 & x^T & . & 0 \\ . & . & . & . \\ 0 & 0 & . & x^T \end{bmatrix} \quad \Delta d = \begin{bmatrix} \Delta d_{11} \\ . \\ . \\ \Delta d_{mn} \end{bmatrix} \quad \Delta g = \begin{bmatrix} \Delta g_{11} \\ . \\ . \\ \Delta g_{mn} \end{bmatrix} \quad \text{Eqn.4.36}$$

This equation requires that the matrix B is known. As previously seen (SISO simulation 4.2), the matrix B can simply be replaced by B_m under some mild conditions. To get the adaptive rules for Δg and Δd , define a Lyapunov function in a similar form to Eqn.4.9:

$$V(e, g, d) = \frac{1}{2} \gamma \cdot e^T \cdot P \cdot e + \frac{1}{2} \Delta g^T \Delta g + \frac{1}{2} \Delta d^T \Delta d \quad \text{Eqn.4.37}$$

We follow the same procedure and compute the time derivative of the Lyapunov function to get the adaptive parameter update rules:

$$\dot{V} = \frac{1}{2} \gamma \cdot \dot{e}^T P \cdot e + \frac{1}{2} \gamma \cdot e^T P \cdot \dot{e} + \Delta d^T \cdot \Delta \dot{d} + \Delta g^T \cdot \Delta \dot{g} \quad \text{Eqn.4.38}$$

Insert the error expression Eqn.4.35 into Eqn.4.38, and after some manipulation gives:

$$\dot{V} = \frac{1}{2} \gamma \cdot e^T (A_m^T \cdot P + P \cdot A_m) e + \Delta d^T \cdot (\gamma \cdot \psi_1^T \cdot P \cdot e + \Delta \dot{d}) + \Delta g^T \cdot (\gamma \cdot \psi_2^T \cdot P \cdot e + \Delta \dot{g}) \quad \text{Eqn.4.39}$$

The first term of equation 4.39 can be replaced by $-Q$ where Q is strictly positive real.

$$\dot{V} = -\frac{1}{2} \gamma \cdot e^T Q \cdot e + \Delta d^T \cdot (\gamma \cdot \psi_1^T \cdot P \cdot e + \Delta \dot{d}) + \Delta g^T \cdot (\gamma \cdot \psi_2^T \cdot P \cdot e + \Delta \dot{g}) \quad \text{Eqn.4.40}$$

A negative Lyapunov function requires that the following conditions are satisfied by equation 4.40:

$$\begin{aligned} \Delta \dot{d} &= -\gamma \cdot \psi_1^T \cdot P \cdot e \\ \Delta \dot{g} &= -\gamma \cdot \psi_2^T \cdot P \cdot e \end{aligned} \quad \text{Eqn.4.41}$$

This gives the two adaptive update equations, note that since g^o and d^o are constant, then $\Delta \dot{d} = \dot{d}$ and $\Delta \dot{g} = \dot{g}$:

$$\begin{aligned} \frac{dd}{dt} &= -\gamma \cdot \psi_1^T \cdot P \cdot e \\ \frac{dg}{dt} &= -\gamma \cdot \psi_2^T \cdot P \cdot e \end{aligned}$$

Eqn.4.42

Where: ψ_1 and ψ_2 are defined by equations 4.36 above. Note the similarity of the two adaptive update equations obtained for the MIMO system when compared with the SISO system given by equations 4.13. The only difference is that the MIMO system requires solving for P given Q and A_m , the Lyapunov Equation: $-Q = A_m^T \cdot P + P \cdot A_m$ which can be directly solved with MATLAB.

The setup is illustrated in figure 4.16 below, simulation results are given on the following page.

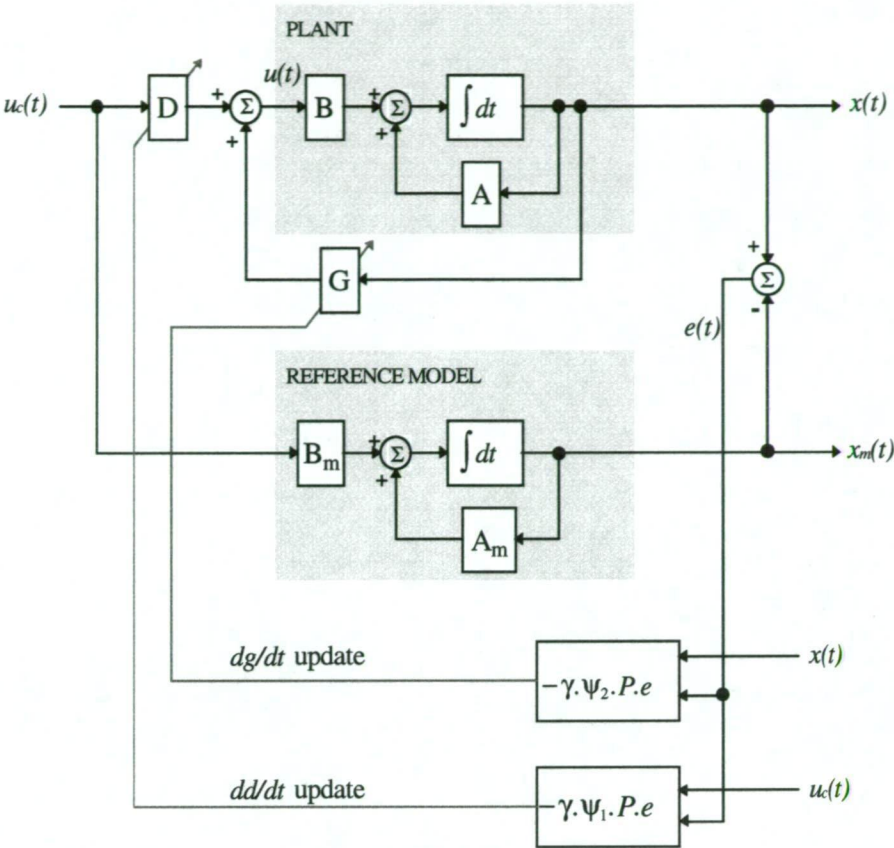


Fig.4.16
MRAC Scheme Using Lyapunov stability (MIMO system)

(ii) Simulation Setup:

For this simulation, the lateral aircraft model from chapter 1 is again used. The open loop dynamics is given by the matrices:

$$\mathbf{A} = \begin{bmatrix} -3.9330 & 0.1260 & -9.9900 & 0 \\ 0.0020 & -0.2350 & 5.6700 & 0 \\ 0.0262 & -0.9997 & -0.1960 & 0.0345 \\ 1.0000 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -45.8300 & -7.6400 \\ -0.9210 & -6.5100 \\ 0.0071 & 0 \\ 0 & 0 \end{bmatrix}$$

The reference model is chosen such that the closed loop response contains desired eigenvalues at the locations $1.5 \pm j1.5$ (roll mode) and $2.0 \pm j1.0$ (Dutch roll mode). This is essentially the eigenstructure assignment problem which was solved in chapter 3, where only the poles are assigned (i.e. a pole placement problem):

$$\mathbf{A}_m = \begin{bmatrix} -4.0000 & -0.0000 & 0.0000 & -5.0000 \\ 0.0234 & -2.8021 & 3.9448 & -0.0038 \\ 0.0262 & -1.0002 & -0.1979 & 0.0353 \\ 1.0000 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{B}_m = \begin{bmatrix} -22.9150 & -11.460 \\ -0.4605 & -9.765 \\ 0.0036 & 0 \\ 0 & 0 \end{bmatrix}$$

The values of G° and D° can be estimated only if we know in advance the matrices A and B , this information can be used only for verification purposes:

$$G^\circ = \begin{bmatrix} -0.002057691 & 0.064508221 & 0.268488544 & -0.111634509 \\ 0.003573822 & -0.403457113 & -0.302988201 & 0.015210675 \end{bmatrix}$$
$$D^\circ = \begin{bmatrix} 0.5 & 0.0 \\ 0.0 & 1.5 \end{bmatrix}$$

(iii) Simulation Results:

(a) *G-Matrix Convergence:* The graph below (Fig.4.17) shows the convergence of the feedback gain matrix G as a function of time, after 10,000 iterations, and computation: 14 MFP convergence is:

$$G = \begin{bmatrix} -0.00194 & 0.06410 & 0.26843 & -0.11162 \\ 0.00245 & -0.40076 & -0.30272 & 0.01498 \end{bmatrix}$$

This agrees with the value G° above. Note that the choice of gamma γ need not be a scalar value, but a matrix diagonal may also be used. In this simulation gamma was manually adjusted for each element of the G matrix until good convergence was obtained for each element of the G matrix.

$$\text{GAMMA } \gamma = 0.133 \cdot \text{diag}([1, 10, 10, 1, 10, 10, 10, 10])$$

Figure 4.17 below illustrates the convergence of the individual elements of the G matrix, convergence can be increased by increasing gamma.

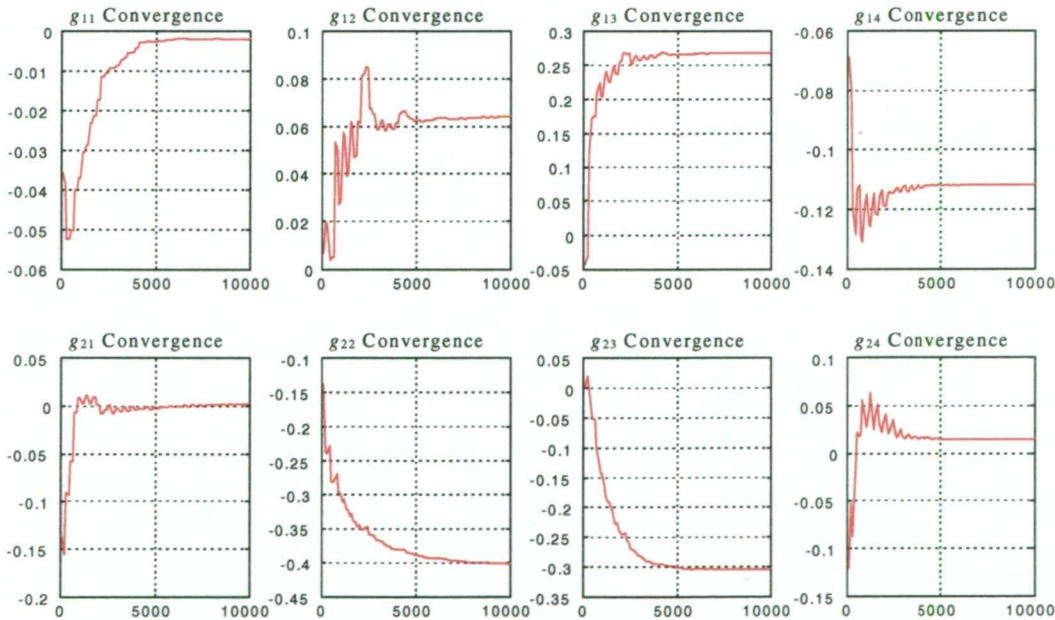


Fig.4.17
MRAC Scheme Using Lyapunov stability (MIMO system) G matrix Convergence

(b) **D-Matrix Convergence:** Figure 4.18 below shows the convergence of the feedback gain matrix D as a function of time, after 10,000 iterations, and computation: 14MFP, convergence is:

$$D = \begin{bmatrix} 0.50000 & 0.00000 \\ -0.00001 & 1.49999 \end{bmatrix}$$

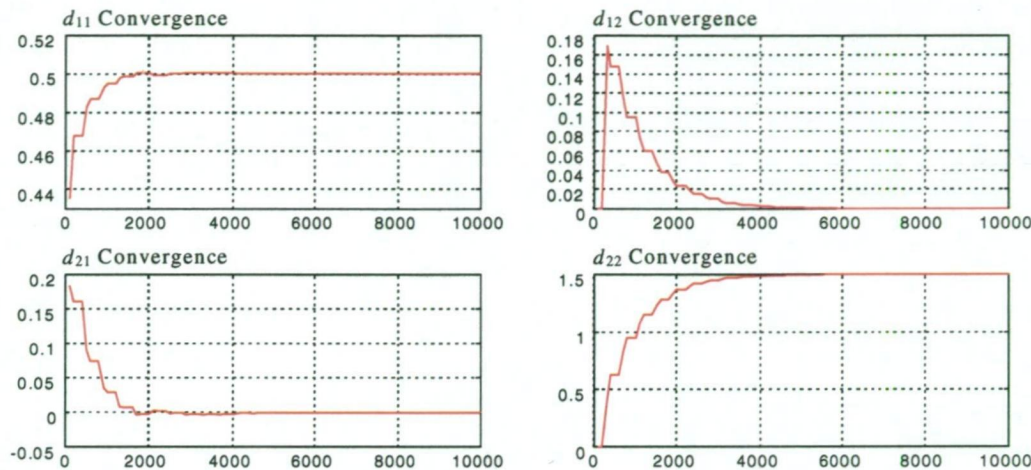


Fig.4.18
MRAC Scheme Using Lyapunov stability (MIMO system) D matrix Convergence

This agrees with the expected value of D^0 above. In this simulation, the choice of gamma was: $\gamma=3$, a scalar was used rather than a diagonal matrix.

(c) **Eigenvalues of $(A-B.G)$ Matrix During Convergence:** Figure 4.19 below illustrates the convergence of the closed loop eigenvalues:

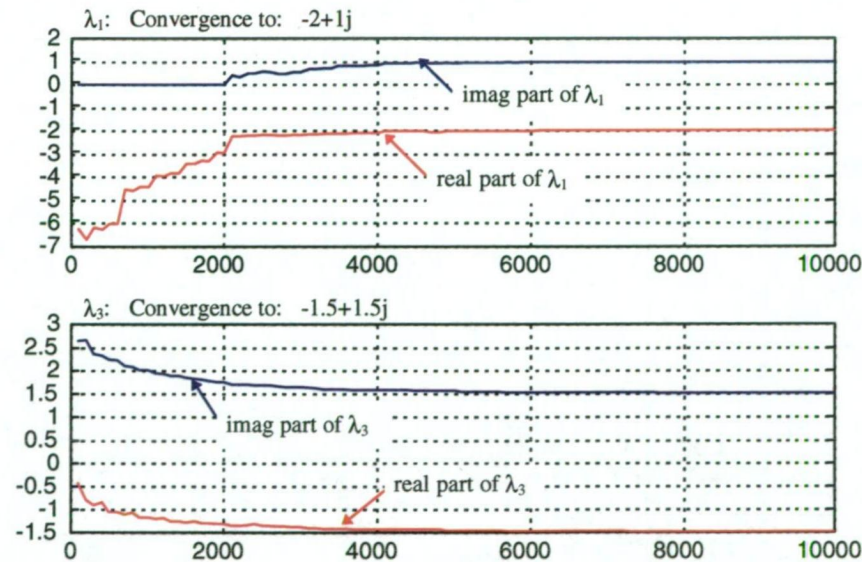


Fig.4.19
MRAC Scheme Using Lyapunov stability (MIMO system) Closed Loop Eigenvalues Convergence

During convergence, the eigenvalues of the closed loop system $(A-B.G)$ should remain stable, a plot of the four (two complex conjugate pairs) is shown below during convergence. It can be seen that the eigenvalues remain stable and that they reach the desired values of $\lambda_{1,2}=1.5\pm j1.5$ (roll mode) and $\lambda_{3,4}=2.0\pm j1.0$ (Dutch roll mode) after approximately 10,000 iterations. Legend: blue = real part of eigenvalue, and red = imaginary part of eigenvalue:

4.3.2 Simulation 4.5: MIT-Gradient Based Method:

(i) Theory:

In this section, parameter update rules for the MIMO system described in 4.3.1 are derived using the MIT-rule. Whilst the derivation is simpler compared to the Lyapunov method, the resulting parameter update equations are slightly more complex. Consider a MIMO system in state variable form as previously described:

Plant:	$\dot{x} = A \cdot x + B \cdot u$
Controller:	$u = D \cdot u_c - G \cdot x$
Closed Loop System:	$\dot{x} = (A - B \cdot G) \cdot x + B \cdot D \cdot u_c$
Reference Model:	$\dot{x}_m = A_m \cdot x_m + B_m \cdot u_c$

Eqn.4.43

Where: $x \in \mathbb{R}^n$, $x_m \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $u_c \in \mathbb{R}^m$. comparing equations 4.43, the plant and reference models are perfectly matched when:

$$\begin{aligned} A_m = (A - B \cdot G) &\Rightarrow G^o = B^{-1} \cdot (A - A_m) \\ B_m = B \cdot D &\Rightarrow D^o = B^{-1} \cdot B_m \end{aligned} \quad \text{Eqn.4.44}$$

Following the same arguments used in section 4.3.2, the parameter update equations are derived for the MIMO system. The MIT rule is:

$$\begin{aligned} \frac{dd}{dt} &= -\gamma \cdot \frac{\partial J}{\partial d} \\ \frac{dg}{dt} &= -\gamma \cdot \frac{\partial J}{\partial g} \end{aligned} \quad \text{Eqn.4.45}$$

Where d and g are vectorized forms (column vectors) of the D and G matrices respectively, not to be confused with the scalar variables used earlier in the SISO simulations. The cost function J is given by:

$$J(d, g) = \frac{1}{2} e^T(d, g) \cdot e(d, g) \quad \text{Eqn.4.46}$$

Computing partial derivatives, gives the following update equations:

$$\begin{aligned} \frac{dd}{dt} &= -\gamma \cdot e^T \cdot \left(\frac{\partial e}{\partial d} \right) \\ \frac{dg}{dt} &= -\gamma \cdot e^T \cdot \left(\frac{\partial e}{\partial g} \right) \end{aligned} \quad \text{Eqn.4.47}$$

Note that the partial derivatives in brackets represent Jacobian matrices. The error function is computed from:

$$e = x - x_m \quad \text{Eqn.4.48}$$

where x and x_m are obtained by taking Laplace transform of the last two equations of 4.43. The error is then given by:

$$e = (sI - (A - B \cdot G))^{-1} \cdot B \cdot D \cdot u_c - (sI - A_m)^{-1} \cdot B_m \cdot u_c \quad \text{Eqn.4.49}$$

Computing partial derivatives (Jacobians) with respect to the G and D matrices gives:

$$\begin{aligned} \frac{\partial e}{\partial D} &= -\frac{B}{sI - (A - B \cdot G)} \cdot u_c \\ \frac{\partial e}{\partial G} &= -\frac{B}{sI - (A - B \cdot G)} \cdot x \end{aligned} \quad \text{Eqn.4.50}$$

Again, since the A and B matrices are unknown, the approximation is made that: $B \approx B_m$ and $(A - B \cdot G) \approx A_m$. The parameter update equations then become:

$$\begin{aligned} \frac{dd}{dt} &= \gamma \cdot e^T \cdot \left(\frac{B_m}{sI - A_m} \cdot u_c \right) \\ \frac{dg}{dt} &= \gamma \cdot e^T \cdot \left(\frac{B_m}{sI - A_m} \cdot x \right) \end{aligned}$$

Eqn.4.51

Implementation Issues: The final implementation is illustrated in figure 4.20 below, note that the actual MATLAB implementation is split into two parts, the computation of the Jacobian matrices: $\partial e / \partial G$ and $\partial e / \partial D$ equations 4.50, and the computation the parameter update equations from 4.47.

Note also that equations 4.51 represent a first order dynamical system which also requires numerical integration. This results in a second order dynamical system. The most practical way to handle this problem is to simply re-write equations 4.50 as a first order differential equation thus:

$$\begin{aligned}\frac{\partial \dot{e}}{\partial D} &= A_m \cdot \left(\frac{\partial e}{\partial D} \right) - B_m \cdot (\delta_{ij}) \cdot u_c \\ \frac{\partial \dot{e}}{\partial G} &= A_m \cdot \left(\frac{\partial e}{\partial G} \right) - B_m \cdot (\delta_{ij}) \cdot x\end{aligned}\tag{Eqn.4.52}$$

where the terms δ_{ij} in the first equation is 1 for the ij entry of the D matrix, and zero for all other entries, similarly for the second expression applying to the G matrix. The simulation then would proceed as follows, for instance for the G matrix:

Algorithm: G matrix:

step-1: first compute

$$\frac{\partial \dot{e}}{\partial G} = A_m \cdot \left(\frac{\partial e}{\partial G} \right) - B_m \cdot (\delta_{ij}) \cdot x\tag{Eqn.4.53.a}$$

step-2: update the Jacobian $\partial e / \partial G$ at the k^{th} time step by numerical integration:

$$\left(\frac{\partial e}{\partial G} \right)_{k+1} = \left(\frac{\partial e}{\partial G} \right)_k + \Delta T \cdot \left(\frac{\partial \dot{e}}{\partial G} \right)\tag{Eqn.4.53.b}$$

step-3: compute the G update equation:

$$\frac{dG}{dt} = -\gamma \cdot \left(\frac{\partial e}{\partial G} \right)^T \cdot e\tag{Eqn.4.53.c}$$

step-4: update the G matrix by numerical integration:

$$G_{k+1} = G_k + \Delta T \cdot \left(\frac{dG}{dt} \right)\tag{Eqn.4.53.d}$$

The same applies for the D matrix. As can be seen, the actual implementation is slightly more complex compared to the Lyapunov method, however the derivation is much simpler. The complete simulation setup is illustrated in figure 4.20. Simulation results are provided below. Again, any cost function could have been used instead of equation 4.46, which must be a function of the error $e(t)$.

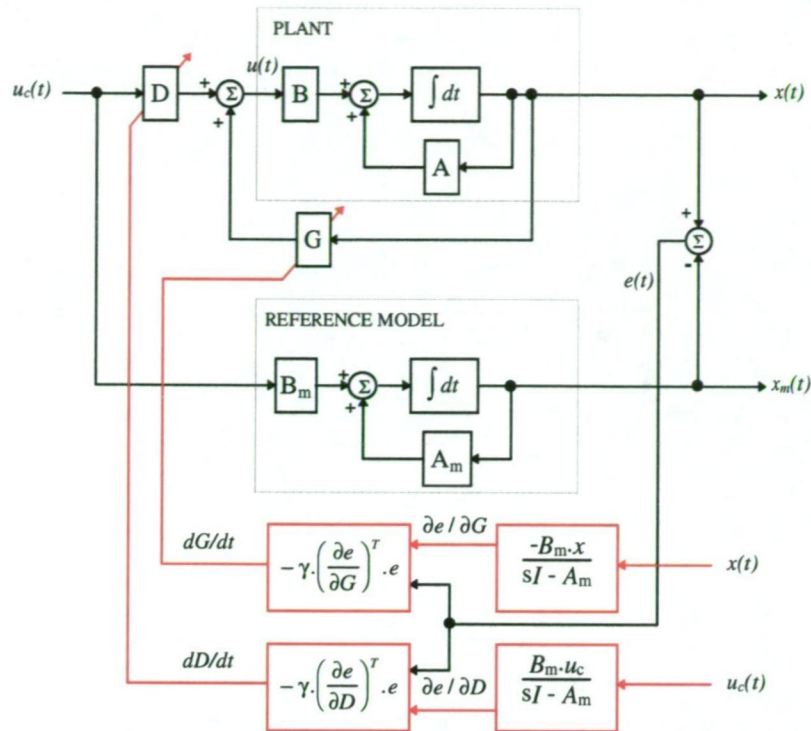


Fig.4.20
MRAC Scheme Using MIT rule (MIMO system)

(ii) Simulation Results:

(a) **G-Matrix Convergence:** The graph below shows the convergence of the feedback gain matrix G as a function of time for three different values of gamma. Convergence is within the first 20,000 iterations. The initial value of the G matrix at $t=0$ is set to zero:

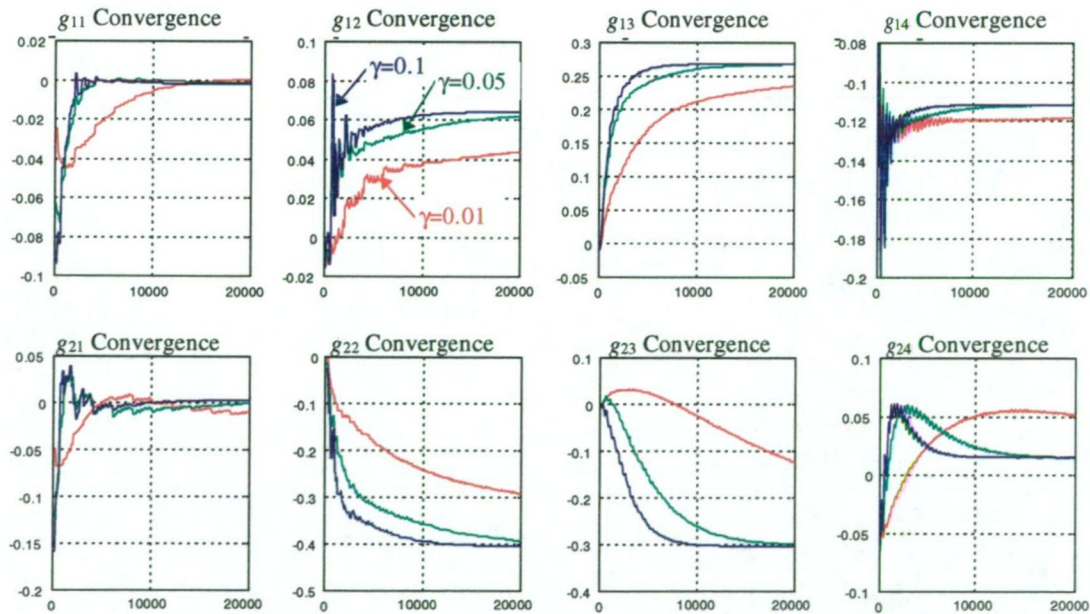


Fig.4.21
MRAC Scheme Using MIT rule (MIMO system) G matrix Convergence

Convergence of the G matrix after 20,000 iterations for the three different values of gamma (gamma is a diagonal matrix thus: $\gamma \times \text{diag}[1,2,1,1,2,2,1]$)

G^0 :				
	-0.0021	0.0645	0.2685	-0.1116
	0.0036	-0.4035	-0.3030	0.0152
G matrix convergence, with $\gamma=0.01$				
	0.0002	0.0435	0.2350	-0.1185
	-0.0099	-0.2906	-0.1240	0.0517
G matrix convergence with $\gamma=0.05$				
	-0.0014	0.0621	0.2677	-0.1117
	-0.0001	-0.3913	-0.2986	0.0157
G matrix convergence with $\gamma=0.10$				
	-0.0020	0.0644	0.2685	-0.1116
	0.0034	-0.4029	-0.3028	0.0152

The diagonal entries were chosen by manually tuning each entry until reasonably good convergence was obtained for each G matrix element.

(b) D-Matrix Convergence: Similarly, for the D matrix, convergence is plotted in figure 4.22 below for three different values of gamma, in this instance gamma is a simply a scalar and not a diagonal matrix.

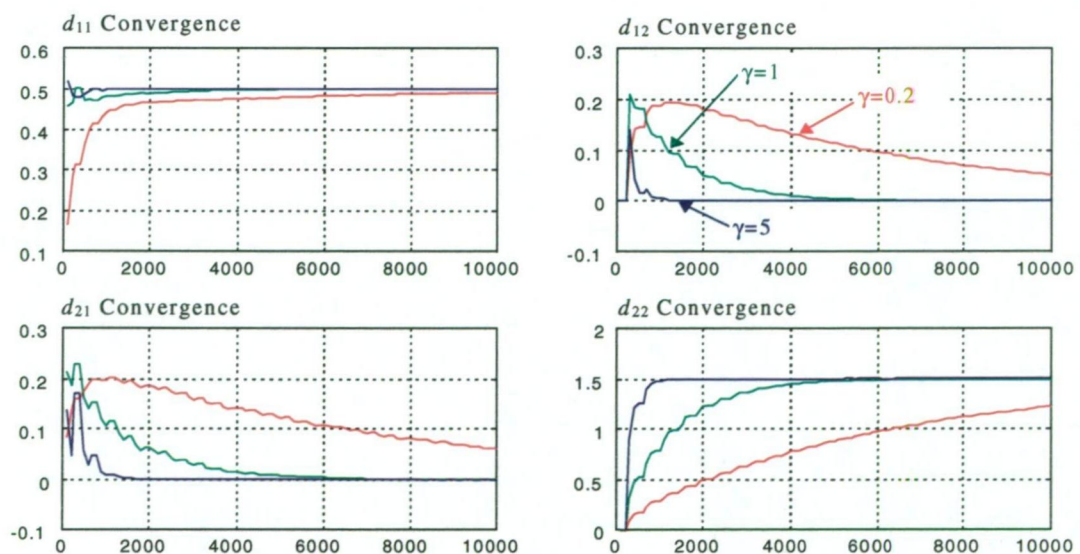


Fig.4.22
MRAC Scheme Using MIT rule (MIMO system) D matrix Convergence

After 10,000 iterations the actual value of D matrix for the three gammas is:

D^o :	0.5000	0.0000
	0.0000	1.5000
D matrix Convergence: $\gamma=0.2$	0.4889	0.0499
	0.0613	1.2237
D matrix Convergence: $\gamma=1$	0.5000	0.0001
	0.0002	1.4996
D matrix Convergence: $\gamma=5$	0.5000	-0.0000
	-0.0000	1.5000

As can be seen, the D matrix also converges to the correct value of D^o , however convergence is also slow, but faster than the G matrix.

(c) **Eigenvalues of (A-B.G) Matrix During Convergence:** The eigenvalues of the closed loop (A-B.G) matrix are plotted during convergence, it can be seen that the closed loop system remains stable, and that they converge to the required closed loop eigenvalues of the reference model.

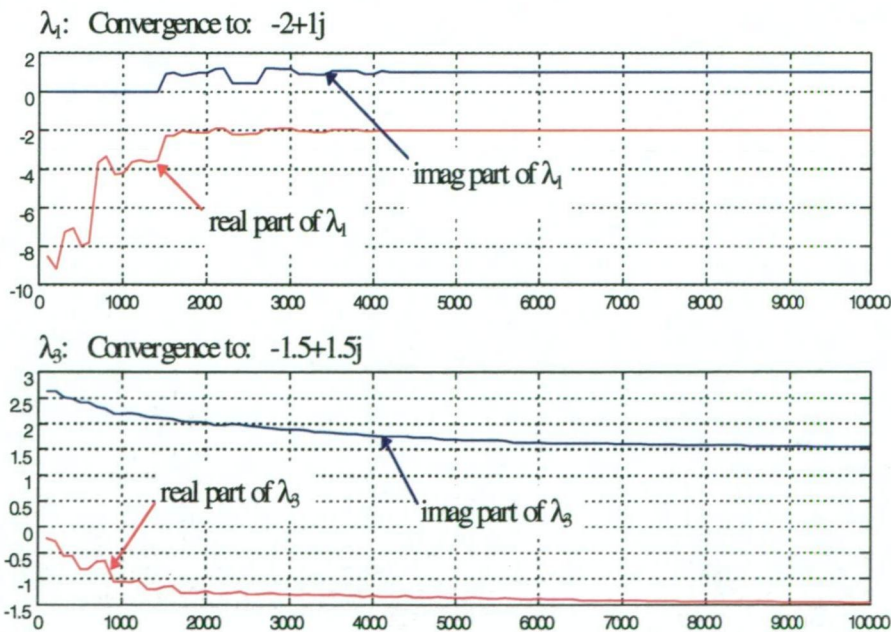


Fig.4.23
MRAC Scheme Using MIT rule (MIMO system) Closed Loop Eigenvalues Convergence

4.3.3 Simulation 4.6: Hybrid Genetic Algorithms:

In this section, the same MIMO problem is solved using hybrid genetic algorithms. Simulation results compare: (i) conventional genetic algorithms, (ii) genetic algorithms + simulated annealing and (iii) genetic algorithms + greedy search. Computational effort and convergence rates are compared. The final configuration is illustrated in figure 4.24 below.

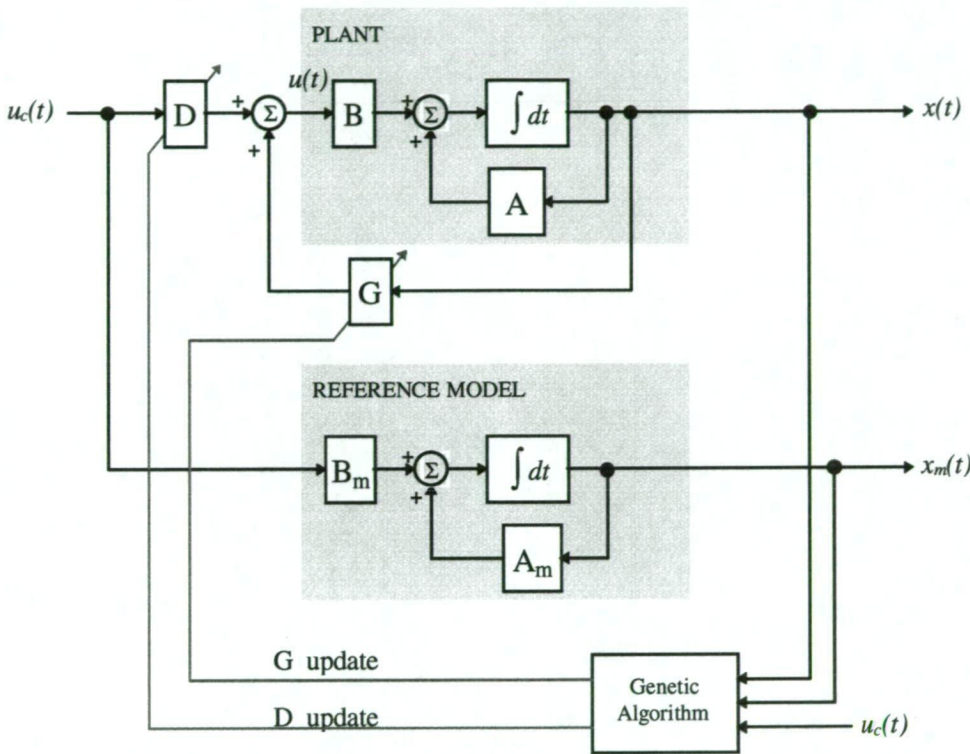


Fig.4.24
MRAC Scheme Using Genetic Algorithms (MIMO system)

(i) **Genetic Algorithms:** Again, the fitness function is chosen to be the inverse of the error function. The error is computed as the RMS value of the output difference $x(t)-x_m(t)$, refer to equation 4.28. The chromosomal representation for this problem is illustrated in figure 4.25 below. The controller parameters to solve for are: D , and G matrices. The fitness is the inverse of the error function: $fitness = 1/error$.

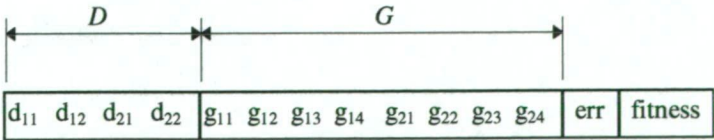


Fig.4.25
MRAC Scheme Using the Genetic Algorithm (MIMO system)

This problem represents a multidimensional unconstrained optimization problem consisting of 12 parameters instead of only 2 compared to the equivalent SISO problem from simulation 4.3. This problem is solved essentially off-line, thus from measurements of x and x_m , the genetic algorithm computes the D , and G values which minimize the error function 4.28.

Simulation results are given below, see simulation 4.6. For this simulation the genetic algorithm setup is as follows: Population: 30, maximum generations=300, data samples=100, crossover probability $P_c=0.6$, and mutation probability $P_m=0.1$, Selection scheme: binary tournament selection.

G matrix convergence: Table 4.4 below summarizes the convergence results obtained using the genetic algorithm. Because genetic algorithms are a probabilistic search based algorithms, the simulation was conducted ten times to observe the stochastic nature of the convergence, results are tabulated below, the first simulation⁽¹⁾ is also plotted on the following page (Fig.4.23).

	g_{11}	g_{12}	g_{13}	g_{14}	g_{21}	g_{22}	g_{23}	g_{24}	error	fitness	norm($G-G^o$)
	-0.0020	0.0645	0.2684	-0.1116	0.0035	-0.4034	-0.3029	0.0152			
300	-0.0023	0.0649	0.2672	-0.1115	0.0045	-0.4047	-0.2972	0.0145	0.0038	263	0.0062 ⁽¹⁾
300	-0.0025	0.0656	0.2681	-0.1110	0.0056	-0.4070	-0.2985	0.0136	0.0113	88	0.0064
300	-0.0022	0.0648	0.2690	-0.1105	0.0044	-0.4043	-0.3058	0.0108	0.0086	116	0.0055
300	-0.0021	0.0646	0.2683	-0.1118	0.0033	-0.4031	-0.3023	0.0163	0.0036	274	0.0014
300	-0.0021	0.0645	0.2685	-0.1117	0.0036	-0.4036	-0.3030	0.0154	0.0006	1810	0.0002
400	-0.0028	0.0660	0.2674	-0.1132	0.0067	-0.4094	-0.2977	0.0209	0.0100	100	0.0105
400	-0.0023	0.0648	0.2675	-0.1123	0.0048	-0.4042	-0.3007	0.0182	0.0074	134	0.0042
400	-0.0021	0.0646	0.2674	-0.1112	0.0041	-0.4040	-0.2978	0.0130	0.0050	198	0.0058
500	-0.0022	0.0647	0.2686	-0.1119	0.0041	-0.4040	-0.3038	0.0160	0.0036	280	0.0014
500	-0.0020	0.0647	0.2682	-0.1118	0.0038	-0.4048	-0.3016	0.0167	0.0048	208	0.0025

Table 4.4
 G matrix convergence for 300,400,500 generations

The target value of G^o is listed in red in the first row table 4.4. The graph below shows the convergence of the feedback gain matrix G as a function of time. Convergence is initially very rapid (within the first 100 generations) and then substantially reduced due to the population reaching steady state.

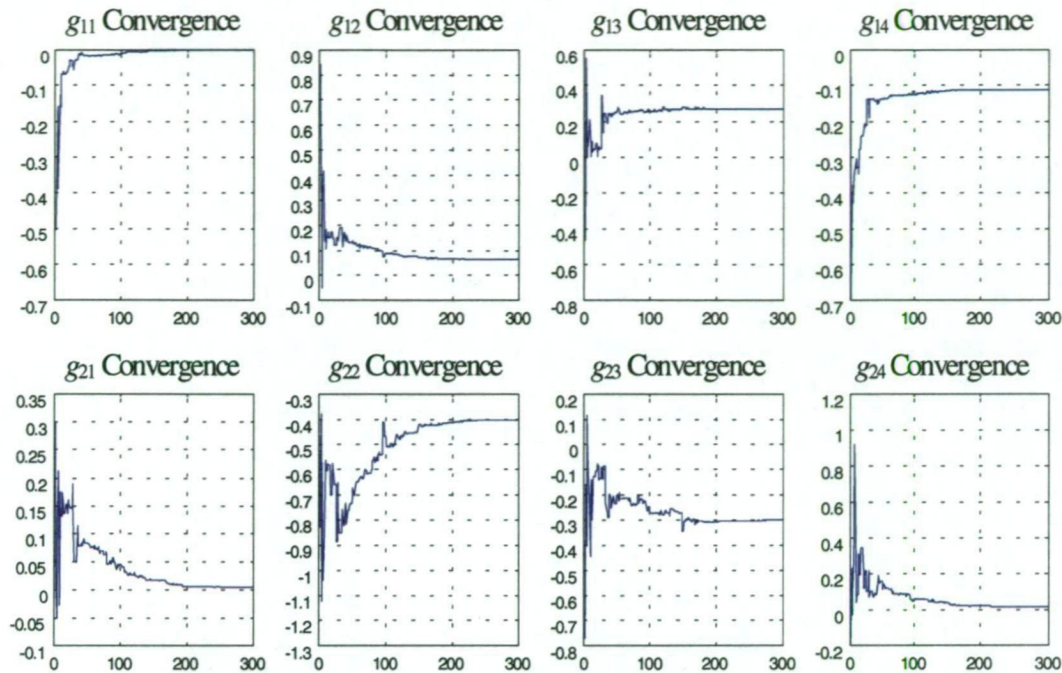


Fig.4.26
MRAC Scheme Using Genetic Algorithms (MIMO system) *G* matrix Convergence

Computational effort: 300 generations \approx approximately 300 MFP. From table 4.4, results from the first simulation (first row 1) are illustrated in figure 4.26 above. Whilst the genetic algorithm is capable of converging near the solution, the final convergence is generally slow. To overcome this, the two hybrid methods discussed earlier are used in the next set of simulations (ii and iii). Figure 4.27 below shows the error convergence as a function of generation for the simulation above:

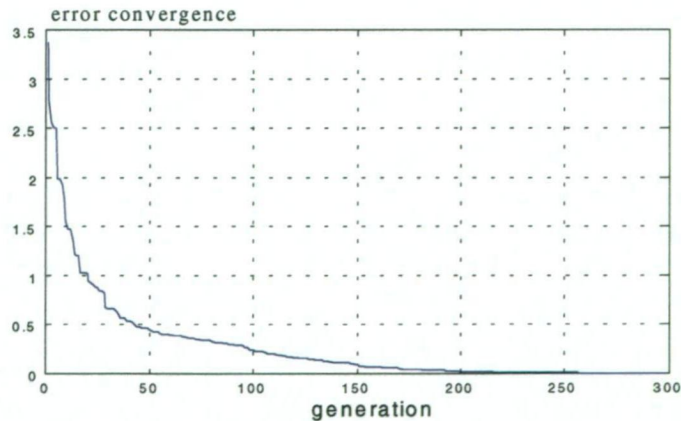


Fig.4.27
Error convergence

D matrix convergence: Table 4.5 below summarizes the convergence results obtained for the *D* matrix. Again the simulation ran ten times to illustrate the stochastic nature of the convergence:

	d_{11}	d_{12}	d_{21}	d_{22}	<i>error</i>	<i>fitness</i>
	0.5000	0.0000	0.0000	1.5000		
150	0.5003	0.0001	-0.0008	1.4992	0.0013	764
150	0.5001	0.0002	-0.0006	1.4992	0.0013	783
150	0.5007	0.0007	-0.0025	1.4964	0.0035	288
150	0.5027	0.0007	-0.0113	1.4980	0.0101	98
150	0.4963	0.0028	0.0152	1.4879	0.0160	62
150	0.4994	0.0001	0.0023	1.4998	0.0023	441
150	0.4978	0.0014	0.0090	1.4941	0.0103	96
150	0.4994	0.0008	0.0043	1.4966	0.0051	197
150	0.4999	-0.0006	-0.0009	1.5018	0.0026	386
150	0.5002	0.0011	-0.0016	1.4955	0.0048	210

Table 4.5
D matrix convergence after 150 Generations

The *D* matrix converges more rapidly than the *G* matrix, due to the fewer parameters to solve for. Figure 4.28 below illustrates the convergence of the *D* matrix.

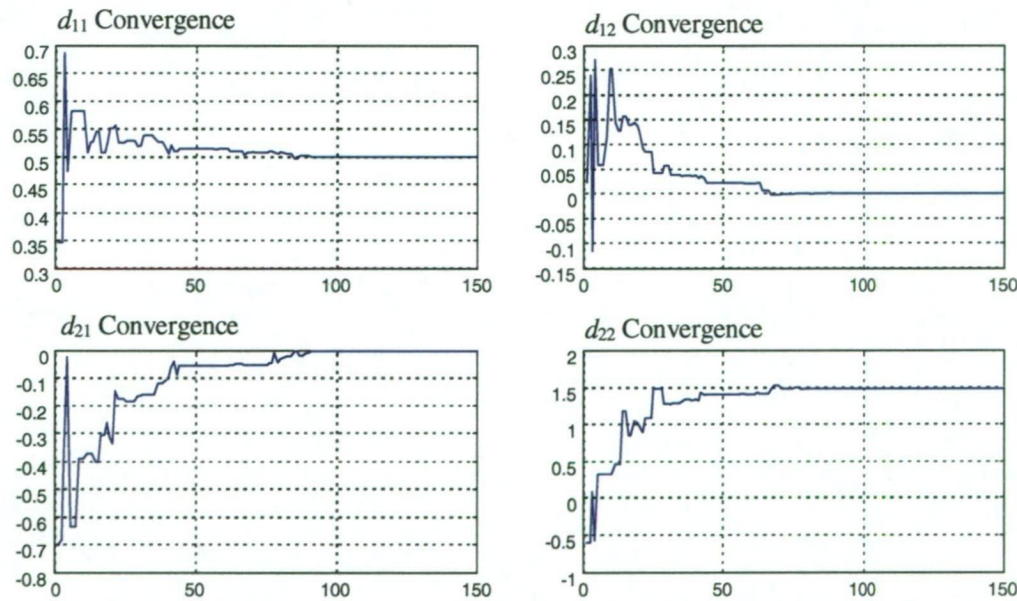


Fig.4.28
MRAC Scheme Using Genetic Algorithms (MIMO system) *D* matrix Convergence

After 150 generations, the *D* matrix has converged. Again, convergence is initially very rapid (first 50 generations), and thereafter considerably reduced.

(ii) **Genetic Algorithms + Simulated Annealing:** The search vector is similarly defined as in conventional genetic algorithms without the error and fitness entries thus:

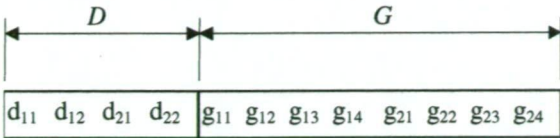


Fig.4.29
MRAC Scheme Using the Hybrid Genetic Algorithm + Simulated Annealing

The temperature annealing schedule is defined identically as in the SISO case (see 4.2.3 part ii). Results for this simulation are summarized below.

G matrix convergence: Table 4.6 below summarizes the G matrix convergence for 10 simulation runs using 100 samples and 3000 SA iterations:

g11:	g12:	g13:	g14:	g21:	g22:	g23:	g24:	error	G-Go:	MFP
-0.0021	0.0645	0.2685	-0.1116	0.0036	-0.4034	-0.3029	0.0152			
-0.0021	0.0645	0.2685	-0.1116	0.0036	-0.4034	-0.3029	0.0153	0.00012	0.00015	737
-0.0020	0.0645	0.2684	-0.1117	0.0035	-0.4033	-0.3024	0.0154	0.00032	0.00061	737
-0.0021	0.0645	0.2685	-0.1116	0.0036	-0.4034	-0.3027	0.0152	0.00022	0.00026	737
-0.0021	0.0645	0.2685	-0.1116	0.0036	-0.4034	-0.3028	0.0152	0.00014	0.00019	737
-0.0021	0.0645	0.2685	-0.1117	0.0036	-0.4035	-0.3029	0.0154	0.00026	0.00025	737
-0.0021	0.0645	0.2685	-0.1116	0.0036	-0.4035	-0.3032	0.0152	0.00025	0.00022	737
-0.0020	0.0645	0.2684	-0.1117	0.0035	-0.4034	-0.3024	0.0154	0.00021	0.00061	737
-0.0021	0.0645	0.2684	-0.1117	0.0036	-0.4034	-0.3024	0.0154	0.00035	0.00062	737
-0.0021	0.0645	0.2685	-0.1116	0.0036	-0.4035	-0.3030	0.0152	0.00006	0.00005	737
-0.0020	0.0645	0.2685	-0.1117	0.0035	-0.4035	-0.3027	0.0154	0.00023	0.00038	737

Table 4.6
G matrix Convergence after 3000 SA Iterations

A typical matrix convergence plot is illustrated below:

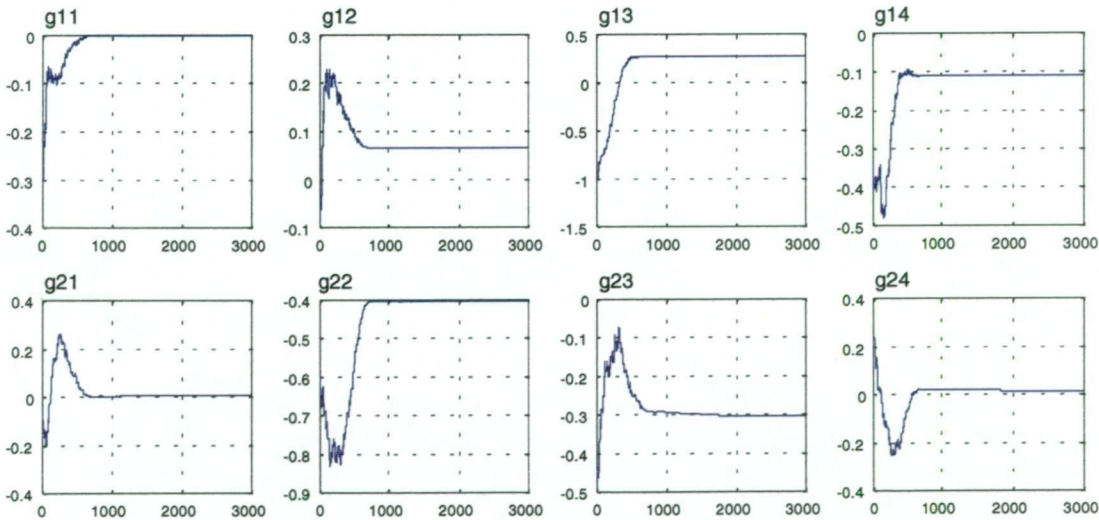


Fig.4.30
MRAC Scheme Using Genetic Algorithms (MIMO system) G matrix Convergence

D matrix convergence: Table 4.7 below summarizes the G matrix convergence for 10 simulation runs:

d11	d12	d21	d22	Error:	D-Do:	MFP:
0.5000	0.0000	0.0000	1.5000			
0.5000	0.0000	0.0001	1.4999	0.00011	0.00017	74
0.5000	-0.0001	-0.0002	1.5002	0.00026	0.00026	74
0.5000	-0.0001	-0.0002	1.5000	0.00025	0.00023	74
0.5000	-0.0001	-0.0001	1.5002	0.00025	0.00024	74
0.5000	0.0000	-0.0000	1.4997	0.00017	0.00026	74
0.5000	0.0000	0.0000	1.4998	0.00011	0.00017	74
0.4999	0.0000	0.0002	1.4999	0.00017	0.00025	74
0.5001	-0.0000	-0.0002	1.5001	0.00018	0.00024	74
0.5000	-0.0000	0.0002	1.5000	0.00026	0.00017	74
0.5000	-0.0000	-0.0002	1.5000	0.00018	0.00018	74

Table 4.7
D matrix Convergence

Typical matrix convergence plots, convergence is within the first 100 iterations.

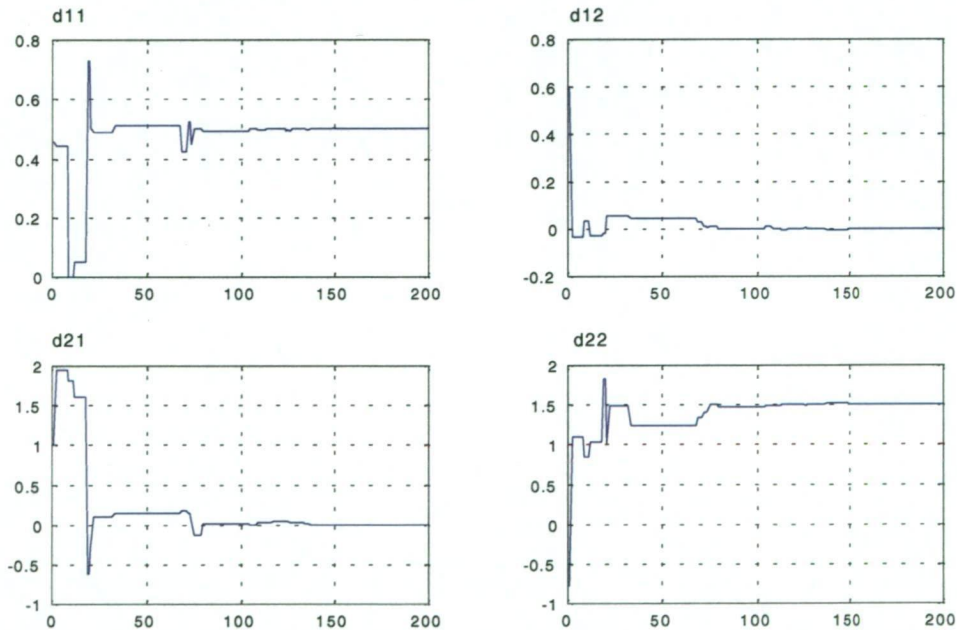


Fig.4.31
MRAC Scheme Using Genetic Algorithms (MIMO system) D matrix Convergence

This hybrid genetic algorithm converges considerably more rapidly when compared with the conventional genetic algorithm in (i).

(iii) **Genetic Algorithms + Greedy search:** The search vector is similarly defined as in simulated annealing, see figure 4.29. Below, results of the G and D matrix convergence are given.

G matrix convergence: Table 4.8 below summarizes the *G* matrix convergence for 10 simulation runs using 100 samples, and 3000 greedy search iterations:

g11:	g12:	g13:	g14:	g21:	g22:	g23:	g24:	Error:	G-Go:	MFP:
-0.0021	0.0645	0.2685	-0.1116	0.0036	-0.4034	-0.3029	0.0152			
-0.0020	0.0645	0.2684	-0.1118	0.0035	-0.4032	-0.3023	0.0164	0.00067	0.00141	231
-0.0021	0.0645	0.2685	-0.1115	0.0036	-0.4033	-0.3033	0.0145	0.00034	0.00079	231
-0.0021	0.0645	0.2684	-0.1117	0.0036	-0.4032	-0.3027	0.0158	0.00032	0.00069	230
-0.0021	0.0645	0.2684	-0.1116	0.0036	-0.4034	-0.3026	0.0152	0.00025	0.00045	230
-0.0021	0.0645	0.2684	-0.1116	0.0036	-0.4034	-0.3026	0.0153	0.00024	0.00041	230
-0.0021	0.0645	0.2684	-0.1116	0.0036	-0.4035	-0.3028	0.0152	0.00020	0.00024	229
-0.0020	0.0645	0.2684	-0.1117	0.0035	-0.4033	-0.3026	0.0154	0.00018	0.00046	231
-0.0021	0.0645	0.2684	-0.1116	0.0036	-0.4035	-0.3028	0.0153	0.00016	0.00022	229
-0.0021	0.0645	0.2685	-0.1116	0.0036	-0.4035	-0.3030	0.0152	0.00001	0.00001	232
-0.0021	0.0645	0.2685	-0.1116	0.0036	-0.4035	-0.3030	0.0152	0.00001	0.00000	231

Table 4.8
G matrix convergence

The figure below illustrates typical convergence properties of the hybrid algorithm:

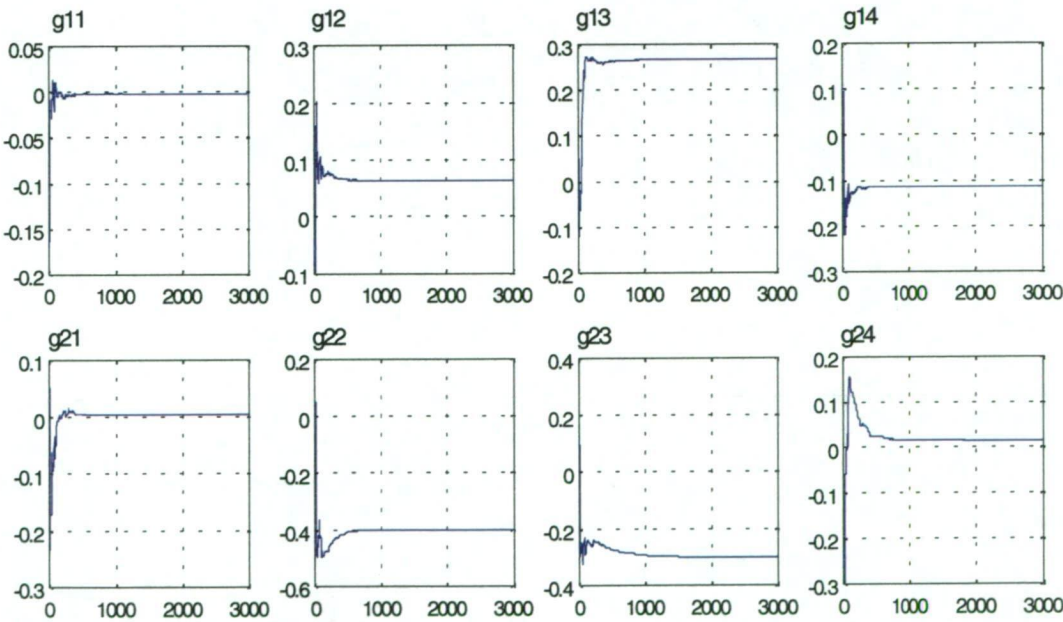


Fig.4.32

MRAC Scheme Using Hybrid Genetic Algorithms + Greedy Search(MIMO system) *G* matrix Convergence

D matrix convergence: Table 4.9 below summarizes the *D* matrix convergence for 10 simulation runs with 100 samples and 220 iterations. The *D* matrix converges very quickly compared with both conventional GA and hybrid GA+simulated annealing methods.

d11	d12	d21	d22	Error:	D-Do:	MFP:
0.5000	0.0000	0.0000	1.5000			
0.5000	0.0000	0.0002	1.5001	5334	0.00024	17
0.5000	0.0000	-0.0002	1.5000	6136	0.00025	17
0.5000	0.0000	-0.0001	1.4998	6330	0.00025	17
0.5000	0.0000	0.0000	1.5000	29718	0.00004	17
0.5000	-0.0000	0.0000	1.5000	30531	0.00003	18
0.5000	0.0000	0.0000	1.5000	46518	0.00004	18
0.5000	-0.0000	-0.0000	1.5000	70323	0.00002	17
0.5000	-0.0000	-0.0000	1.5000	126538	0.00001	18

Table 4.9
D matrix convergence

The figure below illustrates typical convergence properties of the hybrid algorithm, note that convergence is within the first 200 greedy search iterations.

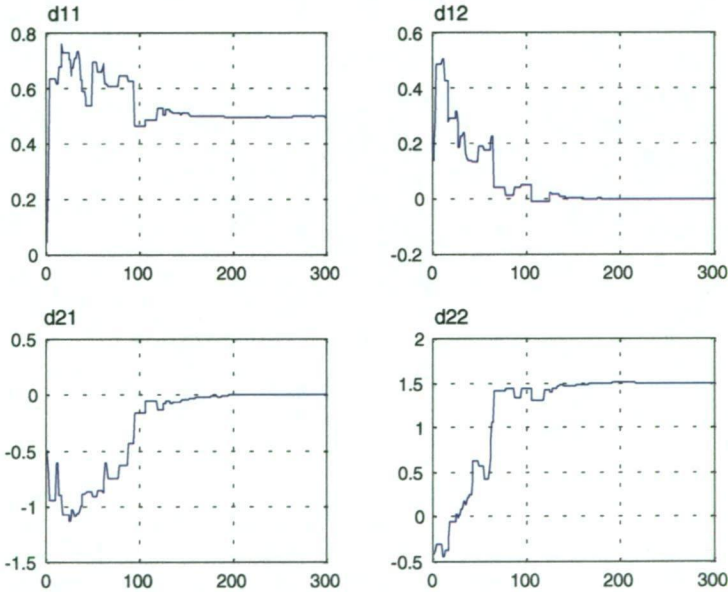


Fig.4.33
MRAC Scheme Using Hybrid Genetic Algorithms + Greedy Search(MIMO system) D matrix Convergence

4.3.4 Convergence Rates:

Greedy algorithms give on average the fastest convergence when compared with both conventional GA and hybrid SA methods. However the convergence of the greedy algorithm is not as consistent as the GA. In many cases the greedy algorithms can converge very rapidly, and in others very slowly. This depends on the initial solution, if the initial value is near the optimum, then convergence is very rapid. One advantage of the genetic search algorithms is that we do not need to specify the gamma parameter as used by both the Lyapunov and MIT rule methods.

The following figures illustrate the convergence properties of the five methods compared:

D matrix convergence:

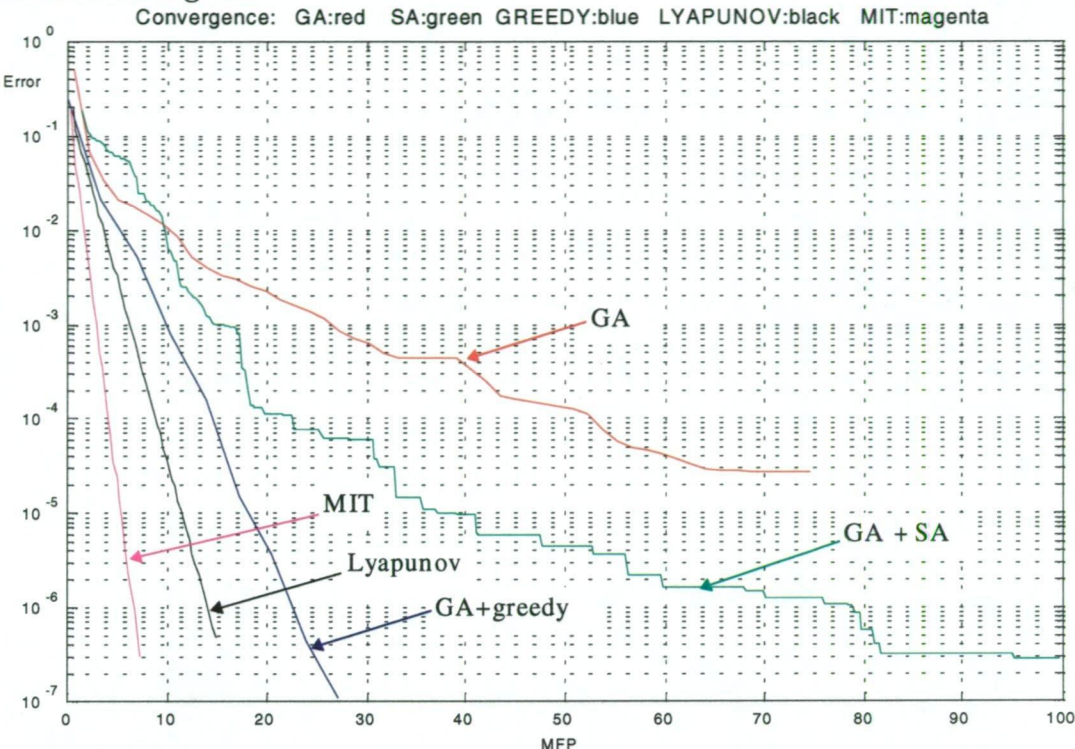


Fig.4.34

G matrix convergence:

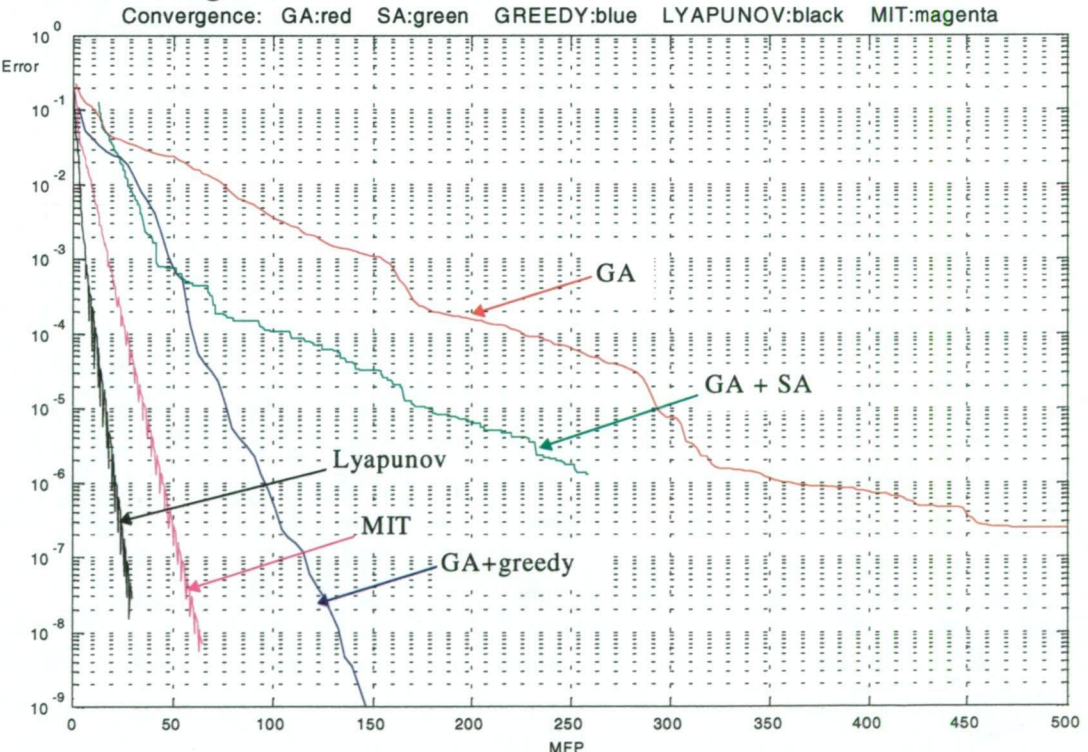


Fig.4.35

4.4 Discussion and Conclusion:

From these simulations, we can see that hybrid genetic algorithms can easily be applied to adaptive control applications and convergence is generally very rapid. For simple SISO systems, convergence results within 50 generations. For multivariable 2th order MIMO systems, the genetic algorithm converges within 200 generations. However, despite the rapid convergence, the computational effort required at each generation is approximately 1 MFP, compared with only 20-30 operations at each time step with conventional Lyapunov or MIT rule based methods. Furthermore, there is no guarantee on the rate of convergence of a hybrid genetic algorithm. This is a critical issue if genetic algorithms are to be accepted as an alternative method of generating parameter update rules in adaptive control applications. On the other hand however, genetic algorithms have fewer restrictions and can also be applied to nonlinear systems. Some key differences are summarized below.

Key Points and Differences:

1. GA is not really recursive, whereas both the Lyapunov method and MIT rule generate new parameters at each sample interval with only the current measurement, the genetic algorithm requires knowledge of past historical data as well as current data. This means that its response is delayed if an abrupt change occurs in the plant A and B matrices. Whilst the genetic algorithm may not be recursive, it can however still operate online.
2. Because the GA works with historical data, it is more immune to the presence of noise in the current measurement. GA method puts equal weights on all samples, whereas the MIT and Lyapunov methods place more emphasis on current data.
3. Constrained problems can also be easily dealt with using genetic algorithms, but more difficult to solve using the conventional MIT and lyapunov methods.
4. Higher computational effort is required with genetic algorithms, typically 20 times or more compared with conventional methods. Hybrid genetic algorithms give comparable performance, in particular the hybrid GA + greedy search converge very rapidly.

5. From the results, we can see that genetic algorithms work well, and have fewer restrictions when compared with more traditional methods such as the MIT gradient based rule and lyapunov stability theory. The GA can easily be extended to more unconventional controller configurations without any change to the genetic algorithm.
6. Convergence is generally faster than our results indicate because not all parameters of the A and B matrices change simultaneously, but only a few matrix parameters change for instance: payload mass of robotic manipulator. This can be encoded into the chromosome, and search conducted for several rather than all matrix parameters.
7. Genetic algorithms are easily extended to solving nonlinear MRAC systems, with any controller structure e.g.: neural networks, fuzzy logic, linear dynamic compensators etc.

Future Work:

Much work needs to be done in order for GA to be accepted in adaptive and MRAC control applications. Currently there are very few papers which address the application of GA to MRAC control. This chapter addresses only the basic concepts of MRAC and attempts to obtain some preliminary results. Some future work would involve:

1. Use measurement feedback instead of full state feedback, and a dynamic compensator (see section 3.2).
1. Modify the GA to act like an online recursive algorithm, rather than searching the entire solution space, use the previous results to generate a narrower search range which would improve convergence. This type of online genetic algorithm can be a topic of future research and is beyond the scope of this thesis.
2. Apply GA to indirect method of MRAC. Only the direct method was used in the simulation, with output feedback instead of full state feedback, including gaussian noise in the output.
3. Applications of GA to nonlinear systems with robustness properties using variable structure model reference adaptive control. Variable structure MRAC is currently an active area of research.
4. Model reference adaptive control problem can also be formulated in a robust control framework, for instance the model matching problem may be written as:

$$error = \min \|T_1 - T_2 \cdot Q\|_{\infty}$$

here T_1 is a model (reference model) and T_2 is the plant, Q is a cascade controller such that the error of the transfer functions is minimized. For linear systems, this may be solved using the Nevanlinna's algorithm. Again this can be easily handled with GA, in which the fitness function can be the inverse of the error.

5. Applications of radial basis function networks for nonlinear systems. For instance consider the following setup, in the figure below, a radial basis function is used to control a nonlinear plant. A linear model is used as a reference. Again, training using genetic algorithms as in chapter 2 can be applied to this problem. We could also use fuzzy logic control to replace the neural network. Genetic algorithms can be used for adjusting the fuzzy rules. For instance see [18].

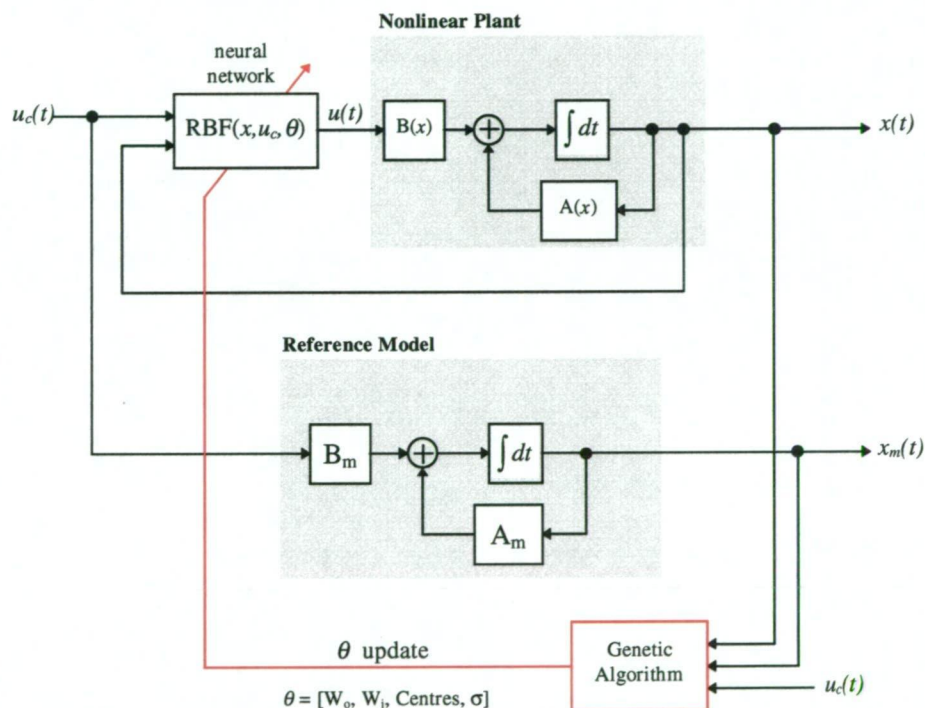


Fig.4.36

MRAC Control for nonlinear systems with neurocontrol

An interesting paper dealing with nonlinear reconfigurable adaptive flight control using genetic algorithms [19], using neural networks/dynamic inversion [20], more general papers [21], adaptive PID control and genetic algorithms [22]. In summary, genetic algorithms can be applied to model reference adaptive control, resulting in good convergence properties. There are clearly many applications including nonlinear neurocontrol and others such as variable structure adaptive control, and fuzzy variable structure control, which offer new and interesting possibilities for research.

4.5 References and Further Reading

- [1] W. S. Levine
Flight Control of Piloted Aircraft. /F-16 Aircraft
The Control Handbook, IEEE CRC Press 1996, Ch.54 pp847-858
- [2] K.J.Astrom,
Theory and Applications of Adaptive Control, A survey
Automatica, Vol.19, No. 5, pp.471-486, 1983
- [3] G.Feng, R.Lozano
Adaptive Control Systems
Newnes, 1999
- [4] M. Bodson, J.E. Groszkiewicz
Multivariable Adaptive Algorithms for Reconfigurable Flight Control
IEEE Transactions on Control System Techonology, Vol.5, No.2, pp.217-229, March 1997
- [5] G. Ambrosino, G.Celentano, F.Garofalo
Variable Structure Model Reference Adaptive Control Systems
International Journal of Control, Vol.39, No.6, pp.1339-1349, 1984
- [6] L. Hsu, R.R.Costa
Variable Structure Model Reference Adaptive Control Using only Input and Output Measurements
International Journal of Control, Vol.49, No.2, pp.399-416, 1989
- [7] P.S.Maybeck, R.D.Stevens
Reconfigurable Flight Control Via Multiple Model Adaptive Control Methods
IEEE Transactions on Aerospace and Electronic Systems, Vol.27, No.3, pp.470-479, May 1991
- [8] S.N. Singh
Nonlinear Adaptive Attitude Control of Spacecraft
IEEE Transactions on Aerospace and Electronic Systems, Vol.AES-23, No.3, pp.470-479, May 1987
- [8] Z.Qije, S.Chunyi
An Adaptive Sliding Mode Control Scheme for Robot Manipulators
International Journal of Control, Vol.57, No.2, pp.261-271, 1993
- [9] M.Bohm, M.A.Demetriou, S.Reich, I.G.Rosen
Model Reference Adaptive Control of Distributed Parameter Systems
SIAM Journal of Control and Optimization, Vol.36, No.1, pp.33-81, January 1998
- [10] K.J.Astrom, J.Hagglund, CC.Hang, W.K.Ho
Automatic Tuning and Adaption for PID Controllers - A Survey
Control Engineering Practice, Vol.1, No.4, pp.699-714, 1993
- [11] M.A.Duarte, K.S.Narendra
A New Approach to Model Reference Adaptive Control
Interational Journal of Adaptive Control and Signal Processing, Vol.3, pp.53-73, 1989
- [12] Chih-Hsin Tsai, Chi-Hsiang Wang, Wei-Song Lin
Robust Fuzzy Model Following Control of Robot Manipulators
IEEE Transactions on Fuzzy Systems, Vol.8, No.4, pp.462-469, August 2000
- [13] Isabelle Rivals, Leon Personnaz
Nonlinear Internal Model Control Using Neural Networks: Applications
IEEE Transactions on Neural Networks, Vol.11, No.1, pp.80-90, January 2000

-
- [14] D.R.Mudgett, A.S.Morse
An Advanced Example of Direct Adaptive Control System Design
International Journal of Adaptive Control and Signal Processing, Vol.4, pp.163-169, 1990
 - [15] K.S.Narendra, J.D.Boskovic
Robust Adaptive Control Using a Combined Approach
International Journal of Adaptive Control and Signal Processing, Vol.4, pp.111-131, 1990
 - [16] M.A.Duarte, K.S.Narendra
A New Approach to Model Reference Adaptive Control
International Journal of Adaptive Control and Signal Processing, Vol.3, pp.53-73, 1989
 - [17] G.Tao, P.A.Ioannou
Robust Model Reference Adaptive Control for Multivariable Plants
International Journal of Adaptive Control and Signal Processing, Vol.2, pp.217-248, 1988
 - [18] E. Sanchez, T,Shibata, L.A.Zadeh
Genetic Algorithms and Fuzzy Logic Systems, Soft Computing Perspectives
World Scientific, 1997
 - [19] M.L.Steinberg, A.B.Page
Nonlinear Adaptive Flight Control with Genetic Algorithm Design Optimization
International Journal of Robust and Nonlinear Control, Vol.9, pp.1097-1115, 1999
 - [20] K.A.Wise, J.S.Brinker, A.J.Calise, D.F.Enns, M.R.Elgersma, P.Voulgaris
Direct Adaptive Reconfigurable Flight Control for a Tailless Advanced Fighter Aircraft
International Journal of Robust and Nonlinear Control, Vol.9, pp.999-1012, 1999
 - [21] K.De Jong
Adaptive System Design, A Genetic Approach
IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-10, No.9, pp.566-574, September 1980
 - [22] W. Zuo
Multivariable Adaptive Control for a Space Station Using Genetic Algorithms
IEE Proceedings on Control theory and Applications, Vol.142, No.2, pp.81-87, March 1995

5

Mixed H_2/H_∞ Controller Synthesis with Hybrid Genetic Algorithms

Contents:

5.1	Introduction	p.5.2
5.1.1	Robust Control Theory.	p.5.3
5.1.2	H_2 Control Theory and State Space Solutions.	p.5.4
5.1.3	H_∞ Control Theory and State Space Solutions.	p.5.7
5.1.4	Mixed H_2/H_∞ Control Theory	p.5.9
5.2	H_2 Controller Synthesis	p.5.11
5.2.1	Simulation Setup.	p.5.11
5.2.2	Conventional State Space Solution.	p.5.13
5.2.3	Solution Using Genetic Algorithms	p.5.15
5.2.4	Convergence Rates for Hybrid Genetic Algorithms	p.5.19
5.3	H_∞ Controller Synthesis	p.5.21
5.3.1	Simulation Setup.	p.5.21
5.3.2	Conventional State Space Solution.	p.5.22
5.3.3	Solution Using Genetic Algorithms	p.5.24
5.3.4	Convergence Rates for Hybrid Genetic Algorithms	p.5.32
5.4	Mixed H_2/H_∞ Controller Synthesis	p.5.33
5.4.1	Simulation Setup.	p.5.33
5.4.2	Solution Using Genetic Algorithms	p.5.35
5.5	Chapter Summary and Conclusion	p.5.41
5.6	References and Further Reading	p.5.42

5.1 Introduction:

In this chapter, we look at how hybrid genetic algorithms can be applied to the synthesis of robust H_2 , H_∞ , mixed H_2/H_∞ linear full order, and reduced order compensators. The results are compared with conventional MATLAB **h2lqg** and **hinf** functions, including standard model reduction techniques for the reduced order compensators.

State space solutions to the H_2 and H_∞ problem for linear systems are well established, requiring only the solution to two Riccati equations. However, this results in compensators with the same order (or higher if shaping filters are used) as the plant, making implementation impractical. The use of model reduction techniques often leads to suboptimal controllers. A further complication is that most specifications are given as multiple or mixed objectives, for instance minimizing the H_2 performance measure of one closed loop transfer matrix, whilst ensuring the $H_\infty < \gamma$ norm bound of another closed loop transfer matrix is also satisfied. As yet, there is currently no direct design method to deal with mixed H_2/H_∞ objectives. In this chapter, we investigate the application of hybrid genetic algorithms to three separate optimal control problems:

- (i) Robust H_2 (linear quadratic gaussian LQR) controllers. Simulation results comparing conventional MATLAB **h2lqg** function with hybrid genetic algorithms is given. The direct implementation of reduced order compensators using genetic algorithms is compared with conventional model reduction techniques.
- (ii) The above procedure is repeated for an H_∞ controller.
- (iii) Parts (i) and (ii) are combined for the implementation of a mixed H_2/H_∞ controller.

A simple linear state-space example is used to illustrate the above concepts. The chapter is divided into four parts: the first part discusses basic concepts of H_2 and H_∞ control theory including standard state space solutions, methods for dealing with mixed H_2/H_∞ problems, fixed order controllers which includes homotopy theory, and genetic algorithms. In the second part, a robust H_2 controller is implemented using genetic algorithms, and compared with conventional state space solutions, the design of reduced order compensators is also included. The same applies for the third part involving a robust H_∞ controller. And lastly, a robust H_2/H_∞ controller with mixed design objectives is implemented.

5.1.1 Robust Control Theory:

The modern robust control paradigm combines the performance and robustness specifications into a single design framework. Thus all the information about a system including plant, external disturbances, noise, plant uncertainties and nonlinearities, can be combined into one single design framework. Robust control theory provides a systematic means of synthesizing controllers within this framework. Additionally, filters and frequency weights can be included into the design. Frequency weights can be used to shape the input noise over some frequency, and can also be used to emphasize the frequency range over which the effects of disturbances are to be minimized. For instance, if we wish to reduce the effects of external disturbances at some frequency range, then the frequency weights are emphasized more over this range. Nonlinearities and unstructured dynamics can be described as a magnitude bounded by some transfer function. For instance, in aircraft control, the airframe deformation produces unmodelled dynamics which can be captured by a norm bound frequency dependent transfer function. Figure 5.1 below illustrates the generalized plant P which includes a nominal plant transfer function, frequency dependent weights, uncertainty models, actuator and sensor dynamics.

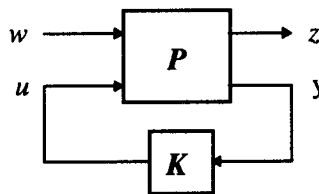


Fig.5.1
Generalized Plant and Controller

The inputs to the plant are: exogenous inputs w , which consist of disturbances, sensor noise, reference commands, and the controlled inputs u to the actuators. The outputs include z , consisting of performance measures, tracking errors, the measured outputs y from the sensors, generally corrupted by noise. The objective is then to minimize the *size* of the transfer function from w to z denoted by $T_{zw}(s)$ by the appropriate choice of the controller K , whilst ensuring internal stability of the closed loop system. The *size* of a transfer function can be represented by a norm. There are two types of norms which are of particular interest in control engineering: the H_2 and the H_∞ norm. Each has a different interpretation and application. The H_2 norm is simply the RMS output at z if w is an independent, zero mean unit intensity white noise source. Thus the H_2 controller is simply an extension (or generalization) of the linear quadratic gaussian (LQG) controller. The H_∞ norm however is defined as the maximum gain (singular value) of the transfer function over all frequencies of interest. Its application is mostly found in dealing with plant uncertainty.

Standard state-space solutions are available for these two types of problems, discussed further in sections 5.1.2 and 5.1.3. These solutions yield controllers which are the same order as the augmented plant P . This sometimes leads to controllers which are physically unrealistic to implement in practice. The application of model reduction techniques can result in sub-optimal controllers. Furthermore, in many practical situations, mixed design objectives are given such as minimizing the norm of: $\|T_{z_2w_2}(s)\|_2$, whilst bounding the norm: $\|T_{z_\infty w_\infty}(s)\|_\infty < \gamma$. This effectively is a constrained optimization problem.

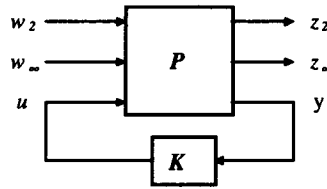


Fig.5.2
Mixed H_2/H_∞ Controller Specifications

At present, there are no direct design methods available to deal with multiobjective H_2/H_∞ control problems. There are a number of iterative numerical schemes available, some are briefly discussed in section 5.1.4. Some excellent introductory references to robust control theory include: [1, 3, 4, 5], a good tutorial paper on H_∞ control can be found in [2], a seminal paper by Zames 1979 [6], and later in which state space solutions were found by Doyle in 1989 [7]. In [8], Doyle demonstrated that H_2 controllers do not necessarily guarantee robustness. Conventional state space solutions to the H_2 and H_∞ are outlined next..

5.1.2 H_2 Control Theory and State Space Solutions:

(i) Definition of the H_2 norm:

Given the following generalized plant and controller (fig 5.3), the H_2 problem is as follows: solve for K such that the H_2 norm of $\|T_{zw}\|$ is minimized, whilst providing internal stability of the closed loop system. Internal stability is defined as follows: when the input is zero $w=0$, then both the states of the plant and controller should approach zero asymptotically: $x \rightarrow 0$ and $x_c \rightarrow 0$

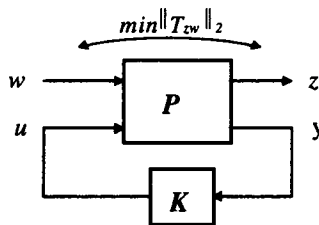


Fig.5.3
Generalized Plant and Controller

The H₂ norm of a transfer function T_{zw} is defined by equation 5.1 below:

$$\|T_{zw}\|_2 = \left(\frac{1}{2\pi} \int_{-\infty}^{\infty} \text{trace} [T_{zw}(jw) \cdot T_{zw}^*(jw)] dw \right)^{1/2} \quad \text{Eqn.5.1}$$

(ii) Compensator using Conventional state Space Methods:

The optimal compensator K which minimizes the above H₂ norm may be computed by the following procedure. Given the following generalized LTI plant in state space form:

$$\left. \begin{aligned} \dot{x} &= A \cdot x + B_1 \cdot w + B_2 \cdot u \\ z &= C_1 \cdot x + D_{11} \cdot w + D_{12} \cdot u \\ y &= C_2 \cdot x + D_{21} \cdot w + D_{22} \cdot u \end{aligned} \right\} \quad \text{Eqn.5.2}$$

assuming $D_{11}=D_{22}=0$ with no loss of generality, the compensator is given by:

$$K = \left[\begin{array}{c|c} \frac{A + B_2 F_2 + L_2 C_2 + L_2 D_{22} F_2}{F_2} & -L_2 \\ \hline & 0 \end{array} \right] \quad \text{Eqn.5.3}$$

where F_2 and L_2 are given by:

$$\left. \begin{aligned} F_2 &= -R_{uu}^{-1} (R_{xu}^T + B_2^T X_2) \\ L_2 &= -(Y_2 C_2^T + V_{xy}) V_{yy}^{-1} \end{aligned} \right\} \quad \text{Eqn.5.4}$$

and X_2 , Y_2 are the positive semidefinite solutions to the two Riccati equations:

$$\left. \begin{aligned} 0 &= X_2 A_r + A_r^T X_2 + R_{xx} - R_{xu} R_{uu}^{-1} R_{xu}^T - X_2 B_2 R_{uu}^{-1} B_2^T X_2 \\ 0 &= A_e Y_2 + Y_2 A_e^T + V_{xx} - V_{xy} V_{yy}^{-1} V_{xy}^T - Y_2 C_2^T V_{yy}^{-1} C_2 Y_2 \end{aligned} \right\} \quad \text{Eqn.5.5}$$

and

$$\left. \begin{aligned} A_r &= (A - B_2 R_{uu}^{-1} R_{xu}^T) \\ A_e &= (A - V_{xy} V_{yy}^{-1} C_2) \end{aligned} \right\} \quad \text{Eqn.5.6}$$

The compensator has the structure of a full order optimal state estimator and a full state optimal controller, expanding equation 5.3 we get the dynamic form of the compensator structure:

$$\left. \begin{aligned} \dot{x}_c &= (A + B_2 F_2 + L_2 C_2 + L_2 D_{22} F_2) \cdot x_c + (-L_2) \cdot y \\ u &= (F_2) \cdot x_c + (0) \cdot y \end{aligned} \right\} \quad \text{Eqn.5.7}$$

The solution to the Riccati equations can be found without iteration. The above implementation is also available using MATLAB's robust control toolbox function **h2lqg**. Results using this method are compared with hybrid genetic algorithms.

(iii) Solution by Genetic Algorithms:

We now outline the method used for solving the H_2 problem using genetic algorithms and a pre-defined (fixed order) compensator. The generalized compensator K is given by (compare this with equation 5.7 above):

$$\left. \begin{aligned} \dot{x}_c &= A_c \cdot x_c + B_c \cdot y \\ u &= C_c \cdot x_c \end{aligned} \right\} \quad \text{Eqn.5.8}$$

The above compensator can be of any order, and need not necessarily be the same as that of the plant. The closed loop system is obtained by combining equations 5.2 with 5.8 into one single augmented system [9], similar in principle to chapter 3:

$$\left. \begin{aligned} \dot{\tilde{x}} &= \tilde{A} \cdot \tilde{x} + \tilde{B} \cdot w \\ z &= \tilde{C} \cdot \tilde{x} \end{aligned} \right\} \quad \text{Eqn.5.9}$$

The system is now in input w output z form, where the composite matrices are:

$$\tilde{x} = \begin{bmatrix} x \\ x_c \end{bmatrix} \quad \tilde{A} = \begin{bmatrix} A & -B_2 \cdot C_c \\ B_c \cdot C_2 & A_c \end{bmatrix} \quad \tilde{B} = \begin{bmatrix} B_1 \\ B_c \cdot D_{21} \end{bmatrix} \quad \tilde{C} = [C_1 \quad D_{12} \cdot C_c] \quad \text{Eqn.5.10}$$

The closed loop transfer function from w to z is given by the Laplace transform of equation 5.9, thus: $T_{zw} = \tilde{C}(sI - \tilde{A})^{-1} \tilde{B}$. If the disturbance w is a zero mean unit intensity white noise, then the H_2 norm from w to z is given by the expression [7]:

$$\left. \begin{aligned} \|T_{zw}\|_2^2 &= \text{trace}(Q \cdot \tilde{B} \cdot \tilde{B}^T) \\ &= \text{trace}(P \cdot \tilde{C} \cdot \tilde{C}^T) \end{aligned} \right\} \text{minimize} \quad \text{Eqn.5.11}$$

where Q and P are the observability and controllability gramians, found by solving either one of the following Lyapunov equations:

$$\left. \begin{aligned} \tilde{A} \cdot P + P \cdot \tilde{A}^T + \tilde{B} \cdot \tilde{B}^T &= 0 \\ \tilde{A}^T Q + Q \cdot \tilde{A} + \tilde{C}^T \cdot \tilde{C} &= 0 \end{aligned} \right\} \text{constraints} \quad \text{Eqn.5.12}$$

This is essentially a constrained optimization problem with one optimization function and one constraint. This offers an alternative method for solving the H_2 problem using genetic algorithms, in which the order of the compensator can be constrained. This problem has been solved using genetic algorithms [9], homotopy theory methods [10,11], and quasi-Newton/continuation methods [12]. Homotopy methods described in 5.1.4, are essentially a gradient based optimization requiring the calculation of a gradient of a lagrangian function (see also chapter 1.5). Note the addition of further constraints such as $Q > 0$ and $P > 0$, which will be discussed in greater detail in section 5.2.

5.1.3 H_∞ Control Theory and State Space Solutions:

(i) Definition of the H_∞ norm:

Given the following generalized plant and controller figure 5.4, the H_∞ problem is as follows: solve for K such that the H_∞ norm of the transfer function $\|T_{zw}\|$ is minimized, whilst ensuring internal stability of the closed loop plant and controller. This problem is conceptually more difficult to solve compared with the previous H_2 norm problem:

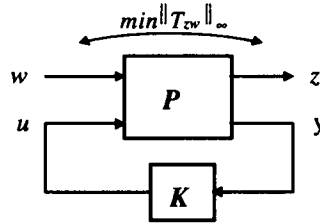


Fig.5.4
Generalized Plant and Controller

The H_∞ is defined by the following equation 5.13 below:

$$\|T_{zw}\|_\infty = \sup_{\omega} \sigma_{\max}[T_{zw}(j\omega)] \quad \text{Eqn.5.13}$$

Thus the H_∞ is simply the maximum value of the singular value plot: $\sigma_{\max}[T_{zw}(j\omega)]$ over all frequencies ω , and sup is the least upper bound of the function (i.e. supremum).

(ii) Compensator using Conventional state Space Methods:

The optimal compensator K which minimizes H_∞ is given by the following expression, in packed matrix notation:

$$K = \left[\begin{array}{c|c} A_\infty & -Z_\infty L_\infty \\ \hline F_\infty & 0 \end{array} \right] \quad \text{Eqn.5.14}$$

where the individual matrices are given by:

$$\begin{aligned} A_\infty &= A + (B_1 + L_\infty D_{21})W_\infty + B_2 F_\infty + Z_\infty L_\infty C_2 + Z_\infty L_\infty D_{22} F_\infty \\ F_\infty &= -R_{uu}^{-1}(R_{xu}^T + B_2^T \cdot X_\infty) \\ W_\infty &= \frac{1}{\gamma^2} B_1^T X_\infty \\ L_\infty &= -(Y_\infty C_2^T + V_{xy})V_{yy}^{-1} \\ Z_\infty &= \left(I - \frac{1}{\gamma^2} Y_\infty X_\infty \right)^{-1} \end{aligned} \quad \text{Eqn.5.15}$$

and where X_∞ and Y_∞ are the solutions to the following two Algebraic Riccati equations:

$$\left. \begin{aligned} 0 &= X_\infty A_r + A_r^T X_\infty + R_{xx} - R_{xu} R_{uu}^{-1} R_{xu}^T - X_\infty (B_2 R_{uu}^{-1} B_2^T - B_1 B_1^T / \gamma^2) X_\infty \\ 0 &= A_e^T Y_\infty + Y_\infty A_e^T + V_{xx} - V_{xy} V_{yy}^{-1} V_{xy}^T - Y_\infty (C_2^T V_{yy}^{-1} C_2 - C_1^T C_1 / \gamma^2) Y_\infty \end{aligned} \right\} \text{Eqn.5.16}$$

Note the presence of the γ parameter. Consequently, the solution to the H_∞ problem requires an iterative search over γ in which γ is minimized and equations 5.16 are satisfied, furthermore, we also require that $X_\infty > 0$, $Y_\infty > 0$ and the solution to 2 Hamiltonian matrices must contain no eigenvalues on the $j\omega$ axis (see reference. 3 pp.654). The H_∞ is then given by: $\|T_{zw}\| < \gamma$.

(iii) Solution by Genetic Algorithms:

In a similar fashion to the H_2 formulation, the H_∞ problem can also be stated as follows: given the generalized dynamic compensator K :

$$\left. \begin{aligned} \dot{x}_c &= A_c \cdot x_c + B_c \cdot y \\ u &= C_c \cdot x_c \end{aligned} \right\} \text{Eqn.5.17}$$

Where the composite matrices $\tilde{A}, \tilde{B}, \tilde{C}$ are given by equation 5.10. The H_∞ compensator can be found from the solution to the following minimization problem:

$$\left. \begin{aligned} \|H\|_\infty^2 &= \text{trace}(Q_\infty \cdot \tilde{B} \cdot \tilde{B}^T) \\ &= \text{trace}(P_\infty \cdot \tilde{C} \cdot \tilde{C}^T) \end{aligned} \right\} \text{minimize} \quad \text{Eqn.5.18}$$

where Q_∞ and P_∞ are the observability and controllability gramians, given by the solution to the following Riccati equations

$$\left. \begin{aligned} \tilde{A} \cdot P_\infty + P_\infty \cdot \tilde{A}^T + \tilde{B} \cdot \tilde{B}^T + \gamma^{-2} P_\infty \tilde{C} \cdot \tilde{C}^T P_\infty &= 0 \\ \tilde{A}^T Q_\infty + Q_\infty \cdot \tilde{A} + \tilde{C}^T \cdot \tilde{C} + \gamma^{-2} Q_\infty \tilde{B} \cdot \tilde{B}^T Q_\infty &= 0 \end{aligned} \right\} \text{constraint} \quad \text{Eqn.5.19}$$

This is essentially a constrained optimization problem with one optimization function and one constraint. In summary, to find the compensator, we require to find the minimum value of γ which will minimize equation 5.18 subject to the constraint given by equation 5.19. The H_∞ is then given by: $\|T_{zw}\| < \gamma$. This problem has been solved using homotopy theory methods [10, 11], and quasi-Newton/continuation methods [12] using a Lagrangian function formulation, and *Linear Matrix Inequalities* methods [21]. Genetic algorithms have not yet been applied to this problem. Note also the presence of further constraints: $Q_\infty > 0$, $P_\infty > 0$, and stability of closed loop system which are discussed in section 5.3.

5.1.4 Mixed H_2/H_∞ Control Theory:

(i) Conventional Methods:

When dealing with mixed H_2/H_∞ control objectives, there are no direct design methods available to finding the compensator K . As most specifications are given as multiobjective optimization problems, then solving this type of problem becomes important. There are currently two methods of dealing with mixed H_2/H_∞ control objectives: *homotopy algorithms* [16] and *Linear Matrix Inequalities* (LMI) in [21]. Both methods are based on numerical optimization techniques and require iterative search algorithms.

-Homotopy algorithms: or continuation methods, are based on algebraic and differential topology theory [15] which can be used as a global search technique for nonlinear problems. Homotopy algorithms can be used to solve complex optimization functions by first solving a simpler and similar function in which a solution can easily be obtained, and then gradually distorting the simpler function back into the original more complex function. At the same time also distorting the solution of the simpler function into the original more complex function. A good introduction can be found in [16]. Applications to H_2 fixed order compensators using homotopy methods can be found in [11, 17, 18], and to mixed H_2/H_∞ problems in [19], and more specifically for H_∞ problems in [20]. Homotopy theory can also be used to solve nonlinear constrained and unconstrained optimization problems. These methods are globally convergent for many complex optimization problems, but can suffer ill-conditioning due to roundoff errors in numerical solution. Convergence is strongly dependent on the ability to accurately track the solution curve which depends on the deformation function.

-Linear Matrix Inequalities (LMI): has gained considerable popularity over the last few years. Currently the theory of robust control is dealt with concepts of Linear Matrix Inequalities (LMI) and convex optimization. Multiobjective H_2 and H_∞ control problems can be cast into a single LMI framework which can be solved numerically with great efficiency using interior point or cutting-plane methods. An excellent introduction to LMI applications and convex optimization in control theory can be found in [23, 24, 30, 31, 43]. Applications using LMI dealing specifically with only H_2 /LQG problems can be found in [21], and to H_∞ problems [27, 33]. Applications to mixed H_2 / H_∞ control objectives using LMI's and convex optimization can be found in [22, 25, 34, 35 40, 44, 45, 46], for SISO systems [48], and discrete time systems [49, 50].

Because LMI uses a finite sum to approximate an infinite dimensional optimization variable (i.e. Ritz approximation), the accuracy of the solution depends on the number of parameters used to approximate the function.

Other variations to the mixed H_2 / H_∞ control problem includes a modified Riccati method [26] in which the inputs w_2 and w_∞ are the same, a state space solution to the mixed H_2/H_∞ control problem is given in [36, 39] under some mild assumptions, and using Lagrange multiplier methods are given in [38]. Necessary and sufficient conditions for the solution to the H_∞/H_2 problem can be found in [47], applications to nonlinear H_∞ control can be found in [29, 51, 52, 53, 54, 55].

(ii) Solution by Genetic Algorithms:

Genetic algorithms have also been applied to the mixed H_2/H_∞ problem. Examples of applications to robust control using genetic algorithms can be found in [13, 14], in particular for H_∞ fixed order compensators [56, 57]. Some examples specific to H_2 control can be found in [9] dealing with fixed order compensators, the mixed H_2/H_∞ control problem has also been investigated using a *two-player Nash differential game theory* [32], and genetic algorithms for SISO systems in polynomial q^{-1} form [37], and multiobjective applications [42].

The method we use is to combine the results obtained in sections 5.1.2 and 5.1.3 into one single multiobjective optimization problem. Then the problem is to minimize the two functions:

$$\left. \begin{aligned} \|T_2\|_2^2 &= \text{trace}(Q_2 \cdot \tilde{B} \cdot \tilde{B}^T) \\ \|T_\infty\|_\infty^2 &= \text{trace}(Q_\infty \cdot \tilde{B} \cdot \tilde{B}^T) \end{aligned} \right\} \text{minimize} \quad \text{Eqn.5.20}$$

subject to the constraints:

$$\left. \begin{aligned} \tilde{A}^T Q_2 + Q_2 \cdot \tilde{A} + \tilde{C}^T \cdot \tilde{C} &= 0 \\ \tilde{A}^T Q_\infty + Q_\infty \cdot \tilde{A} + \tilde{C}^T \cdot \tilde{C} + \gamma^{-2} Q_\infty \tilde{B} \cdot \tilde{B}^T Q_\infty &= 0 \end{aligned} \right\} \text{constraint} \quad \text{Eqn.5.21}$$

Where: T_2 is the transfer function from $w_2 \rightarrow z_2$, and T_∞ is the transfer function from $w_\infty \rightarrow z_\infty$. This is described in more detail in section 5.4. A more relaxed approach would be to minimize the H_2 norm subject to the constraint: $H_\infty < \gamma$. Note also the necessary conditions of closed loop internal stability and positive definite solutions to the Lyapunov and Riccati equations 5.21. To our knowledge, this method has never been investigated using genetic and hybrid algorithms.

5.2 H_2 Controller Synthesis:

5.2.1 Simulation Setup:

For the first simulation, we synthesize a full order H_2 compensator using conventional state space methods, and compare with solutions obtained using hybrid genetic algorithms. For the second part of the simulation, a reduced order compensator is synthesized using hybrid genetic algorithms, and the result is compared with conventional (MATLAB) model reduction techniques on a full order compensator. Convergence rates and computational effort for three hybrid genetic algorithms are compared: (1) conventional genetic algorithms, (2) genetic algorithms and simulated annealing, and (3) genetic algorithms and greedy search.

(i) **Plant Model:** For this simulation, the plant model is taken from reference [45] and is illustrated below:

$$\left. \begin{aligned} \dot{x} &= A \cdot x + B_1 \cdot w + B_2 \cdot u \\ z_2 &= C_1 \cdot x + D_{11} \cdot w + D_{12} \cdot u \\ z_\infty &= C_2 \cdot x + D_{21} \cdot w + D_{22} \cdot u \\ y &= C_3 \cdot x + D_{31} \cdot w + D_{32} \cdot u \end{aligned} \right\} \quad \text{Eqn.5.22}$$

The setup is depicted in Figure.5.5 below:

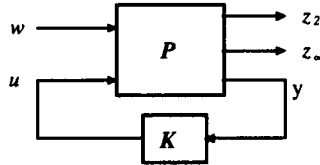


Fig.5.5
Simulation Setup Plant/Compensator

Thus we have a single disturbance input w and two performance output z_2 and z_∞ . For this part, the z_∞ is ignored and only the transfer function from $w \rightarrow z_2$ is considered. Note that one of the conditions required for computing the H_2 norm is that $D_{11}=0$. The plant matrices are given by:

$$A = \begin{bmatrix} 0 & 10 & 2 \\ -1 & -1 & 0 \\ 0 & 2 & -5 \end{bmatrix} \quad C_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad C_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad C_3 = [0 \quad 1 \quad 0]$$

$$B_1 = [1 \quad 0 \quad 1] \quad B_2 = [0 \quad 1 \quad 0]$$

$$D_{11} = [0 \quad 0 \quad 0] \quad D_{12} = [0 \quad 0 \quad 1]$$

$$D_{21} = [0 \quad 0] \quad D_{22} = [0 \quad 1]$$

$$D_{31} = [2] \quad D_{32} = [0]$$

(ii) **Compensator:** The above system represents a 3rd order linear dynamical system, for the simulation we ignore any frequency dependent weights. Thus, the full order compensator would then also be a 3rd order system. The compensator in state space is given by:

$$\left. \begin{aligned} \dot{x}_c &= A_c \cdot x_c + B_c \cdot y \\ u &= C_c \cdot x_c \end{aligned} \right\} \quad \text{Eqn.5.23}$$

For the given plant model above, the compensator matrix sizes are: $A_c \in \mathbb{R}^{3 \times 3}$, $B_c \in \mathbb{R}^{3 \times 1}$, and $C_c \in \mathbb{R}^{1 \times 3}$. Alternatively, the compensator may be written in input/output transfer function form, for a 3rd order system, the transfer function from y to u is given by:

$$K(s) = \frac{c_2 \cdot s^2 + c_1 \cdot s + c_0}{s^3 + a_2 \cdot s^2 + a_1 \cdot s + a_0} \quad \text{Eqn.5.24}$$

The compensator given by equation 5.23 consists of a total of: $9+3+3=15$ parameters, whilst the second compensator of only 6 parameters. Clearly the first compensator is overparameterized. The second compensator is in effect a minimal realization, which can be transformed into either reachable or observable canonical forms, (see reference [59] pp.67) thus:

$$A_c = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 \end{bmatrix} \quad B_c = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad C_c = [c_0 \quad c_1 \quad c_2] \quad \text{Eqn.5.25}$$

Furthermore, note that the B_c compensator matrix remains a constant, and only the A_c and C_c matrices are affected.

(iii) **Closed Loop System:** The closed loop system is obtained by combining equations 5.22 with 5.23 into one single (augmented) system:

$$\left. \begin{aligned} \dot{\tilde{x}} &= \tilde{A} \cdot \tilde{x} + \tilde{B} \cdot w \\ z_2 &= \tilde{C}_2 \cdot \tilde{x} \\ z_\infty &= \tilde{C}_\infty \cdot \tilde{x} \end{aligned} \right\} \quad \text{Eqn.5.26}$$

The system is now in input/output: $w \rightarrow \{z_2, z_\infty\}$ form, where the matrices are:

$$\tilde{A} = \begin{bmatrix} A & -B_2 \cdot C_c \\ B_c \cdot C_3 & A_c \end{bmatrix} \quad \tilde{B} = \begin{bmatrix} B_1 \\ B_c \cdot D_{31} \end{bmatrix} \quad \tilde{C}_2 = [C_1 \quad D_{12} \cdot C_c] \quad \tilde{C}_\infty = [C_2 \quad D_{22} \cdot C_c] \quad \text{Eqn.5.27}$$

For the H_2 simulation, only output z_2 is considered. Simulation results are given in the following pages.

5.2.2 Conventional State Space Solution:

(i) **Full order Compensator:** The above H_2 problem can be readily solved using MATLAB's `h2lqr` function. The full order compensator which minimizes the H_2 norm from input w to output z_2 is given by (shown both in pole-zero and polynomial forms):

$$K(s) = \frac{-0.023046(s + 4.618)(s - 13.99)}{(s + 5.095)(s^2 + 1.383s + 9.912)} \quad \left. \vphantom{\frac{-0.023046(s + 4.618)(s - 13.99)}{(s + 5.095)(s^2 + 1.383s + 9.912)}} \right\} \text{full order compensator}$$

$$= \frac{-0.02305s^2 + 0.2159s + 1.488}{s^3 + 6.478s^2 + 16.96s + 50.5}$$

The open loop H_2 norm is **0.43390**, and the closed loop: **0.40957**. The presence of zero in the positive half plane (13.99) results in a minimum phase compensator. The compensator eigenvalues from $K(s)$ above are given by: **$-0.6916 \pm j3.0714$, -5.0948** . Open and closed loop singular value plots of $\sigma(T_{z_2w}(j\omega))$ are plotted in figure 5.6 below:

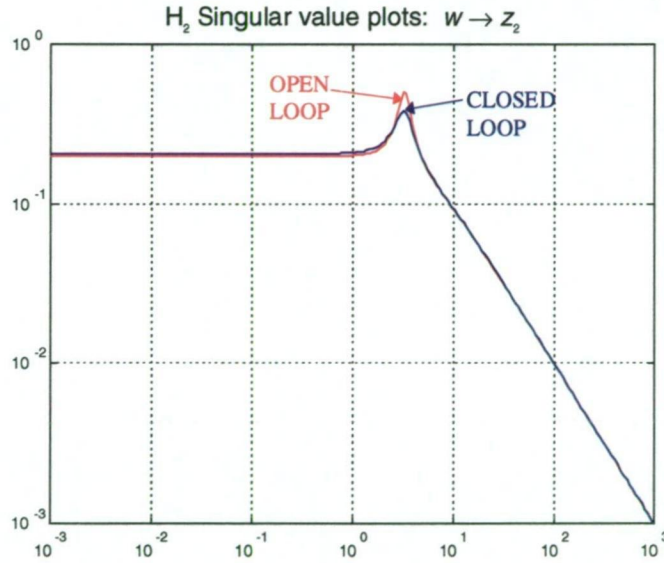


Fig.5.6

Open loop and closed loop singular value plot of $T_{z_2w}(s)$

(ii) **Reduced order Compensator:** Model reduction can be used to obtain a reduced order compensator (2^{nd} order). To perform model reduction the MATLAB function **balreal** is first applied to the compensator which produces a balanced realization of the compensator. In this case, the diagonal entries of the joint gramian are: **0.0450, 0.0300, 0.0003**. Since the last state is weakly coupled to the input/output, this state can be removed by using the MATLAB model reduction function **modred**. This yields the following second order compensator:

$$K(s) = \frac{-0.019745(s - 15.34)}{(s^2 + 1.383s + 10.06)}$$
$$= \frac{-0.01975s + 0.3029}{s^2 + 1.383s + 10.06}$$

} reduced order compensator

With the reduced order compensator, the H₂ norm of the closed loop system is **0.40958**. In this case the effect of model reduction does not degrade the performance of the closed loop transfer function *T_{z2w}*. This can be seen from the singular value plot of the open loop, closed loop using full order and reduced order compensator is shown in figure 5.7 below:

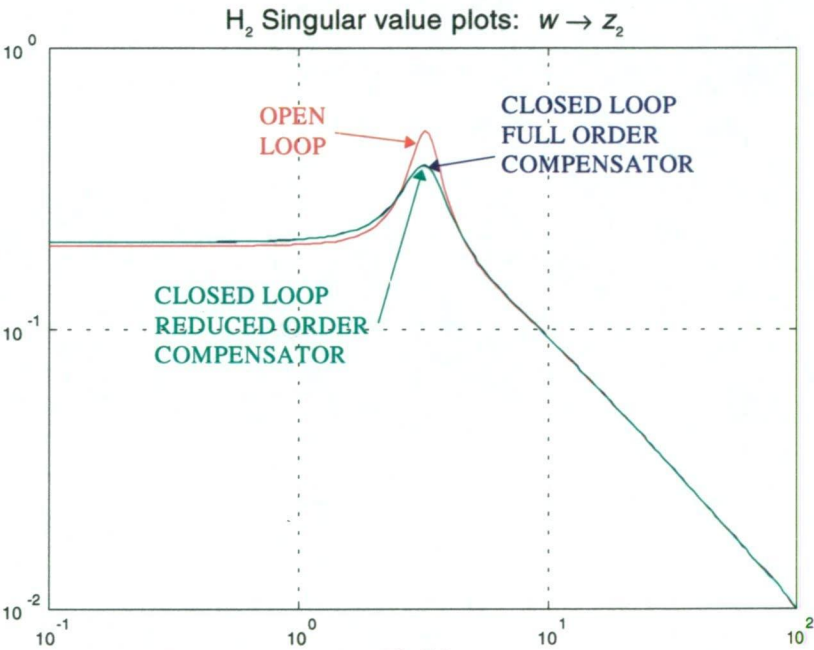


Fig.5.7
Open loop and closed loop using full order and reduced order compensator.

The singular value plot of the closed loop system with a full order compensator (plotted in blue) is overlapped by the closed loop plot using the reduced order compensator (plotted in green). In this instance, the model reduction works well due to the presence of a very weakly coupled state (0.0003). A summary of the three H₂ norms for each instance is tabulated below:

System:	H ₂
Open Loop:	0.43390
Closed Loop (Full order compensator):	0.40957
Closed Loop (Reduced order compensator):	0.40958

Table 5.1

5.2.3 Solution Using Genetic Algorithms:

The above H_2 problem will now be solved using hybrid genetic algorithms as described earlier in section 5.1.2. Both full order and reduced order compensators will be implemented and results compared with those obtained using conventional methods above.

(i) **Full order Compensator:** Solution using genetic algorithms to the H_2 problem was described in section 5.1.2, this is briefly summarized below. The full order compensator is given by the following dynamical system:

$$\left. \begin{aligned} \dot{x}_c &= A_c \cdot x_c + B_c \cdot y \\ u &= C_c \cdot x_c \end{aligned} \right\} \quad \text{Eqn.5.28}$$

where the compensator matrices are given by the minimal realization in canonical form:

$$A_c = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 \end{bmatrix} \quad B_c = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad C_c = [c_0 \quad c_1 \quad c_2] \quad \text{Eqn.5.29}$$

Combining with the plant model given by equation 5.22, we get the following closed loop transfer function from w to z_2 : thus: $T_{z_2w}(s) = \tilde{C}_2(sI - \tilde{A})^{-1} \tilde{B}$, where the composite matrices are:

$$\tilde{A} = \begin{bmatrix} A & -B_2 \cdot C_c \\ B_c \cdot C_3 & A_c \end{bmatrix} \quad \tilde{B} = \begin{bmatrix} B_1 \\ B_c \cdot D_{31} \end{bmatrix} \quad \tilde{C}_2 = [C_1 \quad D_{12} \cdot C_c] \quad \text{Eqn.5.30}$$

The H_2 genetic algorithm is summarized below:

Objective:

The objective of the genetic algorithm then is to find the compensator parameters $\{a_0, a_1, a_2, c_0, c_1, c_2\}$ which will minimize the following function:

$$\|T_{zw}\|_2^2 = \text{trace}(Q_2 \cdot \tilde{B} \cdot \tilde{B}^T)$$

Subject to the following constraints:

1. The existence of the solution to the lyapunov equation:

$$\tilde{A}^T Q_2 + \tilde{Q}_2 \cdot A + \tilde{C}^T \cdot \tilde{C} = 0, \text{ where } Q_2 \text{ is positive definite symmetric ie: } Q_2 > 0.$$

2. The closed loop system must be internally stable, ie: eigenvalues of

\tilde{A} must be stable. Controller must be stable, eigenvalues of A_c must be < 0 . No eigenvalues on the $j\omega$ axis (ie marginally stable closed loop system).

The chromosomal representation for this problem is illustrated in figure 5.8 below:

a_2	a_1	a_0	c_2	c_1	c_0	T_{zw}	Fitness
-------	-------	-------	-------	-------	-------	----------	---------

Fig.5.8
Chromosomal Representation for the H_2 problem

where real number codification is used for $\{a_0, a_1, a_2, c_0, c_1, c_2\}$, the H_2 value is computed as above, and the fitness is the inverse of H_2 . Note that if any of the constraints are violated, then the solution is infeasible and the fitness is set to zero. We chose to discard infeasible solutions rather than attempting to use a repair algorithm because an infeasible solution results in an unstable closed loop system. There are no direct repair algorithms which would directly produce a feasible solution from an unfeasible one. Subsequently attempting to repair the infeasible solution would instead require a second search using the infeasible chromosome as a starting point which would not be as efficient. The solution to the Lyapunov function is found using a Hamiltonian matrix approach. Because this problem is essentially a constrained optimization problem, a second method for solving it would be to minimize the a Lagrangian function (see section 1.5.1), for instance:

$$L(\lambda, A_c, C_c) = \text{trace}\{(Q_2 \cdot \tilde{B} \cdot \tilde{B}^T) + \lambda(\tilde{A}^T \cdot \tilde{Q}_2 + \tilde{Q}_2 \cdot \tilde{A} + \tilde{C}^T \cdot \tilde{C})\} \quad \text{Eqn.5.31}$$

Where L is the Lagrangian function to minimize, and λ is the multiplier matrix. This approach however also requires solving for Q_2 and λ matrices. Given that both Q_2 and λ are symmetric and size 6×6 , a total of 21+21 parameters in addition to the 6 controller parameters. Equation 5.31 can also be solved using gradient based optimization techniques or homotopy theory, see references [9, 10, 19, 28]. However Homotopy or gradient based optimization requires the computation of gradients, for equation 5.28 we have: $\partial L / \partial A_c = 0$, $\partial L / \partial B_c = 0$, $\partial L / \partial \lambda = 0$, $\partial L / \partial Q_2 = 0$, a total of 48 simultaneous equations. Furthermore, the computation of a Hessian of size 48×48 is also required which could be numerically ill-conditioned. Simulation results using the GA algorithm-1 from the previous page are tabulated below. For this algorithm we used: binary tournament selection, crossover probability $P_c=0.5$, mutation probability $P_m=0.2$, full order compensator, population size=100, uniform weighted average crossover. We ran the simulation 5 times due to illustrate the probabilistic nature of the GA, results are tabulated in table 5.2 below. Each simulation runs for 250 generations, the first (red) row is the solution obtained with conventional matlab methods from section 5.2.1. Computational effort is approximately 370MFP for 250 generations.

a_2 :	a_1 :	a_0 :	b_2 :	b_1 :	b_0 :	H_2 :
6.478	16.96	50.5	-0.0230	0.2159	1.488	0.409574
7.165	17.90	57.3	-0.0232	0.2034	1.693	0.409574
6.592	17.12	51.6	-0.0231	0.2144	1.522	0.409574
7.030	17.71	56.0	-0.0233	0.2062	1.652	0.409574
6.948	17.60	55.2	-0.0232	0.2071	1.629	0.409574
7.105	17.81	56.7	-0.0233	0.2044	1.675	0.409574

Table 5.2

Results from table 5.1 indicate that some of the controller parameters have a less influential effect upon the outcome of the computation of the H₂ norm. For instance the parameter b_2 is nearly always the same, whilst the a_2 parameter is more variant. For the second row in table 5.2, the compensator transfer function obtained with genetic algorithms may be written as for comparison with conventional MATLAB solutions:

$$K(s) = \frac{-0.023222(s+5.217)(s-13.98)}{(s+5.784)(s^2+1.382s+9.914)}$$
$$= \frac{-0.02322s^2+0.2034s+1.693}{s^3+7.165s^2+17.9s+57.34}$$

} GA full order compensator

The compensator eigenvalues from K(s) above are given by: **-0.6908 ± j3.0719, -5.7836**. Figure 5.9 below compares the singular value plot of the closed loop system $w \rightarrow z_2$ for the solution obtained with genetic algorithms and conventional methods. The curves are identical and overlap.

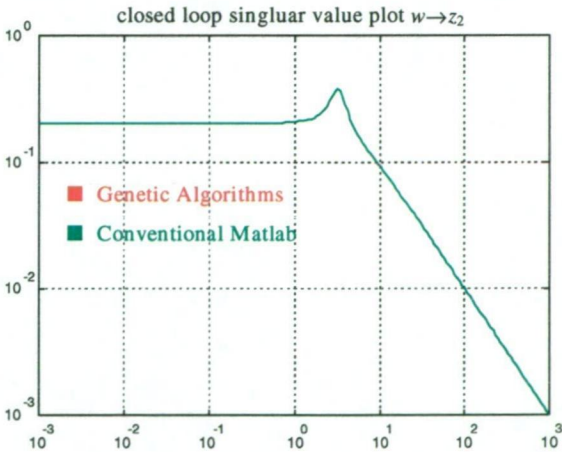


Fig.5.9
Comparing singular value plots: GA and Conventional Methods

(ii) **Reduced order Compensator:** A reduced order compensator (2nd order) can be implemented directly by defining the compensator structure as:

$$\dot{x}_c = A_c \cdot x_c + B_c \cdot y$$
$$u = C_c \cdot x_c$$

} Eqn.5.32

Where the matrices for the reduced order compensator are given by:

$$A_c = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix} \quad B_c = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad C_c = \begin{bmatrix} c_0 & c_1 \end{bmatrix}$$

Eqn.5.33

Again, the simulation is repeated for this system, and results are tabulated in Table.5.3 on the following page. From table 5.3, we can see that the results using genetic algorithms agree well with the results obtained using conventional methods (shown in red in the first row of table 5.3). For this algorithm we used: binary tournament selection, crossover probability $P_c=0.5$, mutation probability $P_m=0.2$, population size=100, uniform weighted average crossover, generations=300. We repeated the simulation 5 times to illustrate the probabilistic nature of the GA. In this case, there is less variation in convergence when compared to the full order compensator (table 5.1), i.e.: the GA algorithm converges to the same solution in all cases.

a_2	a_1	a_0	b_2	b_1	b_0	H_2
	1.383	10.06		-0.0197	0.3029	0.40958
0.000	1.386	10.04	0.000	-0.0196	0.3033	0.409579
0.000	1.386	10.04	0.000	-0.0196	0.3033	0.409579
0.000	1.386	10.04	0.000	-0.0196	0.3033	0.409579
0.000	1.386	10.04	0.000	-0.0196	0.3033	0.409579
0.000	1.386	10.04	0.000	-0.0196	0.3033	0.409579

Table 5.3

Figure 5.10 below illustrates a typical convergence plot of the genetic algorithm. Convergence is generally within the first 50 generations, after which the population has nearly converged to a single solution. Note that this is a plot of H_2-H_{2min} , where H_2 is the fittest value in the population, currently found by the GA and H_{2min} is the target value = 0.4096.

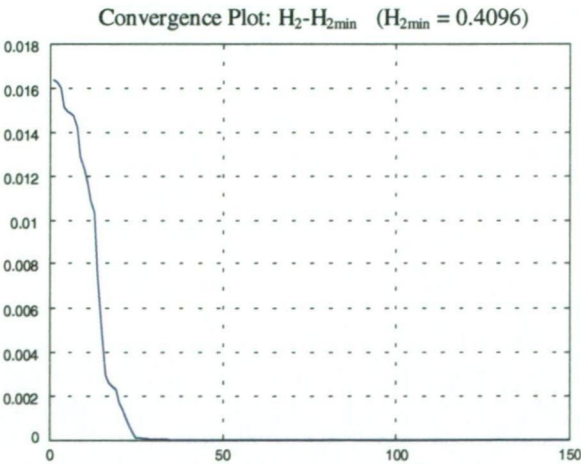


Fig.5.10
GA Typical Convergence Plot

Computational effort: conventional state space solution requires approximately 0.25MFP, and several seconds of computational time (Pentium III/750MHz). Genetic algorithms however require 780MFP and approximately 100 seconds of computation time for 300 generations. Thus whilst genetic algorithms can give a direct solution to fixed and reduced order compensators, the computational overheads are very high. Table 5.2 illustrates the convergence properties of the genetic algorithm. The variation of the compensators found is attributed to the presence of the weakly coupled state (0.0003). This means that the solution has a weak global minimum within a wide global minimum. The figure below illustrates this concept:

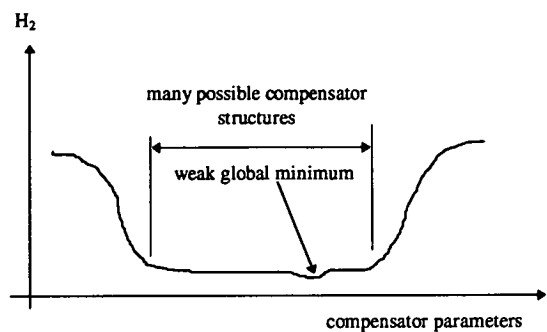


Fig.5.11

This problem is no longer present for the reduced order compensator as seen from table 5.3. The poor gradient prohibits the GA from locating the weak global minimum. Table 5.2 shows that the H_2 is nearly identical in different compensators, and therefore the function minimum is nearly flat.

5.2.4 Convergence Rates for Hybrid Genetic Algorithms:

This last set of simulations compare the convergence rates of the three different hybrid genetic algorithms: (a) conventional genetic algorithms, (b) genetic algorithms and simulated annealing, and (c) genetic algorithms and greedy search. Figure 5.12 below compares the convergence rate of the three algorithms for the full order compensator. Both hybrid methods converge much more rapidly compared with the conventional genetic algorithm. Computational effort compared with conventional matlab H_2 design is summarized in table 5.4 below with accuracy set at 10^{-7} (i.e.: $H_{2min} < 10^{-7}$):

Method:	MFLOPS	Design Time:
Conventional matlab:	0.25	< 1 sec
GA:	80	10 sec
GA + Simulated annealing	30	4 sec
GA + Greedy Search:	25	3 sec

Table.5.4

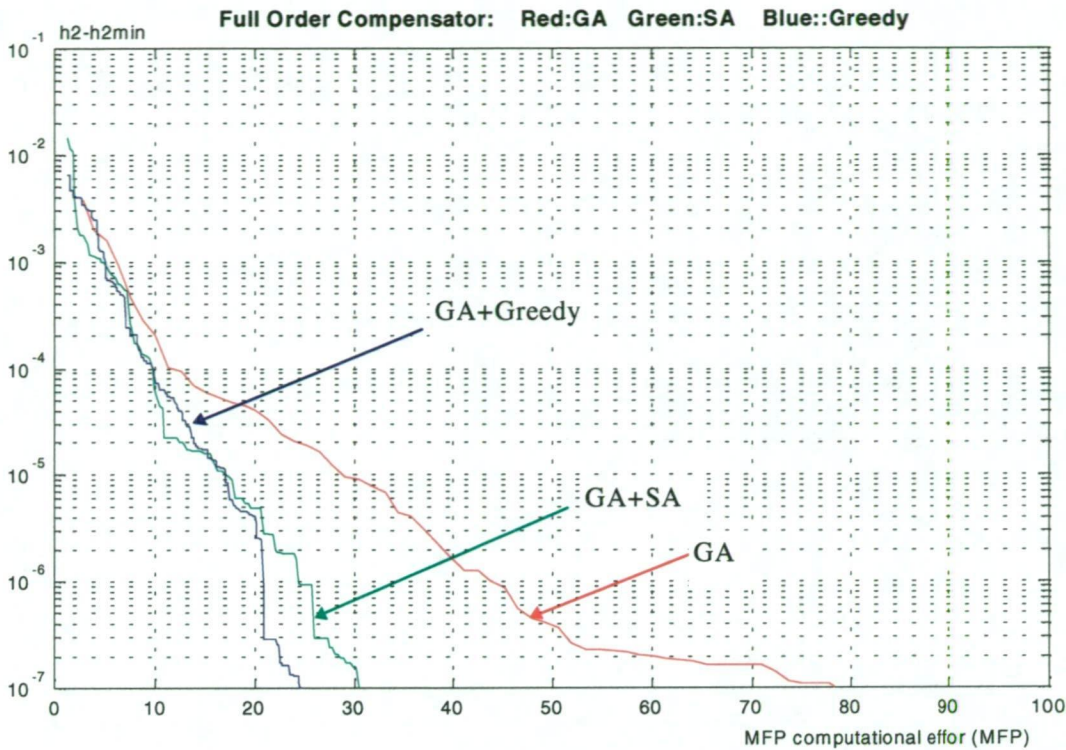


Fig.5.12.A
Convergence rates for: GA, GA+SA, GA+GS (full order compensator)

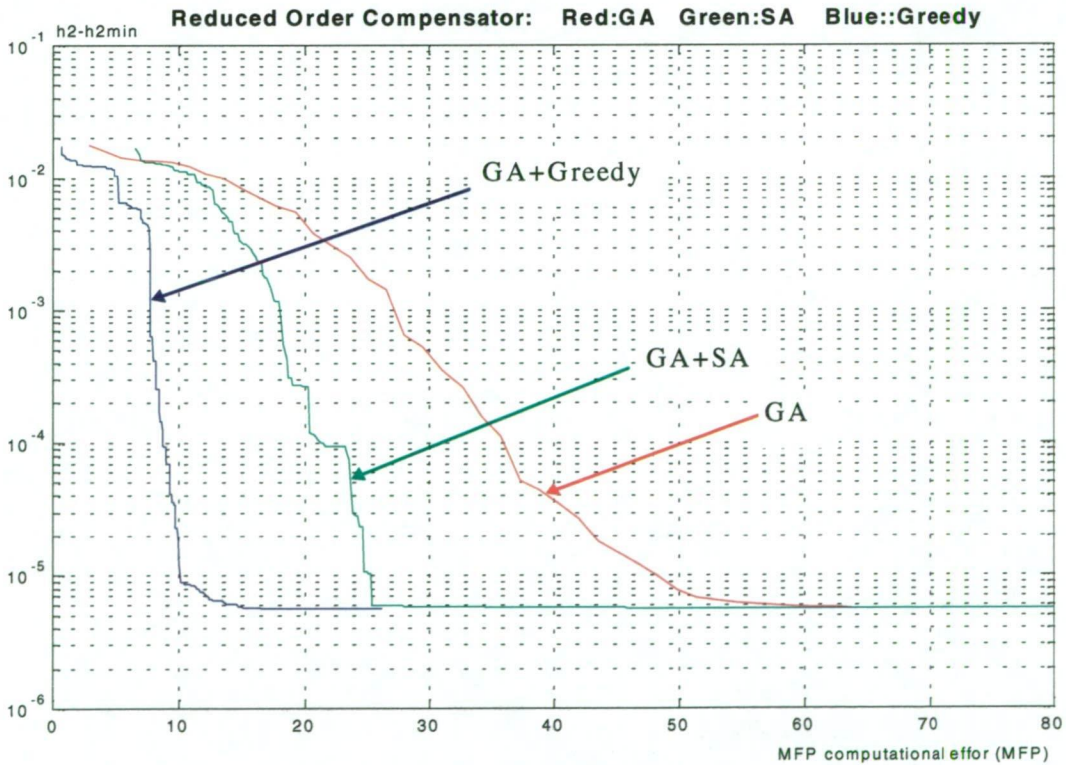


Fig.5.12.B
Convergence rates for: GA, GA+SA, GA+GS (reduced order compensator)

5.3 H_∞ Controller Synthesis:

5.3.1 Simulation Setup:

For these next set of simulations, we apply hybrid genetic algorithms to the design of full order and reduced order H_∞ compensators. Results are compared with conventional state space solutions and model reduction techniques.

(i) **Plant Model:** For this simulation, the same plant model as used in the previous section 5.2 is applied to the H_∞ compensator design:

$$\left. \begin{aligned} \dot{x} &= A \cdot x + B_1 \cdot w + B_2 \cdot u \\ z_2 &= C_1 \cdot x + D_{11} \cdot w + D_{12} \cdot u \\ z_\infty &= C_2 \cdot x + D_{21} \cdot w + D_{22} \cdot u \\ y &= C_3 \cdot x + D_{31} \cdot w + D_{32} \cdot u \end{aligned} \right\} \quad \text{Eqn.5.35}$$

This is illustrated in figure 5.13 below:

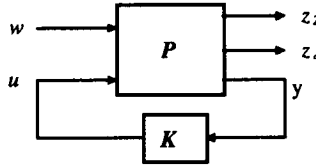


Fig.5.13
Simulation Setup Plant/Compensator

In this instance, we wish to minimize the transfer function from the exogenous input w to the performance output z_∞ . For this part, the z_2 is ignored and only the transfer function from $w \rightarrow z_\infty$ is considered. The transfer function is denoted by: $T_{z_\infty w}(s)$, and plant matrices are identical to the ones in the previous section.

(ii) **Compensator:** Again, a compensator in state-space is sought with the form:

$$\left. \begin{aligned} \dot{x}_c &= A_c \cdot x_c + B_c \cdot y \\ u &= C_c \cdot x_c \end{aligned} \right\} \quad \text{Eqn.5.36}$$

In transfer function form:

$$K(s) = \frac{c_2 \cdot s^2 + c_1 \cdot s + c_0}{s^3 + a_2 \cdot s^2 + a_1 \cdot s + a_0} \quad \text{Eqn.5.37}$$

Where equations 5.32 and 5.33 are related using a minimal realization in reachable canonical form as previously described in section 5.2.

(iii) **Closed Loop System:** The closed loop system is obtained by combining equations 5.22 with 5.23 into one single augmented system:

$$\left. \begin{aligned} \dot{\tilde{x}} &= \tilde{A} \cdot \tilde{x} + \tilde{B} \cdot w \\ z_2 &= \tilde{C}_2 \cdot \tilde{x} \\ z_\infty &= \tilde{C}_\infty \cdot \tilde{x} \end{aligned} \right\} \quad \text{Eqn.5.38}$$

The system is now in input/output form, where the matrices are given by equations 5.27. For the H_∞ simulation, only output z_∞ is considered. Simulation results are given in the following pages.

5.3.2 Conventional State Space Solution:

(i) **Full order Compensator:** The above H_∞ problem can be readily solved using MATLAB's **hinf()** function. The full order compensator which minimizes the H_∞ norm from input w to output z_2 is given by (both in pole-zero and polynomial forms):

$$\left. \begin{aligned} K(s) &= \frac{-0.36083(s-3.124)(s+5.387)}{(s+5.118)(s^2+3.372s+14.66)} \\ &= \frac{-0.3608s^2 - 0.8165s + 6.073}{s^3 + 8.49s^2 + 31.92s + 75.04} \end{aligned} \right\} \quad \text{full order compensator}$$

The open loop H_∞ norm is **1.52705**, and the closed loop: **0.58021**. The presence of zero in the positive half plane (3.124) results in a minimum phase compensator. The compensator eigenvalues from $K(s)$ above are given by: **-1.6862 ± j3.4379, -5.118**. Open and closed loop singular value plots of $\sigma(T_{z_2w}(j\omega))$ are plotted in figure 5.14 below:

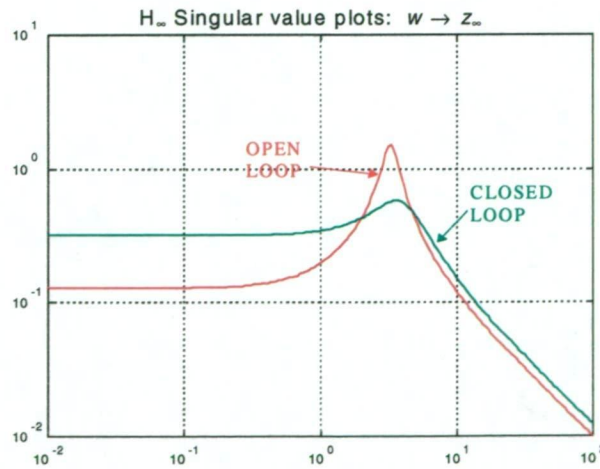


Fig.5.14

Open loop and closed loop singular value plot of $T_{z_2w}(s)$

(ii) **Reduced order Compensator:** Model reduction can be used to obtain a reduced order compensator (2nd order). To perform model reduction the MATLAB function **balreal** is first applied to the compensator which produces a balanced realization of the compensator. In this case, the diagonal entries of the joint gramian are: **0.1082, 0.0682, 0.0005**. Since the last state is weakly coupled to the input/output, this state can be removed by using the MATLAB function **modred**. This yields the following second order compensator:

$$K(s) = \frac{-0.3665(s - 3.114)}{(s^2 + 3.303s + 14.3)} \quad \left. \vphantom{\frac{-0.3665(s - 3.114)}{(s^2 + 3.303s + 14.3)}} \right\} \text{reduced order compensator}$$

$$= \frac{-0.3665s + 1.141}{s^2 + 3.303s + 14.3}$$

With the reduced order compensator, the H_∞ norm of the closed loop system is **0.58124**. In this case the effect of model reduction only mildly degrades the performance of the closed loop transfer function $T_{z\infty w}$. A singular value plot of the open loop, closed loop using full order and reduced order compensator is shown in figure 5.15 below:

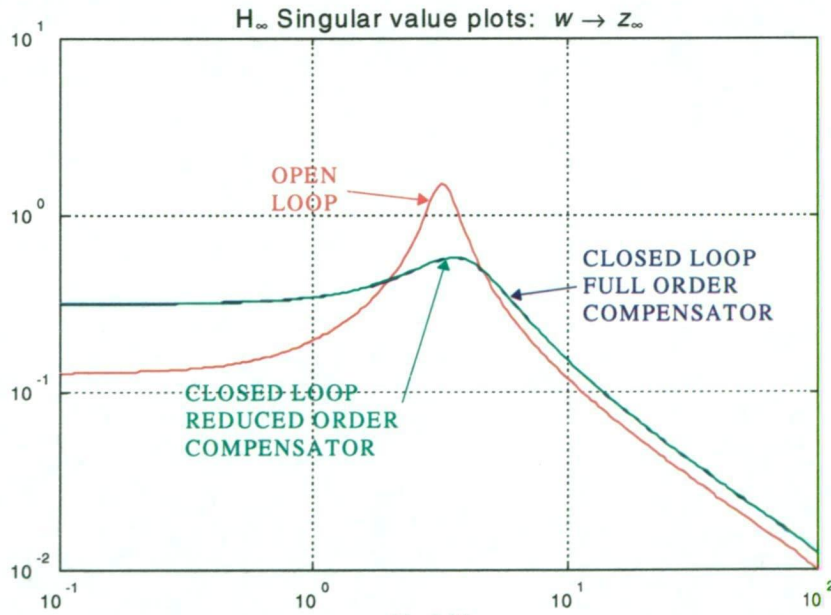


Fig.5.15
Open loop and closed loop using full order and reduced order compensator.

The singular value plot of the closed loop system with a full order compensator (plotted in blue) is overlapped by the closed loop plot using the reduced order compensator (plotted in green). In this instance, the model reduction works well due to the presence of a very weakly coupled state (0.0005).

Table 5.5 below summarizes the results with conventional H_∞ design:

System:	H_∞
Open Loop:	1.52705
Closed Loop (Full order compensator):	0.58021
Closed Loop (Reduced order compensator):	0.58124

Table 5.5

5.3.3 Solution Using Genetic Algorithms:

The above H_∞ problem is now solved using hybrid genetic algorithms as described earlier in section 5.1.3. Both full order and reduced order compensators will be implemented and results compared with those obtained using conventional methods above.

(i) **Full order Compensator:** Solution using genetic algorithms to the H_∞ problem was described in section 5.1.3, this is briefly summarized below. The full order compensator is given by equation 5.29 above, the objective is to minimize the H_∞ norm of the following closed loop transfer function:

$$T_{z\infty w}(s) = \tilde{C}_\infty (sI - \tilde{A})^{-1} \tilde{B} \quad \text{Eqn.5.39}$$

where the composite matrices are:

$$\tilde{A} = \begin{bmatrix} A & -B_2.C_c \\ B_c.C_3 & A_c \end{bmatrix} \quad \tilde{B} = \begin{bmatrix} B_1 \\ B_c.D_{31} \end{bmatrix} \quad \tilde{C}_\infty = [C_2 \quad D_{22}.C_c] \quad \text{Eqn.5.40}$$

This can be accomplished by minimizing the trace of the matrix:

$$\|H\|_\infty^2 = \text{trace}(Q_\infty . \tilde{B} . \tilde{B}^T) \quad \text{Eqn.5.41}$$

subject to the constraint:

$$\tilde{A}^T Q_\infty + Q_\infty . \tilde{A} + \tilde{C}^T . \tilde{C} + \gamma^{-2} Q_\infty \tilde{B} . \tilde{B}^T Q_\infty = 0 \quad \text{Eqn.5.42}$$

where Q_∞ is the observability gramian, given from the solution to the Riccati equation. This is essentially a constrained optimization problem with one optimization function and one constraint. Thus to find the solution, we require to find the minimum value of γ which will minimize equation 5.41 subject to the constraint given by equation 5.42. The H_∞ is then given by: $\|T_{zw}\| < \gamma$. Note also the presence of further constraints such as $Q_\infty > 0$, and stability of closed loop system which are discussed in section 5.2.

Unlike the H_2 problem, the H_∞ is not truly optimal, this is because we are attempting to minimize both γ and equation 5.41. This can be viewed as a multiobjective problem, and as such may lead to a family of solutions. This problem can be solved in a number of ways, using Pareto optimality, Nash equilibria or composite cost function method (see chapter 1.5).

The simplest is to use a composite function as follows: $f_x = (h_x + \alpha \gamma)$, where $h_x = \text{trace}(Q_\infty \cdot \tilde{B} \cdot \tilde{B}^T)$, with alpha being made to vary to see the effects of adding more emphasis on one parameter against the other.

The complete genetic algorithm is summarized below, there are two possible alternative simulations: single objective and multiobjective:

Single - Objective:

The single objective genetic algorithm is to find the compensator parameters $\{a_0, a_1, a_2, c_0, c_1, c_2\}$ which will minimize the following function:

$$\begin{aligned} \text{minimize: } & \|H\|_\infty^2 = \text{trace}(Q_\infty \cdot \tilde{B} \cdot \tilde{B}^T) \\ \text{subject to } & \gamma < \gamma_0, \text{ where } \gamma_0 \text{ is a user specified design goal} \end{aligned}$$

Multi - Objective:

The multi-objective genetic algorithm is to find the compensator parameters $\{a_0, a_1, a_2, c_0, c_1, c_2\}$ which will minimize the following function:

$$\text{minimize: } \|H\|_\infty^2 = \text{trace}(Q_\infty \cdot \tilde{B} \cdot \tilde{B}^T) + \alpha \cdot \gamma$$

Subject to the additional following constraints:

1. The existence of the solution to the Riccati equation:

$$\tilde{A}^T Q_\infty + Q_\infty \cdot \tilde{A} + \tilde{C}^T \cdot \tilde{C} + \gamma^{-2} Q_\infty \tilde{B} \cdot \tilde{B}^T Q_\infty = 0$$
, where Q_∞ is positive definite symmetric ie: $Q_\infty > 0$.
2. The closed loop system must be internally stable, ie: eigenvalues of \tilde{A} must be stable. Controller must be stable, eigenvalues of A_c must be < 0 . No eigenvalues on the $j\omega$ axis (ie marginally stable closed loop system).
3. Verify the stability of the Riccati Solution thus: $A_r = \tilde{A} + \gamma^{-2} \cdot \tilde{B} \cdot \tilde{B}^T \cdot Q_\infty$, where A_r must be positive definite.

Fig.5.16

Thus, the problem can be defined and solved in several different ways, simulation results are given in the following pages. The chromosomal representation for this problem is illustrated below.

a_2	a_1	a_0	c_2	c_1	c_0	γ	h_x	f_x	<i>Fitness</i>
-------	-------	-------	-------	-------	-------	----------	-------	-------	----------------

Fig.5.17
Chromosomal Representation for the H_∞ problem

The genetic algorithm conducts a search over $\{a_0,a_1,a_2,c_0,c_1,c_2,\gamma\}$ using real number codification, and where: $hx = trace(Q_\infty.\tilde{B}.\tilde{B}^T)$, the composite cost functional: $f_x=(h_x+\alpha\times\gamma)$, and the fitness is the inverse: $Fitness=1/f_x$. The presence of the parameter α can be used to see the effects on rate of convergence and final solution by varying alpha. Note that if any of the constraints are violated, then the solution is infeasible and the fitness is made zero. The solution to the Riccati equation Q_∞ is found using a Hamiltonian matrix approach. Because this problem is essentially a constrained optimization problem, a second method for solving it would be to define a Lagrangian function, similar to the H_2 problem discussed in section 5.2.3. Alternatively, rather than attempting to find the minimum value of γ which will minimize equation 5.41, we could simply set a constraint on γ (design goal), for instance $\gamma\leq0.7$, and only minimize equation 5.41. This problem then becomes a single objective rather than multiobjective constrained optimization problem. Simulation results using the GA algorithm-2 from the previous page are tabulated below. For this algorithm we used: binary tournament selection, crossover probability $P_c=0.5$, mutation probability $P_m=0.2$, full order compensator, population size=60, uniform weighted average crossover.

(a) Single Objective: design goal: $\gamma\leq0.7$:

In this simulation, we minimize equation 5.37 and set a constraint on gamma thus: $\gamma\leq0.7$. We ran the simulation 5 times due to illustrate the probabilistic nature of the GA, results are tabulated in Table-5.6 below:

gen:	a_2	a_1	a_0	c_2	c_1	c_0	H_∞
	8.49	31.92	75.04	-0.36	-0.816	6.073	0.5802
500	8.62	34.13	75.25	-0.447	-0.893	6.650	0.5511
500	8.69	34.42	76.41	-0.447	-0.928	6.754	0.5510
500	8.47	33.64	73.20	-0.445	-0.839	6.488	0.5515
500	8.43	33.46	72.49	-0.446	-0.816	6.420	0.5513
500	8.40	33.28	71.81	-0.447	-0.791	6.349	0.5511

Table 5.6

Figure 5.18 below compares the singular value plot of $T_{z_\infty w}$ using conventional state space and genetic algorithms for the design goal: $\gamma \leq 0.7$. Results are nearly identical.

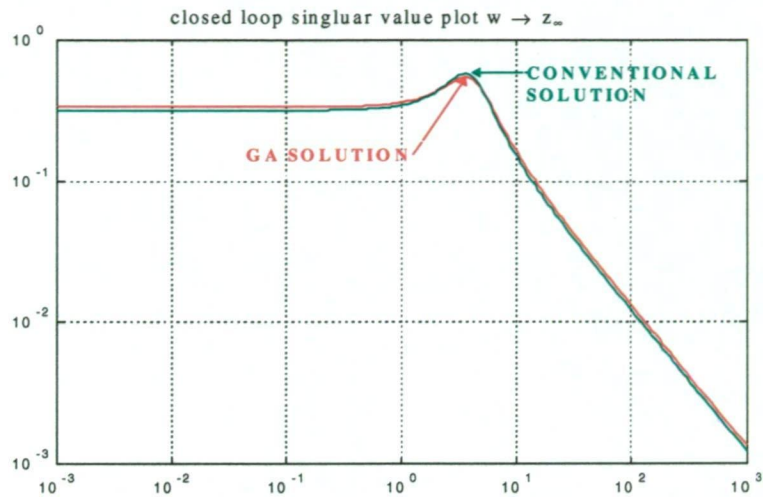


Fig.5.18
Comparing singular value plots: GA and Conventional Methods

This next plot illustrates the effects of decreasing the value of gamma. Values of gamma are: 1, 0.9, 0.8, 0.7, 0.6. The green plot is for $\gamma=1$, with the red plot furthest away at $\gamma=0.6$.

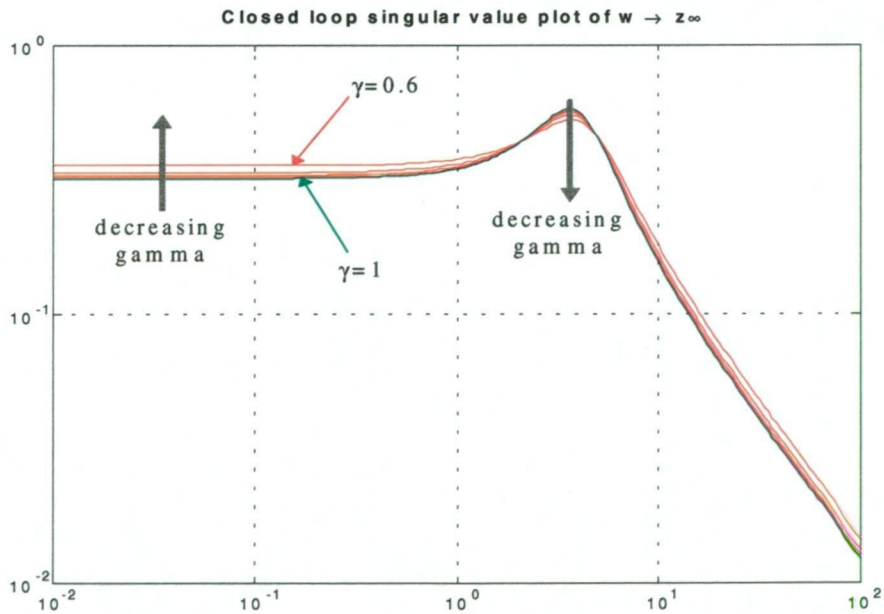


Fig.5.19

Because of the weakly coupled state (0.0005), a family of compensators can be implemented which will satisfy the requirements for $\gamma \leq 0.7$ or any other value of gamma. For gamma $\gamma \leq 0.5$, no solution exists.

(b) **Multiobjective: minimize $f_x=(hx+\alpha \times \gamma)$**

In this simulation, we minimize the composite cost functional which defines a multiobjective problem. We ran the simulation for different values of α to see the effects of α on the final solution. Note the additional constraint of $\gamma < 1$. Results after 500 generations are tabulated in Table-5.7 below:

a_2	a_1	a_0	c_2	c_1	c_0	H_∞	Alpha:
8.49	31.92	75.04	-0.36	-0.816	6.073	0.5802	
8.00	32.32	64.54	-0.48	-0.588	5.902	0.5406	1.0
8.33	33.11	71.10	-0.44	-0.779	6.301	0.5501	0.1
8.01	37.51	59.75	-0.72	-0.600	7.166	0.4865	10

Table 5.7

From table 5.7, in the first simulation, the value of $\alpha=1$ places equal emphasis on both minimizing the function $hx = \tilde{C}_\infty (sI - \tilde{A})^{-1} \tilde{B}$ and gamma. In the second simulation less emphasis is placed on gamma, and in the third simulation greater emphasis is placed on minimizing gamma. The singular value plots of the three simulations is shown in figure 5.20 below:

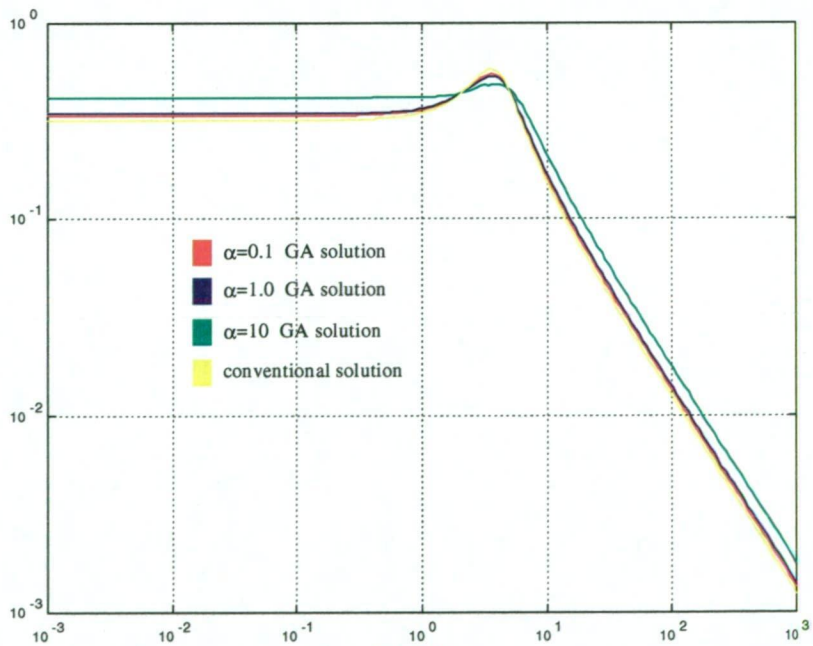


Fig.5.20
Comparing singular value plots: GA and Conventional Methods

Whilst all solutions give an almost identical singular value plots, some shaping can be achieved by the choice of α . In particular, larger values of $\alpha=10$ give a flatter response (green curve). Values of $\alpha < 1$ have little influence on the response.

Figure 5.21 on the following below illustrates a typical convergence plot, convergence is generally within the first 100 generations of the GA.

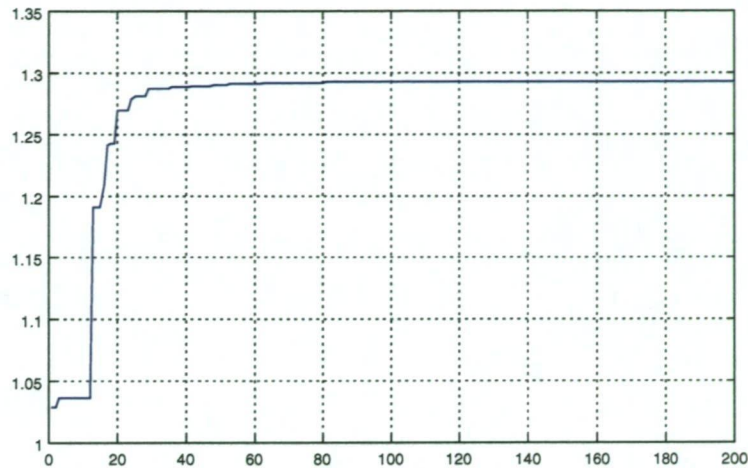


Fig.5.21
Convergence of the genetic algorithm

(ii) **Reduced order Compensator:** A reduced order compensator (2nd order) can also be implemented directly by defining the compensator matrices as:

$$A_c = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix} \quad B_c = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad C_c = [c_0 \quad c_1] \quad \text{Eqn.5.43}$$

Again, the simulation is repeated for this system, and results are given on the following pages.

(a) Single Objective: design goal: $\gamma \leq 0.70$:

In this simulation, we minimize equation 5.41 and set a constraint on gamma thus: $\gamma \leq 0.7$, results are tabulated in Table-5.8 below. From table 5.8, we can see that the results using genetic algorithms agree with the results obtained using conventional methods (shown in red in the first row of table 5.8). For this algorithm we used: binary tournament selection, crossover probability $P_c=0.5$, mutation probability $P_m=0.2$, population size=60, uniform weighted average crossover, generations=1000. We ran the simulation 5 times to illustrate the probabilistic nature of the GA. In this case, there is less variation in convergence when compared to the full order compensator (table 5.7), i.e. the GA algorithm converges to the same solution in all cases. The red values in table 5.8 are the compensator coefficients obtained using MATLAB's H_∞ design and model reduction. The results obtained using genetic algorithms produce a compensator with a lower H_∞ values.

gen:	a_2	a_1	a_0	c_2	c_1	c_0	H_∞
		3.303	14.3		-0.366	1.141	0.58124
1000	0.000	3.863	15.5	0.000	-0.462	1.364	0.55130
1000	0.000	3.822	15.4	0.000	-0.456	1.352	0.55098
1000	0.000	3.849	15.5	0.000	-0.460	1.360	0.55160
1000	0.000	3.858	15.5	0.000	-0.461	1.362	0.55102
1000	0.000	3.861	15.5	0.000	-0.461	1.364	0.55113

Table 5.8

Figure 5.22 below compares the singular value plot of the reduced order compensator obtained using genetic algorithms with conventional model reduction techniques:

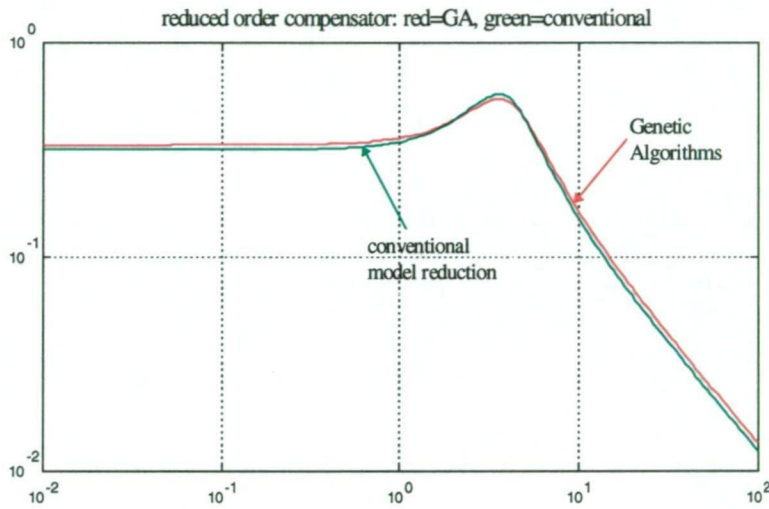


Fig.5.22

Comparing singular value plots: GA and Conventional Methods Reduced order compensator

Figure 5.22 shows that whilst the genetic algorithm compensator has a slightly lower peak (H_∞ norm), it has a higher singular value over most of the frequency range from 10^{-2} to 10^{+2} .

(b) Multiobjective: minimize $f_x=(h_x+\alpha \times \gamma)$

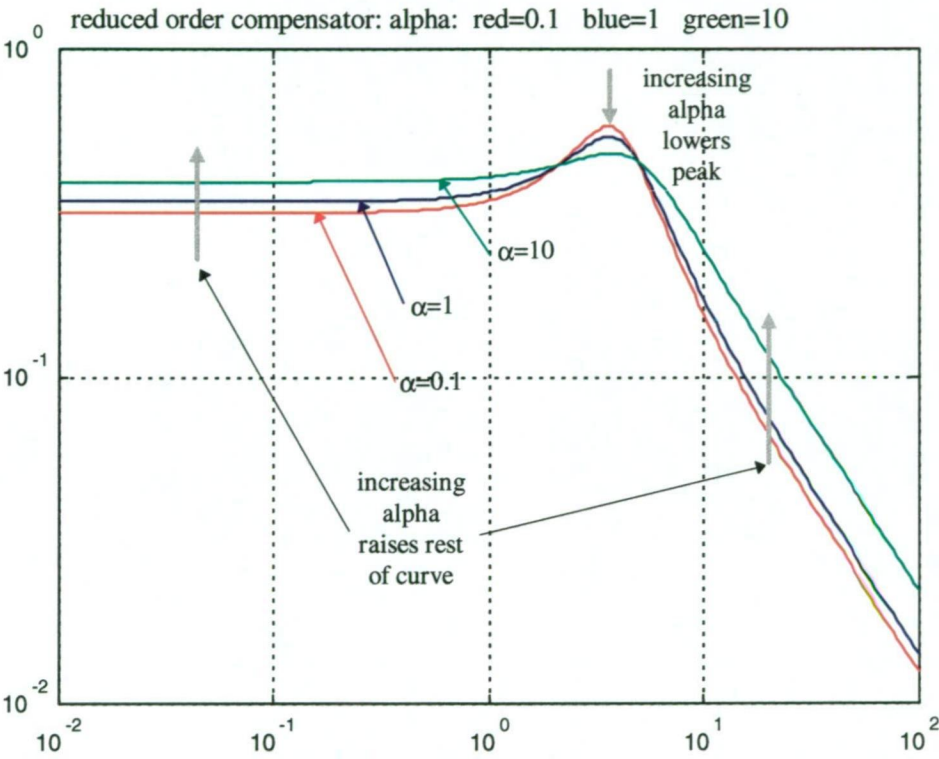
In this simulation, we minimize the composite cost functional for 3 different values of α : (1, 0.1, 10). Results for this simulation after 1000 generations are tabulated in table-5.9 below:

a_2	a_1	a_0	c_2	c_1	c_0	γ	H_∞	α
	3.303	14.3		-0.366	1.141		0.58124	
0.000	4.032	15.9	0.000	-0.494	1.441	0.6485	0.54139	$\alpha=1.0$
0.000	3.337	14.3	0.000	-0.369	1.146	1.0000	0.58123	$\alpha=0.1$
0.000	6.599	22.4	0.000	-0.991	2.477	0.4931	0.48132	$\alpha=10$

Table 5.9

Comparing the results of the simulation from table-5.9, we can clearly see that when $\alpha=0.1$, the compensator obtained with genetic algorithms is identical to that obtained with conventional model reduction techniques. This indicates that the choice of α should be less than one if implementing reduced order compensators with genetic algorithms.

Figure 5.23 below is a singular value plot of the results obtained using genetic algorithms for the reduced order compensator for the 3 different values of α :



Comparing singular value plots for different alphas.

From figure 5.23, increasing the value of α simply places more emphasis or penalty on γ , thus it would be expected that for large values of α , the peak of the singular value plot is lowered at the expense of raising the rest of the curve.

In summary, implementing H_∞ compensators with genetic algorithms, there are two possible scenarios, the first would be to minimize equation 5.41 subject to the constraint given by equation 5.42. The compensator would then have the property: $H_\infty < \gamma$. Note that if γ is chosen too small, a solution may not exist. A second method is to minimize the composite cost functional: $f_x=(h_x+\alpha \times \gamma)$, where $h_x = trace(Q_\infty.\tilde{B}.\tilde{B}^T)$, with α set to less than one.

5.3.4 Convergence Rates for Hybrid Genetic Algorithms:

This last set of simulations compares the convergence rates of the three different hybrid genetic algorithms: (a) conventional genetic algorithms, (b) genetic algorithms and simulated annealing, and (c) genetic algorithms and greedy search. In all cases, the convergence rates differ significantly.

Figure 5.24 below compares the convergence rate of the three algorithms for the full order compensator. The red plot is the conventional genetic algorithm, the green plot is the hybrid genetic algorithm + simulated annealing, and the blue plot is the hybrid genetic algorithm + greedy search strategy. Both hybrid methods converge much more rapidly compared with the conventional genetic algorithm.

Full order compensator:

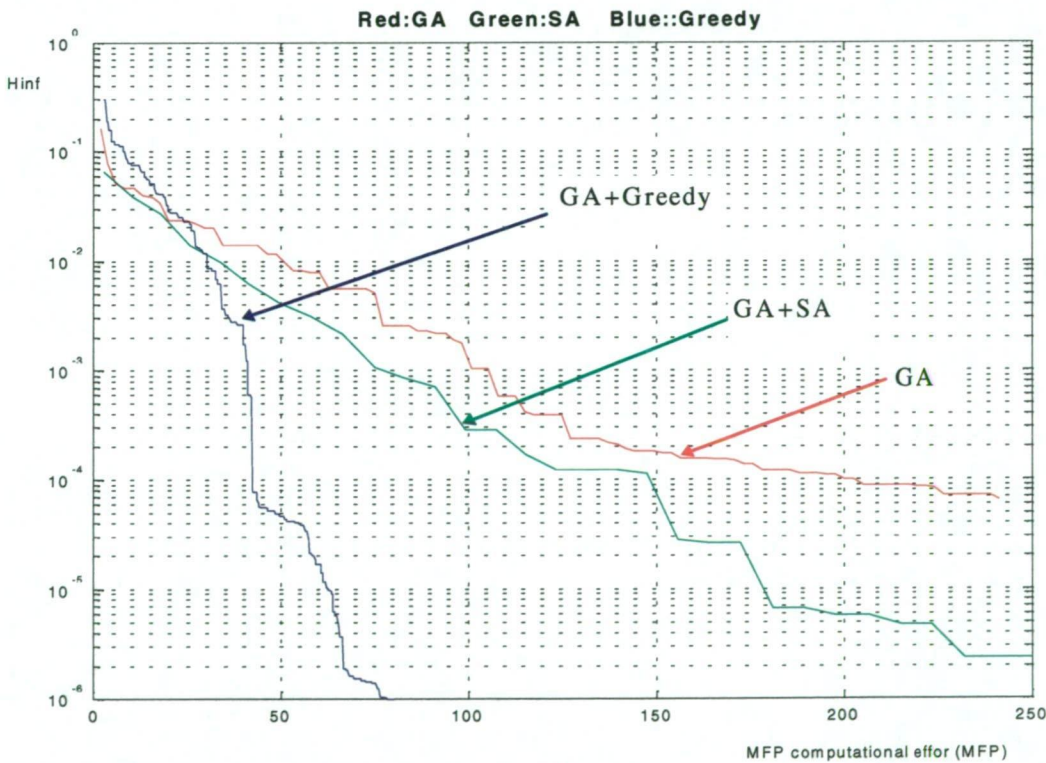


Fig.5.24

Figure 5.25 below shows the convergence properties of the 3 genetic algorithm methods for the reduced order compensator.

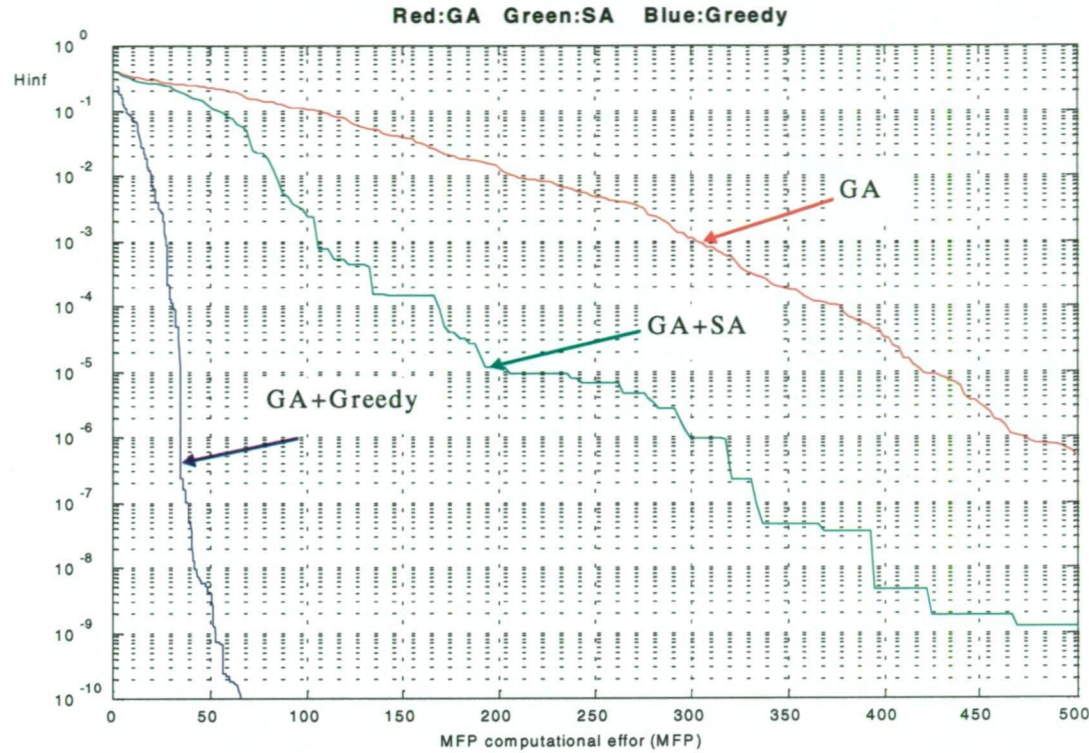


Fig.5.25

5.4 Mixed H_2/H_∞ Controller Synthesis:

5.4.1 Simulation Setup:

For the last set of simulations, we apply genetic algorithms to the design of full order and reduced order compensators with mixed H_2/H_∞ specifications. This is in essence a combination of the two previous methods, and represents a multiobjective optimization problem. Currently there is no direct design solution to this problem. The only two iterative numerical optimization methods are *Homotopy theory*, and *linear matrix inequalities* (a convex optimization approach). These were discussed earlier in section 5.1.4.

(i) **Plant Model:** For this simulation, the same plant model as used in the previous section 5.2 is used for H_∞ design:

$$\left. \begin{aligned} \dot{x} &= A \cdot x + B_1 \cdot w + B_2 \cdot u \\ z_2 &= C_1 \cdot x + D_{11} \cdot w + D_{12} \cdot u \\ z_\infty &= C_2 \cdot x + D_{21} \cdot w + D_{22} \cdot u \\ y &= C_3 \cdot x + D_{31} \cdot w + D_{32} \cdot u \end{aligned} \right\}$$

Eqn.5.45

This is illustrated in figure 5.26 below:

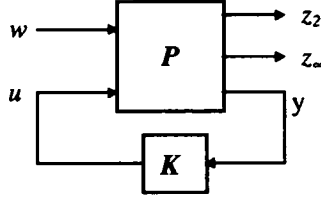


Fig.5.26
Simulation Setup Plant/Compensator

The objective is to minimize the H_2 norm from the exogenous input w to the performance output z_2 . Additionally, we wish to either minimize the H_∞ from w to z_∞ , or a more relaxed approach would be to satisfy the constraint: $H_\infty < \gamma$, where γ is some design goal. Again, to minimize the H_2 norm we simply minimize the function:

$$f_2 = Q_2 \cdot \tilde{B} \cdot \tilde{B}^T \quad \text{Eqn.5.46}$$

and to minimize the H_∞ , norm we minimize the function:

$$f_\infty = \text{trace}(Q_\infty \cdot \tilde{B} \cdot \tilde{B}^T) + \alpha \cdot \gamma \quad \text{Eqn.5.47}$$

(ii) **Compensator:** Again, a compensator in state-space is sought with the form:

$$\left. \begin{aligned} \dot{x}_c &= A_c \cdot x_c + B_c \cdot y \\ u &= C_c \cdot x_c \end{aligned} \right\} \quad \text{Eqn.5.48}$$

In transfer function form:

$$K(s) = \frac{c_2 \cdot s^2 + c_1 \cdot s + c_0}{s^3 + a_2 \cdot s^2 + a_1 \cdot s + a_0} \quad \text{Eqn.5.49}$$

Where equations 5.43 and 5.44 are related using a minimal realization in reachable canonical form as previously described in section 5.2 (equations 5.28).

(iii) **Closed Loop System:** The closed loop system is obtained by combining equations 5.22 with 5.23 into one single (augmented) system:

$$\left. \begin{aligned} \dot{\tilde{x}} &= \tilde{A} \cdot \tilde{x} + \tilde{B} \cdot w \\ z_2 &= \tilde{C}_2 \cdot \tilde{x} \\ z_\infty &= \tilde{C}_\infty \cdot \tilde{x} \end{aligned} \right\} \quad \text{Eqn.5.50}$$

The system is now in input/output: $w \rightarrow \{ z_2, z_\infty \}$ form, where the matrices are given by equations 5.27. In the mixed H_2/H_∞ simulation, both z_∞ and z_2 outputs must be considered.

5.4.2 Solution Using Genetic Algorithms:

Both full order and reduced order compensators will be implemented and results compared with those obtained using conventional methods above. The complete genetic algorithm is summarized below:

Multi-Objective:

The multi-objective genetic algorithm is to find the compensator parameters: $\{a_0, a_1, a_2, c_0, c_1, c_2, \gamma\}$ which will minimize the following functions:

$$f_2 = Q_2 \cdot \tilde{B} \cdot \tilde{B}^T$$

$$f_\infty = \text{trace}(Q_\infty \cdot \tilde{B} \cdot \tilde{B}^T) + \alpha \cdot \gamma$$

Subject to the additional following constraints:

1. The existence of the solution to the Riccati equation, where Q_∞ is positive definite symmetric: $Q_\infty > 0$.

$$\tilde{A}^T Q_\infty + Q_\infty \cdot \tilde{A} + \tilde{C}_\infty^T \cdot \tilde{C}_\infty + \gamma^{-2} Q_\infty \tilde{B} \cdot \tilde{B}^T Q_\infty = 0,$$

2. The existence of the solution to the Lyapunov equation: where Q_2 is positive definite symmetric: $Q_2 > 0$.

$$\tilde{A}^T Q_2 + \tilde{Q}_2 \cdot A + \tilde{C}_2^T \cdot \tilde{C}_2 = 0$$

3. The closed loop system must be internally stable, ie: eigenvalues of \tilde{A} must be stable. Controller must be stable, eigenvalues of A_c must be < 0 . No eigenvalues on the $j\omega$ axis (ie: marginally stable closed loop system).

4. Verify the stability of the Riccati solution, where A_r must be positive definite:

$$A_r = \tilde{A} + \gamma^{-2} \tilde{B} \cdot \tilde{B}^T Q_\infty$$

Fig.5.27

(i) **Full order Compensator:** The chromosomal representation for this problem is illustrated below:

a_2	a_1	a_0	c_2	c_1	c_0	γ	f_2	f_∞	<i>Fitness</i>
-------	-------	-------	-------	-------	-------	----------	-------	------------	----------------

Fig.5.28

Chromosomal Representation for the mixed H_2/H_∞ problem

The genetic algorithm conducts a search over $\{a_0, a_1, a_2, c_0, c_1, c_2, \gamma\}$ using real number codification, and where: the fitness is given by the inverse of the composite cost functional:

$$Fitness = \frac{1}{f_2 + \kappa \cdot f_\infty} \quad \text{Eqn.5.51}$$

If the solution is feasible, however if any of the constraints are violated, then the solution is infeasible and the fitness is made zero. Note the additional constraint $\gamma < 1$. The presence of the parameter $\kappa (>0)$ can be used to see the effects of the final solution by varying relative emphasis on the H_2 or H_∞ components. In the following simulations, the value of κ is set to: 0.1, 1, 10, the value of alpha is fixed $\alpha=0.1$. For this algorithm we used: binary tournament selection, crossover probability $P_c=0.5$, mutation probability $P_m=0.2$, population size=50, uniform weighted average crossover, generations=1000. Results are tabulated in Table-5.10, 5.11, 5.12 for values of $\kappa=0.1, 1, 10$ respectively below:

$\kappa=0.1$: In this simulation, more emphasis is placed upon minimizing the H_2 norm of the $w \rightarrow z_2$ transfer function by choosing $\kappa=0.1$. Subsequently, the results would give a compensator which is closer to the H_2 compensator obtained in section 5.2. Looking at figure 5.29, the top graph compares the response of the mixed H_2/H_∞ compensator with that of only the H_2 compensator obtained in section 5.2, the bottom graph compares the response of the mixed H_2/H_∞ compensator with that of only the H_∞ compensator obtained in section 5.3. We can see that the closed loop response matches the H_2 response better as more emphasis was placed on minimizing the f_2 function. Note the additional constraint that $\gamma \leq 1$. Total iterations=400.

$\kappa=1.0$: In this simulation, equal emphasis is placed on both minimizing the H_2 norm of the $w \rightarrow z_2$ transfer function and the H_∞ norm of the $w \rightarrow z_\infty$ transfer function by choosing $\kappa=1$. Subsequently, the results would give a compensator which is a compromise between the H_2 compensator obtained in section 5.2, and the H_∞ compensator obtained in section 5.3.

Looking at figure 5.30, the top graph compares the response of the mixed H_2/H_∞ compensator with that of only the H_2 compensator obtained in section 5.2, the bottom graph compares the response of the mixed H_2/H_∞ compensator with that of only the H_∞ compensator obtained in section 5.3. We can see that the closed loop response is an attempt to simultaneously match both the H_2 response and H_∞ . Total iterations=400 in each simulation.

$\kappa=10$: In this simulation, more emphasis is placed upon minimizing the H_∞ norm of the $w \rightarrow z_\infty$ transfer function by choosing $\kappa=10$. Subsequently, the results would give a compensator which is closer to the H_∞ compensator obtained in section 5.3. Looking at figure 5.31, the top graph compares the response of the mixed H_2/H_∞ compensator with that of only the H_2 compensator obtained in section 5.2, the bottom graph compares the response of the mixed H_2/H_∞ compensator with that of only the H_∞ compensator obtained in section 5.3. We can see that the closed loop response matches the H_∞ response better as more emphasis was placed on minimizing the f_∞ function. Total iterations=400.

Table 5.10 below gives the compensator coefficients for the three values of κ . Note that as the value of κ increases from 0.1 to 10, more emphasis is placed on reducing the H_∞ norm, subsequently this value is smallest when $\kappa=10$. However as the value of H_∞ decreases, the value of H_2 invariably increases. The choice of κ determines the compensator coefficients. Consequently, when synthesizing a mixed H_2/H_∞ compensator, first select the desired upper value of $H_\infty < \gamma$, and then compute the compensator coefficients to minimize the H_2 norm.

The red values of H_2 and H_∞ in table 5.10 are computed from individual H_2 and H_∞ designs (section 5.2 and 5.3) and do not represent a mixed H_2/H_∞ design. However as $\kappa \rightarrow 0$, then $H_2 \rightarrow 0.4096$, however as $\kappa \rightarrow \infty$, then $H_\infty \rightarrow 0.5802$.

κ	a_2	a_1	a_0	c_2	c_1	c_0	H_2	H_∞
							0.4096	0.5802
0.1	8.782	21.570	74.957	-0.054	0.034	3.357	0.4161	1.0000
1	8.272	26.675	71.140	-0.179	-0.346	5.111	0.4556	0.8798
10	7.949	30.786	68.717	-0.319	-0.688	5.802	0.5020	0.8591

Table 5.10

$\kappa=0.1$:

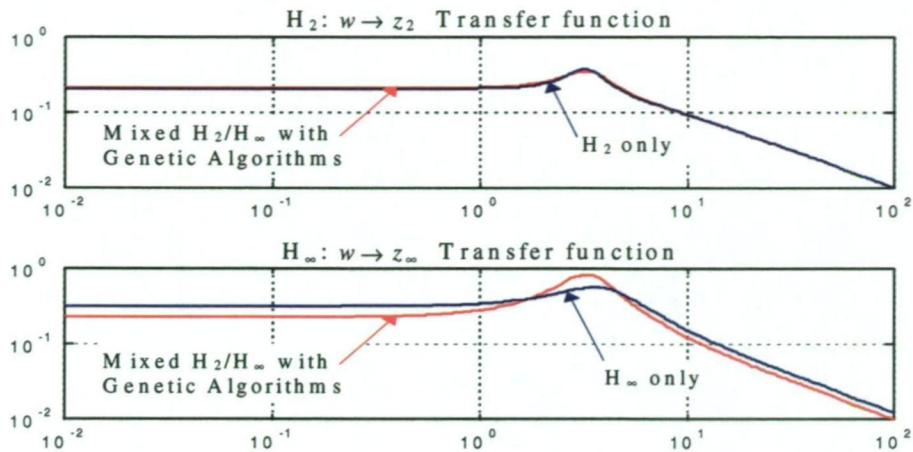


Fig.5.29

$\kappa=1$:

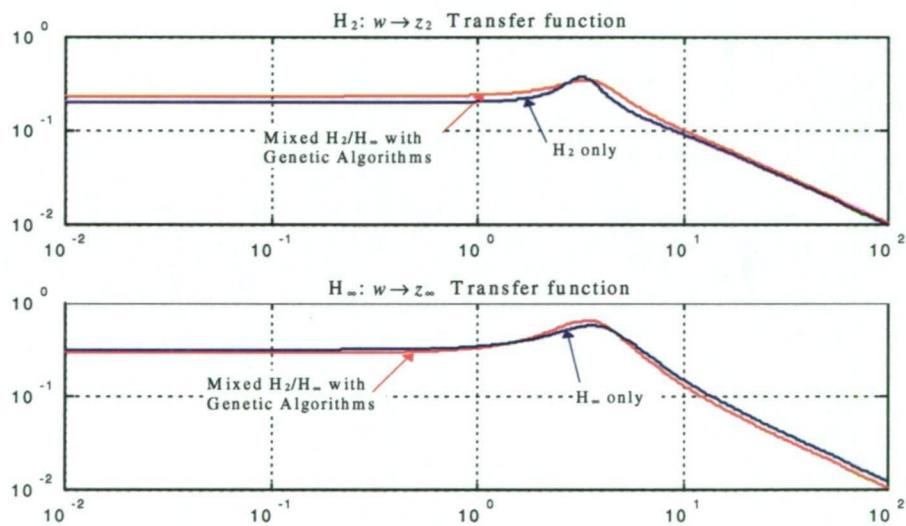


Fig.5.30

$\kappa=10$:

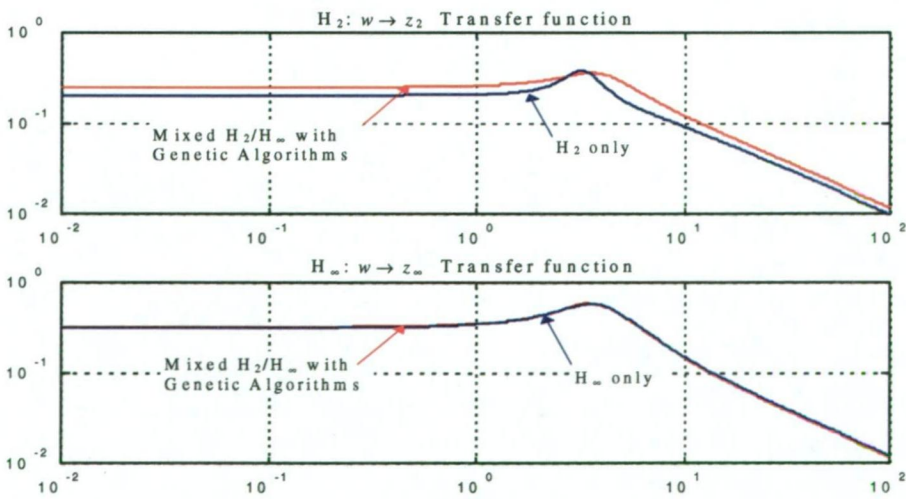


Fig.5.31

(ii) **Reduced order Compensator:** A reduced order compensator (2nd order) can be implemented directly by defining the compensator matrices as before:

$$A_c = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix} \quad B_c = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad C_c = [c_0 \quad c_1]$$

Eqn.5.52

Again, the simulation is repeated for this system, and results are tabulated in Table.5.13 on the following page. For this algorithm we used: binary tournament selection, crossover probability $P_c=0.5$, mutation probability $P_m=0.2$, population size=60, uniform weighted average crossover, generations=400.

κ	a_2	a_1	a_0	c_2	c_1	c_0	H_2	H_∞
							0.4096	0.5802
0.1	0.000	1.544	10.311	0.000	-0.057	0.458	0.4161	1.0000
1	0.000	2.477	12.009	0.000	-0.194	0.846	0.4560	0.8796
10	0.000	3.207	13.957	0.000	-0.340	1.103	0.5014	0.8591

Table 5.11

Table 5.11 above summarizes the results obtained with genetic algorithms for the reduced order compensator with values of $\kappa=0.1, 1, 10$ respectively. For each value of κ , the simulation was conducted 5 times to observe any variation in convergence, and in each instance the results were identical, consequently only one result is given for each κ . Figures 5.32, 5.33, 5.34 below compare the reduced order mixed H₂/H_∞ compensator obtained using genetic algorithms with the conventional H₂ and H_∞ compensator obtained using state space solutions. Again, with $\kappa=0.1$, the compensator is much like the H₂ compensator. With $\kappa=1$, the compensator is an in-between compromise between the H₂ and H_∞ compensator, and finally when $\kappa=10$, the compensator becomes much more like the H_∞ compensator.

$\kappa=0.1$:

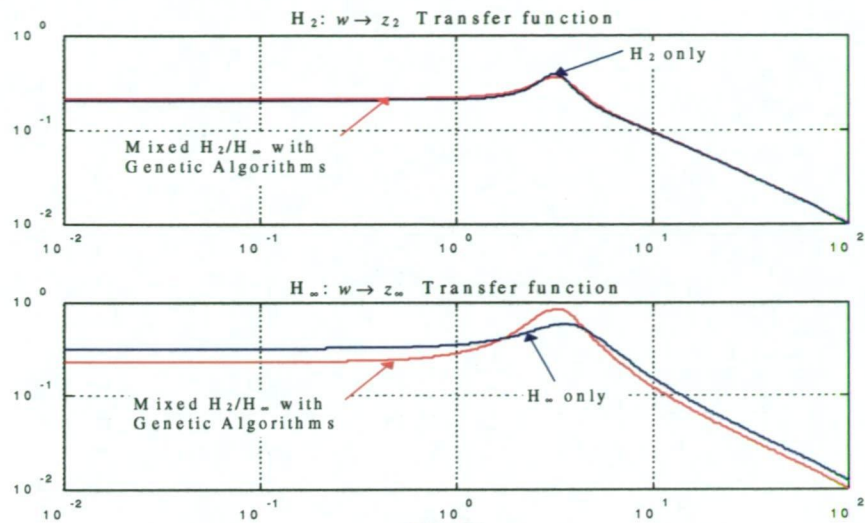


Fig.5.32

$\kappa=1$:

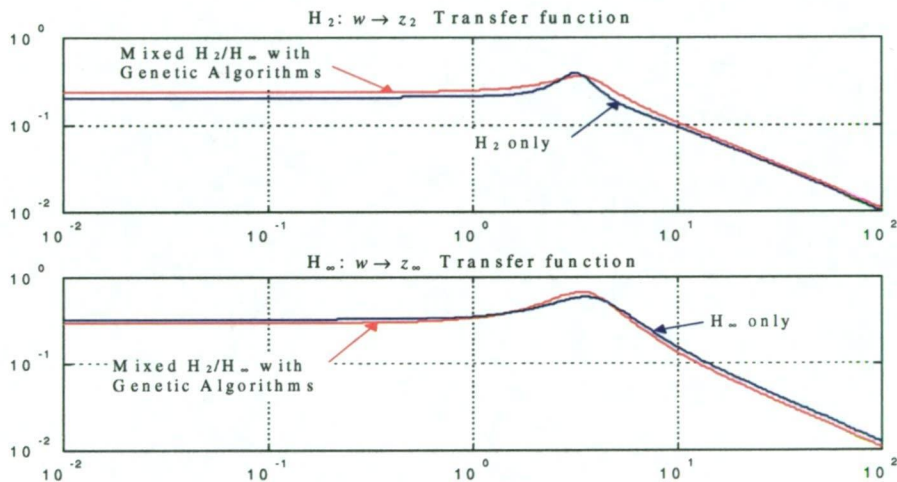


Fig.5.33

$\kappa=10$:

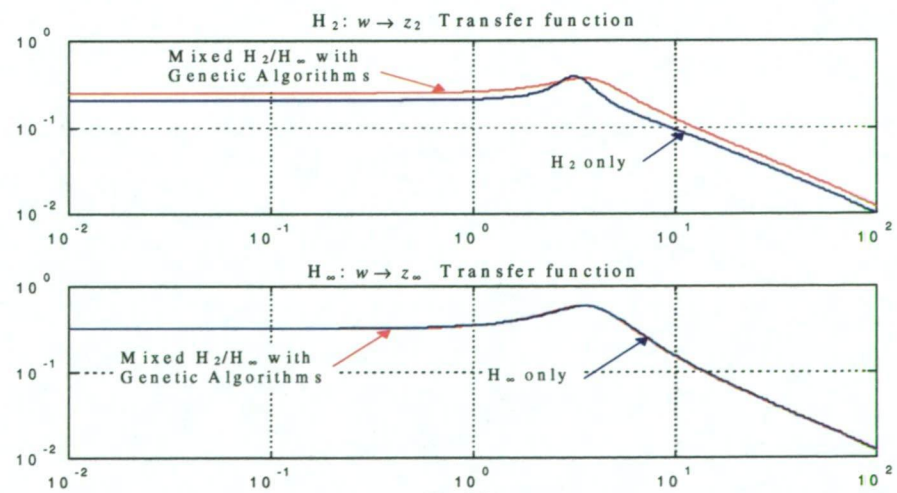


Fig.5.34

5.5 Chapter Summary and Conclusion:

(i) Summary:

Simulation results indicate that genetic algorithms can be successfully applied to the design of full order and reduced order H_2 , H_∞ , and mixed H_2/H_∞ compensators. Results agree well with those obtained using conventional state space solutions, and conventional model reduction techniques. In most cases, the GA converged within 400 generations. In all simulations we used: binary tournament selection, crossover probability $P_c=0.5$, mutation probability $P_m=0.2$, population size=50 to 100, uniform weighted average crossover. The mutation gain was gradually reduced over the simulation run for the conventional genetic algorithm. Genetic algorithms are conceptually elegant, simple and applicable to a wide range of robust control and multiobjective constrained optimization problems. In this applications, solution to the H_2 or H_∞ problem required only a single objective constrained optimization. The solution to the mixed H_2/H_∞ is a multiobjective constrained optimization problem which leads to a family of solution. By proper selection of the scalar weight κ , more or less emphasis can be placed on the optimization of either the H_2 or H_∞ specifications. This gives the user some design freedom in implementation. Note as a further extension, the scalar weight κ can be frequency dependent $\kappa(j\omega)$.

(ii) future work:

1. Replace the H_2 or H_∞ compensator with a RBF network trained using genetic algorithms, the figure below illustrates a typical setup for a H_2 optimal controller:

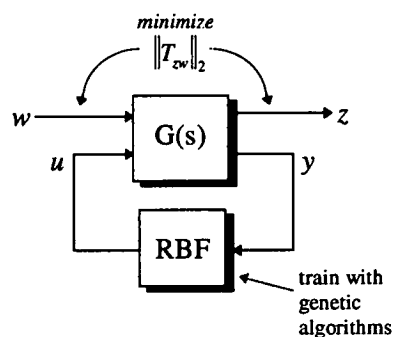


Fig.5.34
Using a RBF H_2 and H_∞ compensator

2. For the mixed H_2/H_∞ simulation, use linear matrix inequalities and convex optimization comparing solutions with genetic algorithms.
3. Addition of frequency dependent weights which can also be designed using hybrid genetic algorithms.

5.6 References and Further Reading:

- [1] J.C. Doyle, B.A. Francis, A.R. Tannenbaum
Feedback Control Theory
New York, Macmillan Publishing Co. 1992
- [2] Huibert Kwakernaak
Robust Control and H_∞ Optimization - Tutorial Paper
Automatica Vol. 29, No. 2, pp.255-273, 1993
- [3] W.S. Levine
The Control Handbook (chapter 40)
CRC Press 1996
- [4] M.Green, D.J.N. Limebeer
Linear Robust Control,
Prentice Hall Information and System Science Series, 1995
- [5] J.M. Maciejowski
Multivariable Feedback Design
Addison Wesley Publishing Company 1989
- [6] G. Zames,
Feedback and Optimal Sensitivity: Model Reference Transformations, Weighted Seminorms, and Approximate Inverses
Proc. 17th Allerton Conf., pp.744-752, 1979
- [7] J.C. Doyle, K. Glover, P.P. Khargonekar, B.A. Francis
State-Space Solutions to Standard H_∞ and H_2 Control Problems
IEEE Transactions on Automatic Control, Vol.34, No.8, pp.831-846, August 1989
- [8] J.C. Doyle,
Guaranteed Margins for LQG Regulators
IEEE Transactions on Automatic Control, Vol. AC-23, No.4, pp.756-757, August 1978
- [9] C.L. Karr, L.M. Freeman
Genetic Algorithms for H_2 Controller Synthesis
Industrial Applications of Genetic Algorithms CRC Press 1999 (pp 35-48)
- [10] M. Whorton, H. Buschek, A.J. Calise
Homotopy Algorithm for Fixed Order Mixed H_∞/H_2 Design
Journal of Guidance, Control and Dynamics, Vol.19, No.6, pp.1262-1269, Nov-Dec. 1996
- [11] M. Mercadal
Homotopy Approach to Optimal, Linear Quadratic, Fixed Architecture Compensation
Journal of Guidance, Control and Dynamics, Vol.14, pp.1224-1233, Nov-Dec. 1991
- [12] J.R. Corrado, R.S. Erwin, D.S. Bernstein, W.M. Haddad
Stable H_2 Optimal Controller Synthesis.
Optimal Control Applications and Methods, Vol.21, pp.107-124, 2000
- [13] S.S. Ge, T.H. Lee, G. Zhu
Genetic Algorithm tuning of Lyapunov Based Controllers: An Application to a Single Link Flexible Robot System.
IEEE Transactions on Industrial Electronics, Vol.43, No.5, pp.567-573, October 1996
- [14] C.I. Marrison, R.F. Stengel
Robust Control System Design Using Random Search and Genetic Algorithms
IEEE Transactions on Automatic Control, Vol.42, No.6, pp.835-839, June. 1997

-
- [15] M.C. Gemignani
Elementary Topology
Addison Wesley Publishing Company 1972, (Book)
- [16] S.L.Richtcher, R.A.DeCarlo
Continuation Methods: Theory and Applications
IEEE Transactions on Circuits and Systems, Vol.CAS-30, No.6, pp.347-352, June. 1983
- [17] J.R.Corrado, R.S.Erwin, D.S.Bernstein, W.M.Haddad
Stable H_2 Optimal Controller Synthesis
Optimal Control Applications and Methods, Vol.21, pp.107-124, 2000
- [18] E.G.Collins, L.D.Davis, S.Richter
Design of Reduced-order, H_2 Optimal Controllers Using a Homotopy Algorithm
International Journal of of Control, Vol.61, No.1, pp.97-126, 1995
- [19] M.Whorton, H.Buschek, A.J.Calise
Homotopy Algorithm for Fixed Order Mixed H_∞/H_2 Design
Journal of Guidance, Control and Dynamics, Vol.19, No.6, pp.1262-1269, Nov-Dec. 1996
- [20] B.C.Fabien
Output Feedback Stabilizing Control With an H_∞ Bound on Disturbance Attenuation
Journal of Dynamic Systems Measurement and Control, Vol.115, pp.531-535, September 1993.
- [21] E.Feron
Analysis of Robust H_2 Performance using Multiplier Theory
SIAM Journal of Control and Optimization, Vol.35, No.1, pp.160-177, January 1997
- [22] X.Chen, J.T.Wen
A Linear Matrix Inequality Approach to the General Mixed H_∞/H_2 Control Problem
American Control Conference, pp.1443-1447, June 1995
- [23] S.Boyd, L.E.Ghaoui
Linear Matrix Inequalities in Systems and Control Theory
SIAM Studies in Applied Mathematics, Vol.15, 1994 (book)
- [24] S.Boyd, V.Balakrishnan, E.Feron, L.E.Ghaoui
Control System Analysis and Synthesis Via Linear Matrix Inequalities
American Control Conference, pp.2147-2154, 1993
- [25] D.E.Walker, D.B.Ridgely
Uniqueness of the General Mixed H_∞/H_2 Optimal Controller
American Control Conference, pp.1453-1457, June 1995
- [26] D.S.Bernstein, W.M.Haddad
LQG Control with an H_∞ Performance Bound: A Riccati Equation Approach.
IEEE Transactions on Automatic Control, Vol.34, No.3, pp.293-305, March 1989
- [27] P.Gahinet, P.Apkarian
A Linear Matrix Inequality Approach to the H_∞ Control
International Journal of Robust and Nonlinear Control, Vol.4, pp.421-488, 1994
- [28] F.S.Kramer, A.J.Calise
Fixed Order Dynamic Compensation for Multivariable Systems
Journal of Guidance Control and Dynamics, Vol.11, No.1, pp.80-85, Feb.1988
- [29] A.J. van der Schaft
On a State Space Approach to Nonlinear H_∞ Control
Systems and Control Letters, Vol.16, pp.1-8, 1991

-
- [30] S.Boyd, V.Balakrishnan, E.Feron, L.E.Ghaoui
History of Linear Matrix Inequalities in Control Theory
American Control Conference, pp.31-34, June 1994
 - [31] E.G.Collins, D.Sadhukhan, L.T.Watson
Robust Controller Synthesis via Non-linear Matrix Inequalities
International Journal of Control, Vol.72, No.11, pp.971-980, 1999
 - [32] Chang-Shi Wu, Bor-Sen Chen, Ying-Wen Jan
Unified Design for H_∞/H_2 and Mixed Control of Spacecraft
Journal of Guidance, Control and Dynamics, Vol.12, No.6, pp.884-896, Nov-Dec. 1999
 - [33] R.J.Niewhoener, I.Kaminer
Linear Matrix Inequalities in Integrated Aircraft/Controller Design
American Control Conference, pp.177-181 June 1995
 - [34] P.P.Khargonekar, M.A.Rotea
Mixed H_∞/H_2 Control: A Convex Optimization Approach
IEEE Transactions on Automatic Control, Vol.36, No.7, pp.824-837, July 1991
 - [35] Mario Sznaier
An Exact Solution to General SISO Mixed H_∞/H_2 Problems via Convex Optimization
IEEE Transactions on Automatic Control, Vol.39, No.12, pp.2511-2517, December 1994
 - [36] M.A. Rotea, P.P.Khargonekar
 H_2 -optimal Control with an H_∞ -constraint: The State Feedback Case
Automatica, Vol.27, No.2, pp.307-316, 1991
 - [37] K.J.Hunt
Polynomial LQG and H_∞ Controller Synthesis: A Genetic Algorithm Solution
IEEE Proceedings of the 31st Conference on Decision and Control, Tucson, Arizona pp.3604-3609, Dec. 1992
 - [38] J.Doyle, K.Zhou, K.Glover, B.Bodenheimer
Mixed H_2 and H_∞ Performance Objectives II: Optimal Control
IEEE Transactions on Automatic Control, Vol.39, No.8, pp.1575-1586, August 1994
 - [39] L.Xie, Y.Chai Soh
Robust LQG Control of Uncertain Linear Systems Via Simultaneous H_2 and H_∞ Approach
Optimal Control Applications and Methods, Vol.18, pp.49-58, 1997
 - [40] Z.Hu, S.E.Salcudean, P.D.Loewen
A Numerical Solution to the Multiobjective Control Problems
Optimal Control Applications and Methods, Vol.19, pp.411-422, 1998
 - [41] J.Doyle, K.Zhou, K.Glover, B.Bodenheimer
Mixed H_2 and H_∞ Performance Objectives I: Robust Performance Analysis
IEEE Transactions on Automatic Control, Vol.39, No.8, pp.1564-1574, August 1994
 - [42] A.Chipperfield, P.Flemming
Multiobjective Gas Turbine Engine Controller Design Using Genetic Algorithms
IEEE Transactions on Industrial Electronics, Vol.34, No.5, pp.583-587, October 1996
 - [43] R.E.Avedon, B.A.Francis
Digital Control Design via Convex Optimization
Journal of Dynamic Systems Measurement and Control, Vol.115, pp.579-586, December 1993
 - [44] I.Masubuchi, A.Ohara, N.Suda
LMI-Based Output Feedback Controller Design
American Control Conference, pp.3473-3477 June 1995

-
- [45] C.Scherer, P.Gahinet, M.Chiladi
Multiobjective Output Feedback Control via LMI Optimization
IEEE Transactions on Automatic Control, Vol.42, No.7, pp.896-911, July 1997
- [46] C.Scherer
Multiobjective H_2/H_∞ Control
IEEE Transactions on Automatic Control, Vol.40, No.6, pp.1054-1062, June 1995
- [47] His-Han Yeh, S.S.Banda, Bor-Chin Chang
Necessary and Sufficient Conditions for the Mixed H_2 and H_∞ Optimal Control
IEEE Transactions on Automatic Control, Vol.37, No.3, pp.355-358, March 1992
- [48] Z.Hu, S.E.Salcudean, P.D.Loewen
Numerical Solution of the Multiple Objective Control System Design Problem for SISO Systems
American Control Conference, pp.1458-1462, June 1995
- [49] M.Sznaier
A Mixed H_2/H_∞ Optimization Approach to Robust Controller Design
SIAM Journal of Control and Optimization, Vol.33, pp.1086-1101, July 1995
- [50] J.C.Geromel, P.L.D.Peres, S.R.Souza
A Convex Approach to the Mixed H_2/H_∞ Control Problem for Discrete Time Uncertain Systems
SIAM Journal of Control and Optimization, Vol.33, No.6, pp.1816-1833, November 1995
- [51] J.A.Ball, J.W.Helton, M.L.Walker
 H_∞ Control for Nonlinear Systems with Output Feedback
IEEE Transactions on Automatic Control, Vol.38, No.4, pp.546-559, April 1993
- [52] A.Isidori, W.Kang
 H_∞ Control via Measurement Feedback for General Nonlinear Systems
IEEE Transactions on Automatic Control, Vol.40, No.3, pp.466-472, March 1995
- [53] A.J. van der Schaft
 L_2 Gain Analysis of Nonlinear Systems and Nonlinear State Feedback H_∞ Control
IEEE Transactions on Automatic Control, Vol.37, No.6, pp.770-784, June 1992
- [54] Guang-hong Yang, James Lam, Jianliang Wang
Reliable H_∞ Control for Affine Nonlinear Systems
IEEE Transactions on Automatic Control, Vol.43, No.8, pp.1112-1116, August 1998
- [55] A. Isidori, A.Astolfi
Disturbance Attenuation and H_∞ Control Via Measurement Feedback in Nonlinear Systems
IEEE Transactions on Automatic Control, Vol.37, No.9, pp.1283-1293, September 1992
- [56] Bor-Sen Chen, Yu-Min Cheng
A Structure Specified H_∞ Optimal Control Design for Practical Applications: A Genetic Approach.
IEEE Transactions on Control System Technology, Vol.6, No.6, pp.707-718, Nov. 1998
- [57] K.S.Tang, K.F.Man, D.W. Gu
Structured Genetic Algorithm for Robust H_∞ Control System Design
IEEE Transactions on Industrial Electronics, Vol.43, No.5, pp.575-582, October. 1996
- [58] T. Back, D.B.Fogel, Z. Michalewicz
Handbook of Evolutionary Computation
Institute of Physics Publishing and Oxford University Press, , 1997 (Book)
- [59] F.L.Lewis
Applied Optimal Control and Estimation, Digital Design and Implementation
Prentice Hall, Digital Signal Processing Series, 1992 Texas Instruments (Book)

6

Fault Detection and Isolation Using Hybrid Genetic Algorithms

Contents:

6.1	Fault Detection and Isolation	p.6.2
6.1.1	Introduction	p.6.2
6.1.2	Fault Detection and Isolation - Survey	p.6.4
6.1.3	Signal Based Methods	p.6.7
6.1.4	Model Based Methods	p.6.7
6.1.5	Observer Based Methods	p.6.9
6.1.6	Modeling Faults in Systems, Residual Generation.	p.6.12
6.1.7	Parity Space Methods - Theory	p.6.15
6.2	Detecting Faults With Hybrid Genetic Algorithms	p.6.17
6.2.1	Theory	p.6.17
6.2.2	Detecting Input/Output Faults in Linear Systems	p.6.20
6.2.3	Detecting Internal Faults.	p.6.29
6.3	Chapter Summary and Conclusion	p.6.31
6.4	References and Further Reading	p.6.32

6.1 Fault Detection and Isolation:

6.1.1 Introduction:

This chapter examines the subject of fault detection and isolation (FDI). Its purpose is twofold: (1) to provide an initial outline and summary on a number of traditional and active areas of research involving fault detection and isolation, and (2) the application of hybrid GA and neural networks to the detection and identification of faults. Simulation results comparing genetic algorithms and conventional fault detection methods is presented. A comprehensive survey on FDI can be found in reference [1].

The objectives of fault detection and isolation can range from simple diagnosis of non critical components, to more complex life critical systems such as aircraft, nuclear power plants, and medical equipment. Faults, when detected early can be used to prevent major damage to equipment, loss of operation or income, or loss of human lives. Faults which occur gradually over a long period of time (*incipient*), generally indicate equipment or component mechanical wear, deterioration and contamination. Sudden (*abrupt*) faults on the other hand are generally easier to detect, but can be more damaging if not detected quickly. When a fault has been detected, various contingencies can be initiated from simple manual component replacement to more complex control reconfiguration. Nowadays, fault detection and isolation has become an integral part of the operation of ships, submarines, aircraft, spacecraft and industrial plants. Fault detection and isolation has evolved from simple limit checking, to more sophisticated analytical methods involving plant models and state estimators, knowledge databases, expert diagnostic systems, and artificial intelligence. Fault detection and isolation can be broadly classified into three main categories: *signal based*, *model based* (*qualitative* and *quantitative*), and *observer based*.

(1) Signal based: These methods are the simplest and generally the more commonly used in industry. For example comparing readings from a multiply-redundant sensor system, limit and threshold checking, trend checking, and spectrum checking. Suitable for manual inspection and operation.

(2) Model based: Model based methods fall into two categories: qualitative and quantitative.

(i) Qualitative: A model is required, however not necessarily a mathematical one. This includes Artificial Intelligence (AI), Artificial Neural Networks (ANN) black box models, Expert Systems, Fuzzy Logic Systems (FLS), fault trees, topological and rule-based methods are generally a combination of the above.

For instance, neural networks can be trained to classify data into healthy or faulty. Neural network methods have been slow to emerge in the area of FDI due to the long training times involved. There is ample and varied literature on these methods (see section 6.14 below for references) .

(ii) Quantitative: These methods are well established for linear systems, and all require an accurate mathematical model of the process. Methods include *fault detection filters* (FDF), *parity space* and optimally robust parity methods, *unknown input observers* (UIO's), eigenstructure assignment, *influence matrix methods* and robust H_∞ (normed spaces) methods. All these methods use the concept of *analytical redundancy* in which the output of any one sensor can be reconstructed from measurements of the other (healthy but dissimilar) sensors, and *a-priori* knowledge of the plant. Analytical redundancy requires no extra hardware (i.e. sensors/actuators) when compared to *hardware redundancy* (or parallel redundancy), in which sensors/actuators are physically duplicated. All these methods (excluding H_∞) suffer from model uncertainties, disturbances and noise.

(3) Observer based: these methods include: Kalman filters and Luenberger observers. Newer techniques include: robust sliding mode *Variable Structure System* (VSS) observers, nonlinear observers such as extended Kalman filters (EKF) for stochastic systems, and extended Luenberger observers for deterministic systems, Lie-algebraic methods, robust H_∞ methods, and pseudolinearization methods. The advantages include robustness to disturbances and model uncertainties, and ability to deal directly with the nonlinearities of the systems. In many cases, the nonlinearity is treated as an unknown bounded disturbance. Observer based methods are also affected by model uncertainties and disturbances, however robustness issues can be included in the overall part of the observer design. The application of nonlinear and variable structure observers to FDI is presently an active area of research.

There are three stages to detecting faults: (i) *Fault Detection*: knowing that a fault has occurred and generating an alarm condition, (ii) *Fault Isolation*: locating the fault i.e. deciding which sensor or component has failed, and (iii) *Fault Identification*: estimating the extent or size of the fault, and any time dependent behavior. The first part is the simplest, detecting a fault condition. This is accomplished by calculating a *residual vector*, which is the difference between the measured and calculated outputs (from a mathematical model) of the system.

In the absence of faults, the residual is zero, when a fault occurs, the residual is nonzero. Problems arise when external disturbances, model uncertainties and noise exist, resulting in a nonzero residual for the fault-free condition. Robust, adaptive thresholding, and statistical techniques have been developed to address this problem. The concept of residuals is described in more detail in section 6.1.6. The second part is more difficult, to locate (i.e. isolate) the faulty sensor or component. Presently the method of Multiple Hypothesis Testing and Maximum Likelihood ratio testing are used. These are all statistical methods requiring multiple models to be tested simultaneously, each model assumes a specific fault, including one model which is fault free. The residual from each model is statistically tested against the actual process output to infer the cause of the fault. The third part (size of fault) can be estimated once the fault has been located. Note that both model based and observer based methods involve the calculation of a residual vector.

The prompt detection of faults is becoming more vital, as the complexity and interdependence on automation is increasing, and in many cases becoming more life-critical such as nuclear power plants, medical equipment, and aircraft control. For instance, control system failures have contributed to a number of aircraft incidents: blocked pressure ports which produced erroneous air data leading to a crash [2], a false stall warning because of a stuck angle-of-attack detector leading to damage to the aircraft [3], and separation of an engine pylon caused loss of power, loss of hydraulic systems, and asymmetrical flap settings leading to a crash [4], failed inertial reference unit (gyro) caused the ARIANE-5 (Flight-501, June 1996) to crash [2].

This chapter is primarily concerned with detecting and isolating faults occurring in the plant sensor and actuator part of the control loop. Both linear and nonlinear systems are investigated. Simulations using genetic algorithms are compared with more traditional methods of parity space. This chapter is not intended to be a comprehensive survey of fault detection and isolation, however a brief introduction and overview will be provided, and many references are also provided at the end of the chapter for the interested reader on this subject.

6.1.2 Fault Detection and Isolation - Survey:

A large number of survey papers have been published on fault detection and isolation, refer to [5, 6, 7, 8, 9, 10, 11, 12] dealing with parity space and dedicated observers, fault detection filters, residual generation, robust observers for linear systems, expert systems, artificial intelligence, fuzzy logic and neural network based methods, and statistical methods are discussed in [80].

An excellent introductory textbook is by Chen and Patton [1]. Before discussing techniques for fault detection and isolation, some definitions are provided which are used throughout the FDI literature:

Fault: An unpermitted deviation of at least one characteristic property or parameter of the system from the acceptable/usual/standard condition.

Failure: A permanent interruption of a system's ability to perform a required function under specified operating conditions.

Residual: A fault indicator, based on a deviation between measurements and mode-equation based computations. Residuals can be computed from either states or measured outputs.

Analytical Redundancy: Use of two or more (but not necessarily identical) ways to determine a variable, where one way uses a mathematical process model in analytical form. As opposed to physical/hardware redundancy in which sensors/actuators are duplicated, and a voting scheme is required.

As indicated earlier, fault detection and isolation methods can be broadly classified into three categories: (i) *Signal Based*, (ii) *Model Based (qualitative, quantitative)*, and (iii) *Observer Based*. Figure 6.1 on the following page illustrates this classifications.

A failure detection and identification scheme must possess certain fundamental characteristics in order to reliably detect faults, and minimize false alarms: (i) Robustness to modeling uncertainties. (ii) Robustness to disturbances and noise. (iii) Ability to isolate faults. (iv) Ability to detect *incipient* (gradual) and *abrupt* (sudden) faults. (v) Detection of both additive and multiplicative faults. (vi) Applicability to non-linear systems.

Both linear and nonlinear observer (variable structure) methods offer the greatest potential to fault detection and isolation due to the robustness properties. Although model based and observer based methods require *a-priori* knowledge of the plant dynamics, and the concept of analytical redundancy, the main difference is that model based methods do not require a knowledge of the plant states, only input/output measurements. Consequently, observer based methods can potentially diagnose faults more reliably due to the extra information from the plant state vector.

Variable structure observer based methods are gaining popularity due to their robustness properties and application to nonlinear systems. They are described briefly in section 6.1.5(iii). Figure 6.1 below illustrates the various fault detection and isolation classification schemes:

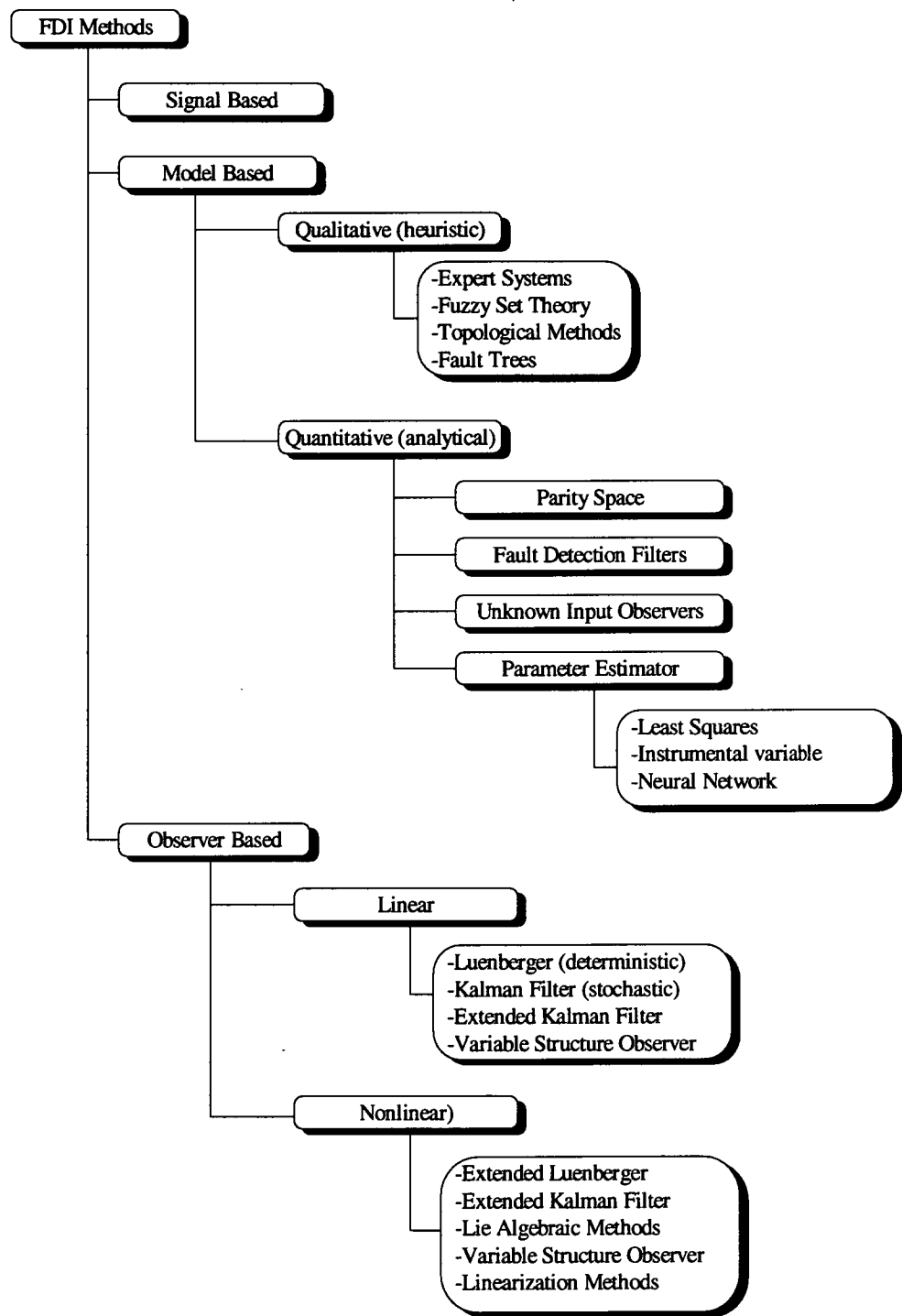


Fig.6.1
Classifications of FDI Methods

6.1.3 Signal Based Methods:

These are methods which do not require any mathematical models of the plant. The simplest method of fault detection is limit checking. Other methods use special or multiple sensors such as strain gauges on critical structures, redundant sensors, and voting systems, frequency domain analysis (signature analysis), although this requires knowledge of the spectra for normal and failed operation. These methods are very common in industrial plants. A very popular method includes the use of *Multi Valued Influence Matrices* (MVIM) in which fault diagnosis is performed by matching a measurement vector against the columns of the influence matrix, generally used in complex systems [13]. Note that MVIM methods require some training/learning techniques to construct the influence matrix. A neural network equivalent classifier can be used in an identical manner. See also [14, 15] for variations involving the use of influence matrices. Others include Fault Tree Analysis (FTA) and Event Tree Analysis (ETA).

In summary, signal based methods include:

- Topological approaches such as fault trees/directed graphs [64];
- Multivalued influence matrix approach [13, 14, 15].

6.1.4 Model Based Methods:

Methods which require a mathematical model of the plant. These methods are more extensive, and do not necessarily require additional, or special hardware to identify faults. They can give more accurate estimation of faults, and can use analytical redundancy such as sensor fusion techniques. These fall into two categories: *Qualitative* or heuristic and *Quantitative* or analytical methods.

(i) *Qualitative*: Applicable to large scale systems in which the dynamics of the process is not well known. These methods require a black box model or heuristic model of the plant, these include:

- Expert systems and knowledge databases: [8, 59, 60, 61, 62].
- Artificial intelligence methods [63].
- Fuzzy logic identification. In most real world systems, as the complexity of a system increases, our ability to create accurate and precise models about such systems decreases. In these instances, fuzzy models of the system can be constructed where the physical processes are poorly understood but in which linguistic, intuitive knowledge and degree of vagueness of the the variables of the system are available. Fuzzy model identification and fault diagnosis has been applied to: modeling internal combustion engines [56], nonlinear systems [57], process industries [81], survey papers on fuzzy modeling and control: [55, 58].

- Neural networks have become an active area of research in fault diagnosis. Neural networks have robustness and generalization capabilities, with the ability to adapt and learn. Neural networks can be trained to classify faults from input/output training data sets, and learn complex nonlinear transfer functions for modeling applications. Introductory papers can be found in: [67, 68, 70];
- Hopfield/ART-1/ARTMAP classifiers [65, 66, 69]; neurofuzzy methods: [55]; modeling and state estimation [71]. An issue of concern however with neural network is that the training algorithms may suffer from local and slow convergence. Currently, qualitative methods using neural networks for fault detection and isolation is a promising area of research.

(ii) **Quantitative:** Require an accurate mathematical model of the plant. Quantitative methods operate in two stages: - *residual generation* and *residual analysis*. Many well established methods are available, these are:

- Parity space is one of the most popular [1, 17, 19], with variations using fuzzy logic [51, 52], optimal parity vectors [16], optimally robust parity relations [6, 18], continuous parity space [20], generalized parity space [21],
- Parameter estimation/system identification methods see [34, 35, 36, 53],

The more common and well established methods of Parity space will be discussed in further detail due to their simplicity, ease of use to linear systems in state variable form, and for simulation comparison with genetic algorithms. Figure 6.2 illustrates the relationship between different fields of science and their relative degree of model accuracy (i.e. depth of knowledge):

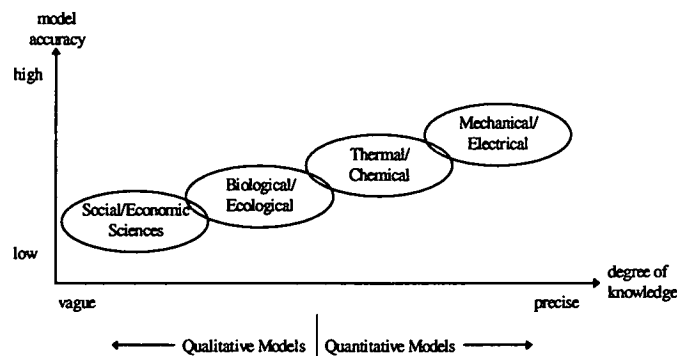


Fig.6.2
Model Accuracy Versus Field of Knowledge

Qualitative models in which accurate mathematical models are not available include biological, social, economic systems. Quantitative models in which an accurate mathematical model can be developed include thermal, electrical, mechanical and chemical processes.

6.1.5 Observer Based Methods:

Observer based methods are the most powerful, extensive, and can be applied to nonlinear systems. These methods use an observer which estimates the states of the plant, residuals can then be calculated from the difference of the measured and estimated states. There are several types of observers: *Luenberger* (deterministic), *Kalman filters* (stochastic), *Extended Kalman filters* (non linear stochastic systems) and *Sliding mode observers*. There are many different variations, in particular see [49] for general introduction to nonlinear observers. Observer based methods include:

- *Fault Detection Filters* (FDF) see [1, 22], robust [23, 24, 54], and simplified design method [25, 26].
- *Unknown Input Observers* (UIO) see [1, 27, 28, 30], stochastic systems [29], and robust [31], and *Dedicated Observer Schemes* (DOS): which requires multiple observers for each sensor, for LTI systems see [32, 33],
- *Linear Kalman Filters* and *Linear (KF) Luenberger Observer*: [48, 72, 73, 74] are well established in the literature.
- *Extended Kalman Filter* [40] (EKF) and *Extended Luenberger observer*: also known as nonlinear Luenberger observers, [46, 47].
- *Thau Observer*: which is more of a verification method rather than design, see: [37].
- *Lie Algebraic Methods*: newer but more difficult to apply, control affine form only.
- *Adaptive Observers*.
- *Variable Structure Observers*: which includes the *Walcott and Zak* observer [41, 42], the *Utkin observer* [44], the *discontinuous observers* [43]. Variable structure observers apply to both linear and nonlinear systems [45].
- *High Gain Observers*.

(i) **Extended Luenberger observers**: [46, 47] and extended Kalman filters [40, 49] provide a natural extension of conventional Kalman and Luenberger observers to nonlinear systems. The extended Luenberger observer uses an extended linearization method which is independent of the operating point (pseudolinearization). This pseudolinearization is made independent of the operating point by nonlinear state transformation to observer canonical form. The mathematical description of the extended Luenberger observer is however complex. The Extended Kalman Filter (EKF) works by constantly linearizing the system at the current plant operating point, the EKF filter minimizes the trace of the covariance matrix of the estimation errors.

Its main deficiencies include: requiring perfect system knowledge, no a priori knowledge on stability, no robustness against modeling errors can be guaranteed, and computer intensive real time implementation. An enhancement of the extended Kalman filter is the *Constant Gain Extended Kalman Filter* which addresses the robustness issues and real time implementations.

(ii) **Thau Observers:** [37, 49], describes a method of verification rather than a direct design method. Sufficient conditions for the convergence of the observer are given but no information on how to design the observer is provided. It can be applied to control affine nonlinear systems. The Thau method does not address the problem of modeling errors, and lacks robustness, however its main advantage is in its simplicity.

(iii) **Variable structure Observers:** (VS) observers are a relatively new type of observers which owe their design and implementation to Lyapunov stability theory and variable structure/sliding mode control theory. Because of this close affinity to VS control, the properties of robustness, invariance in sliding mode and applicability to nonlinear systems are all equally valid to VS observers. There are at present three main types of VS observers: (i) The *Walcott and Zak observer*, (ii) The *Utkin observer* and (iii) The *Discontinuous observer*. A brief description is given below. For an introductory treatment see [43].

- (a) **The Walcott and Zak Observer:** Perhaps one of the easiest VS observer is the Walcott and Zak observer [41, 42]. The only condition imposed is that the disturbance/nonlinearity is *matched*, and the solution to two Lyapunov equations. This observer is identical to a standard Luenberger observer with a VS term which eliminates the disturbance term. Relatively easy to apply to linear systems, however a boundary layer is necessary to prevent chattering.
- (b) **The Utkin Observer:** The Utkin observer [43, 44] requires a similarity transformation so that the outputs $y(t)$ appear as components of the states. The plant is then in a symmetric (observer canonical) form. It can be shown that if the output error $e_y(t) = \hat{y}(t) - y(t)$ converges to zero, then the state error $e_x(t) = \hat{x}(t) - x(t)$ also converges to zero asymptotically. Proof is by defining a Lyapunov function candidate. It does not require matching conditions on the disturbances.
- (c) **The Discontinuous (structured) Observer:** This is a general form of the VS observer, it requires a similarity transformation to symmetric canonical form and the solution to two Lyapunov equations. Much more difficult to apply, but more general with fewer restrictions.

Variable structure observers offer the greatest potential due to their robustness properties, able to deal with system nonlinearities and external disturbances. In the applications to linear systems with matched disturbances, the VS observer is relatively straightforward to apply. Chattering must also be considered during implementation.

(iv) **Adaptive observers:** [49, 50] were designed to be for a certain class of nonlinear second order systems with bounded coefficients and having bounded time variation of the form:

$$\ddot{y} + a_1(y, \dot{y}, t) \cdot \dot{y} + a_2(y, \dot{y}, t) = b(y, \dot{y}, t) \cdot u + f(y)$$

where a_1, a_2, b are unknown functions of time, and $f(y)$ is a known functions of y . Applying a transformation to canonical form, results in a seventh-order ordinary differential equation. Because of its restricted applicability to second order systems, and amount of real-time computation, the adaptive observer is not very common.

(v) **High gain observers:** (HGO), also known as the *Gauthier-Kupta* observer, it gives very fast convergence of the estimation error. Works only with single output systems, this is generally not a drawback because multiple HGO can be designed for each individual output. There are restrictions to the structure of the system dynamics, and is not very general. Only applicable to certain class of nonlinear systems.

(vi) **Fault detection filters:** FDF (also known as fault sensitive filters) are a special class of Luenberger observers which generate directional residuals for the purpose of fault isolation. Fault detection filters are full order state estimators with a special choice of observer gain, chosen such that when a particular fault occurs, the residual is constrained in a single direction or plane. The residuals are therefore the innovations terms of the filter. To detect a fault, the norm of the residual is compared to a threshold, if greater than the threshold then a fault is detected. Fault isolation requires comparing the direction of the residual with pre-defined fault directions (or signatures). It is possible to design the observer such that the residuals are insensitive to the unknown disturbances, but sensitive to faults, this is the robust form of the fault detection filter. The fault detection filter developed by Beard (1971), also known as the Beard Fault Detection Filter (BFDF).

6.1.6 Modeling Faults in Systems, Residual Generation:

(i) **Modeling Faults:** Faults can appear in the input (actuator faults), system (internal or component faults) and output (sensor faults). It is common practice to model faults as a combination of additive and multiplicative faults. Figure 6.3 below illustrates this concept. In most instances, faults occur in either the input or output, thus component faults are generally not considered in most research. The simplest fault model applies to linear deterministic systems with no external disturbances. To build a mathematical model, we need to use the conventional form of state space.

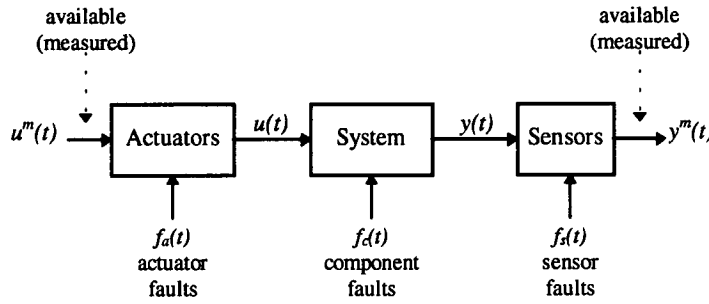


Fig.6.3
Fault Model for Linear Systems

Looking at figure 6.3, actuator and sensor faults can be modeled as a combination of multiplicative and additive faults, in the continuous time domain, the state-space plant model is given by:

Plant State Model:

$$\begin{aligned} \dot{x} &= A.x + B.u \\ y &= C.x + D.u \end{aligned} \quad \text{Eqn.6.1}$$

Input/Output Fault models:

$$\begin{aligned} u &= \alpha_u . u^m + \Delta u \\ y &= \alpha_y . y^m + \Delta y \end{aligned} \quad \text{Eqn.6.2}$$

where: $u \in \mathbb{R}^r$: input control vector of the plant, $y \in \mathbb{R}^m$: output measurement vector of the plant, $u^m \in \mathbb{R}^r$: is the measured value of the input control vector, $y^m \in \mathbb{R}^m$: is the measured value of the output vector. The diagonal matrices α_u and α_y are multiplicative faults represented by square diagonal matrices, Δu and Δy column vectors representing additive faults:

$$\alpha_u = \begin{bmatrix} \alpha_{u_1} & 0 & \dots & 0 \\ 0 & \alpha_{u_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \alpha_{u_r} \end{bmatrix} \quad \alpha_y = \begin{bmatrix} \alpha_{y_1} & 0 & \dots & 0 \\ 0 & \alpha_{y_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \alpha_{y_m} \end{bmatrix} \quad \Delta u = \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \\ \vdots \\ \Delta u_r \end{bmatrix} \quad \Delta y = \begin{bmatrix} \Delta y_1 \\ \Delta y_2 \\ \vdots \\ \Delta y_r \end{bmatrix} \quad \text{Eqn.6.3}$$

In the fault free case, all elements of the diagonal are set to 1, it is customary to consider only a single fault in the input or a single fault at the output. This sets all entries to 1, except for one single entry in each matrix corresponding to a multiplicative fault. The column vectors: Δu and Δy represent additive faults, in the fault free case all entries are zero. In the fault case, only one entry will be nonzero corresponding to the additive fault. To simplify the problem, the faults can be assumed time independent (constant).

(ii) Residual Generation: The residual $r(t)$ is simply the difference between the measured output and estimated output (or state) from an observer or model of the plant. In the case of an observer, the residual is also the innovation term of the observer dynamics. The residual vector may have a specific signature or direction (*directional residuals*) as in the case of the fault detection filter (FDF). Ideally, in the absence of faults the residual should be zero. When faults are present, the residual is nonzero. To detect a fault, we calculate the magnitude of the residual (Euclidean norm), and compare it to some threshold value. Due to modeling errors, uncertainties, disturbances and noise, the residual is nonzero even in the absence of faults. The magnitude of the residual provides an indication of the size of the fault. To locate the fault, the structure or signature of the residual must be compared with a set of known patterns of known fault modes. The use of neural networks classification techniques have been used such as the Hopfield and ART networks. Statistical testing is often used, such as Sequential Probability Ratio Testing (SPRT), Bayesian testing, Multivariate testing, Hypothesis testing, and Maximum Likelihood testing.

When dealing with robust residual generation, most methods attempt to maximize some performance or cost functional, this cost function is usually the ratio of the transfer function norm from: *fault-to-residual* $G_{rf}(s)$ to *disturbance-to-residual* $G_{rd}(s)$, for instance using Laplace notation:

$$J = \max \left(\frac{G_{rf}(s)}{G_{rd}(s)} \right) \quad \text{Eqn.6.4}$$

This leads to a multiobjective optimization problem in which we try to maximize the transfer function $J_1 = \max \|G_{rf}(s)\|$ and minimize $J_2 = \min \|G_{rd}(s)\|$. To see how residuals are applied to fault detection and isolation, consider a linear LTI system with faults at the input and outputs as follows, the general expression is:

$$\left. \begin{aligned} \dot{x} &= A.x + B.u + R_1.f \\ y &= C.x + D.u + R_2.f \end{aligned} \right\} \quad \text{Eqn.6.5}$$

Where R_1 and R_2 are known input and output fault transfer matrices, and f is the fault to be estimated. The input/output transfer function using Laplace transform is given by: (bearing in mind that $y(s)$ is actually a different variable to $y(t)$)

$$y(s) = G_u(s).u(s) + G_f(s).f(s) \quad \text{Eqn.6.6}$$

Where:

$$\left. \begin{aligned} G_u(s) &= C.(sI - A)^{-1}.B + D \\ G_f(s) &= C.(sI - A)^{-1}.R_1 + R_2 \end{aligned} \right\} \quad \text{Eqn.6.7}$$

This is the most commonly used form of representing input and output faults for linear systems. All model based FDI methods are designed to estimate the fault f with possible variations which include noise, disturbances, nonlinearities, unmodeled dynamics and unknown inputs. The residual can be defined mathematically as a combination of the input and output measurement sequences, see figure 6.4:

$$r(s) = H_u(s).u(s) + H_y(s).y(s) \quad \text{Eqn.6.8}$$

substituting the equation for $y(s)$ given previously into the above expression results in a general form of the residual generator:

$$r(s) = [H_u(s) + H_y(s).G_u(s)].u(s) + H_y(s).G_f(s).f(s) \quad \text{Eqn.6.9}$$

the requirements are that when no faults are present: i.e. $f=0$, then the residual should also be zero: $r=0$, therefore we have two requirements which must be satisfied:

$$\left. \begin{aligned} H_u(s) + H_y(s).G_u(s) &= 0 \\ H_y(s).G_f(s) &\neq 0 \end{aligned} \right\} \quad \text{Eqn.6.10}$$

The above is a generalized representation of fault detection using residuals for linear systems. The first requirement implies that the residual is independent of the input, and the second requirement specifies *detectability*. This applies to all forms including model based, observer based and parity space methods.

Most literature relating to fault detection and isolation for linear systems deals with the above equations, or variations of equation 6.9. These include the presence of disturbances, noise, and plant uncertainty. In these circumstances, statistical testing of the residual may be required. This can involve: (i) Weighted sum-squared residual (WSSR), (ii) Chi-squared testing (iii) Sequential probability ratio testing SPRT, (iv) Generalized Likelihood ratio testing GLR, and (v) Multiple Hypothesis Testing.

The basic concept of residual generation is illustrated in figure 6.4 below:

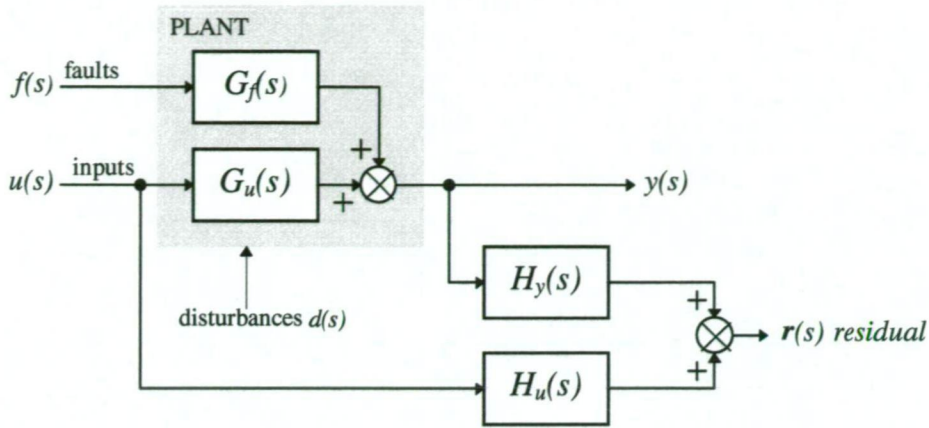


Fig. 6.4

Generic Form of Residual Generators for Linear Deterministic Systems

One of the most popular methods for detecting faults in linear systems is *parity space*. Parity space is used as a comparison with hybrid genetic algorithms. The theory is outlined next.

6.1.7 Parity Space Methods - Theory:

The parity space method (see ref. [1], pp.38-44) is one of the most commonly used model based approach for generating residuals. There are two types of parity space: *direct redundancy* and *temporal redundancy*. Direct redundancy is of limited applicability and is not considered as it requires more sensors than states, i.e. for: $x(t) \in \mathbb{R}^n$, $y(t) \in \mathbb{R}^m$ requires $m > n$. Temporal redundancy may be applied to a wider range of problems and is not restricted to the above condition. Parity space was developed in discrete time domain, but has now been extended to continuous time domain. Given the following system model in discrete time:

$$\left. \begin{aligned} x(k+1) &= A.x(k) + B.u(k) + R_1.f(k) \\ y(k) &= C.x(k) + D.u(k) + R_2.f(k) \end{aligned} \right\} \quad \text{Eqn.6.11}$$

Taking a window of s previous samples and the current sample, a total $s+1$ equations is constructed, combining these equations recursively into one another gives:

$$\begin{bmatrix} y(k-s) \\ \vdots \\ y(k-1) \\ y(k) \end{bmatrix} - H. \begin{bmatrix} u(k-s) \\ \vdots \\ u(k-1) \\ u(k) \end{bmatrix} = W.x(k-s) + M. \begin{bmatrix} f(k-s) \\ \vdots \\ f(k-1) \\ f(k) \end{bmatrix} \quad \text{Eqn.6.12}$$

where:

$$H = \begin{bmatrix} D & 0 & \dots & 0 \\ CB & D & \dots & 0 \\ CAB & CB & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ CA^{s-1}B & CA^{s-2}B & \dots & D \end{bmatrix} \quad W = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^s \end{bmatrix} \quad M = \begin{bmatrix} R_2 & 0 & \dots & 0 \\ CR_1 & R_2 & \dots & 0 \\ CAR_1 & CR_1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ CA^{s-1}R_1 & CA^{s-2}R_1 & \dots & R_2 \end{bmatrix} \quad \text{Eqn.6.13}$$

Note the presence of the observability matrix W , re-writing equation 6.12 gives:

$$Y(k) - H.U(k) = W.x(k-s) + M.F(k) \quad \text{Eqn.6.14}$$

Note also that $Y(k)$ represents $s+1$ measurements of y vectors, and $U(k)$ likewise. However $x(k-s)$ is a single unknown vector. The key idea of parity space is to multiply each side by a matrix V thus:

$$V.[Y(k) - H.U(k)] = V.[W.x(k-s) + M.F(k)] \quad \text{Eqn.6.15}$$

where V is chosen to be the left nullspace of the matrix W such that: $V.W=0$, then the equation becomes independent of the state $x(k-s)$, this simplifies to:

$$V.[Y(k) - H.U(k)] = V.M.F(k) \quad \text{Eqn.6.16}$$

the residual is defined to be the right hand part of the above equation:

$$r(k) = V.[Y(k) - H.U(k)] \quad \text{Eqn.6.17}$$

in the absence of faults $F(k)=0$, the residual should be zero $r(k)=0$; however in the presence of faults $F(k) \neq 0$ the residual is nonzero $r(k) \neq 0$. In summary, the parity space method for linear systems has two parts: it first requires the calculation of the residual from the equation: i.e. *computational form of residual*, obtained from past measurements:

$$r(k) = V.[Y(k) - H.U(k)] \quad \text{Eqn.6.18}$$

we can then estimate the fault $F(k)$ from the *evaluational form of residual*:

$$r(k) = V.M.F(k) \quad \text{Eqn.6.19}$$

This method can easily detect the presence of faults, but locating the fault is more difficult. Note also that the parity space equations are equivalent to a *deadbeat observer*. There are other limitations also with parity space as we shall see later.

6.2 Detecting Faults With Hybrid Genetic Algorithms:

From the brief survey presented above, there are many ways in which genetic and heuristic search algorithms may be applied to fault detection and isolation problems. Below we present a direct and straightforward manner of using genetic algorithms for fault detection and isolation.

In these next set of simulations, genetic algorithms are compared with conventional parity space methods. Both additive and multiplicative faults in the sensor and actuator are compared. Two types of faults are simulated:

- (i) Input and output faults using the linearized open loop longitudinal aircraft model described in the appendix. Both single input/output and multiple input/output faults are considered. Results obtained using hybrid genetic algorithms are compared with conventional parity space methods described earlier.
- (ii) The second simulation investigates internal (component) faults using hybrid genetic algorithms for the nonlinear aircraft system.

This is essentially a model based fault diagnosis system. We use a radial basis function network to provide a reference (fault free) model the plant. The radial basis function network is initially trained using genetic algorithms. The residual generated is used by the genetic algorithm to predict the fault. The basic underlying theory behind detecting faults with GA is discussed next. The final configuration is illustrated in figure 6.7.

6.2.1 Theory:

The basic concept behind detecting faults with genetic algorithms is as follows: the population of chromosomes is initialized with many different fault combinations, this can also include a fault-free condition. The fitness of the chromosome is then evaluated by calculating the difference between the measured plant output (of the faulty plant), and the predicted plant output using the fault information contained within the chromosome. Note that this is essentially a model based approach. Consequently, the chromosome which best predicts the actual faulty plant output provides the best estimate of the occurring fault.

The concept is illustrated Fig.6.5 below, for the given plant model:

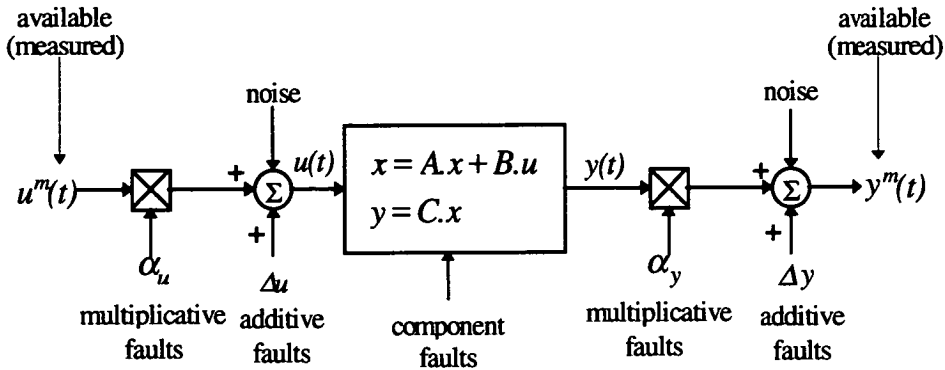


Fig. 6.5
Input-Output Faults in a Linear System

The GA codification for this problem could be something like fig.6.6 below:

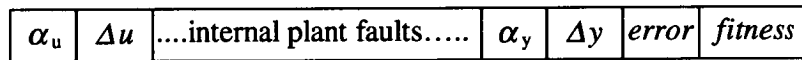


Fig. 6.6
Typical Codification for detecting input/internal/output faults

Where the faults Δu , α_u , Δy , α_y , can represent scalar or vectored variables. This also means that we can code any plant fault including internal faults as well as input-output faults into the chromosome. Internal plant faults must be explicitly defined parametrically as part of the plant dynamics, for instance: $\dot{x} = A.x + B.u + D.f$, where f =fault. The fitness of the chromosome can then be simply computed as the inverse of the error function:

$$error = \sum_{j=1}^N (y_j^m - y_{GAj})^2 \quad \text{Eqn.6.20}$$

where y_j^m is the actual measured plant output containing the fault at sample number: j , and y_{GAj} is the predicted plant output computed using the fault information contained within chromosome. Thus we can search through a large fault space by simply defining a large population of chromosomes, each with a different value and type of fault.

For instance a population of 100 chromosomes can initially search through 100 different fault configurations simultaneously. Figure 6.7 below illustrates a typical setup in which a genetic algorithm is used to detect and estimate faults. Note that the RBF represents the fault free model of the plant.

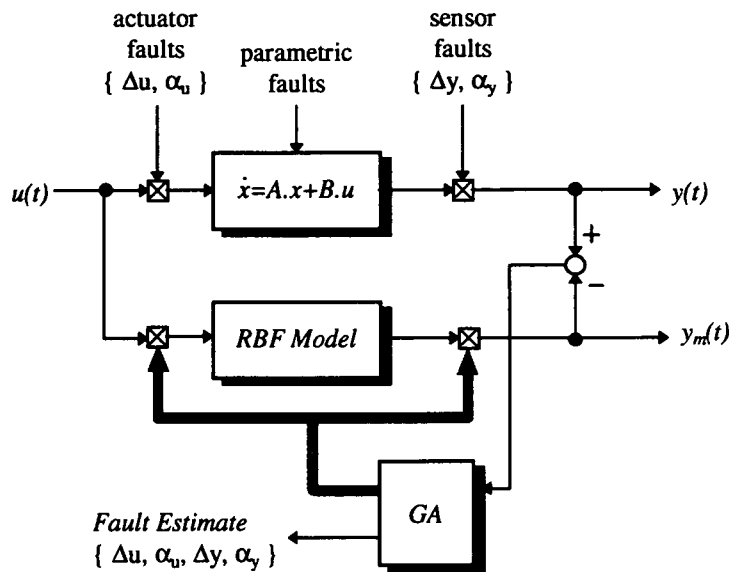


Fig. 6.7
Using Genetic Algorithms to Detect Faults

From figure 6.7, the genetic algorithm essentially injects the fault information contained within each chromosome into the input/output of the RBF model. The predicted model output is then computed and compared with the actual plant output. The resulting error is used to compute the corresponding fitness of the chromosome.

One of the advantages of using genetic algorithm search is the inherent robustness to noise. Furthermore, constraints can also be imposed, which can narrow the search space and produce a faster convergence. For instance a faulty sensor can produce a maximum output within its saturation range. This constrains the maximum allowable fault range. Methods of constrained optimization were discussed in chapter 1. A disadvantage of GA is that this algorithm is computationally intensive because the error calculation requires running a full simulation of the model output (over N samples) for each chromosome. Thus for a population of 100 chromosomes, we need to run a complete model simulation 100 times at each generation step. However as we shall see, convergence is generally very rapid.

The accuracy of the predicted fault depends largely upon the accuracy of the reference model, the presence of noise and external disturbances. A further advantage of such a method shown in figure 6.7 is the potential applicability to nonlinear systems. Simulation results using the longitudinal aircraft model are presented next, comparing genetic algorithms with conventional parity space methods.

6.2.2 Detecting Input/Output Faults in Linear Systems:

For the first simulation, a model based FDI technique described earlier in 6.1.4 is used. The model is constructed using radial basis function networks, this is essentially a black box model. We use genetic algorithms for two parts of this simulation:

- (i) To train (offline) the radial basis function network to model the longitudinal aircraft dynamics.
- (ii) To detect, locate and estimate single input, single output and multi input/output faults.

The linearized open loop longitudinal dynamics described in the appendix is used, and is again illustrated below. For the state space system: $\dot{x} = A.x + B.u$, where: $x = [q \ \alpha \ \theta]^T$ and $u = \delta_e$, q : pitch rate (deg/sec.), α : angle of attack (deg.), and θ : pitch angle (deg.), the control δ_e is the elevator surface command.

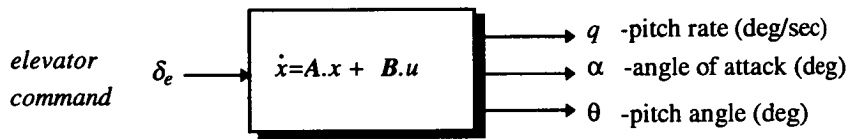


Figure 6.8
Open loop Longitudinal Dynamics

Linearizing the model about $x_0 = [0, 1.5, 0]^T$ gives the following state space matrices:

$$A = \begin{bmatrix} -0.9870 & -22.9501 & 0 \\ 1.0000 & -1.3290 & 0 \\ 1.0000 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -28.3409 \\ -0.1680 \\ 0 \end{bmatrix}$$

Note that we could also have used the nonlinear model, however for parity space comparison, a linearized model in state space form is required.

(i) Training the RBF with GA:

For the first part of the simulation, we need to initially train the radial basis function network (RBF) to model the linearized aircraft dynamics. The radial basis function network was described in detail previously in chapter 2. We use a genetic algorithm to train the RBF network as in chapter 2.

The longitudinal aircraft model has 3 states (outputs) and therefore requires 3 separate radial basis function networks. This is illustrated in figure 6.9 below. In this instance, random training data was generated from the linearized model. The data is then used to train the RBF network using genetic algorithms.

Modeling the linearized aircraft with a radial basis function network. The z^{-1} are time delay (or shift) operators:

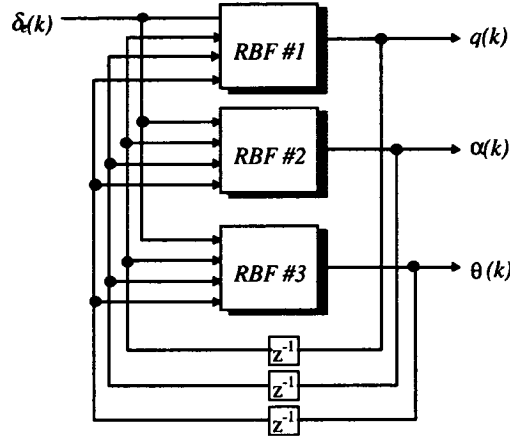


Fig.6.9
RBF Model of the Longitudinal Aircraft Dynamics

The chromosomal representation of the training problem is illustrated below Fig.6.10. Note that all three RBF are trained simultaneously.

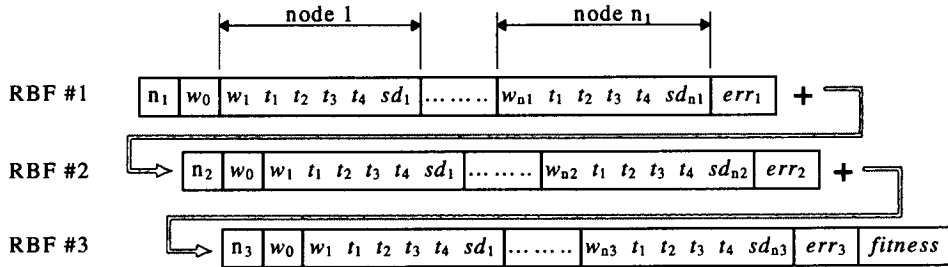


Fig.6.10
Chromosomal Representation of RBF Training for Longitudinal Aircraft Model

Where n_1 =number of hidden nodes in the first RBF#1, n_2 =number of hidden nodes in the second RBF#2, and n_3 =number of hidden nodes in the third RBF#3. The error values err_1 , err_2 , and err_3 are given by:

$$err_1 = \sum_{j=1}^N (q_j - q_{RBF_j})^2 \quad err_2 = \sum_{j=1}^N (\alpha_j - \alpha_{RBF_j})^2 \quad err_3 = \sum_{j=1}^N (\theta_j - \theta_{RBF_j})^2 \quad \text{Eqn.6.21}$$

The fitness is evaluated as the inverse of the sum of the 3 error values: $fitness = 1/(err_1 + err_2 + err_3)$. This means that all three RBF networks are trained simultaneously. Results of the training is illustrated on the following pages. The setup for the genetic algorithm is as follows: Hidden neurons: 10 in each RBF, population: 10, Probability of crossover $P_c=0.6$, Probability of mutation $P_m=0.1$, and binary tournament selection. We can see that the genetic algorithm converges within 100 generations.

This fast convergence is due to the linear quadratic nature of the cost function. Referring to figure 6.11, there are two columns, the left column is used for testing (verification) the radial basis function network using a step response, and the right column is used to train the radial basis function using random data. The blue plot is the RBF output, and the red plot is the linearized aircraft output. The two graphs agree well. Note however the presence of a slight DC drift on the pitch angle (bottom left graph). It is impossible and impractical for a model to follow the actual open loop plant precisely.

Legend: ■ RBF ■ Linearized Aircraft

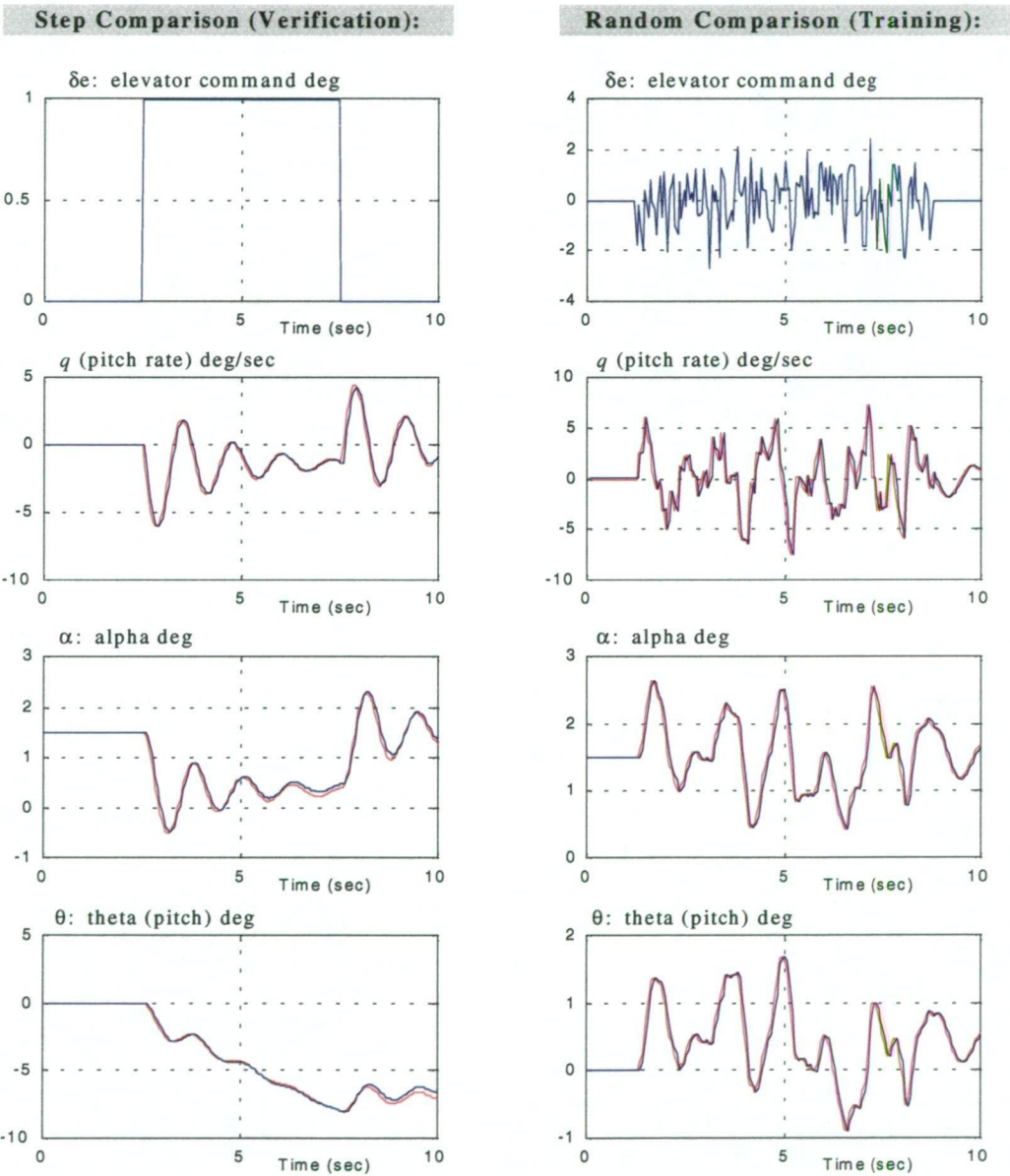


Fig.6.11
Comparing the RBF model output with Plant

(ii) **Fault Detection with Hybrid genetic algorithms:**

Using the previously trained RBF, the RBF can now be applied to the task of fault detection and isolation. In these next set of simulations, the genetic algorithm is used to both detect and quantify the following faults: single input faults, single output faults, multiple input/output faults, single input faults with time of occurrence. Before presenting the results, the simulation setup is depicted below in figure 6.12 in greater detail:

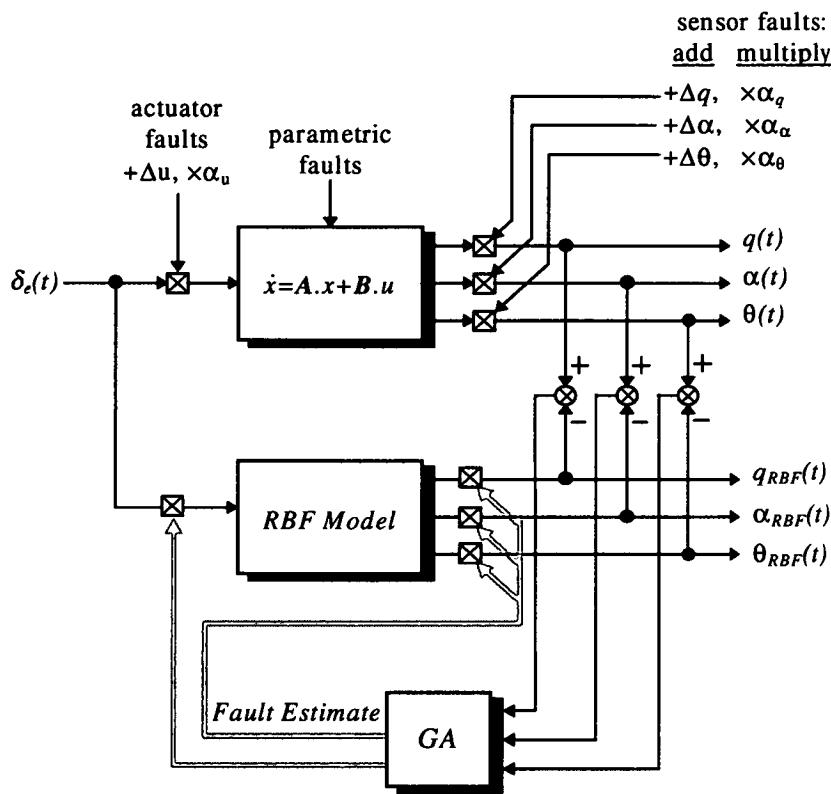


Fig.6.12
Fault detection using genetic algorithms and radial basis functions

It is assumed that all the states are available for measurement, thus C matrix is the identity matrix: $y(t)=x(t)$. Additive and multiplicative faults are both shown in fig.6.12. The Chromosomal representation of the above problem is shown below fig 6.13. The error is computed as the sum of equations 6.21, the fitness is the inverse of the error. No constraints are imposed on this simulation problem.

$\times\alpha_u$	$+\Delta u$	$\times\alpha_q$	$+\Delta q$	$\times\alpha_\alpha$	$+\Delta\alpha$	$\times\alpha_\theta$	$+\Delta\theta$	error	fitness
------------------	-------------	------------------	-------------	-----------------------	-----------------	-----------------------	-----------------	-------	---------

Fig.6.13
Chromosomal representation for the above problem

Detecting faults with GA as illustrated above works as follows: initially assume that the system is free from faults, and the RBF model output exactly matches the plant dynamics $x(t)=x_m(t)$. When a fault occurs, the genetic algorithm will begin to search through all possible *fault-spaces* until a particular fault is found (or combination of faults). At this point, the RBF model + fault best matches the faulty plant output $x(t)$. Note that the above FDI methodology is effectively independent of any controller. Simulation results are given below.

(iii) Single Input Faults:

In this simulation, single input faults are detected. A multiplicative fault of 0.75 and additive fault of 0.00 is introduced into the input thus: $\Delta u=0$ $\alpha_u=0.75$. The genetic algorithm correctly locates the fault after 150 generations. Refer to figure 6.14. The three hybrid genetic algorithms are compared with results obtained using conventional parity space methods. A summary is illustrated below in table 6.1 including computational effort:

Method:	Error:	MFP
Parity Space	0	< 1
GA	< 10^{-2}	22
GA + Simulated Annealing	< 10^{-2}	25
GA + Greedy Search	< 10^{-2}	7

Table.6.1
Comparison of conventional Parity space with hybrid GA

The error is defined as the RMS difference between the actual fault and estimated fault. Note that the error can never be zero due to the modeling inaccuracies of the RBF neural network used in the simulation. The above simulations are typical, with the greedy algorithm consistently giving better results. The setup for the genetic algorithm is as follows: population: 20, $P_c=0.6$, $P_m=0.2$, binary tournament selection is used. Convergence properties of the GA and hybrid methods are illustrated below. In all three cases, all methods converge to the correct fault.

Figure 6.14 compares the convergence rate of the three methods. Note that both conventional genetic algorithms and hybrid GA + simulated annealing resulted in almost comparable convergence rates. However the hybrid GA + greedy search resulted in a factor of four improvement in convergence over the conventional genetic algorithm.

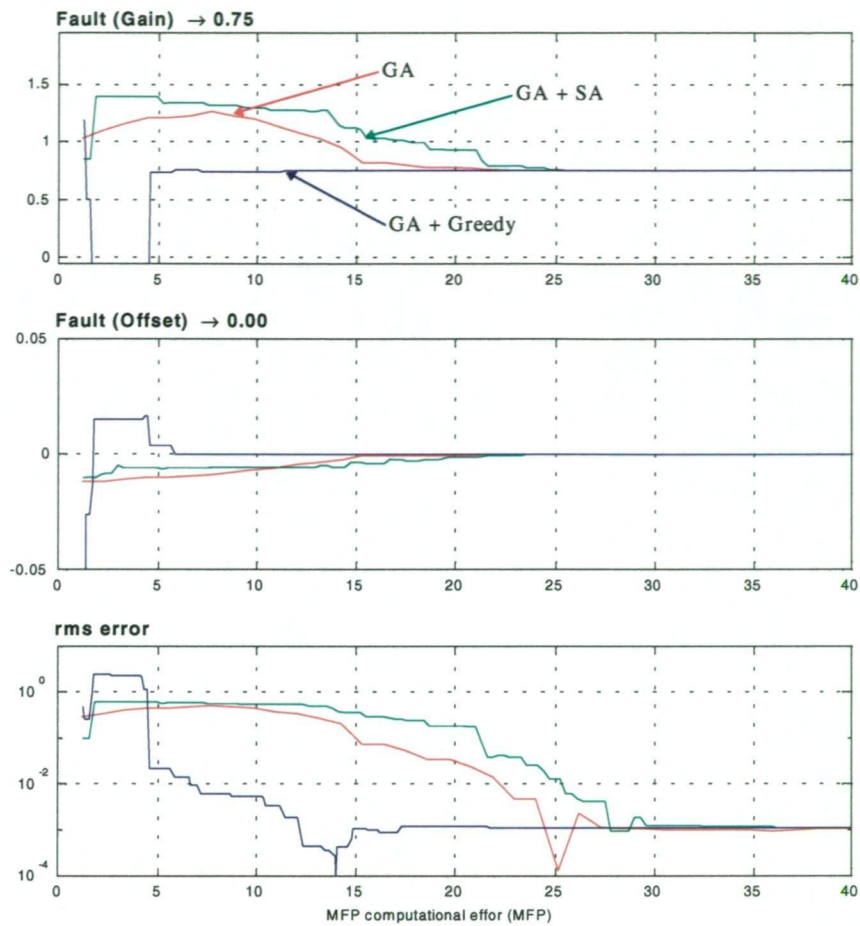


Fig.6.14
Single input faults

(iv) Single Output Fault:

In this simulation, single output faults are detected. A multiplicative fault of 0.85 and additive fault of 0.00 is introduced into the first output thus: $\Delta q=0$ $\alpha_q=0.85$. The genetic algorithm correctly locates the fault after 100 generations.

Method:	Error:	MFP
Parity Space	0	< 1
GA	< 10 ⁻⁴	20
GA + Simulated Annealing	< 10 ⁻⁴	25
GA + Greedy Search	< 10 ⁻⁴	13

Table.6.2

The setup for the genetic algorithm is as follows: population: 20, Pc=0.6, Pm=0.2, binary tournament selection is used. Typical convergence plot is illustrated in figure 6.15 below:

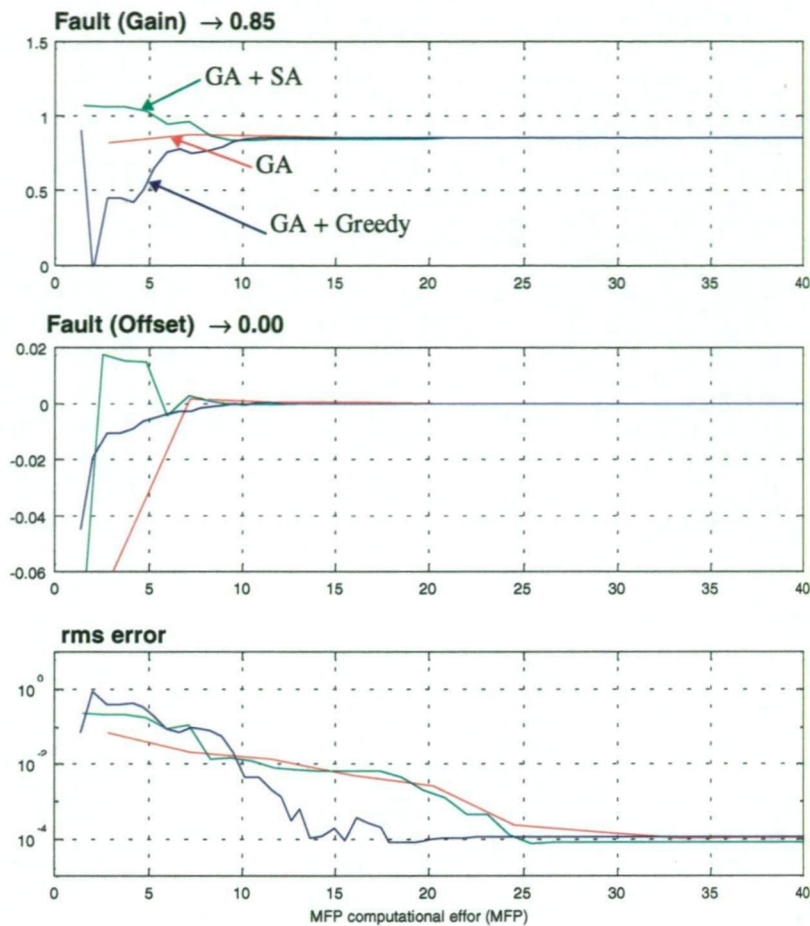


Fig.6.15
Single output faults

(v) Multiple Input - Output Fault:

In this simulation, multiple input - output faults are estimated. A multiplicative fault of 0.75 and additive fault of 0.00 is introduced into the input ie: $\Delta u=0$ $\alpha_u=0.75$, and a second fault is also introduced into the output: $\Delta q=0$ $\alpha_q=0.85$. All three hybrid genetic algorithms correctly locate both faults after 700 generations, refer to figure 6.16. Table 6.3 below compares the results:

Convergence at error < 10 ⁻² :					
Method:	MFP	GAIN → 0.75 (input)	OFFSET → 0.0 (input)	GAIN → 0.85 (output)	OFFSET → 0.0 (output)
GA:	250	0.7592	0.0018	0.8481	0.0020
SA	418	0.7416	-0.0024	0.8511	-0.0029
Greedy	60	0.7480	0.0053	0.8515	0.0059

Table.6.3

Typical convergence plots of the three methods:

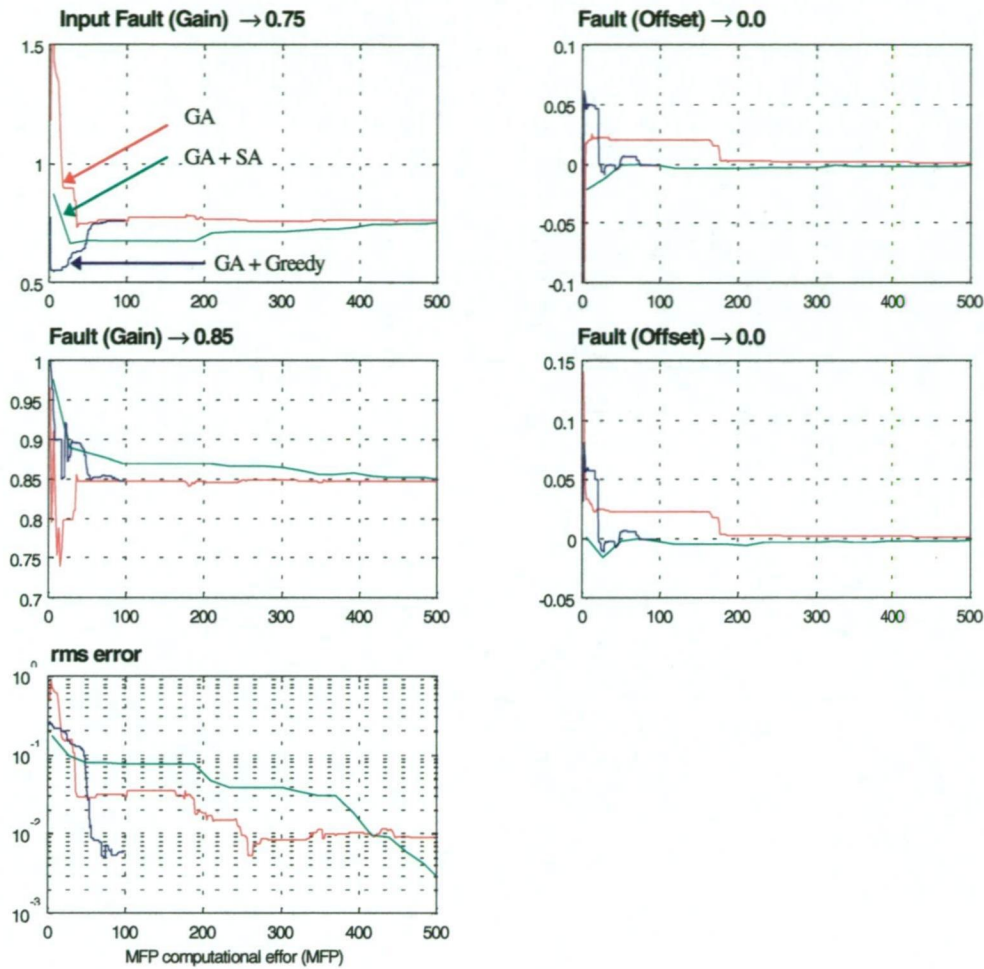


Fig.6.16
Multiple input output faults

(vi) Detecting time of fault:

The time at which the fault occurred can also be estimated, by introducing an extra time parameter into the chromosomal representation of the faults, thus T_{fault} . Chromosomal representation for this problem is illustrated below:

$\times \alpha_u$	$+\Delta u$	$\times \alpha_q$	$+\Delta q$	$\times \alpha_\alpha$	$+\Delta \alpha$	$\times \alpha_\theta$	$+\Delta \theta$	T_{fault}	error	fitness
-------------------	-------------	-------------------	-------------	------------------------	------------------	------------------------	------------------	--------------------	-------	---------

Fig.6.17
Codification for detecting time of fault

Where T_{fault} refers to a sample number which must be an integer between 1 to N. This is a simple constraint which can be handled by a repair algorithm. The error and fitness are again calculated according to equations 6.21.

For this simulation, a multiplicative fault of 0.75 and additive fault of 0.00 is introduced into the input ie: $\Delta u=0$ $\alpha_u=0.75$. Furthermore, the time of fault is set to occur at the 50th sample. Results are summarized below:

Convergence at error < 10 ⁻² :				
Method:	MFP	GAIN → 0.75 (input)	OFFSET → 0.0 (input)	Time of Fault→ 50 (sample)
GA	66	0.7563	-0.0000	50.0000
GA + SA	152	0.7568	-0.0001	50.0000
GA + Greedy	21	0.7519	-0.0000	49.0000

Table.6.4

Note that both input gain and offset faults are floating point numbers, however the time of fault must be an integer (sample number). Table 6.4 compares the convergence of the three genetic algorithm methods. For this type of problem, the parity space technique offers no direct methodology for solving the time of fault.

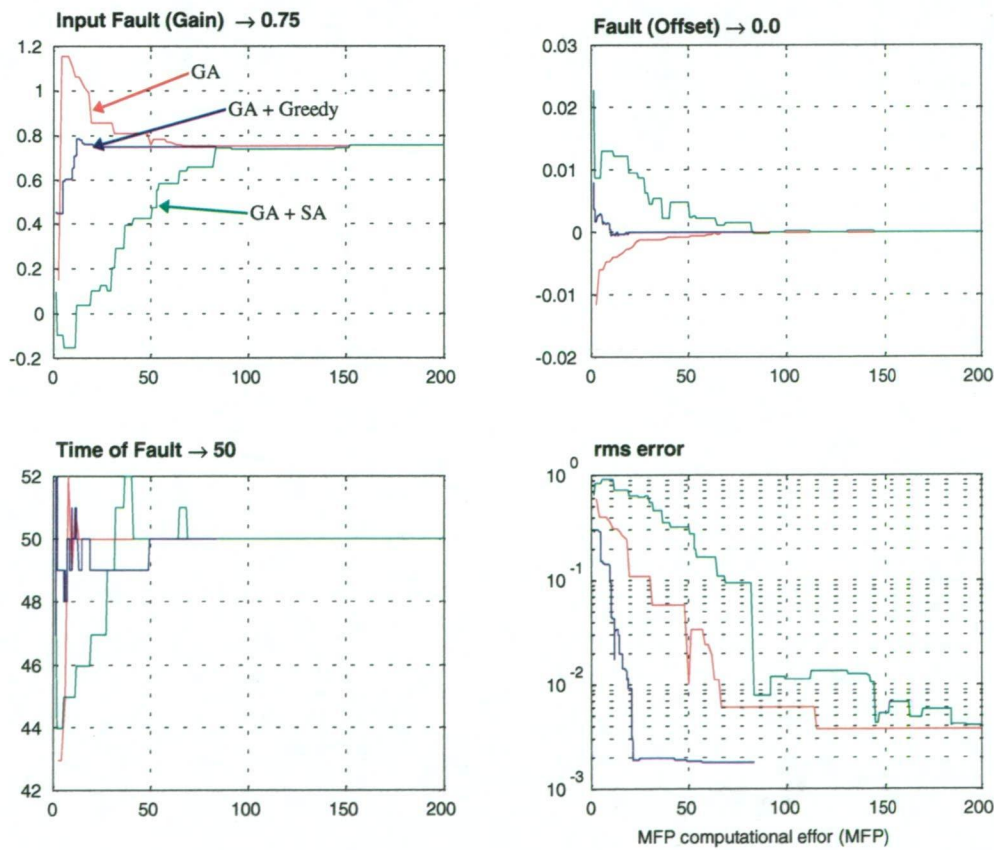


Fig.6.18

6.2.3 Detecting Internal Faults:

In this last simulation, the nonlinear aircraft longitudinal model is used. Genetic algorithms are applied to the problem of detecting internal plant faults. Given the generalized dynamical system representing the aircraft longitudinal dynamics:

$$\dot{x} = A(x, p) + B(x).u \quad \text{Eqn.6.22}$$

in which the vector p consists of components representing potential internal plant faults in parametric form. If this system is linearized, the following linear model is obtained:

$$\dot{x} = A.x + B.u + D.p \quad \text{Eqn.6.23}$$

The faults now appear parametrically as inputs to a linear system. The full simulation setup is illustrated below:

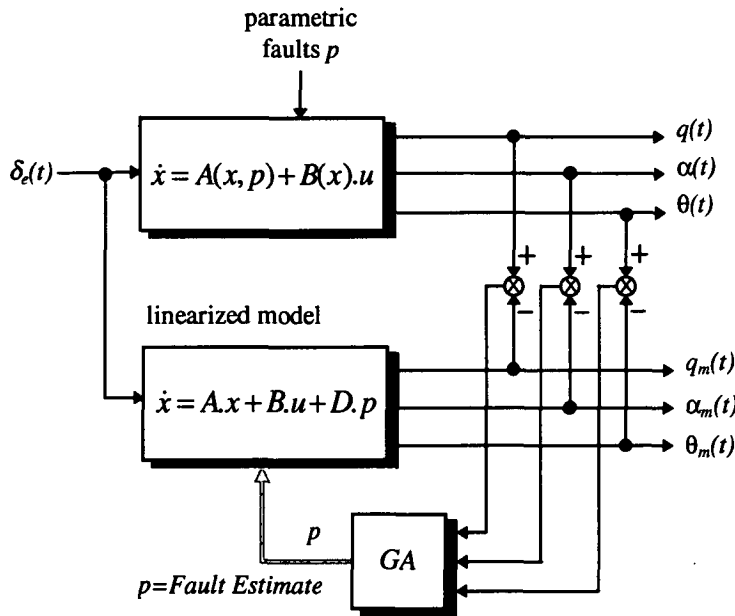


Fig.6.19

Fault detection using a linearized parametric form of plant internal faults

For this simulation, the faults are assumed to occur in the following coefficients of the aircraft equation of motion: $q = \{\tilde{M}_\alpha, \tilde{M}_q, Z_\alpha\}$, refer to appendix 8.1 for a description of these aerodynamic coefficients. Further, for this simulation we assume that only multiplicative faults occur, and that there are no additive faults. Thus we denote the multiplicative fault for each parameter as:

$FAULT_ \tilde{M}_\alpha$: multiplicative fault in \tilde{M}_α

$FAULT_ \tilde{M}_q$: multiplicative fault in \tilde{M}_q

$FAULT_Z_\alpha$: multiplicative fault in Z_α

Then the chromosomal representation for this problem can be simply defined as:

$FAULT_ \tilde{M}_\alpha$	$FAULT_ \tilde{M}_q$	$FAULT_Z_\alpha$	error	fitness
----------------------------	-----------------------	-------------------	-------	---------

Fig.6.20

Codification for detecting internal plant faults above

Again, the error is computed as per equations 6.21. The first and second faults can be interpreted as altered pitch aerodynamic due to airframe change, the third fault as altered yaw aerodynamics due to airframe change. Note that a value of 1 in each indicates a fault free condition. Results are given below:

In this simulation, the following multiplicative faults are assumed to occur:

$$FAULT_ \tilde{M}_\alpha = 1.1$$

$$FAULT_ \tilde{M}_q = 0.8$$

$$FAULT_Z_\alpha = 0.9$$

Results for this simulation are illustrated in figure 6.21, convergence is within 250 generations. After 250 generations, the values found by the GA search are: 1.100, 0.8002, 0.8999, which agree well with the above faults.

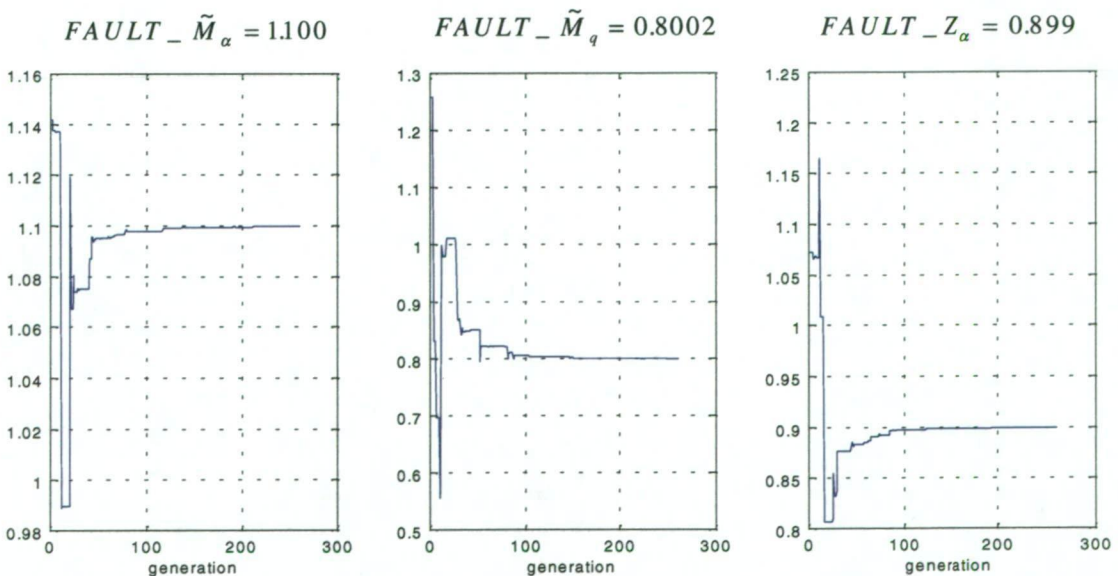


Fig.6.21

Convergence properties of the GA, the fault is correctly predicted

6.3 Chapter Summary and Conclusion:

(i) Results: From simulation results, genetic algorithms can be readily applied to fault diagnosis of linear systems with input and output faults. We have used parity space as a basis for comparison. Parity space methods are easy to apply but suffer from serious limitations such as inability to detect multiple input/output faults. GA methods do not have these limitations, and convergence is generally rapid. Results using hybrid genetic algorithms show that the combined GA+greedy search outperforms the conventional GA and the hybrid GA+simulated annealing algorithm. We have also shown that genetic algorithms can be used to detect the time at which the fault occurred.

(ii) Advantages: Genetic algorithms can be used to detect multiple input and output faults. Furthermore, fault diagnosis using genetic algorithms is not restricted to linear systems, it can also be applied to nonlinear time varying systems, provided that the fault be modeled parametrically. Additionally, genetic algorithms can also be applied in detecting internal plant faults in instances where the internal faults may be described parametrically.

The underlying concept of using genetic algorithms in fault diagnosis enables hundreds of possible fault combinations to be searched over a wide parameter space. In addition, the use of chromosomal representation enables greater freedom in describing a fault, which may appear as a discontinuous function, for instance an ON/OFF condition may be detected, or some discrete function. We also have a greater choice in selecting a fitness function which may be linear, nonlinear or discontinuous. In our simulation, a simple Euclidean distance metric was used.

(iii) Disadvantages: As with any GA, computational effort required was much greater than conventional parity space. From simulation results, the computational effort was approximately two orders of magnitude greater when genetic algorithms were used. This is an issue which needs to be addressed if genetic algorithms are to be accepted in real time and online fault diagnosis, in particular life-critical systems such as aircraft and medical applications. A further disadvantage is that an input/output (black box) model of the system is required. The reason for the higher computational effort is because the genetic algorithm runs full model simulation in order to compute the fitness function for each member of the population over each generation. Perhaps a means to reduce computational effort is to use a simpler fitness function or a variation such as a statistical function. This may be a topic of future research.

(iv) **Future work:** From this brief survey, the model based methods offer interesting applications for research in fault detection and isolation. Other methods including observer based which have yet to be investigated, such as variable structure/sliding observers and nonlinear (lie algebraic) methods. The advantages of using these techniques in control are well established, including robustness to uncertainties, and invariance to disturbances. These advantages can be carried over to the design of robust observer applications.

There are clearly a multitude of methods for fault diagnosis as seen from the survey, a full discussion would be impractical. The popular methods of parity space are well established and offer a simple and reliable methodology of detecting faults for linear systems.

Although model based and observer based methods require *a-priori* knowledge of the plant, the main difference is that model based methods do not require a knowledge of the plant states, only input/output measurements. Consequently, more information is necessary for observer based methods.

From the literature on FDI, we can see that there are many techniques and variations of fault detection, in which GA can be applied. Currently, research and state-of-the-art FDI focuses primarily on nonlinear systems and robust methods with the application of neural networks and fuzzy set theory.

The application of fault diagnosis comparing (or hybridizing) genetic algorithms with other statistical and fuzzy methods described in section 6.1 would be an interesting topic for future research with applications to genetic algorithms.

6.4 References and Further Reading:

- [1] J. Chen, R.J. Patton,
Robust Model Based Fault Diagnosis for Dynamic Systems
Kluwer Academic Publishers, 1999
- [2] McKenna J.T.,
Blocked Static Ports Eyed in Aeroperu 757 Crash
Aviation Week and Space Technology, pp. 76, November 11, 1996
- [3] Learmont, D.
Airline Safety Review
Flight International, Jan.27, 1993
- [4] National Transport Safety Board NTSB-AAR 79-17
Aircraft Accident Report, American Airlines Inc. DC-10-10, N110AA
Chicago-O'Hare International Airport, Chicago Illinois, May 25, 1979, Jan.27, 1993
- [5] P.M. Frank
Fault Diagnosis in dynamic Systems Using Analytical and Knowledge-based Redundancy - A survey and
Some New Results, Automatica Vol. 26, No. 3, pp. 459-474, 1990
- [6] E.Y.Chow, A.S.Willsky
Analytical Redundancy and the Design of Robust Failure Detection Systems
IEEE Transactions on Automatic Control, Vol. AC-29, No.7, pp.603-614, July 1984
- [7] R. Isermann
Fault Diagnosis of Machines via Parameter Estimation and Knowledge Processing - A Tutorial paper
Automatica, Vol. 29, No. 4, pp. 815-835, 1993
- [8] R. Milne
Strategies for Diagnosis
IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-17, No.3, pp. 333-339, May/June 1987
- [9] J.J. Gertler
Survey of Model Based Failure Detection and Isolation in Complex Plants,
IEEE Control Systems Magazine, Vol.8, No.6, pp. 3-11, December 1988
- [10] R. Isermann
Process Fault Detection Based on Modeling and Estimation Methods - A Survey
Automatica, Vol. 20, No. 4, pp. 387-404, 1984
- [11] M.A.Massoumnia, G.C.Vergheze, A.S.Willsky
Failure Detection and Identification
IEEE Transactions on Automatic Control
Vol. 34, No.3, pp.316-321, March 1989
- [12] A.S.Willsky
A Survey of Design Methods for Failure Detection in Dynamic Systems
Automatica Vol.12, pp. 601-611, 1976
- [13] K.Danai, H.Chin
Fault Diagnosis With Process Uncertainty.
Journal of Dynamic Systems, Measurement and Control, Vol.113, pp.339-343, September 1991
- [14] H.Chin, K.Danai
A Method of Fault Signature Extraction for Improved Diagnosis
Journal of Dynamic Systems, Measurement and Control
Vo.113, pp.634-638, December 1991

-
- [15] T.Ono, T.Kumamaru, A.Maeda, S.Sagara, K.Kumamaru
Influence Matrix Approach to Fault Diagnosis of Parameters in Dynamical Systems.
IEEE Transactions on Industrial Electronics. Vo.IE-34, No.2, , pp.285-291, May 1987
 - [16] Hong Jing, Hong Yue Zhang
Optimal Parity Vector Sensitive to Designated Sensor Fault
IEEE Transaction on Aerospace and Electronic Systems, Vol.35, No.4, pp.1122-1128, October 1999
 - [17] J.Gertler, D.Singer
A New Structural Framework for Parity Equation based Failure Detection and Isolation
Automatica, Vo.26, No.2, pp.381-388, 1990
 - [18] Xi-Cheng Lou, A.S.Willsky, G.C.Verghese
Optimally Robust Redundancy Relations for Failure Detection in Uncertain Systems
Automatica, Vo.22, No.1, pp.333-344, 1986
 - [19] M.P.Frank, R.J. Patton, R. Clark
Fault Diagnosis in Dynamic Systems - Theory and Applications
Kluwer Academic Publishers, 1989
 - [20] A. Medvedev
Fault Detection and Isolation by a Continuous Parity Space Method
Automatica, Vo.31, No.7, pp.1039-1044, 1995
 - [21] J.F.Magni, P.Mouyon
On Residual Generation by Observer and Parity Space Approaches
IEEE Transactions on Automatic Control, Vol. 39, No.2, pp.441-447, February 1994
 - [23] H.Wang, H.Kropholler, S.Daley
Robust Observer Based FDI and its Application to the Monitoring of a Distillation Column.
Transactions on of the Institute of Measurement and Control, Vo.15, No.5, pp.221-227, 1993
 - [24] Randal K. Douglas, Jason L. Speyer
Robust Fault Detection Filter Design
Proceedings of the American Control Conference, pp.91-96, 1995
 - [25] Jaehong Park, Giorgio Rizzoni
A New Interpretation of the Fault Detection Filter Part.1, Closed Form Algorithm
International Journal of Control, Vol.60, No.5, pp.767-787, 1994
 - [26] Jaehong Park, Y.Halevi, Giorgio Rizzoni
A New Interpretation of the Fault Detection Filter Part.2, The Optimal Detection Filter
International Journal of Control, Vol.60, No.6, pp.1339-1351, 1994
 - [27] J.Chen, R.J.Patton, H.Y.Zhang
Design of Unknown Input Observers and Robust Fault Detection Filters
International Journal of Control
Vo.63, No.1, pp.85-105, 1996
 - [28] M. Hou, P.C. Muller
Design of Observers for Linear Systems with Unknown Inputs
IEEE Transactions on Automatic Control, Vol. 37, No. 6, pp.871-875, June 1992
 - [29] Y. Park, J.L. Stein
Steady State Optimal State and Input Observer for Discrete Stochastic Systems
Journal of Dynamic Systems, Measurement and Control, Vo.111, pp.121-127, June 1989
 - [30] M.Saif, Y.Guan
A Novel Approach to the Design of Unknown Input Observers
IEEE Transactions on Automatic Control, Vol. 36, No.5, pp.632-635, May 1991

-
- [31] M.Saif, Y.Guan
A New Approach To Robust Fault Detection and Identification
IEEE Transaction on Aerospace and Electronic Systems, Vol.29, No.3, pp.685-695, July 1993
 - [32] R.N.Clark
A Simplified Instrument Failure Detection Scheme
IEEE Transaction on Aerospace and Electronic Systems, Vol.AES 14, No.4, pp.558-563, July 1978
 - [33] R.N.Clark
Instrument Fault Detection
IEEE Transaction on Aerospace and Electronic Systems, Vol.AES 14, No.3, pp.456-465, May 1978
 - [34] K.J.Astrom, P.Eykhoﬀ
System Identification - A Survey
Automatica, Vol. 7, pp. 123-162, 1971
 - [35] K.J. Hunt
A Survey of Recursive Identification Algorithms
Transactions of the Institute of Measurement and Control, Vo.8, No.5, pp.273-278, October-December 1986
 - [36] L.A.Pineiro, D.J.Biezad
Real-Time Parameter Identification Applied to Flight Simulation
IEEE Transaction on Aerospace and Electronic Systems, Vol.29, No.2, pp.290-301, April 1993
 - [37] F.E. Thau
Observing the State of Nonlinear Dynamic Systems.
International Journal of Control, Vol.17, No.3, pp.471-479, 1973
 - [38] R. Mehra, S. Seereeram, D. Bayard, F. Hadaegh
Adaptive Kalman Filtering, Failure Detection and Identification for Spacecraft Attitude Estimation
Fourth International Conference, pp. 176-181
 - [40] S.M.Joshi
Robustness of Extended Kalman Type Observers
International Journal of Control, Vol.45, No.5, pp.1857-1866, 1987
 - [41] B.L.Walcott, S.H.Zak
Comparative Study of Nonlinear State Observation Techniques
International Journal of Control, Vol.45, No.6, pp.2109-2132, 1987
 - [42] B.L.Walcott, S.H.Zak
Combined Observer Controller Synthesis for Uncertain Dynamical Systems with Applications
International Journal of Control, Vol.18, No.1, pp.88-104, 1988
 - [43] Christopher Edwards, Sarah K. Spurgeon
On the Development of Discontinuous Observers
International Journal of Control, Vol.59, No.5, pp.1211-1229, 1994
 - [44] Ibrahim Haskara, Umit Ozguner, Vadim Utkin
On Sliding Mode Observers via Equivalent Control Approach
International Journal of Control, Vol.71, No.6, pp.1051-1067, 1998
 - [45] J.J.E.Slotine, J.K.Hedrick, E.A.Misawa
On Sliding Observers for Nonlinear Systems
Journal of Dynamic Systems, Measurement and Control, Vo.109, pp.245-252, September 1987
 - [46] J. Birk, M.Zeitz
Extended Luenberger Observer for nonlinear Multivariable Systems
International Journal of Control, Vol.47, No.6, pp.1823-1836, 1988

-
- [47] Nikolaos Kazantzis, Costas Kravaris
Nonlinear Luenberger type Observer with Applications to Catalyst Activity Estimation
Proceedings of the American Control Conference, pp.1756-1761, 1995
 - [48] D.G. Luenberger
An Introduction to Observers
IEEE Transactions on Automatic Control, Vol. AC-16, No. 6, pp.596-602, December 1971
 - [49] J.K.Hedrick, E.A.Misawa
Nonlinear Observers, State of the art Survey
Journal of Dynamic Systems, Measurement and Control, Vol.111, pp.344-352, September 1989
 - [50] May-Win L. Thein, Eduardo. A. Misawa
Comparison of the Sliding Observer to Several State Estimators Using a Rotational Inverted Pendulum.
Proceedings of the 34th Conference on Decision and Control, Vol. 4, pp.3385-3390, December 1995, New Orleans LA.
 - [51] G.Schram, S.M.Gopisetty, R.F.Stengel
A Fuzzy Logic Parity Space Approach To Actuator Failure Detection and Identification
American Institute of Aeronautics and Astronautics 1998, (internet download)
 - [52] S.M. Gopisetty, R.F.Stengel
Detecting and Identifying Multiple Failures in a Flight Control System
AIAA-98-4488 conference Boston aug.1988, (internet download)
 - [53] J.X.Xu, H. Hashimoto
Parameter Identification Methodologies based on Variable Structure Control
International Journal of Control, Vol.57, No.5, pp.1207-1220, 1993
 - [54] Randal K. Douglas, Jason L. Speyer
An H_{∞} Bounded Fault Detection Filter.
Proceedings of the American Control Conference, pp.86-90, 1995
 - [55] Y.Jin, J.Jiang, J.Zhu
Neural Network Based Fuzzy Identification and Its Application to Modeling and Control of Complex Systems.
IEEE Transactions on Systems, Man and Cybernetics, Vol.25, No.6, pp. 990-997, June 1995
 - [56] E.G. Laukonen, K.M.Passino, V.Krishnaswami, G.C.Luh, G.Rizzoni
Fault Detection and Isolation for an Experimental Internal Combustion Engine via Fuzzy Identification.
IEEE Transactions on Control System Technology, Vol.3, No.3, pp. 347-355, September 1995
 - [57] K.Liu, F.L.Lewis
Adaptive Tuning of Fuzzy Logic Identifier for Unknown Non Linear Systems
International Journal of Adaptive Control and Signal Processing, Vol.8, No.3, pp. 573-586, 1994
 - [58] R. Isermann
On Fuzzy Logic Applications for Automatic Control, Supervision, and Fault Diagnosis
IEEE Transactions on Systems, Man and Cybernetics, Vol. 28, No. 2, pp. 221-235, March 1998
 - [59] P.K. Fink, J.C.Lusth
Expert Systems and Diagnostic Expertise in the Mechanical and Electrical Domains
IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC 17, No.3, pp. 341-349, May/June 1987
 - [60] M.J. Pazzani
Failure-Driven Learning of Fault Diagnosis Heuristics
IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC 17, No.3, pp. 380-394, May/June 1987
 - [61] R. Isermann, B.Freyermuth
Process Fault Diagnosis Based on Process Model Knowledge - Part 1, Principles for Fault Diagnosis With Parameter Estimation.
Journal of Dynamic Systems, Measurement and Control, Vol.113, No.2, , pp.620-626, December 1991

-
- [62] R. Isermann, B.Freyermuth
Process Fault Diagnosis Based on Process Model Knowledge - Part II, Case Study Experiments
Journal of Dynamic Systems, Measurement and Control, Vol.113, No.2, , pp.627-633, December 1991
 - [63] R.F.Stengel
Intelligent Failure-Tolerant Control
IEEE Control Systems Magazine, pp.14-23, June 1991
 - [64] Y.Ishida, N. Adachi, H. Tokumaru
A Topological Approach to Failure Diagnosis of Large-Scale Systems
IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC 15, No.3, pp. 327-333, May/June 1985
 - [65] A. Srinivasan, C.Batur
Hopfield/ART-1 Neural Network Based Fault Detection and Isolation
IEEE Transactions on Neural Networks, Vol.5, No.6, pp. 890-899, November 1994
 - [66] T.Sorsa, H.N.Koivo, H.Koivisto
Neural Networks in Process Fault Diagnosis
IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No.4, pp.815-825, August 1991
 - [67] T.Sorsa, H.N.Koivo
Application of Artificial Neural Networks in Process Fault Diagnosis
Automatica, Vol. 29, No.4, pp.483-489, 1993
 - [68] S.R.Naidu, E.Zafiriou, T..J.McAvoy
Use of Neural Networks for Sensor Failure Detection in a Control System
IEEE Control Systems Magazine, pp. 49-55, April 1990
 - [69] T.H.Guo, J.Musgrave
Neural Network based Sensor Validation for Reusable Rocket Engines
American Control Conference, pp.1367-1372, 1995
 - [70] M.R.Napolitano, C.I.Chen, S.Naylor
Aircraft Failure Detection and Identification Using Neural Networks
Journal of Guidance, Control and Dynamics, Vol.16, No.6, pp. 999-1009, Nov-Dec 1993
 - [71] M.R.Napolitano, V.Casdorph, C.Neppach, S.Naylor, M.Innocenti, G.Silvestri
Online Learning Neural Architectures and Cross-correlation Analysis for Actuator Failure Detection and Identification.
International Journal of Control, Vol.63, No.3, pp.433-455, 1996
 - [72] J.C.DeLaat, W.C.Merrill
A Real Time Microcomputer Implementation of Sensor Failure Detection for Turbofan Engines
IEEE Control Systems Magazine, pp.29-37, June 1990
 - [73] G.F.Steven
Kalman Filtering - noise corrupted signal processing
Electronics and Wireless World, pp.1083-1085, Nov. 1988
 - [74] J.C.Deckert, M.N.Desai, J.J.Deyst, A.L.Willsky
F-8 DFBW Sensor Failure Identification Using Analytic Redundancy
IEEE Transactions on Automatic Control, Vol. AC-22, No. 5, pp.795-803, October 1977
 - [75] T.Takagi, M.Sugeno
Fuzzy Identification of Systems and Its Application to Modeling and Control
IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-15, No.1, pp. 116-132, January/Feb. 1985
 - [76] W.S. Levine
The Control Handbook
IEEE Press 1996

-
- [78] A.J. Fossard, D.Normand-Cyrot
Nonlinear Systems, Volume-1, Modeling and Estimation
Chapman & Hall 1995
 - [79] Jean-Jacques E. Slotine, Weiping Li
Applied Nonlinear Control
Prentice Hall 1991
 - [80] Michele Basseville
Detecting Changes in Signals and Systems- A Survey
Automatica, Vol. 24, No.3, pp.309-326, 1988
 - [81] R.M.Thong, M.B.Beck, A.Latten
Fuzzy Control of the Activated Sludge Wastewater Treatment Process
Automatica, Vol. 16, pp.695-701



Summary and Conclusions:

<i>Contents:</i>	
7.1	Conclusion p.7.2
7.2	Challenges and Future Development p.7.10
7.3	References and Further Reading p.7.14

7.1 Conclusion:

(i) Genetic Algorithms:

Genetic algorithms (GA) are a powerful generalized multiparameter search scheme. However, they cannot simply be applied to any problem blindly. Any method which can be used to enhance the performance of a GA (i.e. gradient or hybrid methods) should be considered. Thus careful application of the GA algorithm is always recommended. This can mean the difference between successful convergence or poor convergence. The rate of convergence is strongly affected by the shape of the fitness function. The selection of a proper fitness function is also important. Careful choice of mutation, crossover, and selection operators is also critical. However, when the GA search is performed over a search space of low dimensionality as seen in chapter 4 (SISO system), then the choice of mutation and crossover are less critical. One of the main concerns associated with genetic algorithms is *premature convergence*. This is caused by a superfit individual quickly proliferating throughout the population, reducing genetic diversity and generally resulting in rapid convergence to a sub-optimal solution. Conversely, using high mutation probabilities and selection schemes with low *selection pressure* can result in excessive genetic diversity and unacceptably slow convergence. A genetic algorithm should always be implemented as a judicious balance between avoiding premature convergence, and avoiding wasteful searching by ineffective selection schemes or excessive mutation probabilities. Some lessons learned from the simulation studies are:

1. ***Fitness functions***: For fitness functions which are quadratic, simple convex functions, or have few minima, convergence can be very rapid. The shape of the fitness function and the number local minima has a profound influence upon the convergence of the genetic algorithm. For example, designing a LQR using a genetic algorithm generally results in fast convergence. The fitness function should always be carefully selected. If a GA fails to converge, or takes unusually long to converge, the improper choice of fitness function is generally the primary cause.

This concept can be extended to constrained optimization problems in which a penalty function is used. Dynamic penalty functions require a procedure for scheduling (or increasing) the penalty coefficient. A simpler method would be to use a static penalty function in which the penalty coefficient remains constant. In our simulations, we made use of an infinite value of penalty parameter, in other words, if the solution is infeasible (i.e. constraint is not met) then the fitness is simply set to zero.

-
- 2. Codification:** According to the original schema theorem developed by Holland [2], binary coding was used to derive convergence rates and population takeover time for the basic GA. Unfortunately, the original schema theorem cannot be directly applied to floating point chromosomal representation. However, despite this, the majority of current publications on GA research use floating point representation. Refer to [6] for a comparison on binary and floating point representation. In all our simulations, floating point representation was used and found to work well.
- 3. Selection Operator:** The choice of selection operator has a strong influence upon the rate of convergence. Proportional fitness selection such as *roulette wheel selection* is common. This however can result in premature convergence and reduce initial search. Most selection operators however must bear some direct relation between probability of selection and fitness value. The two methods found to be most effective are: *ranking selection* and *tournament selection*, binary tournament selection was used throughout all the simulations and found to work well. Premature convergence can be avoided by reducing the *selection pressure*, for instance: reducing the number of sub-individuals m chosen from the population n in tournament selection can substantially reduce the rate of convergence. Since in general, a GA requires at least two parents, two different selection operators may be used, one for each parent. The first parent may be selected using a tournament selection scheme, the second using *pure random selection*. This method can sometimes prevent premature convergence, leading to a better search. Random selection simply selects a parent from the population at random without regard to its fitness value. Thus any individual can be selected with equal probability. Ranking selection is also found to be a very effective selection scheme. Ranking selection eliminates the problem associated with premature convergence because only relative fitness is used and not the absolute fitness. Additionally, because only relative fitness is required, the computation of fitness values can be simplified, or in applications in which a continuous fitness function may not be available, for instance classification problems. Another selection operator which is very effective is *stochastic universal sampling*, often used with multiple parents (more than 2). This is regarded as an optimal sampling scheme.
- 4. Mutation Operator:** The choice of mutation operator, mutation probability P_m and mutation intensity κ all have a strong influence upon the performance of any GA. Whilst the concept of mutation is identical in both binary and floating point representation, the actual implementations differ.

In binary representation, mutation simply swaps bits from 1 to 0 or 0 to 1 with finite probability P_m . This probability is generally very low eg: $P_m=0.01$. When dealing with floating point representation, we have many more variations by which mutation may be implemented. The most obvious is to add a uniformly distributed random number thus: $x_j=x_j+\kappa \times rand$, where κ is the mutation gain or intensity. A second method would be to use: $x_j=x_j \times (1+\kappa \times rand)$ where $rand$ has a gaussian distribution. The first method has the potential to search a wider space, the second a narrower search space. Furthermore the mutation intensity κ may be chosen to be a function of time $\kappa(t)$, and generally decrements as the algorithm converges closer to the solution. In many instances, κ may be manually controlled.

In general, randomly switching between both schemes resulted in rapid convergence. As a further observation, the probability of mutation P_m should be higher when using floating point representation than if using binary representation. For example if a number x is represented as a 20 bit binary string, the probability of mutating this binary string number is $20 \times P_m$.

If floating point representation is used for x , then we should use $P_m=20 \times 0.01=0.2$. This is why high mutation probabilities are used in most simulations between P_m : 0.1-0.4. Note that a high mutation probability P_m helps to retain genetic diversity and prevent premature convergence. On the other hand, a high mutation rate can also slow down convergence considerably. Again, the choice of mutation probability and intensity sometimes requires much trial-and-error work. Some authors have used self adaptive genetic algorithms with fuzzy search control, see [5]. and below section 7.2 for details. As a rule of thumb, the mutation probability should be chosen to be the inverse of the dimensions of the search space. The algorithm for mutation is:

```

for j=1 to n
  if (rand <  $P_m$ )
     $x_j = x_j + \kappa \times rand$  (or)
     $x_j = x_j \times (1 + \kappa \times rand)$ 
  end
end

```

where n =length of chromosome, and $rand$ =random number generator with uniform or gaussian probability distribution.

5. **Crossover Operator:** The choice of crossover operator is less critical. If using binary string representation, a two point crossover is generally a more effective search method than single point crossover.

When using floating point representation, a uniform weighted average crossover was found to give better convergence results than any other method tested. The crossover operator for floating point (or integer) representation is: $z_j = \alpha x_j + (1-\alpha)y_j$, where x_j and y_j are the two parents, z_j is the offspring and α is a uniformly distributed random number $[0,1]$, and j is the j^{th} components of the chromosome. With $\alpha=0.5$ fixed, averaging crossover can also yield good results. However simple swapping crossover (as used in binary strings) generally leads to poor results. Probability of crossover P_c is generally higher than mutation, values between 0.6-0.9 are recommended. The addition of the *Hooke-Jeeves* pattern search crossover (refer to chapter 1.4.2) was also found to improve the rate of convergence. The algorithm for the uniform weighted average crossover operator is:

```

for j=1 to n
  if (rand < Pc)
     $z_j = \alpha x_j + (1-\alpha)y_j$ 
  end
end

```

where n =length of chromosome, and rand =random number generator with uniform probability distribution $[0,1]$. The *Hooke-Jeeves* crossover operator is:

```

if (fitnessA > fitnessB) then
  offspring =  $2.x_A - x_B$ 
else
  offspring =  $2.x_B - x_A$ 

```

When dealing with complex numbers, the crossover operator should be implemented as if the real and imaginary parts are two separate numbers.

- 6. Population Inversion:** Generating a new population from the old population is commonly referred to as population inversion. Given a population of n parents, the GA generally produces a second population of n offspring at each generation step. There are generally 3 accepted methods of population inversion: (i) The most common method is to completely replace the old population (parents) with the new population (offspring). (ii) Another variation would be to replace a subpopulation of m (parent) individuals with m fittest offspring. (iii) Or combine the new and old population into one, and then select the n most fit individuals.

From simulations, the third method was found to be very effective in most applications. However one point to consider: the first method generally results in slower convergence, but retains greater genetic diversity. The third method converges quicker, losing potential genetic information early, some trial and error may be necessary in the choice of population inversion operator. Whichever method is used however, the concept of *Elitism* was found to be indispensable in any population inversion scheme.

7. **Computational Effort:** The calculation of fitness values is by far the most computationally intensive part of any GA, this is particularly true in training RBF networks (chapter 2), in MRAC (chapter 4) control, and FDI (chapter 6). The high computational burden stems from the parallel nature of genetic algorithms. For example, when training a RBF network using 200 training samples, a population of 50 individuals and 10 hidden nodes, this results in $200 \times 50 \times 10 = 100,000$ computations for each hidden node at each generation step (note that a single node computation may require many FLOPS). Furthermore, because the RBF weights are computed by least squares, this means that in addition, it also requires 50 matrix inversions at each generation. In such instances, vectoring the algorithm or using multiple processors (e.g. systolic array) would be preferable as the GA is easily adapted for parallel processing hardware. As a consequence, the GA must be made to converge as efficiently as possible in the fewest possible generations. To overcome this problem, hybrid GA methods were adopted.

Increasing population size reduces the number of generations required for convergence, however not necessarily as a linear function. This means that a tradeoff between population size and number of generations to converge is generally needed on a single processor machine. If running on a parallel machine with multiple processors for instance, then the population size may be increased to the full number of available processors.

8. **Robustness Qualities:** Genetic algorithms show good robustness properties to noisy data. This robustness quality can be attributed to the large amount of measured data generally required to be processed by the GA in order to compute fitness. For instance when training a RBF network for modeling or control, the quantity of training data and the fact that all data carries equal weight reduces the effect of noise present. Noise is assumed to be uncorrelated with a zero mean.

9. Population Initialization: Initial search is essential for rapid convergence. The population should always be initialized within and as close to the solution space as possible. Choose a narrower search space if the solution is known to lie within a particular region. If the GA initially encounters a local minima, the entire population may soon lie within this local minima. Whilst the GA is capable of emerging from the local minima, this is generally not very efficient. If however, the initial search is prevented from converging too quickly, it will have greater opportunity to find a global minimum and subsequently converging towards it.

10 Constraint Handling: Genetic algorithms can handle constrained related optimization problems directly without the need to restructure the problem. Repair algorithms are found to work well in all simulations. The static penalty function has been also found to work well, the difficulty arises in the choice of a value of the penalty coefficient. When light penalties are used, they fail to accurately enforce the constraint. However when heavy penalties are used, that portion of the population which violates the constraints will quickly vanish. This reduces the search space and can lead to an excessive number of unfeasible solutions..

In summary, genetic algorithms cannot be simply applied blindly to any problem. Each GA implementation must be carefully considered taking into account any problem specific information and some trial-and-error work. As a last point at hand, we note that according to the no-free-lunch theorem (NFL) by Wolpert and Macready [1], states that without problem and domain specific information, there is no way to justify claims that one search algorithm is better than all others. This implies that the proper choice of search algorithm is strongly dependent upon the problem to be solved.

(ii) Applications to Identification and Control:

Some key results obtained in the implementation of control systems with genetic algorithms and radial basis function networks is briefly summarized below:

1. RBF Applications: From the results obtained in chapter 2, training using genetic algorithms can produce a RBF network with superior performance compared to conventional training. However training times are excessive. To reduce training time, we require some variation of the GA tailored specifically for RBF networks. The hybrid combination of conventional genetic algorithms and greedy local search was found to improve convergence.

2. Eigenstructure Assignment: The generalized robust eigenstructure assignment problem formulated in chapter 3 can be directly solved with hybrid genetic algorithms, generally with fast convergence results. If using conventional gradient based optimization, the presence of constraints results in a problem formulation requiring *Lagrange multiplier* methods. The Lagrange multiplier formulation (chapter 1) produces an excessive number of equations (five matrix differential equations), requiring gradient calculations for each, including having to solve for the lagrange multiplier matrix. Furthermore, convergence is local only. Genetic algorithms can deal with this problem directly, using penalty functions to handle the constraints.

Eigenstructure assignment has also been extended to the problem of reconfigurable control [7]. In this instance, the feedback controller K is modified such that the closed loop eigenstructure remains unchanged under the influence of plant changes: ΔA and ΔB . This can be a subject of further research with the application of genetic algorithms.

3. MRAC Control: From chapter-4, we can see that hybrid genetic algorithms can easily be applied to adaptive control applications, and convergence is generally very rapid. For simple SISO systems, convergence results within 50 generations. However, despite the rapid convergence, the computational effort is excessive. The use of hybrid genetic algorithms helps to alleviate this problem. However, there is no guarantee that the genetic algorithm will converge at all. This is a critical issue if genetic algorithms are to be accepted as an alternative in MRAC control applications. On the other hand however, genetic algorithms have fewer restrictions and can also be applied to nonlinear systems. Note also that the GA is not really recursive, whereas both the Lyapunov method and MIT rule generate new parameters at each sample interval with only the current measurement, the genetic algorithm requires knowledge of past historical data as well as current data. This means that its response is delayed if an abrupt change occurs in the plant A and B matrices. Whilst the genetic algorithm may not be recursive, it can however still operate online. From the results, we can see that genetic algorithms work well, and have fewer restrictions when compared with more traditional methods such as the MIT gradient based rule and Lyapunov stability theory. The GA can easily be extended to more unconventional controller configurations without any change to the genetic algorithm design. Genetic algorithms are easily extended to solving nonlinear MRAC systems, utilizing any controller structure e.g.: neural networks, fuzzy logic, linear dynamic compensators etc. The GA offers many new and novel possibilities for implementing robust adaptive control systems, in particular areas of intelligent control systems.

4. Mixed H_2/H_∞ Control: Results from chapter-5 show that genetic algorithms can be successfully applied to the design of full order, reduced order H_2 , H_∞ , and mixed H_2/H_∞ compensators. Results agree well with those obtained using conventional state space solutions, and conventional model reduction techniques. In most cases, the GA converged within 400 generations. Genetic algorithms are conceptually elegant, simple and applicable to a wide range of robust control and multiobjective constrained optimization problems. In this applications, solution to the H_2 or H_∞ problem required only a single objective constrained optimization. The solution to the mixed H_2/H_∞ is a multiobjective constrained optimization problem which leads to a family of solution. By proper selection of the scalar weight κ , more or less emphasis can be placed on the optimization of either the H_2 or H_∞ specifications.

5. Fault Detection and Isolation: Fault detection and isolation (FDI) can be viewed as a system identification problem. From the survey in chapter-6, there are a multitude of methods for fault diagnosis as evidenced from the survey. The popular methods of parity space are well established and offer a simple and reliable methodology of detecting faults for linear systems. However parity space methods suffer serious limitations such as inability to detect output or multiple faults. Simulation results indicate that genetic algorithms using a model based FDI system can be used to detect input, output, and internal plant faults with rapid convergence. Genetic algorithms do not suffer from restrictions prevailing most traditional FDI methods. Furthermore, nonlinear systems can be diagnosed as long as a model of the system is available. Because genetic algorithms deal with population of individuals, a large variety of probable faults can be quickly analyzed and evaluated, whilst providing good immunity to sensor noise. Given the plethora of currently available methods [8], one would expect wide-ranging applications of genetic algorithms and heuristic search, in the field of fault detection and isolation. This would be an ideal topic for future PhD research.

In summary, we have seen that genetic and hybrid algorithms are a powerful tool for solving problems including constrained optimization and machine learning. We have shown that the GA can converge very rapidly in all applications. This feature, coupled with need for control systems to be more autonomous, reliable, and adaptive makes the genetic algorithm an ideal mechanism for evolving control systems.

7.2 Challenges and Future Development:

Some directions for possible future research are briefly outlined below. In particular, applications of evolutionary concepts to areas of conventional control and intelligent control. Although, the work in this thesis focused primarily on genetic algorithms, evolutionary computation encompasses a broader class of evolutionary theories as discussed in chapter 1. This provides a greater scope for many different potential applications to control engineering.

(i) Genetic Algorithms:

1. Mathematical framework: Evolutionary algorithms have demonstrated the ability to rapidly solve problems in which classical optimization methods fail or are inadequate. However there is a lack of strong mathematical framework, particularly in the areas of convergence rates and stability proofs.

2. Multiobjective optimization: This is a relatively new area in which evolutionary computation can play a significant role. Currently there are two methods of multiobjective optimization using genetic algorithms: Pareto dominance principle and Nash Equilibria [9]. A good review on multiobjective optimization (MOP) using genetic algorithms is found in [13].

Nash Equilibria is a relatively new concept of game theory in genetic algorithms, which has more robust and faster convergence properties. Nash equilibria which originated in 1951 [10], is inspired from Games Theory and economics, produces a single solution rather than a family of solutions. Also referred to as *Non-Cooperative approaches*. A good introduction to Nash equilibria is given by [11]. A similar strategy using *asynchronous* (less frequent) exchange of data exists, this is called the *Stackelberg Equilibria* [12] in which one player plays before the other, taking into account its reaction. All these techniques are referred to as *evolutionary game theory*, and offer new avenues of research in genetic algorithms [11]. Niching and coevolutionary theory also relate to multiobjective optimization concepts. This is currently an active area of research with potential applications in control theory.

3. Self Adaptation: Self adaptation is a relatively new concept in evolutionary strategies. Self adaption involves dynamically modifying the genetic operators such as crossover and mutation probabilities, and mutation intensity, in order to improve convergence.

Self adaptation of *strategy parameters* involves encoding the strategy parameters i.e. mutation and crossover probabilities, mutation and crossover intensities as part of the search space. In other words, the strategy parameters are included as part of the chromosome. Consequently, an individual consists of two parts: an object variable vector x and a strategy parameter vector s as part of its chromosomal structure. Subsequently, s may be defined by the 4-tuple: $s = \{P_m, P_c, \sigma_m, \sigma_c\}$. A typical chromosomal representation would be something like:



Fig.7.1

The simplest mechanism of self adaptation would be to first recombine and mutate the strategy parameters s yielding s' , and then using these updated parameters to recombine and mutate the solution vector x yielding x' . Thus rather than using constant strategy parameters (or modified by some deterministic rule), they are themselves modified by evolutionary means. The strategy parameters are continuously updated and the rate of convergence improved, as the simulation progresses. Note that the speed of adaptation of the strategy parameters is controlled by *learning-rates* generally under some manual control.

In theory, all four parameters of s can evolve, in practice this would be inefficient and only the mutation operator is generally considered. The mutation operator plays a significant role in the convergence of GA, for instance at the start of a GA search, the mutation intensity should be high for efficient search over a wide solution space. However when a global extremum has been found, the intensity should be reduced to enable narrow search within this region.

4. **Fuzzy Search Control:** This is a variation of the above method. Rather than including the strategy parameters s into the chromosome, an external fuzzy logic system is used. The fuzzy logic system monitors the progress of the genetic algorithm, and adapt the strategy parameters s according to the time-evolution of the fitness function. The method is illustrated below, the advantage of such method is the partitioning of the two entities: the chromosomal representation of the search space x and the adaptation of the strategy parameters s .

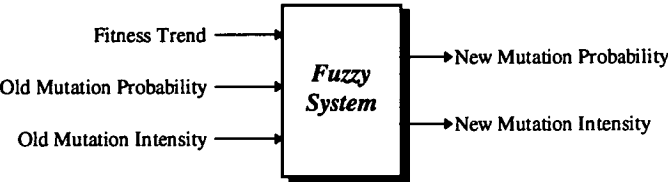


Fig.7.2

(ii) Applications to Identification and Control:

1. **Neural network and fuzzy logic systems:** Practical applications of evolutionary computation to the training of neural networks and fuzzy logic controllers [14] are necessary, in order to gain greater industry acceptance. We have seen that genetic algorithms can produce good results with RBF networks. However training times are in general excessive. To reduce training time, we require some variation of the GA tailored specifically to train RBF networks, hybrid GA methods help to ameliorate this problem. The application of hybrid GA methods can also be extended to training many different types of neural networks, however this is still an active area of research.
2. **Eigenstructure assignment:** Eigenstructure assignment has also been extended to the problem of reconfigurable control [7]. In this instance, the feedback controller K is modified such that the closed loop eigenstructure remains unchanged under the influence of plant changes: ΔA and ΔB . This is a subject of further research with applications to genetic algorithms.
3. **MRAC and Adaptive Control:** In chapter 4, we found genetic algorithms to work well when applied to MRAC control applications. Whilst convergence was generally very rapid, it required high computational effort. With hybrid GA, the computational effort was significantly reduced and comparable to conventional MRAC schemes. Genetic algorithms are easily extended to solving nonlinear MRAC systems, with any controller structure e.g.: neural networks, fuzzy logic, linear dynamic compensators etc. Further areas of research would include:
 - Modify the GA to be a recursive algorithm, rather than searching the entire solution space, use the previous results to generate a population with a narrower search range which would aid in convergence.
 - Apply GA to indirect method of MRAC. Only the direct method was used in the simulations, with output feedback instead of full state feedback.
 - Applications of GA to nonlinear systems with robustness properties using variable structure model reference adaptive control. Variable structure MRAC is currently an active area of research.

4. H_2/H_∞ Control: From chapter-5, some possibilities for future research include:

- Replace a dynamic H_2 or H_∞ compensator with a RBF network trained with genetic algorithms, the figure below illustrates a typical setup for a H_2 optimal controller:

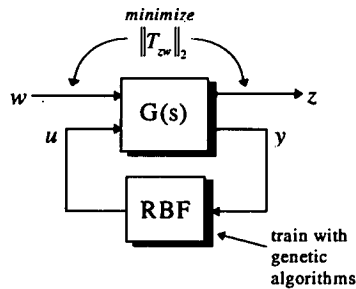


Fig.7.3
Using a RBF implementation of H_2 and H_∞ compensators

- Use homotopy theory for training RBF networks, compare with genetic algorithms.
- For the mixed H_2/H_∞ simulation, use linear matrix inequalities and convex optimization comparing solutions with genetic algorithms.

5. Fault Detection and isolation: From this brief survey, the observer based methods offer interesting applications for research in fault detection and isolation. Several methods which have yet to be investigated include variable structure/sliding observers and nonlinear (lie algebraic) methods. The advantages of using these techniques in control are well established, including robustness to uncertainties, and invariance to disturbances. These advantages can be carried over to the design of robust observer applications. From the literature (chapter 6) on FDI, we can see that there are many techniques and variations, in which GA can be applied.

As systems become more complex, the application of conventional controllers may become inadequate. The application of artificial intelligence, neural networks, expert systems and evolutionary theory to produce better and more robust intelligent control systems is inevitable. Genetic algorithms offer alternatives to solving control system problems dealing with intelligent control. Intelligent control systems have the ability to adapt to changes in both plant and environment. Another feature of intelligent control is the ability to diagnose and/or predict potential faults from the behavior of the system, and if possible automatically reconfigure the control laws. This produces a high level of autonomy and self reliance. Evolutionary algorithms offer a feasible alternative by which such systems may attain practical implementation.

7.3 References and Further Reading:

- [1] D.H.Wolpert, W.G.Macready
No Free Lunch Theorems for Search
Technical Report SFI-TR-95-02-010, Santa Fe Institute
- [2] J.H.Holland,
Adaption in Natural and Artificial Systems.
University of Michigan Press, Ann Arbor, 1975.
- [3] W. Pedrycz
Fuzzy Evolutionary Computation
Kluwer Academic Publishers, 1997
- [4] J. Yen, R.Langari, L.A.Zadeh
Industrial Applications of Fuzzy Logic and Intelligent Systems.
IEEE Press, 1995
- [5] T. Back, D.B.Fogel, Z. Michalewicz
Handbook of Evolutionary Computation
Institute of Physics Publishing and Oxford University Press, , 1997 (Book)
- [6] C.Z.Janikow, Z. Michalewicz,
An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms
Proceedings fo the 4th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, California. pp.31-36, 1991.
- [7] J. Jiang
Design of Reconfigurable Control Systems Using Eigenstructure Assignments
International Journal of Control Vol.59, No.2, pp.395-410, 1994
- [8] J. Chen, R.J. Patton,
Robust Model Based Fault Diagnosis for Dynamic Systems
Kluwer Academic Publishers, 1999
- [9] D.Quagliarella, J.Periaux, C.Poloni, G.Winter
Genetic Algorithms and Evolution Strategies in Engineering and Computer Science
Recent Advances and Industrial Applications, John Wiley and Sons, 1998 (Book)
- [10] J. Nash
Non-Cooperative Games
Annals of Mathematics, Vol.54, pp.286-295, 1951
- [11] Robert Gibbons
A Primer in Game Theory
Harvester Wheatsheaf, 1992 (Book)
- [12] Joao Pedro Pedroso
Numerical Solution of Nash and Stackelberg Equilibria: and Evolutionary Approach
- [13] H. Tamaki, H. Kita, S. Kobayashi
Multi-Objective Optimization by Genetic Algorithms: A Review
Proceedings of the 1996 IEEE International Conference on Evolutionary Computation pp.517-522, 1996
- [14] M. Russo
Genetic Fuzzy Learning
IEEE Transactions on Evolutionary Computation, Vol.4, No.3, pp.259-273 September 2000
- [15] S.N. Singh,
Asymptotically Decoupled Discontinuous Control of Systems and Nonlinear Aircraft Maneuver.
IEEE Transactions on Aerospace and Electronic Systems, Jan 1989, Vo. 25, No. 3, pp.380-390



Appendix:

Contents:

8.1 Aircraft Mathematical Model	p.8.2
8.2 Partial Eigenstructure Assignment	p.8.6
8.3 Bioreactor Mathematical Model	p.8.8
8.4 Hooke-Jeeves Search Flowchart.. . . .	p.8.10

8.1 Aircraft Mathematical Model:

(i) Full Nonlinear Model:

In order to define the mathematical model, we first need to define the coordinate system and notation used in the model. In a three dimensional Cartesian coordinate system, for a rigid object in motion, there are three position (x,y,z) components, three linear velocity components (u,v,w) , three angular rates (p,q,r) and three rotational orientation components (Φ,θ,Ψ) also known as *Euler angles*.

There are thus a total of 12 state variables to fully describe an aircraft in motion (in a uniform atmosphere). In many cases, these can be greatly simplified for steady state level equilibrium (or trim) flight. There are several systems of coordinates: *principal or body axes* (aligned with aircraft body axes), *stability axes* (aligned with wind velocity vector), fixed *earth reference axes*. Figures 8.1, 8.2, 8.3, 8.4 below illustrate the notation used for a body axis system.

The model used is for a swept-wing fighter high performance aircraft taken from reference [15] (see chapter 7): described by a nonlinear system in control affine form:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} (L_{p\alpha}(\alpha-\alpha_o)+L_{p\beta}\beta+L_{pq}q+L_{pr}r+L_{pp}p+L_{p\alpha}(\alpha-\alpha_o).r-I_1.q.r \\ \tilde{M}_{\alpha}(\alpha-\alpha_o)+\tilde{M}_q.q+I_2.p.r-M_{\alpha}p\beta+M_{\alpha}(g/V).(\cos\theta\cos\phi-\cos\theta_o) \\ N_{\beta}\beta+N_r.r+N_p.p+N_{p\alpha}p(\alpha-\alpha_o)-I_3.p.q+N_q.q \\ q-p\beta+Z_{\alpha}(\alpha-\alpha_o)+(g/V).(\cos\theta\cos\phi-\cos\theta_o) \\ Y_{\beta}\beta+p(\sin\alpha_o+\alpha-\alpha_o)-r.\cos\alpha_o+(g/V).\cos\theta\sin\phi \\ p+q.\tan\theta\sin\phi+r.\tan\theta\cos\phi \\ q.\cos\phi-r.\sin\phi \\ q.\sin\phi.\sec\theta+r.\cos\phi.\sec\theta \end{bmatrix} + \begin{bmatrix} L_{\delta_a}+L_{\delta_{\alpha}}(\alpha-\alpha_o) & L_{\delta_r} & 0 \\ 0 & 0 & \tilde{M}_{\delta_r} \\ N_{\delta_a}+N_{\delta_{\alpha}}(\alpha-\alpha_o) & \tilde{N}_{\delta_r} & 0 \\ 0 & 0 & Z_{\delta_r} \\ Y_{\delta_a} & Y_{\delta_r} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_a \\ \delta_r \\ \delta_e \end{bmatrix}$$

Where: Position, velocity, and orientation components are:

- u, v, w - Linear velocity components along body axes m/sec.
- $\dot{u}, \dot{v}, \dot{w}$ - Linear acceleration components along body axes. m/sec²
- p, q, r - Angular velocity components for each body axes, rad/sec.
- $\dot{p}, \dot{q}, \dot{r}$ - Angular acceleration components for each body axes, rad/sec²
- ϕ, θ, ψ - Euler angles (rotational orientation) in radians.
- α, β - Angle of attack and sideslip angles, typically measured by air data probes.

Control Surfaces:

- δ_e : - Elevator (horizontal tail) command in degrees, used mainly for pitch control.
- δ_a - Aileron commands in degrees, used mainly for roll control.
- δ_r : - Rudder (vertical tail) command in degrees, used mainly for yaw control.

Inertial Coefficients: $I_1=0.7270$; $I_2=0.9490$; $I_3=0.7160$ along each of the body axes.
note below that: $\alpha_0=1.5$ degrees is constant.

$$\left. \begin{aligned} \tilde{M}_\alpha &= M_\alpha + M_{\dot{\alpha}} \cdot Z_\alpha \\ \tilde{M}_q &= M_q + M_{\dot{\alpha}} \\ \tilde{M}_{\delta e} &= M_{\delta e} + M_{\dot{\alpha}} \cdot Z_{\delta e} \\ \tilde{L}_{\delta a} &= L_{\delta a} + L_{a\delta a} \cdot (\alpha - \alpha_o) \\ \tilde{N}_{\delta a} &= N_{\delta a} + N_{a\delta a} \cdot (\alpha - \alpha_o) \end{aligned} \right\} \quad \text{Eqn.8.2}$$

Data for two different flight conditions is supplied:

Flight Condition I:				Flight Condition II							
Y_β	=	-0.196;	M_α	=	-23.180;	Y_β	=	-0.280;	M_α	=	-10.700;
Z_α	=	-1.329;	$M_{\dot{\alpha}}$	=	-0.173;	Z_α	=	-1.746;	$M_{\dot{\alpha}}$	=	-0.251;
L_β	=	-9.990;	M_q	=	-0.814;	L_β	=	-20.910;	M_q	=	-1.168;
L_p	=	-3.933;	$M_{\delta e}$	=	-28.370;	L_p	=	-5.786;	$M_{\delta e}$	=	-31.640;
L_q	=	0.107;	N_β	=	5.670;	L_q	=	0.108;	N_β	=	8.880;
L_r	=	0.126;	N_p	=	0.002;	L_r	=	0.221;	N_p	=	0.013;
$L_{r\alpha}$	=	8.390;	$N_{p\alpha}$	=	-1.578;	$L_{r\alpha}$	=	13.160;	$N_{p\alpha}$	=	-1.583;
$L_{\beta\alpha}$	=	-684.400;	N_r	=	-0.235;	$L_{\beta\alpha}$	=	-543.800;	N_r	=	-0.377;
$L_{\delta a}$	=	-45.830;	$N_{\delta a}$	=	-0.921;	$L_{\delta a}$	=	-60.270;	$N_{\delta a}$	=	-1.282;
$L_{a\delta a}$	=	63.500;	$N_{a\delta a}$	=	1.132;	$L_{a\delta a}$	=	64.600;	$N_{a\delta a}$	=	2.459;
$L_{\delta r}$	=	-7.640;	$N_{\delta r}$	=	-6.510;	$L_{\delta r}$	=	-10.050;	$N_{\delta r}$	=	-8.300;
$Y_{\delta a}$	=	0.0071;	$Z_{\delta e}$	=	-0.168;	$Y_{\delta a}$	=	0.0119;	$Z_{\delta e}$	=	-0.224;
g/V	=	0.0345;	N_q	=	0.223;	g/V	=	0.0412;	N_q	=	0.223;
$Y_{\delta r}$	=	0.000;				$Y_{\delta r}$	=	0.000;			

Note that since all angular components are in radians, the control input (rudder, elevator, ailerons) must also be converted to radians before input into the aircraft equation. Equation 8.1 on the previous page can be written in traditional control affine form, for nonlinear control design.

$$\dot{x}(t) = A(x) + B(x).u(t) \quad \text{Eqn.8.3}$$

(ii) Linearized Longitudinal Dynamics:

The linearized longitudinal dynamics is given by:

$$\dot{x} = A.x + B.u \quad \text{Eqn.8.4}$$

where: $x = [q \ \alpha \ \theta]^T$ and $u = \delta_e$, All other state variables in the full model are set to zero.

Linearizing the model about $x_0 = [0, 1.5, 0]^T$ gives A and B matrices:

$$A = \begin{bmatrix} -0.9870 & -22.9501 & 0 \\ 1.0000 & -1.3290 & 0 \\ 1.0000 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -28.3409 \\ -0.1680 \\ 0 \end{bmatrix}$$

(ii) Linearized Lateral Dynamics:

The linearized lateral dynamics is given by:

$$\dot{x} = A.x + B.u \quad \text{Eqn.8.5}$$

where: $x = [p \ r \ \beta \ \phi]^T$ and $u = [\delta_a^c \ \delta_r^c]^T$, All other state variables in the full model are set to zero. Linearizing the model about $x_0 = [0, 1.5, 0]^T$ gives A and B matrices:

$$A = \begin{bmatrix} -3.9330 & 0.1260 & -9.9900 & 0 \\ 0.0020 & -0.2350 & 5.6700 & 0 \\ 0.0262 & -0.9997 & -0.196 & 0.0345 \\ 1.0000 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -45.8300 & -7.6400 \\ -0.9210 & -6.5100 \\ 0.0071 & 0 \\ 0 & 0 \end{bmatrix}$$

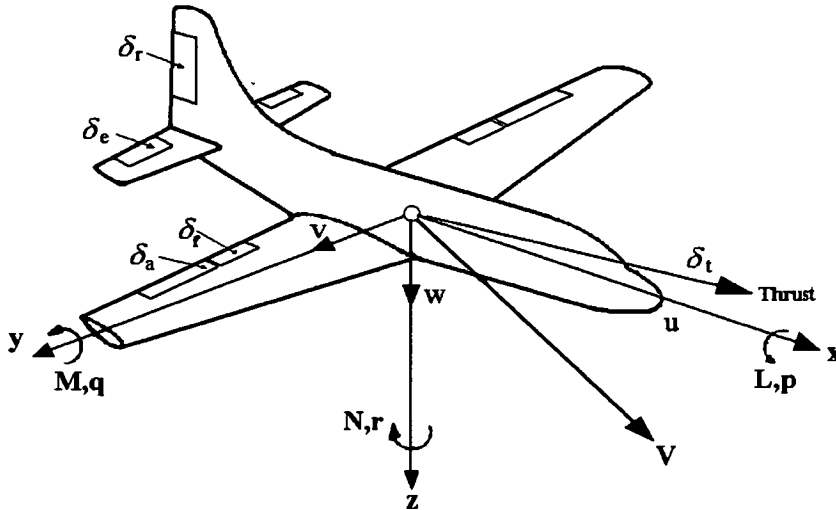


Fig.8.1
Aircraft Axes and Symbols

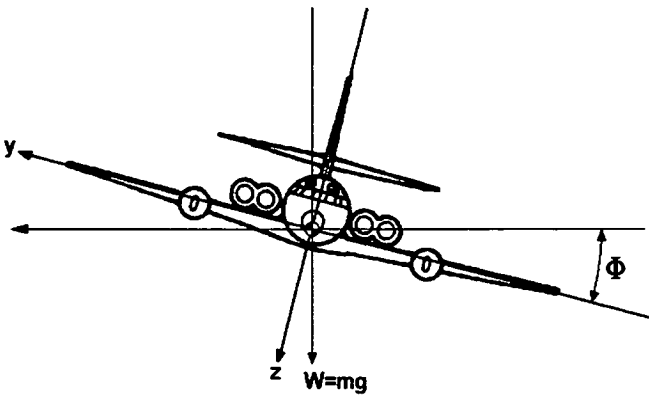


Fig.8.2
Aircraft roll component

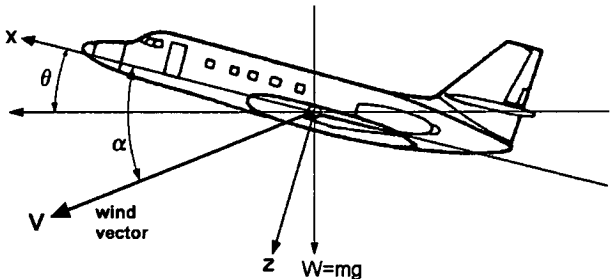


Fig.8.3
Longitudinal Dynamics
Aircraft pitch θ and angle of attack α definitions

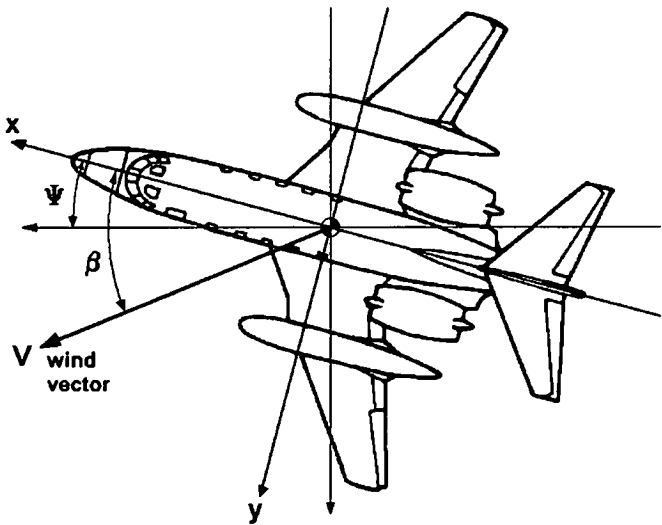


Fig.8.4
Lateral Dynamics
Aircraft yaw ψ and sideslip angle β definitions

8.2 Partial Eigenstructure Assignment:

This method was used as a comparison against results obtained with genetic algorithms in chapter 3. This is the conventional partial eigenstructure assignment algorithm. Its description is outlined below by way of an example instead of giving a general formulation. The description is for a single eigenvalue λ and eigenvector \mathbf{v} component. The procedure must be repeated for all of the eigenvalues/eigenvectors. From chapter 3, we have seen that all achievable closed loop eigenvectors \mathbf{v}^a must belong to the subspace spanned by the columns of $S = (\lambda.I - A)^{-1}.B$, in other words the vector \mathbf{v}^a must correspond to the subspace: $\mathbf{v}^a = S.\mathbf{g}$ where \mathbf{g} is a vector to be solved for, the minimization now becomes:

$$\text{minimize } \|\mathbf{v} - S.\mathbf{g}\|_2^2 \quad \text{Eqn.8.6}$$

Matrix dimensions used in simulations for chapter 3 are: $\mathbf{v} \in \mathbb{R}^4$ is a column vector, $S \in \mathbb{R}^{4 \times 2}$ is a matrix, and $\mathbf{g} \in \mathbb{R}^2$ is a column vector. The minimization of the norm can be achieved by least squares solution of the \mathbf{g} vector. Assuming that $\{\lambda, \mathbf{v}\}$ forms part of a complex conjugate pair, implies that \mathbf{g} vector has also a complex conjugate. If we define all $\mathbf{v}, S, \mathbf{g}$ in more detail we get the following expression, for the roll mode eigenvalue/eigenvector specification (see chapter 3):

$$\mathbf{v} = \begin{bmatrix} X + j1 \\ 0 + j0 \\ 0 + j0 \\ 1 + jX \end{bmatrix} \quad S = \begin{bmatrix} a_{11} + jb_{11} & a_{12} + jb_{12} \\ a_{21} + jb_{21} & a_{22} + jb_{22} \\ a_{31} + jb_{31} & a_{32} + jb_{32} \\ a_{41} + jb_{41} & a_{42} + jb_{42} \end{bmatrix} \quad \mathbf{g} = \begin{bmatrix} x_1 + jy_1 \\ x_2 + jy_2 \end{bmatrix} \quad \text{Eqn.8.7}$$

where X =don't care state in the eigenvector, all other specified entries in the eigenvector must be assigned as closely as possible. The problem becomes that of minimizing:

$$\min \left\| \begin{bmatrix} X + j1 \\ 0 + j0 \\ 0 + j0 \\ 1 + jX \end{bmatrix} - \begin{bmatrix} a_{11} + jb_{11} & a_{12} + jb_{12} \\ a_{21} + jb_{21} & a_{22} + jb_{22} \\ a_{31} + jb_{31} & a_{32} + jb_{32} \\ a_{41} + jb_{41} & a_{42} + jb_{42} \end{bmatrix} \begin{bmatrix} x_1 + jy_1 \\ x_2 + jy_2 \end{bmatrix} \right\|_2^2 \quad \text{Eqn.8.8}$$

Remove the norm symbol only for notational convenience, and separating the real/imaginary parts gives:

$$\left(\begin{bmatrix} X \\ 0 \\ 0 \\ 1 \end{bmatrix} + j \begin{bmatrix} 1 \\ 0 \\ 0 \\ X \end{bmatrix} \right) - \left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{41} & a_{42} \end{bmatrix} + j \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix} \right) \cdot \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + j \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \right) \quad \text{Eqn.8.9}$$

Note there are 4 unknowns: x_1, x_2, y_1, y_2 to solve for.

Expanding equation 8.9 gives:

$$\left(\begin{bmatrix} X \\ 0 \\ 0 \\ 1 \end{bmatrix} + j \begin{bmatrix} 1 \\ 0 \\ 0 \\ X \end{bmatrix} \right) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{31} & a_{42} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{31} & b_{42} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + j \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{31} & a_{42} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + j \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{31} & b_{42} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{Eqn.8.10}$$

Matching real parts and imaginary parts gives two expressions 8.11a and 8.11b respectively:

$$\begin{bmatrix} X \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{31} & a_{42} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{31} & b_{42} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \text{Eqn.8.11a}$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ X \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{31} & a_{42} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{31} & b_{42} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{Eqn.8.11b}$$

Writing 8.11a and 8.11b in matrix form gives two sets of equations:

$$\left. \begin{aligned} u &= A.x - B.y \\ v &= B.x + A.y \end{aligned} \right\} \quad \text{Eqn.8.12}$$

Where $A = \text{real}(S)$, $B = \text{imag}(S)$, and the matrix $S = (\lambda.I - A)^{-1} \cdot B$. The x and y vectors are as in Eqn.8.11. Combine into one expression to give: (not to be confused with A,B matrices of the LTI state variable system):

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} A & -B \\ B & A \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{Eqn.8.13}$$

All we need to do is to remove the X don't care rows from the above expression and then solve for the x and y vectors by conventional least squares:

$$\begin{array}{c|c|c|c|c} \begin{bmatrix} X \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ X \end{bmatrix} & = & \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{31} & a_{42} \end{bmatrix} & - & \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{31} & b_{42} \end{bmatrix} & \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \end{array} \quad \begin{array}{l} \leftarrow \text{Remove} \\ \text{these rows} \end{array} \quad \text{Eqn.8.14}$$

After the first and last rows have been removed, write the above equation 8.12 as a simple linear matrix function: $w = T \cdot \bar{g}$, where w , T and \bar{g} are the respective components of equation 8.12.

Solving by least squares gives:

$$\bar{g} = (T^T \cdot T)^{-1} \cdot T^T \cdot w \quad \text{Eqn.8.15}$$

From the solution of: $\bar{g} = [x_1 \ x_2 \ y_1 \ y_2]^T$, the g vector can be formed: $g = \begin{bmatrix} x_1 + jy_1 \\ x_2 + jy_2 \end{bmatrix}$,

achievable eigenvectors can then be computed $S \cdot g$ and the minimization equation 8.6 solved.

8.3 Bioreactor Mathematical Model:

The bioreactor represents another physical system which was used for simulations studies. A brief introduction is given below. The bioreactor has been chosen for simulations due to its wide use in industry and portrays a simpler nonlinear system (lower order) compared to the aircraft model. The mathematical model we used is taken from chapter-2 references [5, 6, 7, 8]. The bioreactor consists of a tank containing water, nutrients (or substrate) and biomass (or cells). Nutrients and biomass are added to the tank (via the inlet), the nutrients are consumed by the biomass thereby increasing the overall biomass concentration in the tank. Furthermore, biomass is removed from the tank via an outlet, at the same flow rate as the inlet. The overall volume of the liquid in the tank is made to remain constant. This is illustrated schematically below Fig.8.5:

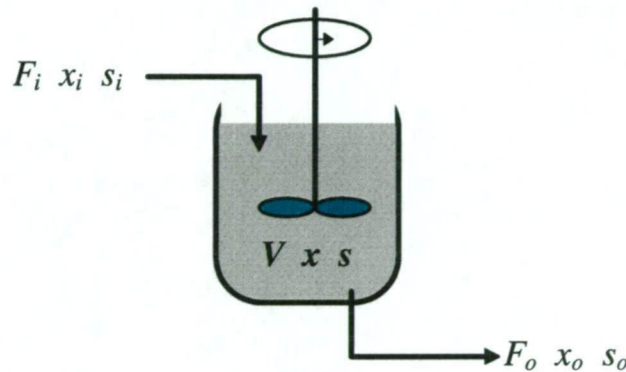


Fig.8.5
Schematic Diagram of a Bioreactor

Figure 8.5 illustrates the basic elements of a bioreactor tank. A complete derivation is omitted, however the reader is referred to chapter 2 references on bioreactors. The following state equations describe the dynamics of a bioreactor as a set of second order nonlinear differential equations:

Bioreactor Dynamics:

$$\begin{aligned}\frac{ds}{dt} &= -\kappa_1 \cdot \left(\mu_m \frac{s}{\kappa_s + s} \right) \cdot x + \frac{F_i}{V} \cdot (s_i - s) \\ \frac{dx}{dt} &= \left(\mu_m \cdot \frac{s}{\kappa_s + s} - \frac{F_i}{V} \right) \cdot x\end{aligned}\quad \text{Eqn 1.16}$$

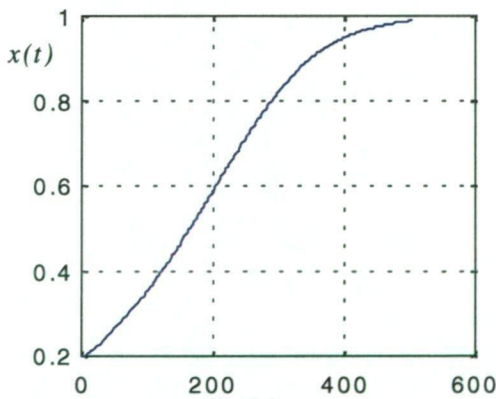
Where:

- x_i : Input biomass concentration=0.
- s_i : Input nutrient concentration.
- F_i : Input flowrate (constant)
- x : Biomass concentration inside tank. = output biomass
- s : Nutrient concentration inside tank. = output concentration

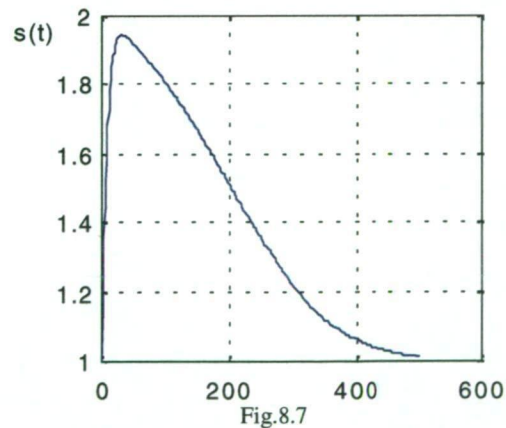
In continuous operation, the bioreactor runs at some steady state operating point, we assume that the flow rates are constant and identical ie: $F_i = F_o$, thus the volume of liquid inside the tank remains constant. We also assume that the output biomass and nutrient are the same as the biomass and nutrient ie: $x_o = x$, $s_o = s$ inside the tank, we also assume the input has no biomass $x_i = 0$. Typical values for the saturation constant and growth rate coefficients are: $\mu_m = 0.3$ and $\kappa_s = 0.1$ to 0.4 , $\kappa_1 = 1.25$, the initial conditions: $s(0) = 1.0$, $x(0) = 0.2$. The equations can be represented in non-linear state variable form shown below. Let $x_1 = x$, $x_2 = s$, $u = s_i$, then together with the above assumptions we can write in more traditional form:

$$\left. \begin{aligned} \dot{x}_1 &= \mu_m \cdot \left(\frac{x_2}{\kappa_s + x_2} - \frac{F_i}{V} \right) \cdot x_1 \\ \dot{x}_2 &= -\kappa_1 \cdot \left(\mu_m \frac{x_2}{\kappa_s + x_2} \right) \cdot x_1 + \frac{F_i}{V} \cdot (u - x_2) \end{aligned} \right\} \quad \text{Eqn.1.17}$$

Figure Fig.8.6 and 8.7 shows a typical step response simulation of the bioreactor open loop dynamics to nutrient input.



Open Loop Step Response: $x(t)$: Biomass Output



Open Loop Step Response: $s(t)$: Nutrient Output

The response is relatively intuitive, when a step input (in nutrient) is applied to the tank, assuming perfect and instantaneous mixing, the nutrient in the tank and hence output nutrient is initially high, but the nutrient is gradually consumed by the biomass (fig. 8.7) thus reduce with time. At the same time, the biomass concentration increases as an exponential function i.e. fig.8.6 due to nutrient uptake.

8.4 Hooke-Jeeves Search Flowchart:

