Robust electronic circuit design using evolutionary and Taguchi methods

Prepared by : **Muhammud A. Owadally**
Postgraduate student
Department of Electrical and Electronic Engineering
University of Cape Town

Prepared for: **Assoc. Prof. JR Greene**
Department of Electrical and Electronic Engineering
University of Cape Town

30 September 1997

Thesis prepared in partial fulfilment of the requirements for the Degree of Msc in Electrical and Electronic Engineering

Acknowledgements

I would like to thank the following people :

Prof John Greene, for his enthusiasm and guidance throughout this whole thesis

My friends, for their support.

My parents, for their faith in me.

TERMS OF REFERENCE

This thesis was commissioned by Assoc. Prof. J.R. Greene of the electrical engineering department at the University of Cape-Town, in February 1996.

It was requested that a software algorithm based on the principle of robust electronic circuit design be developed by genetic search.

The specific instructions were :

1. Analyse electronic circuits using discrete component value selection obtained from a genetic search.
2. Develop a software program based on genetic search to find robust electronic circuits.
3. Submit the thesis as a report on 30 September 1997.

SYNOPSIS

In engineering, there is a wide range of applications where genetic optimizers are used. Two genetic optimizers used in this thesis namely, Population Based Incremental Learning (PBIL) and Cross generational selection Heterogeneous crossover Cataclysmic mutation (CHC), are tested on a series of circuit problems to find if robust electronic circuits can be built from evolutionary methods.

The evolutionary algorithms were used to search the space of discrete component values from a range of manufactured preferred values to obtain robust electronic circuits. Parasitic effects were also modelled in the simulation to provide for a more realistic circuit.

The two optimizers were modified into hybrid genetic algorithms to find the sensitivity of a particular set of component values given a certain tolerance during the local search stage. Methods used to find the sensitivity were: Full factorial perturbation of the component values, the Taguchi method, the Taguchi method with Lamarckian learning.

The full factorial search of the neighbourhood of a set of component values is computer intensive and as such, computing time was large. The Taguchi method predicts a tangent to the surface of the set of component values using a fractional factorial approach. This means that this method is less computer intensive. Finally the method combining Taguchi with Lamarckian learning was used to guide the evolutionary optimizers towards better designs during the search.

The method of evolving robust circuit using genetic optimizers by choosing preferred values is better than conventional circuit design where usually nearest preferred values are used in circuits.

The hybrid CHC algorithm worked faster than the hybrid PBIL algorithm when using the Taguchi method only during the local search stage. The hybrid PBIL algorithm outperforms the hybrid CHC algorithm when using both the Taguchi method and Lamarckian learning during the local search stage.

Table of Contents

List of Illustrations

Page

Figures

Tables

Chapter 1

Introduction

In many engineering design problems, the analysis of an existing design is simpler than its synthesis. Designers are often confronted with problems that conventional methods are unable to solve. Modern computational methods make use of stochastic search algorithms. This method of problem solving involves searching through the space of all possible solutions to obtain an optimal result for a particular problem. Population Based Incremental Learning (PBIL) and Cross generational selection Heterogeneous crossover Cataclysmic mutation (CHC) are two of the search methods used in function optimization design problems.

The aim of this thesis is to investigate whether evolutionary algorithms can be used to build robust electronic circuits.

The evolutionary algorithms were tested, as optimizers, on five different electronic circuits. The evolutionary algorithms were also used, in three other circuits, to search for discrete component values from a range of manufactured preferred values. A new robust hybrid genetic algorithm combining Taguchi techniques in local search stage with a traditional genetic algorithm was developed to cope with the tolerance of component values.

This thesis starts with a description of optimization in the engineering field. Then in Chapters 3 and 4, two evolutionary algorithms, namely Population Based Incremental Learning (PBIL) and Cross generational selection Heterogeneous crossover Cataclysmic mutation (CHC) are developed. Chapter

5 deals with different optimization techniques applied to some circuit problems. Examples of robust electronic designs using evolutionary methods are analysed in Chapter 6. The Taguchi methodology is then discussed in Chapter 7. A new robust hybrid genetic algorithm is developed in Chapter 8. In Chapter 9, the new algorithm is tested. Finally, conclusions are drawn in Chapter 10.

Chapter 2

Optimization methods in electronic circuit design

This chapter starts with an overview of optimization and then different optimization methods are discussed.

2.1 Overview of optimization

The application of optimization theory to engineering presents a new way to solve problems for which no formal design methods exist. Optimization is a computer-oriented technique which has been in use since the mid 1960s. Designers were not keen to use this method at that time as computers were scarce and slow in computation. However, we have now entered a new era where low-cost computers have reduced to a great extent the computational time in solving engineering problems.

Every engineer has to deal sooner or later with a design problem that has no solution derived from analytical procedures. Such problems could only be treated, before the mid 1960s, by approximation or over-design, hoping that the specification targets would still be met. However, the design problems can be solved by making use of the optimization theory.

When the constraints of a design problem are expressed as an optimization problem, the problem can then be solved by an optimizer. In other words, optimization provides a powerful and general design tool which complements and extends the capabilities of existing design techniques such as

design/synthesis techniques. Optimization is used extensively in, for example, printed circuit board layout and interconnection, integrated circuit design and layout amongst others.

The optimization strategy can be summarized as follows :

1. guess a solution;
2. analyse the solution;
3. compare results with requirements:
   - if satisfactory, stop;
   - if not, change the values of one or more variables;
4. repeat steps 2 and 3 until results meet requirements.

Figure 1 shows a block diagram representation of the optimization strategy.

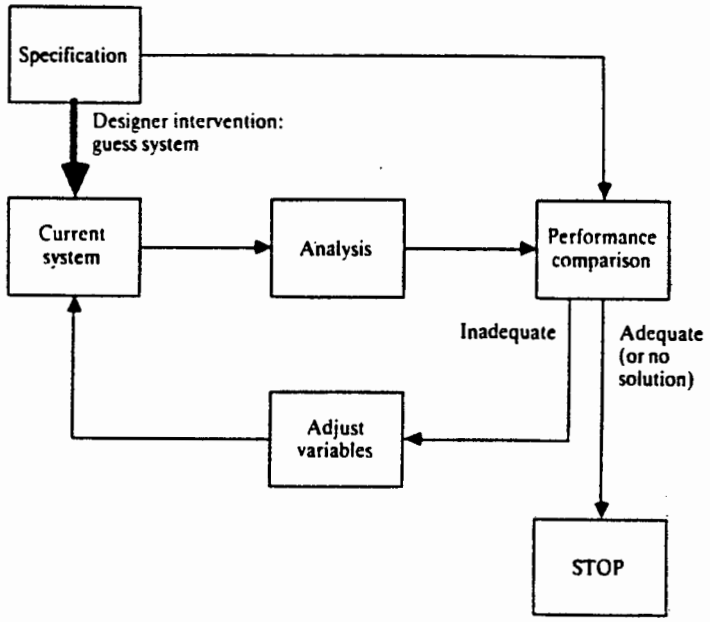


Figure 1: Optimization strategy

The optimization scheme begins with an initial guess solution, involving a fixed circuit configuration with an initial set of parameters. As shown in Figure 1, an analysis procedure then evaluates the circuit for any given set of parameters with respect to the specification of the problem.

The performance obtained from the analysis procedure is then compared to the required performance. If the specification is met, the design is satisfactory and the optimization process ends. However, if the actual performance does not meet the required specification, the design variables are altered and the process is then repeated until either a satisfactory solution is achieved or a solution that satisfies the requirements does not exist.

This iterative strategy provides *design-by-analysis* and shows that the analysis of a complex problem is easier compared to its synthesis. The efficiency of the optimization scheme depends heavily on how well the variables are adjusted.

The 'performance comparison' stage shown in Figure 1 evaluates the error measure. The latter should have the smallest possible value for an exact solution to be obtained for a particular design problem. In some cases no exact solution is found, hence a nonzero final error exists.

2.2 The error function

There are different ways in which an error measure function can be formulated, for example, error measure function using moduli and error measure function using sum-of-squares.

The error measure function, $\Phi(x_1, x_2, ..., x_n)$, using moduli is given by the general equation :

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{m} | A_{si} - A_i(x_1, x_2, \ldots, x_n) |$$

where x_j, represents the design variables, $j = 1, 2, 3, \ldots, n$

n represents the number of design variables

m represents the number of points defining the specified performance

A_{si} represents the specified performance

A_i represents the actual performance

The moduli operator in the error function is used so that negative terms ($A_i > A_{si}$) do not cancel positive terms ($A_i < A_{si}$). Thus, a zero error solution is obtained only if the specified and actual responses agree at all the *m* chosen points.

The error measure function, $\Phi(x_1, x_2, \ldots, x_n)$, using the sum-of-squares is given by the general equation :

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{m} (A_{si} - A_i(x_1, x_2, \ldots, x_n))^2$$

A zero error solution is obtained, $\Phi = 0$, if and only if $A_{si} = A_i$, for all *i*.
This error function is better than the moduli error function as it avoids the mathematical complexities of dealing with moduli.

2.3 The fitness function

The fitness function for an optimization problem is often defined as the inverse of the error function. This means that if an exact solution is found by the optimizer, then, at that particular point, the error function would have a value

of zero. This also means that the fitness of the exact solution would be maximum.

The fitness function can also be defined as the negative value of the error function. This is equivalent to the inverse of the error function. Throughtout this thesis, the fitness function has been defined as the negative of the error function. When the fitness value of a particular solution is quoted, the absolute value is given.

2.4 Direct search optimization methods

There are three main types of direct search algorithms namely tabulation, sequential and linear methods. These methods require only the computation of objective function values to select suitable search directions.

2.5 Gradient optimization methods

Gradient methods use first- and/or higher-derivatives of the objective function to determine a suitable search direction. There are three main methods:

- Steepest descent (use of first-order derivatives)
- Newton's method (use of second order derivatives)
- Quasi-Newton methods (use of conjugate gradient methods)

2.6 Genetic optimization methods

Genetic optimizers use bits to represent design variables. The bits are evolved to find better design solutions. Genetic optimizers can tackle a large variety of problems and are usually harder to get trapped to local minima, in contrast to the other optimization techniques mentionned before. The genetic algorithm is one such optimization method. The next two chapters discusses two different genetic optimizers namely, Population-Based Incremental Learning (PBIL) and the Cross-generational selection Heterogeneous crossover Cataclysmic mutation (CHC).

Chapter 3

The Population Based Incremental Learning (PBIL) algorithm

This chapter discusses the basic structure of the PBIL algorithm. Then it gives a comparison between the PBIL and GA algorithm. Finally the mechanisms of the PBIL algorithm are explained.

3.1 The basic PBIL structure

The Population Based Incremental Learning algorithm (PBIL) as described by S.Baluja [1;2] is a search method which combines the general genetic algorithm and competitive learning into a simple function optimizer. The object of the algorithm is to create a real valued probability vector which, when sampled, reveals high quality solution vectors with high probability. Initially, the values of the probability vector are set to 0.5. Each value represents an equal probability of either a 1 or a 0 occurring in that position.

A population of random solutions is generated using the probability vector. The probability vector is updated after each generation where it is pushed towards the fittest solution vector by the learning rate.

As search progresses, the values in the probability vector gradually shift to represent high evaluation solution vectors. This is accomplished as follows:
A number of solution vectors are generated based upon the probabilities specified in the probability vector. The probability vector is pushed towards the

generated solution vector(s) with the highest evaluation obtained from the desired fitness function. The distance the probability vector is pushed depends upon the learning rate parameter. After the probability vector is updated, a new set of solution vectors is produced by sampling from the updated probability vector, and the cycle is continued. As the search progresses, entries in the probability vector move away from their initial settings of 0.5 towards either 0.0 or 1.0. The probability vector may then look like this :

[0.01 0.01 0.99 0.01 0.01 0.01 0.99 0.01 0.99 0.99]

The probability vector can be viewed as a prototype vector for generating solution vectors which have high evaluations with respect to the available knowledge of the space.

The PBIL algorithm (see Appendix A for the PBIL program) in contrast to the GA, does not use selection probabilities at each generation, rather, the probability vector is updated through the search procedure by using a few of the best performing individuals. The manner in which the updates to the probability vector occur is very similar to the weight update rule in supervised competitive learning networks. The probability update rule is described as follows:

$$Prob_i = (Prob_i * (1 - LR)) + (LR * Vector_i)$$

where

$Prob_i$: Probability generating a 1 in bit position i.

$Vector_i$: ith position in the solution vector which the probability vector is moved towards.

LR : Learning Rate.

3.2 Comparison between PBIL and GA algorithm

One feature of genetic optimization is the parallelism in the search; many diverse points are represented in the population of early generations. As the search progresses, the population of the GA tends to converge around a good solution vector in the function space. PBIL attempts to create a probability vector that is a prototype for high evaluation vectors for the function space being explored. As search progresses in PBIL, the values in the probability vector move away from 0.5, towards either 0.0 or 1.0. Analogously to genetic search, PBIL converges from initial diversity to a single point where the probabilities are close to either 0.0 or 1.0. At this point, there is a high degree of similarity in the vectors generated.

As PBIL uses a single probability vector, it may seem to have less expressive power than a GA using a full population that can represent a large number of points simultaneously. For example, in Figure 2, the vector representations for populations #1 and #2 are the same although the members of the two populations are quite different. This appears to be a fundamental limitation of the PBIL; a GA would not treat these two population the same.

Population #1	**Population #2**
0 0 1 1	1 0 1 0
1 1 0 0	0 1 0 1
1 1 0 0	1 0 1 0
0 0 1 1	0 1 0 1
Representation	**Representation**
0.5,0.5,0.5,0.5	0.5,0.5,0.5,0.5

Figure 2: The probability representation of two small populations of 4-bit solution vectors with a population size of four.

PBIL has mechanisms that are different from the GA, such as the learning rate and the forgetting factor and does not have neither a crossover operator nor a mutation operator.

3.3 The mechanisms of PBIL algorithm

The PBIL algorithm containes the following parameters :
Population size, learning rate and forgetting factor.

3.3.1 The role of population size.

The population size helps the PBIL algorithm to be more resistant to getting caught in local minima; this occurs as various solution vectors from the population are obtained and only the best solution vector from that group is chosen for the update in the probability vector of the next generation.

3.3.2 The role of learning rate.

The learning rate affects how fast the prototype vector is shifted to resemble a correctly classified point. Since in PBIL, the probability vector is used to generate the next set of sample points, the learning rate also affects which portions of the function space will be explored. The setting of the learning rate has a direct impact on the trade-off between exploration of the function space and exploitation of the exploration already conducted.

In this context, exploration is the ability of the algorithm to search the function space thoroughly, while exploitation refers to the algorithm's ability to use the information it has gained about the function space to narrow its future research. For example, if the learning rate is 0, there is no exploitation of the information

gained through search. As the learning rate is increased, the amount of exploitation increases, and the ability to search large portions of the function space diminishes.

3.3.3 The role of forgetting factor.

The forgetting factor makes the prototype vector move slightly towards 0.5 at each position in the prototype vector. This mechanism is similar to mutation in a GA algorithm. Moving all the bits of the probability vector towards 0.5 is an attempt at avoiding the solution vectors from getting trapped into local regions of the search space.

At the end of each generation, the prototype vector is biased by the learning rate and then the forgetting factor operator is also exerted on the prototype vector.

Chapter 4

The Cross-generational selection Heterogeneous crossover Cataclysmic mutation (CHC) algorithm

This chapter gives a comparison between the CHC and GA algorithm. Then it explains the different mechanisms of the CHC algorithm.

4.1 Comparison between CHC and GA algorithm

The only common point between CHC [3] and the traditional GA is the randomization of the initial population. The CHC algorithm (see Appendix B for the CHC algorithm) is based on an elitist selection process where the fittest member of the current population survives through to the next generation. The GA algorithm is, however, based on a reproduction-selection process. CHC prevents mating between similar individuals, known as incest prevention. In the GA algorithm, no mechanism controls the reproduction process. The recombination operator of CHC is a variant of uniform crossover, whereas that of GA is either a one- or two-point crossover. Finally, at the recombination stage in the CHC algorithm, no mutation is performed but diversity is maintained by restarts of the algorithm whenever convergence is detected. In the GA algorithm, a low rate of mutation is used in the recombination stage to maintain diversity.

4.2 Elitist selection

Elitist selection is a process whereby the fittest member (best solution) of the current population survives through to the next generation. There are two procedures used during elitist selection, namely : selection for reproduction and survival selection.

During selection for reproduction, each member of the old parent population, P(t-1), where t represents the current generation, is copied in random order to a child population, C(t). This means that C(t) is identical to P(t-1) except that the order of the members is different. The members of C(t) are then paired for reproduction, giving a new child population, C'(t).

During survival-selection, the newly created children population C'(t) and the members of the old parent population P(t-1) must compete against each other for survival. This is known as cross-generational competition. The members of P(t-1) and C'(t) are combined and ranked according to fitness. If a member of P(t-1) and that of C'(t) have the same fitness, the member of P(t-1) takes precedence. The new population, P(t), is formed by retaining the best M ranked members, where M represents the population size, of the combined population. Thus, only the best M members will survive to the next generation.

4.3 Uniform crossover

The recombination operator of the CHC algorithm is slightly different from uniform crossover (UX). It takes into account the tradeoff between effective recombination and preservation of schemata. Uniform crossover occurs during reproduction when bits are randomly taken from each parent at different bit

positions to produce new children. The uniform crossover mechanism is shown in Figure 3 :

Parent 1 : 0 0 0 0 0 0 0 0 0 0
Parent 2 : 1 1 1 1 1 1 1 1 1 1

After uniform crossover, the children are :
Child 1 : 1 0 1 1 0 0 1 1 0
Child 2 : 0 1 0 0 1 1 0 0 1

Figure 3: Uniform crossover mechanism

Recombination of high valued schemata from both parents will usually result in better offsprings. However, copying more schemata from one parent will increase the chance of breaking up good schemata from the other parent. To solve this problem, a crossover operator that crosses half the differing bits (between parents) at random is used. This method of crossover, known as HUX, is shown in Figure 4. The high valued schemata of both parents is thereby disturbed in the least sense in the children, compared to the uniform crossover.

Parent 1 : 0 0 1 0 1 0 1 1 1 0 1 1
Parent 2 : 1 0 0 0 1 1 0 1 1 1 0 1
Differing bits : * * * * * * (6 bits)
Cross 3 bits at
random : x x x

Child 1 : 1 0 1 0 1 1 1 1 1 0 0 1
Child 2 : 0 0 1 0 1 0 0 1 1 1 1 1

Figure 4: HUX Crossover mechanism of CHC algorithm

4.4 Incest prevention

During the reproduction step, each parent is selected randomly, without replacing it back to the population, and paired for mating. Parents mate if half the hamming distance (number of places where the bits of the parents differ) exceeds a difference threshold. The latter is initially set to L/4 where L is the number of bits of one parent. This initial threshold is half the expected hamming distance between two randomly generated members. However, parents, whose hamming distance does not exceed the difference threshold, do not mate and are removed from the child population. The procedure mentioned above is known as incest prevention.

4.5 Restart procedure

Whenever the new parent population P(t) is the same as the old population P(t-1), the difference threshold is decremented. When the difference threshold drops to zero, restart occurs as the population has almost converged. Restart is a procedure where the population is reinitialized, keeping the fittest member of the old population. The new members are generated by flipping a fixed proportion of the fittest member's bits.

The CHC algorithm works well on a large range of problems using the same parameter settings due to the reinitialization process. Usually, the best solution, on an easy problem, is found in the first initialization cycle, whereas on hard problems, it is found after repeated restarts.

Chapter 5

Examples of electronic circuit optimization techniques using evolutionary methods

In this chapter, various optimization techniques are used with the evolutionary methods to solve different circuit problems.

5.1 Optimization using the technique of curve matching

The aim of this technique is to determine the component values of a circuit so that the system performance matches a set of requirements at specific values of frequencies [4]. In this problem, a third order LC circuit is used and shown in Figure 5.

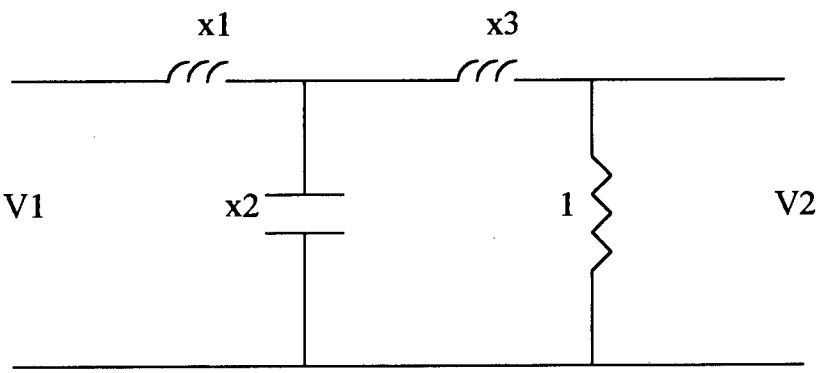


Figure 5: A third order LC ladder circuit

The network shown in Figure 5 has the following transfer function given by :

$T(j\omega) = V_2/V_1 = 1 / (1 - \omega^2 x_1 x_2 + j\omega(x_1 + x_3 - \omega^2 x_1 x_2 x_3))$

where V_1 is the input voltage

V_2 is the output voltage

x_1 represents inductance

x_2 represents capacitance

x_3 represents inductance

The set of requirements defines a curve (see Appendix C1) which is to be matched. The requirements, R, at different frequencies, for this problem is given in Table 1.

ω	R
0.1	0.99999
0.2	0.99996
0.5	0.9922
1.0	0.707
2.0	0.124
5.0	0.008
10.0	0.001

Table 1: Requirements for the LC ladder circuit.

The error function, E, for the evolutionary search is given by :

$$E = \sum_{k=1}^{7} (R_k - |T(\omega_k)|)^2$$

This problem can be solved by an evolutionary method to find the component values of the circuit. The evolutionary method would treat this circuit as an optimization problem and hence minimize the error function to obtain the best possible set of component values. The best set of component values for this

circuit with such a set of requirements is given by : $x_1 = 1.5$ H, $x_2 = 1.33$ F, $x_3 = 0.5$ H.

RESULTS

The components, namely, x_1, x_2 and x_3 are allowed to vary according to the genetic optimizer. Twelve bits were used to represent each component. The search was done on component values in the range between zero and ten.

For the PBIL run, the following parameters were used :
maximum no. of evaluations (maxeval) : 15 000; population (pop) : 100; best fitness (fmax) : 0.001
For the CHC run, the following parameters were used :
maxeval : 15 000; r : 0.35; pop : 50; fmax : 0.001

Comparison between PBIL and CHC runs :

PBIL over 10 runs
It found the solution eight times before 15 000 evaluations
bestfitness = 2.92e-4
worstfitness = 7.26e-3
best no. of evaluations = 1200
worst no. of evaluations = 15000

The best set of component values for this circuit, obtained by PBIL is :
$x_1 = 1.47$ H; $x_2 = 1.36$ F; $x_3 = 0.50$ H

CHC over 10 runs
It found the solution all ten times before 15 000 evaluations
bestfitness = 7.02e-5
worstfitness = 6.78e-4
best no. of evaluations = 506

worst no. of evaluations = 5647

The best set of component values for this circuit, obtained by CHC is :
$x_1 = 1.51$ H; $x_2 = 1.32$ F; $x_3 = 0.51$ H

This shows that both optimizers are able to find the best set of component values of the circuit. Circuit response plots of the values obtained by both PBIL and CHC can be found in Appendix C1.

5.2 Optimization using the technique of nonlinear equation solving

This optimization method matches the coefficients of the transfer function of the same third order LC ladder circuit shown in Figure 5 to some specified values [4].

The voltage transfer function of the circuit is :

$$T(s) = V_2(s)/ V_1(s)$$
$$= (1 / (x_1x_2x_3)) \ / \ (1 / (x_1x_2x_3) + s(x_1+x_3)/(x_1x_2x_3) + s^2/ x_3 + s^3)$$

The defining relations are :

$g_1 = 1/(x_1x_2x_3)$

$g_2 = (x_1+x_3)/(x_1x_2x_3)$

$g_3 = 1/x_3$

g_1, g_2, g_3 represent a set of nonlinear equations. The requirements will be assumed to be those for a third order maximally flat magnitude Butterworth function. Thus we have as requirements, $r_1 = 1$, $r_2 = 2$, $r_3 = 2$ (specified values).

The error function, E, for the evolutionary search is given by :

$$E = \sum_{i=1}^{3} (R_i - G_i)^2$$

where i represents coefficient number

R represents r_1, r_2, r_3

and G represents g_1, g_2, g_3

An evolutionary method can be used to find the component values of the circuit by treating this circuit as an optimization problem. It would minimize the error function to obtain the best possible set of component values. The best set of component values for this circuit with such a set of requirements is similar to the problem in section 5.1 and given by : $x_1 = 1.5$ H, $x_2 = 1.33$ F, $x_3 = 0.5$ H.

RESULTS

The components, namely, x_1, x_2 and x_3 are allowed to vary according to the genetic optimizer. Twelve bits were used to represent each component. The search was done on component values in the range between zero and ten.

For the PBIL run, the following parameters were used :
maxeval : 15 000; pop : 100; fmax : 0.001

For the CHC run, the following parameters were used :
maxeval : 15 000; r : 0.35; pop : 50; fmax : 0.001

Comparison between PBIL and CHC runs :

PBIL over 10 runs
It found the solution twice before 15 000 evaluations
bestfitness = 6.63e-4
worstfitness = 1.55e-2
best no. of evaluations = 2700

worst no. of evaluations = 15000

The best set of component values for this circuit, obtained by PBIL is :
x_1 = 1.53 H; x_2 = 1.34 F; x_3 = 0.50 H

CHC over 10 runs
It found the solution all ten times before 15 000 evaluations
bestfitness = 1.73e-4
worstfitness = 9.86e-4
best no. of evaluations = 642
worst no. of evaluations = 1144

The best set of component values for this circuit, obtained by CHC is :
x_1 = 1.51 H; x_2 = 1.32 F; x_3 = 0.50 H

It could be seen that this method of formulating the optimization problem is more difficult for both PBIL and CHC to solve than the technique of curve matching as they use more evaluations to obtain the best set of component values. Circuit response plots of the values obtained by both PBIL and CHC can be found in Appendix C2.

5.3 Optimization using the technique of curve matching of the real and imaginary part of a general second order rational polynomial function

This technique shows how powerful evolutionary optimization can be as a tool to design circuits. The network considered for solving this problem is Levy's function given by :

$$F = (1 + i\omega) / (1 + (i\omega)/10 + (i\omega /10)^2)$$

The general rational polynomial fitting Levy's function is given by :

$$Z_g = (a_0 + a_1 s + a_2 s^2 + ... + a_p s^p) / (1 + b_1 s + b_2 s^2 + ... + b_n s^n)$$

Here, this general equation is approximated to form a second order rational polynomial function given by :

$$Z = (a_0 + a_1 s + a_2 s^2) / (b_0 + b_1 s + b_2 s^2)$$

In general, this problem is solved by finding the magnitude function of F at different frequencies and then approximating the magnitude to the general rational polynomial Z, hence obtaining the unknown coefficients. However, the resulting set of simultaneous nonlinear equations are badly conditioned and extremely difficult to solve. Gradient optimizers cannot find a solution to this problem (according to Jong and Shanmugam, 1977). E. C. Levy has a method that uses the Gauss-Jordan algorithm to find the set of unknown coefficients. The set of coefficients for this problem is :

$a_0 = 0.9993$; $a_1 = 1.0086$; $a_2 = -1.59e\text{-}5$

$b_0 = 1$; $b_1 = 0.10097$; $b_2 = 0.0100$

Here, this problem is solved using evolutionary optimizers. The real and imaginary part of Levy's function F are approximated to real and imaginary part of a general second order rational polynomial Z respectively to form an optimization problem.

The error criterion E, over the frequency range (1,..., m), is given by :

$$E = \sum_{k=1}^{m} (\text{real} [F(\omega_k) - Z(\omega_k)])^2 + (\text{imag} [F(\omega_k) - Z(\omega_k)])^2$$

where $k = 1, ..., m$

RESULTS

The coefficients, namely, a_0, a_1, a_2 , b_1 and b_2 are allowed to vary according to the genetic optimizer. Twelve bits were used to represent each coefficient. The search was done on coefficient values in the range between zero and one. The frequency range is between 0 and 16 Hz.

For the PBIL run, the following parameters were used :

maxeval : 15 000; pop : 100; fmax : 0.001

For the CHC run, the following parameters were used :

maxeval : 15 000; r : 0.35; pop : 50; fmax : 0.001

Comparison between PBIL and CHC runs :

PBIL over 10 runs

It found the solution twice before 15 000 evaluations

bestfitness = 4.41e-4

worstfitness = 140.01

best no. of evaluations = 8300

worst no. of evaluations = 15000

The best set of coefficient values for this circuit, obtained by PBIL is :

a_0 = 9.99e-1; a_1 = 1; a_2 = 0; b_1 = 1.00e-1; b_2 =1.00e-2 ;

CHC over 10 runs

It found the solution twice before 15 000 evaluations

bestfitness = 3.61e-4

worstfitness = 99.78

best no. of evaluations = 13543

worst no. of evaluations = 15000

The best set of coefficient values for this circuit, obtained by CHC is :

$a_0 = 9.98e\text{-}1$; $a_1 = 9.99e\text{-}1$; $a_2 = 0$; $b_1 = 9.99e\text{-}2$; $b_2 = 1.00e\text{-}2$;

This shows that evolutionary optimizers can solve problems that ordinary gradient optimizers cannot. Circuit response plots of the values obtained by both PBIL and CHC can be found in Appendix C3.

5.4 Optimization of an electronic circuit containing a lossy inductor

The electronic circuit for this problem is a third order low pass filter with a lossy inductor as shown in Figure 6.

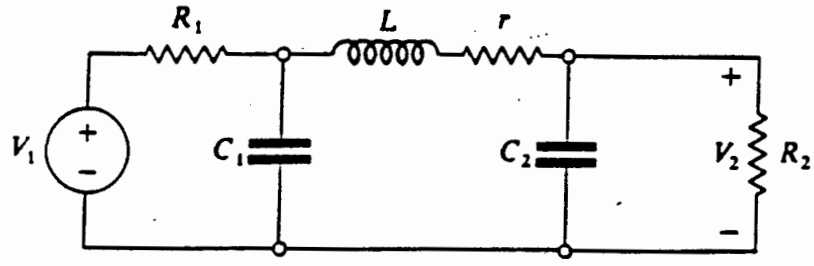


Figure 6: Low pass filter with a lossy inductor

The general expression for such a network is :

$$Z = V_1/V_2$$

$$= R_1LC_1C_2\,s^3 + L[\,(R_1/R_2)\,*C_1 + C_2\,]\,s^2 + [\,L/R_2 + R_1(C_1 + C_2\,)\,]\,s + R_1/R_2 + 1$$

For this problem, the specified transfer function is given by :

$$T = V_1/V_2 = 1.5s^3 + 3s^2 + 3s + 1.5$$

In general, for a typical third order lossless low pass filter, the electronic components can be calculated by the Newton's technique. The component values in this case would be : $R_1 = 1\ \Omega$, $L = 3$ H, $C_1 = 1$ F, $C_2 = 0.5$ F, $R_2 = 2\ \Omega$

As the inductor is lossy, it has a quality factor Q_L at $\omega = 1$ rad/s as :

$\omega L / r = 10$

where r is the resistive part of the inductor representing the loss of inductance.

Such a circuit can still be solved by Newton's technique. However, this requires a lot of mathematical derivations. This circuit can be solved easily by an evolutionary optimizer.

The four coefficients of the general network function Z are matched to that of the specified transfer function T to form an optimization problem. This shows that in this method, one does not need to know the effect of a lossy inductor on the dynamics of such a circuit.

The error criterion E is given by:

$$E = \sum_{i=1}^{4} (Z(i) - T(i))^2$$

where i = 1, ..., 4

RESULTS

The resistance R_1 is fixed at 1Ω and the other components, namely, L, C_1, C_2 and R_2 are allowed to vary according to the genetic optimizer. It should be noted that the value of r can be determined from the relation $r = L / 10$. Twelve bits were used to represent each component. The search was done on component values in the range between zero and ten.

For the PBIL run, the following parameters were used :

maxeval : 50 000; pop : 100; fmax : 0.0001

For the CHC run, the following parameters were used :
maxeval : 50 000; r : 0.35; pop : 50; fmax : 0.0001

Comparison between PBIL and CHC runs :

PBIL over 10 runs
It found the solution once before 50 000 evaluations
bestfitness = 7.18e-6
worstfitness = 4.32e-2
best no. of evaluations = 5500
worst no. of evaluations = 50000

CHC over 10 runs
It found the solution all ten times before 50 000 evaluations
bestfitness = 5.6e-5
worstfitness = 9.9e-5
best no. of evaluations = 3102
worst no. of evaluations = 20796

Both PBIL and CHC found the same two sets of component values to solve this problem; one set having similar values to the Newton's method and the other set having completely different values. This shows that both PBIL and CHC are more powerful as optimizers when compared to Newton's method which found only one set of solution. Further, both these evolutionary methods require little mathematical derivation to obtain a satisfactory solution for this problem.

The Newton's set of component values for the network shown in Figure 6 is :
1. $L = 3.644$ H, $C_1 = 0.943$ F, $C_2 = 0.436$ F, $R_2 = 2.729\ \Omega$

PBIL's best sets of component values for the same network are :

1. $L = 3.629$ H, $C_1 = 0.940$ F, $C_2 = 0.439$ F, $R_2 = 2.716\ \Omega$
2. $L = 1.895$ H, $C_1 = 0.664$ F, $C_2 = 1.206$ F, $R_2 = 2.269\ \Omega$

CHC's best sets of component values for the same network are :

1. $L = 3.595$ H, $C_1 = 0.943$ F, $C_2 = 0.444$ F, $R_2 = 2.713\ \Omega$
2. $L = 1.805$ H, $C_1 = 0.637$ F, $C_2 = 1.309$ F, $R_2 = 2.364\ \Omega$

The PBIL algorithm found the second set of component values twice over ten runs whereas the CHC algorithm found the same set of values seven times over ten runs. Circuit response plots of the values obtained by both PBIL and CHC can be found in Appendix C4.

5.5 Optimization of the 8th order butterworth low pass filter

This problem tackles an 8th order doubly terminated ladder network Butterworth function. The latter is known to have more than one set of component solution. This means that it is a more difficult problem for an evolutionary algorithm to solve. The circuit is given in Figure 7.

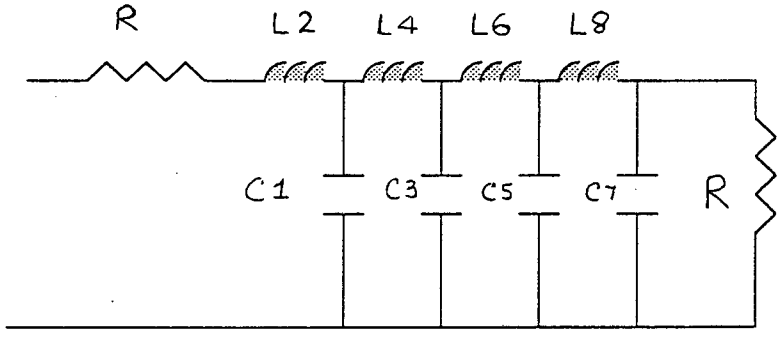


Figure 7: 8th order Butterworth filter

This problem is solved by matching the known coefficients of the Butterworth filter with coefficients generated from an evolutionary method simulating the circuit in Figure 7. Then the component values can be found from the evolutionary algorithm. The technique used is similar to section 5.2. The symbolic representation of the coefficients of this circuit was calculated by the 'derive' package.

RESULTS

The resistances are each set to 1Ω and all the other components are allowed to vary according to the genetic optimizer. Twelve bits were used to represent each component. The search was done on component values in the range between zero and ten.

For the PBIL run, the following parameters were used :
maxeval : 50 000; pop : 100; fmax : 0.0001

For the CHC run, the following parameters were used :
maxeval : 50 000; r : 0.35; pop : 50; fmax : 0.0001

Comparison between PBIL and CHC runs :

PBIL over 10 runs
It did not find the solution before 50 000 evaluations (with terminating criterion fmax=0.001).
bestfitness = 2.25e-1
worstfitness = 10.79
best no. of evaluations = 50000
worst no. of evaluations = 50000

The best set of coefficient values for this circuit, obtained by PBIL is :

$a_0 = 3.15e\text{-}1$; $a_1 = 8.70e\text{-}1$; $a_2 = 1.22$; $a_4 = 1.52$; $a_5 = 2.13$; $a_6 = 1.86$; $a_7 = 1.72$; $a_8 = 6.40e\text{-}1$

CHC over 10 runs

It did not find the solution before 50 000 evaluations (with terminating criterion fmax=0.001).

bestfitness = 3.96e-3

worstfitness = 6.86e-2

best no. of evaluations = 50000

worst no. of evaluations = 50000

The best set of coefficient values for this circuit, obtained by CHC is :

$a_0 = 3.83e\text{-}1$; $a_1 = 1.12$; $a_2 = 1.42$; $a_4 = 2.13$; $a_5 = 1.68$; $a_6 = 2.03$; $a_7 = 1.16$; $a_8 = 3.83e\text{-}1$

Therefore, it is important to find out if the optimizers have found similar or different sets of solutions.

The best three fitnesses provided by the CHC optimizer are analysed having respective fitnesses of 0.008, 0.007 and 0.004.The centroid of these three points was determined by averaging the three points in each dimension individually. Then 'fmins', a simplex algorithm from Matlab, was used to improve each point individually. After 'fmins' was run, the new fitnesses were 0.0025, 0.002 and 0.0014 respectively. It is seen that in all three cases, the fitness has improved.

The euclidean distances of each point, obtained from the optimizer, and the centroid point were computed. The euclidean distances were : 0.6602, 0.6487 and 0.3565 respectively. Further, the euclidean distances of each new point,

obtained from 'fmins', and the centroid were computed. Their respective euclidean distances were : 3.0276, 5.7253 and 2.6475.

The aim of comparing the euclidean distances of the optimizer and of 'fmins' is to find if the euclidean distances increased or decreased. If the euclidean distance of one data point increases, it means that the data point moved away from the centroid point. However, if the euclidean distance of one data point decreases, it means that the data point moved towards the centroid point. In this case, it can be seen that all three euclidean distances increased. This means that all three points moved away from the centroid point. One can conclude that these three points obtained from the optimizer would not converge onto the same minimum, hence these three points are different solution sets to the Butterworth 8th order filter function.

A similar study was done on the PBIL data. No new set of component values were found. The simulation program for this circuit can be found in Appendix C5.

Chapter 6

Examples of robust electronic circuit design by evolutionary methods

This chapter shows how evolutionary methods can be used to choose preferred component values for different circuits. Furthermore, circuits are modelled to take parasitic effects into consideration.

6.1 Component value selection for active filters

Electronic components are normally manufactured in a range of 'preferred' values in order to standardize component values. This helps to reduce the cost of manufacture. However, when a circuit is designed, some of the required component values are not available. So the designer usually selects the nearest preferred value from the range, causing imperfections in the response of the circuits.

There can be a set of component values that will result in a better circuit response. This set is unfortunately found in a huge solution space. Stochastic optimizers (such as PBIL and CHC) can been used to search this space.

The problem tackled in [5] is a fully discrete second order state variable active filter, with low pass frequency response, having six resistors and two capacitors as shown in Figure 8. The specification chosen is a passband cut-off frequency of $\omega_0 = 1591.55$ rad/sec and a selectivity factor of $Q = 1.41421$. The passband

gain H is not critical and thus has been left unconstrained during the solution search.

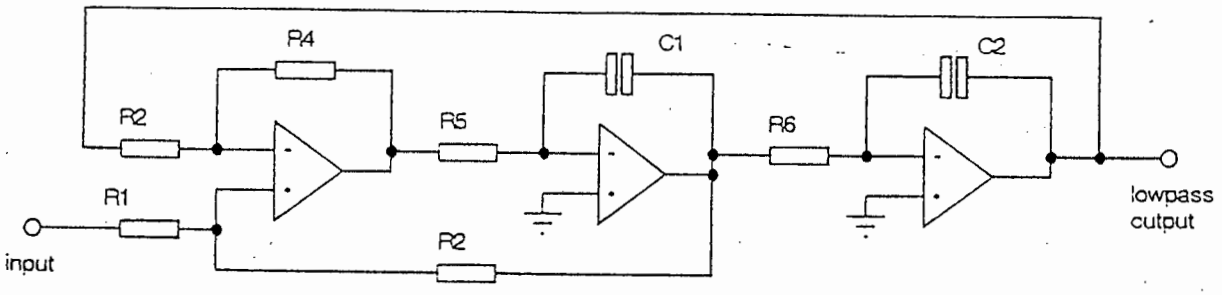


Figure 8: State Variable Filter with Low Pass Frequency Response

Implementation

The preferred values of the electronic components, namely resistors and capacitors, are taken from the 'E-12 series'. Components are chosen from the E-12 series over a four decade range, with the solution space being 2.8e14 points, and are modelled for the filter from the following equations:

$\omega_0 = \text{sqrt}((R4/R3)*(1/(C1C2R5R6)))$

and

$Q = ((R3(R1+R2))/(R1(R3+R4)) * \text{sqrt}((C1R4R5)/(C2R3R6)))$

The error function of the filter is given by :

$$Error = a_1*(|\delta\omega| / \omega_0) + a_2*(|\delta Q| / Q)$$

where a_1 and a_2 are assumed to be equal as in this case both design tolerances for cut-off frequency and selectivity are equal; $\delta\omega$ is error in frequency and δQ is error in quality factor.

RESULTS

Six bits are used for each component value of the filter to specify any preferred value over the four decade range. Two of the bits are used to signify the decade

and the remaining four bits to indicate the value of the E-12 series in that range. (see Appendix D1 for program simulation of circuit)

Table 2 shows the results obtained from the nearest preferred values of the circuit (using the conventional method), a PBIL run and a CHC run.

For the PBIL run, the following parameters were used :
maxeval: 15000; pop : 100; fmax : 0.001

For the CHC run, the following parameters were used :
maxeval : 15000; r : 0.35; pop : 50; fmax : 0.001

	Nearest preferred	PBIL	CHC
Q	1.37234	1.41296	1.41447
$\omega 0$	1773.05	1592.53	1591.52
R_1	4700	68000	5.6e5
R_2	8200	3900	1.8e5
R_3	4700	1.2e6	5.6e4
R_4	4700	27000	2700
R_5	4700	2.2e5	4.7e5
R_6	4700	4700	2700
C_1	1.2e-7	3.9e-9	1e-9
C_2	1.2e-7	2.2e-9	1e-8
Error (%)	7.1824	7.5e-4	1.0e-4

Table 2: Comparison between nearest preferred values and genetic optimizers on the filter problem

As it can be seen from Table 2, both PBIL and CHC have better solutions than the nearest preferred values.

Comparison between PBIL and CHC runs

PBIL over 10 runs :

It found the solution only once at exactly 15000 evaluations.

CHC over 10 runs :

It found the solution 5 times before 15000 evaluations.

The best number of evaluations was : 5324

However, CHC found the worst fitness when compared to PBIL's worst.

This shows that genetic optimizers reduce design errors when compared to nearest preferred value method.

6.2 Genetically derived filter circuit using preferred component values

It is common practice to design LC circuits in which the permitted component values are assumed to be unrestricted. Then, the circuit is converted to a practical one by simply rounding off the exact component values to the nearest value in the permitted set, because of costs. Usually, the circuit performance will differ from the ideal. It may then be necessary to repeat the design with a more stringent specification or to use a more closely spaced set of permitted values, both of which can have cost implications. In general, a better set of preferred values will exist. However this set will be in a solution space of all component-value combinations that is normally huge.

In section 6.1, it has been shown that Genetic Algorithms (GAs) can be used to search this space. There, the application is to a simple second order active filter

specified by its transfer function parameters. In this case, an LC Chebychev filter is analysed and the optimal search is carried out directly on the frequency-response template specification rather than on a specified approximating ideal transfer function as discussed in [6]. This helps to avoid any additional source of approximation.

A seventh order low pass response is considered with template specified by a 1 dB passband ripple with a passband edge at 10^5 rad/sec, and a stopband attention of -150 dB at a stopband edge of 10^6 rad/sec and the circuit is given in Figure 9. The evolutionary algorithm is used to generate the component values of the LC ladder structure.

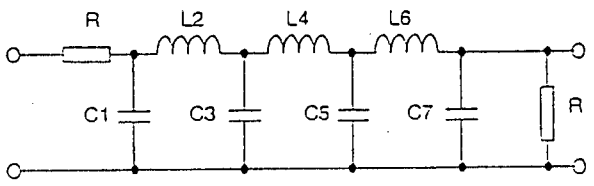


Figure 9: Low pass all-pole LC filter

The circuit was simulated by solving a symbolic representation of the Nodal Admittance Matrix (NAM) for each set of components. This is done in order to minimize computing time in the program.

In this application, the components to be varied are represented by groups of six bits to specify each component value. This allows components to be selected by the optimizer from a range of sixty four permitted values. This range is narrower than the range of preferred values commonly used for discrete components, which span many decades. However this was not a restriction since in practice the solutions obtained were bunched and suitable initial range scaling was easily chosen to centre the component values produced within a sixty four valued range.

The fitness function is defined from the amplitude response error. The latter is calculated as the sum squared excess error at frequencies where the amplitude response in dB falls outside the template for any set of component values. A linear grid of one hundred frequencies is chosen in the passband together with one hundred frequencies in the stopband.

Results

Circuits were designed by the evolutionary approach and for comparison, also by the conventional design approach. In this investigation, capacitors and inductors were chosen from the Twelve-series of preferred values, 10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82, 100, ...

In the conventional approach for LC design, the lowest-order standard polynomial transfer function is first selected that meets the specification. The standard LC filter is then synthesized leading to exact component values. These are then rounded to the nearest preferred values.

The component values, for the source and load resistors, are both assumed to be 100 Ω and were not involved in the stochastic search. The other component values are shown in Table 3 with PBIL and CHC results. The resulting responses are shown in Appendix D2.

	Ideal values	Nearest preferred	PBIL values	CHC values
C1	0.2166μF	0.22μF	0.12μF	0.12μF
L2	1.112mH	1.0mH	1.2mH	1.0mH
C3	0.3094μF	0.33μF	0.18μF	0.18μF
L4	1.174mH	1.0mH	1.0mH	0.56mH
C5	0.3094μF	0.33μF	0.1μF	0.15μF
L6	1.112mH	1.0mH	1.0mH	0.82mH
C7	0.2166μF	0.22μF	0.15μF	0.22μF

Table 3: Components for LC filter

It is seen that the practical response no longer meets the template specification. It is interesting to observe, from Table 3, that the evolutionary derived circuit values are, in general, several values away from the nearest preferred value. This shows that searching combinations of simple nearest rounded up and down values will not produce a global optimum.

For the PBIL run, the following parameters were used :
maxeval : 15 000; pop : 100; fmax : 0.0001

For the CHC run, the following parameters were used :
maxeval : 15 000; r : 0.35; pop : 50; fmax : 0.0001

Comparison between PBIL and CHC runs :

PBIL over 10 runs
It found the solution all ten times before 15 000 evaluations
bestfitness = 0
worstfitness = 8.15e-5

best no. of evaluations = 1800

worst no. of evaluations = 9200

CHC over 10 runs

It found the solution all ten times before 15 000 evaluations

bestfitness = 6.06e-10

worstfitness = 8.15e-5

best no. of evaluations = 1298

worst no. of evaluations = 2845

Another circuit is also tested using the same technique. This circuit is a frequency dependent negative resistance (FDNR) RC active filter shown in Figure 10 and uses the same template as in the above problem.

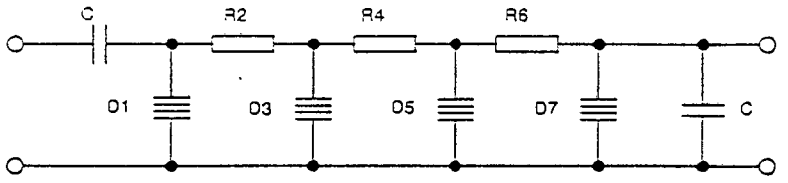


Figure 10: Low pass all pole FDNR filter

Each frequency dependent negative resistor can be represented by the generalized impedance convertor circuit shown in Figure 11.

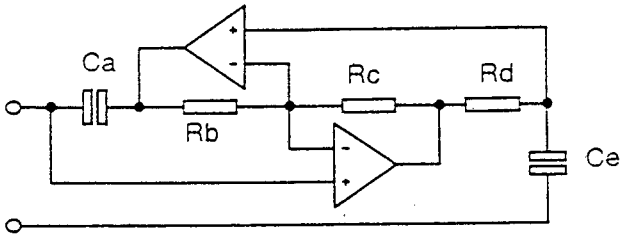


Figure 11: Frequency dependent negative resistor

This circuit uses more components and hence the search space for an evolutionary algorithm is greater.

Results

Six bits were used to represent each component value. The component values, for the source and load capacitors, are both assumed to be 0.1μF and were not involved in the stochastic search. The other component values are shown in Tables 4 and 5 with PBIL and CHC results respectively. The resulting responses are shown in Appendix D2

	Ladder values	Ca μF	Rb Ω	Rc Ω	Rd Ω	Ce μF
D1	1.06e-12	0.47	27	10	1.2e4	100
R2	120Ω					
D3	1.80e-12	1.5	82	22	1.8e4	12
R4	120Ω					
D5	1.32e-12	0.82	27	180	1e4	3.3
R6	120Ω					
D7	4.12e-13	4.7	39	15	1.2e3	0.18

Table 4: Component values for FDNR filter circuit derived by PBIL

	Ladder values	Ca μF	Rb Ω	Rc Ω	Rd Ω	Ce μF
D1	7.25e-13	0.27	270	120	8.2e4	6.8
R2	120Ω					
D3	1.92e-12	2.7	390	39	1.2e4	0.56
R4	120Ω					
D5	2.31e-12	3.3	39	220	3.3e3	0.27
R6	120Ω					
D7	1.59e-12	2.2	33	150	5.6e3	0.82

Table 5: Component values for FDNR filter circuit derived by CHC

It can be seen from the Tables 4 and 5 and Appendix D2 (Response) that with even a bigger seach space, the optimizers can find good solutions for the FDNR filter.

For the PBIL run, the following parameters were used :
maxeval : 15 000; pop : 100; fmax : 0.0001

For the CHC run, the following parameters were used :
maxeval : 15 000; r : 0.35; pop : 50; fmax : 0.0001

Comparison between PBIL and CHC runs :

PBIL over 10 runs
It found the solution all ten times before 15 000 evaluations
bestfitness = 3.18e-6
worstfitness = 9.08e-5
best no. of evaluations = 3050
worst no. of evaluations = 9950

CHC over 10 runs
It found the solution all ten times before 15 000 evaluations
bestfitness = 0
worstfitness = 9.32e-5
best no. of evaluations = 3768
worst no. of evaluations = 7024

6.3 Genetically derived filter circuit modelling parasitic effects using preferred component values

The technique used in this section to design a filter circuit is similar to the previous section except that it allows the genetic search to incorporate parasitic effects [7]. The circuit under consideration is the seventh order LC chebychev filter as shown in Figure 9.

Practical components differ from ideal ones due to parasitic effects. The parasitic components for this filter circuit are modelled as shown in Figure 12 :

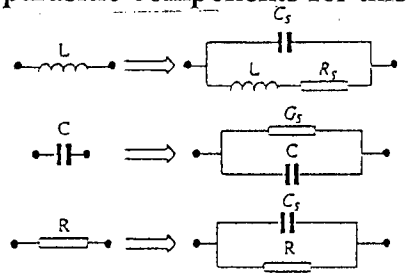


Figure 12: Equivalent circuits modelling parasitic effects

where C_s represents stray capacitances

R_s represents equivalent loss resistance of the inductor

G_s represents equivalent dielectric loss conductance of the capacitor

Circuit interconnections can be non-ideal and are modelled as stray capacitances between adjacent node pairs.

Results

Six bits are used to represent each component value. The parasitic capacitances of all resistors, capacitors and inductors were assumed to be 10, 10 and 20 pF respectively. The inductor equivalent loss resistance was 10 Ω and the dielectric loss conductance of the capacitor was assumed to be negligible. Interconnection parasitics between the main nodes and ground were assumed to be 20 pF.

For the PBIL run, the following parameters were used :
maxeval : 15 000; pop : 100; fmax : 0.0001

For the CHC run, the following parameters were used :
maxeval : 15 000; r : 0.35; pop : 50; fmax : 0.0001

Comparison between PBIL and CHC runs :

PBIL over 10 runs
It did not find the solution four times before 15 000 evaluations
bestfitness = 8.71e-8
worstfitness = 3.09e6
best no. of evaluations = 3700
worst no. of evaluations = 15000

CHC over 10 runs
It found the solution all ten times before 15 000 evaluations
bestfitness = 8.71e-8
worstfitness = 9.70e-5
best no. of evaluations = 1622
worst no. of evaluations = 6476

Both the PBIL and CHC algorithms found the same best set of component values. The latter is :
C1=1e-7; L2=1.2e-3; C3=1.2e-7; L4=1.2e-3; C5=1.2e-7; L6=1.2e-3; C7=1e-7

The resulting response is shown in Appendix D3. When the parasitic components are added to the circuit used in section 6.2, the response obtained does not satisfy the template. However, using the method described in this section, in which parasitic components are included in the evolutionary search, a solution which satisfies the template is obtained.

Chapter 7

Taguchi techniques for robust designs

In this chapter, the Taguchi methodology is discussed.

7.1 Overview of Taguchi technique

The Taguchi technique [8-10] permits the designer to determine optimum parameters, having fully investigated the sensitivity of the system specification to the factors causing variability. Usually, only factorial experimentation of the system leads the designer to determine exactly the optimum set of parameters. This method is not convenient as a large number of experiments is required. However, by using the methods of statistical experimental design, it is possible to find the effects of many parameters within a few carefully selected experiments. This is known as the Taguchi technique.

The Taguchi technique is a method to design fractional factorial experiments based on Latin squares (balanced square arrangements required for unbiased statistical experimentation). Taguchi's approach to the design of experiments utilizes Robust Design, which can be applied to a wide variety of problems. Robust Design reduces economically the variation of a product's function in the customer's environment so that high-quality products can be produced quickly and at low cost.

The Taguchi method differs from the classical methods, developed by R.A. Fisher, in that the time and cost required to learn and use it are minimal.

Furthermore, it requires little training of engineers in applied probability and statistics. Fisher's methods are often cumbersome to implement in manufacturing industrial experimentation because of certain assumptions and procedural emphasis.

7.2 Design Process

The aim of experimentation during the design process is to devise ways of minimizing the deviation of a quality characteristic, such as durability, from its target value. This can be done only by identifying the factors which affect the quality characteristic and by changing the appropriate factor levels. From a quality point of view, experimentation seeks to determine the best factors which will produce a desired quality characteristic taking cost into account.

7.2.1 Noise factor

A parameter that causes the deviation of a quality characteristic from its target value is the noise factor. There are three main types of noise :

- External noise - factors in the environment that influence the behaviour of a product
- Internal noise - factors that cause the deterioration of a product with age
- Unit-to-unit noise - factors that make differences to occur between individual products that are manufactured to the same specifications

7.2.2 Function of quality control processes

During a design process, there are four quality control stages :

- Product design

- Production process design,
- Manufacturing, and
- Customer usage.

The aim at each stage is to minimise noise factors. Quality control processes prior to manufacturing are called off-line quality control while quality control processes in manufacturing are called on-line quality control. The Taguchi technique is mainly applied on the production process stage, which consists of : system, parameter and tolerance design.

7.2.2.1 System design

The conceptual stage of any product development is system design. The latter requires technical knowledge and extensive experience in an area of specialization to initially design a product. It is a search for the best available technology. It includes the selection of materials, tentative product parameter values and the selection of production equipment. The strategy behind system design is to take new ideas and convert them into something that can work.

7.2.2.2 Parameter design

The aim of parameter design is to take the innovation which works in system design stage and enhance it so that it will consistently function as intended. The nominal values from the system design are tested over specified ranges and the best combination of levels is determined. Parameter design provides the means of reducing costs and improving quality by making use of experimental design methods. It determines the product parameter values which are less sensitive to change in environmental conditions and other noise factors. This step is very important in robust design.

7.2.2.3 Tolerance design

After the system has been designed and the nominal values of its parameters have been determined, the tolerances of the parameters are then set. Tolerance design helps to control factors that affect the target value. This entails ar increase in cost as better grade components have to be used. The methodology is different than in parameter design. In the latter, factor levels that make the quality characteristic insensitive to noise factors are identified. In the tolerance design stage, the noise factors are controlled by keeping them within narrow tolerances. Cost calculations are used to determine the tolerances.

7.2.3 Factors affecting a quality characteristic

A number of factors other than the noise factor can affect the response variable of a product. These factors are :

- Control factors - these are parameters whose values are controlled by the design engineer.
- Signal factors - these are factors which change the true values of the quality characteristic to be measured.
- Scaling factors - these are factors which adjust the mean level of a quality characteristic to make the functional relationship between a signal factor and the quality characteristic possible.

7.2.4 Properties of factors

There are a number of properties of factors that need to be considered :

- factor levels,

- number of factor levels,
- effect of the number of factor levels on the number of experiments

7.2.4.1 Factor levels

These are levels of values given to a factor, which may be a noise, control, signal or scaling factor. A control factor such as *force* may be represented as in Figure 13. A factor must have at least two levels because at least two measurements are needed for a comparison. In Figure 13, the factor *force* is being studied at two levels, coded A1 and A2 corresponding to levels 1 and 2. Their respective forces are 100 and 300 newtons.

Response	Code	Level 1	Level 2	Units
Force	A	100	300	newton

Figure 13: Factor levels

7.2.4.2 Number of factor levels

The levels used for each factor selected for the experiment play an important role in planning. The number of levels for the qualitative factors will usually be determined from the problem being investigated. However, choosing the appropriate levels for the quantitative factors is a more difficult task. The number of levels selected depends on the amount of information gathered about the product. For example, if a new product is being studied, it may be desirable to run three levels for some of the variables to evaluate non-linearity over the range of the variables. However, if the effect of certain variables is known, then only two levels are sufficient to extract the desired response from the experimental results.

7.2.4.3 Effect of the number of factor levels on the number of experiments

For a 2-level factor at least one measurement is made at each level. If in an experiment there are two factors each being studied at two levels, then there must be four measurements. In general, if there are *m* factors at two levels, there will be 2^m measurements. Similarly, if there are *n* factors at three levels, there will be 3^n measurements. When *m* two level and *n* three level factors are found in an experiment, there will be $2^m * 3^n$ measurements.

7.3 Matrix selection for experiments

A matrix is an array of numbers treated as a single quantity. A matrix experiment consists of a set of experiments where the factors and levels are changed according to the matrix. These factors and levels are the settings of the various product parameters to be investigated. The effect of several parameters can be determined efficiently from matrix experiments using special matrices, known as orthogonal arrays. The latter is an important technique in Robust Design.

7.3.1 Orthogonal arrays

An orthogonal array is a matrix of numbers set in columns and rows. Each column represents a specific factor that can be changed from experiment to experiment. Each row represents the state of the factors in a given experiment. The array is called orthogonal because the levels of the various factors are balanced and can be separated from the effects of the other factors within the experiment. The orthogonal array is a fractional factorial array which assures a balanced and fair comparison of levels of any factor or interaction of factors whereby all columns can be evaluated independently of one another.

Orthogonal arrays have a number of unique properties:

- equal proportions of experiments,
- equal proportions of remaining factor levels,
- equal proportions of combinations of factor levels.

The $L_8(2^7)$ orthogonal array shown in Figure 14 is often used in the design of experiments.

The L notation in $L_8(2^7)$ represents an orthogonal array.

The subscript 8 represents the number of experiments.

The number 2 indicates the number of factor levels.

The superscript 7 indicates the number of factors.

Exp	A	B	C	D	E	F	G
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

Figure 14: The $L_8(2^7)$ orthogonal array.

7.3.1.1 Equal proportions of experiments

In column A, there are four ones and four twos in this column. Similarly, in all other columns there are also four ones and four twos.

7.3.1.2 Equal proportions of remaining factor levels

Consider column A. The factor level 1 (denoted A1) appears in experiments 1, 2, 3 and 4. While factor A is in level 1, half of factor B is in level 1 and the other half is in level 2. Similarly, while factor A is in level 1, half of factor C in level 1 and the other half is in level 2. This is true for all the remaining factors. This is also true if we started with any other factor.

7.3.1.3 Equal proportions of combinations of factor levels

Consider columns A and B. In experiments 1 and 2, factor A is in level 1 and factor B is in level 1. We denote this as (1,1). In experiments 3 and 4, factor A is in level 1 and factor B is in level 2. We denote this as (1,2). Similarly, there are (2,1) and (2,2) in experiments 5, 6 and 7, 8 respectively. These combinations (1,1), (1,2), (2,1) and (2,2) occur an equal proportion of times. This is true for any two pairs of the columns A to G. It is this balanced property for which the array is called orthogonal. These unique properties of the $L_8(2^7)$ orthogonal array thus enable seven fair comparisons of factors, namely, A1 against A2, B1 against B2, and so on, to G1 against G2.

7.3.2 Degrees of freedom for factors and levels

The degrees of freedom is the number of independent measurements available to estimate sources of information. The number of degrees of freedom indicates the number of independent comparisons that may be made within a set of data. In general, the number of degrees of freedom associated with a factor (ν_{fl}) is equal to one less than the number of levels for that factor.

$$\nu_{\text{fl}} = \textit{number of levels} - 1$$

7.3.3 Degrees of freedom of orthogonal arrays

Conducting experiments with orthogonal arrays yield only a certain number of independent comparisons for factors. This is the degrees of freedom in an orthogonal array (v_{oa}) which is always one less than the number of experiments because one degree of freedom is taken up by the overall mean.

$$v_{oa} = number\ of\ experiments - 1$$

7.3.4 Comparison of degrees of freedom for experimentation

In a standard orthogonal array, the number of degrees of freedom of factors and levels must be matched with the number of degrees of freedom for that array. When choosing an orthogonal array for experimentation, we must ensure that the number of degrees of freedom for the factors and levels (v_{fl}) is equal to or less than the number of degrees of freedom for the orthogonal array (v_{oa}). If an experiment does not use all the degrees of freedom of an orthogonal array can provide, orthogonality is still preserved and the experiment can proceed.

7.3.5 Choosing an orthogonal array

Taguchi has formed 18 orthogonal arrays as seen in Figure 15 which, in some cases, can be used directly to plan an experiment.

2 levels	3 levels	4 levels	5 levels	mixed levels
$L_4(2^3)$	$L_9(3^4)$	$L_{16}(4^5)$	$L_{25}(5^6)$	$L_{18}(2^1*3^7)$
$L_8(2^7)$	$L_{27}(3^{13})$	$L_{64}(4^{21})$		$L_{32}(2^1*4^9)$
$L_{12}(2^{11})$	$L_{81}(3^{40})$			$L_{36}(2^{11}*3^{12})$
$L_{16}(2^{15})$				$L_{36}(2^3*3^{13})$
$L_{32}(2^{31})$				$L_{54}(2^1*3^{25})$
$L_{64}(2^{63})$				$L_{50}(2^1*5^{11})$

Figure 15: Taguchi orthogonal arrays

7.4 Interactions between factors

An interaction occurs when two or more factors together have an effect on the quality characteristic that is different from those of the factors taken individually. When such an effect is strong, the task of predicting the effect of a factor selection becomes difficult. The effect of interactions can be annulled by performing an additivity experiment which consists of :

- predict the quality characteristic
- conduct a confirmation experiment
- compare prediction against confirmation

7.4.1 Prediction experiment of the quality characteristic

Using the results of the orthogonal array experiment, we calculate the predicted value of the quality characteristic $\mu_{predicted}$ based on the mean effects of factor levels.

7.4.2 Confirmation experiment of the quality characteristic

The confirmation experiment is very important in the design of an experiment. The validation experiment is conducted at the optimum factor level settings and the quality characteristic $\mu_{confimation}$ at the optimum settings is obtained.

7.4.3 Comparison of prediction with confirmation

The most important part of a design of an experiment is in the comparison of $\mu_{predicted}$ with $\mu_{confirmation}$. If the confirmation value is within +/- 5% of the predicted value when we compare $\mu_{predicted}$ to $\mu_{confirmation}$, then we can assume that the values are similar. In this case, additivity is present and interaction effects cannot be dominant. However, if the confirmation value is not within the required range, then additivity is not present, in which case, interaction effects may be dominant.

7.5 Aim of the Additivity principle (Confirmation experiment)

The main purpose of a confirmation experiment is to detect when there are strong interaction effects. Hence, additivity and the reproducibility of experimental results are poor. If the predicted optimum conditions derived from orthogonal array experimental results are not validated by a confirmation experiment, laboratory optimization may be inadequate for usage. Thus, a confirmation experiment should always be conducted to ensure reproducibility and thereby prevent faulty product designs.

7.6 Formal mathematical treatment of Taguchi's approach

In general, for a particular set containing *i* experiments, there would be *i* results R_j with j = 1 to i.The mean result, μ_{result}, of this set of experiment is given by :

$$\mu_{result} = 1/i \sum_{j=1}^{i} R_j \quad (j = 1, ..., i)$$

In that experiment, consider *m* factors f, each factor having *x* levels; hence there are *m*x* factor levels represented as: $[f_1(1), ..., f_1(x)]$; $[f_2(1), ..., f_2(x)]$ to $[f_m(1), ..., f_m(x)]$.

Each factor, in an orthogonal array, causes a certain response at any one level known as factor effect F. The factor effect of a particular level can be calculated by substracting the mean result of all similar factor levels from the overall mean result of all factor levels.

For a particular level *k* of factor f_p, the factor effect $F(f_p(k))$ is given by :

$$F(f_p(k)) = \sum_{l=fp(k(1))}^{fp(k(n))} R_l / n - \mu_{result}.$$

where $l = f_p(k(1)), ..., f_p(k(n))$

$n = i/x$

$\{ p: (1 < p \leq m) \}$

$\{ k: (1 < k \leq x) \}$

and $\{ l: (1 < l \leq n) \}$

The mean factor effect μ_F of a particular factor f_p of all factor levels is given by:

$$\mu_{F(fp)} = \sum_{y=1}^{x} F(f_p(y) / x$$

where $y = (1, ..., x)$

The level effect L of a factor is the sensitivity analysis of each individual level. The level effect L for a given factor f_p at a certain level can be calculated by substracting the factor effect of that particular level from the mean factor effect.

For a particular level k, with factor f_p, the level effect L ($f_p(k)$) is given by :

$$L(f_p(k)) = F(f_p(k)) - \mu_{F(fp)}$$

The worst possible combination, W, of factor levels is found by finding the maximum level effect from all levels for all factors.

$$W = \max \{ L(f_1(y)), L(f_2(y)), ..., L(f_m(y)) \}$$

where $\{y: (1 < y \leq x)$

Hence the predicted experiment can be calculated from the following formula :

$$\mu_{predicted} = \mu_{result} + \sum_{z=1}^{m} f_z(W)$$

where $z = (1, ..., m)$

The confirmation experiment is calculated from the quality characteristic function γ using the worst case combination factor levels. The confirmation experimental value $\mu_{confirmation}$ can be calculated from the following formula :

$$\mu_{confirmation} = \gamma (W)$$

where γ represents the mathematical equivalence for the quality characteristic function.

The Additivity principle holds if the confirmation value is within +/- 5% of the predicted value. The additivity principle formula, AP is given by :

$$AP = | \mu_{confirmation} - \mu_{predicted} | / \mu_{confirmation}$$

Chapter 8

A new robust hybrid genetic algorithm combining the Taguchi techniques in local search stage with a traditional genetic search algorithm

This chapter discusses a new robust hybrid genetic algorithm with various techniques used during the local search stage.

8.1 Genetic search and robustness

As we have seen, genetic evolution and stochastic search tend to lead to near optimal performance in electronic circuit designs. However such designs are not necessarily robust with respect to perturbations of the nominal component values. It is therefore of interest to explore possibilities of incorporating robustness as part of the fitness evolution process, thereby developing a process which leads to the evolution of inherently robust designs.

8.2 Hybrid genetic algorithms incorporating local learning

Hybrid genetic algorithms have been devised which combine local search with the more traditional version. The mechanism of a hybrid genetic algorithm can be formulated as follows:

1. Generate a population of different solutions (as in the case of a traditional search).

2. Perform local search using each member of that population.
3. If better solutions are found during the local search, these solutions replace the original solutions into the population. - acquired or learned behavior.
4. Repeat step 1 (using the improved population) until termination criterion is achieved.

Genetic search can be modified by the use of local learning (for example using a hill-climbing algorithm on each individual of the population).

Depending on the use made of the additional information gained, this is known as Lamarckian or Baldwinian learning. Lamarckian learning describes the procedure in which, after the learning step, the modified individual replaces the original version in the population - that is to say, the evolutionary process (as is not the case in nature) works with genetic material which has been modified by the learning process.

Baldwinian learning is closer to natural process : here the learning step serves only to evaluate the fitness of an individual - an individual able to improve its performance through learning (hill-climbing) is thereby credited with a greater fitness (in that concluded after learning). However the 'genetic material' is not altered. The original individual is replaced into the mating pool.

Usually better convergence and better optimal solution are achived using the hybrid genetic algorithm than with a traditional genetic optimizer [11;12]. However, a traditional hybrid genetic algorithm presents the same problem as a traditional genetic optimizer; the unrobust nature to the variability of the electronic circuit components.

8.3 Robust hybrid genetic algorithm using full factorial combination during local search stage

A robust hybrid genetic algorithm is a possible solution to account for the variability of electronic circuit components. During the local search, for a given tolerance on component values, a full factorial combination (covering all possible combinations) of all tolerance component values can be performed on each member of the population. This is similar to a sensitivity analysis of the neighbourhood of the proposed solution.

The worst point in the neighbourhood of the proposed solution, known as the worst fitness neighbourhood point, f_w, is defined by a combination of tolerance component values, known as the worst combination,W. A solution is termed as robust, when the difference between the solution's fitness, f_s, and the worst fitness neighbourhood, f_w, is minimum, as shown in the following equation :

$S = \delta f = (f_s - f_w) \rightarrow 0 \Rightarrow$ Robust design

where S is the macroscopic sensitivity of the neighbourhood

$f_s > f_w$

In this new hybrid genetic algorithm, the population is first ranked according to the fitness of their solution. Then a local search of the neighbourhood of each member of the population is performed and f_w determined for each member.

During the search, both robust and optimal characteristics are needed to obtain a final solution which is both optimal and robust to variability of component values. For each point in the search space, the local neighbourhood is examined (by exhaustive perturbation) and the fitness accorded that point is that of the worst perturbed variant.

To find the desired solution, a procedure is performed where the worst fitness neighbourhood point of the first member f_{w1} of the ranked population is tested against the solution fitness of the second member f_{s2} of the ranked population. If $f_{w1} > f_{s2}$ occurs, then the desired solution of the population is found (see Figure 16). If $f_{w1} < f_{s2}$ occurs, then the worst fitness neighbourhood point of the second member f_{w2} of the ranked population is tested against the solution fitness of the third member f_{s3} of the ranked population (see Figure 17). And the test is done, down the ranked population until the desired solution is found.

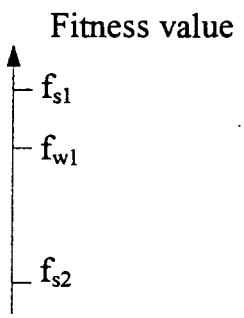


Figure 16: Worst fitness point of f_{s1} > fitness solution f_{s2}

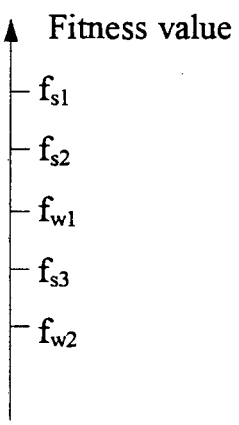


Figure 17: Worst fitness point of f_{s1} < fitness solution f_{s2}

In general, the desired solution is found as follows :

1. Rank population in order of fitness f_i, where i represents the ith ranked member of the population.

2. Take member with fitness f_i and perform local search (sensitivity analysis of neighbourhood region), and find fitness of worst combination f_{wi}.
3. If $f_{wi} > f_{i+1}$, desired solution of population is found
4. Else, go to step 2.

This algorithm successfully solves the problem of variability of components. However, as a full factorial approach is performed to find the sensitivity of the solution, a lot of computing time is taken. Further, this method becomes more inefficient with an increase in component values in a circuit. In general, for a circuit containing n component values, a full factorial experimentation of the neighbourhood of a design solution would require 2^n experiments.

It should be noted that this technique of using a hybrid genetic algorithm is different from the traditional hybrid genetic algorithm, as here, the robustness characteristic of the final solution is improved as compared to a traditional one, where only the fitness of the final solution itself is improved.

8.4 Role of Taguchi techniques

The method of experimental design popularised by Genichi Taguchi perform an approximate neighbourhood sensitivity analysis by a fractional factorial process. This requires much less computing time.

An orthogonal array is used to design the fractional factorial experiments such that the experiments on the neighbourhood are as far apart as possible and the experimental variables enter the calculation in a balanced way. This often allows us to approximate the sensitivity of the neighbourhood accurately. For example, on a three dimensional problem, the neighbourhood of a proposed solution can be viewed as a cube. A full factorial set would require eight

experiments that is the eight corners of the cube. However, in a fractional factorial set, only four experiments are required as seen from Figure 18.

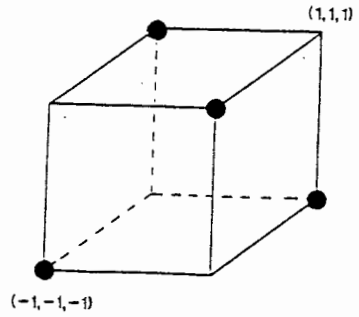


Figure 18: Three dimensional problem neighbourhood.

These four points are used to calculate the sensitivity of the neighbourhood, hence determine the gradient of the neighbourhood. Then, the worst point in the neighbourhood is predicted. A validation experiment is used to test if the worst point is approximated with sufficient accuracy. If it is, then the effects of interactions between variables can be ignored and the model is said to be additive.

8.5 Robust hybrid genetic algorithm using Taguchi techniques during the local search stage

This algorithm is similar to the one mentioned in section 8.3 except that instead of the full factorial approach during the local search, only a fractional approach based on the Taguchi technique is used.

This robust hybrid genetic algorithm uses the Taguchi techniques to predict the gradient of the local neighbourhood of a particular solution, and if the validation experiment holds, then the Taguchi techniques are used to predict the worst point in that neighbourhood. However, if the validation experiment does not hold, then a full factorial search of that neighbourhood is done and the

worst point of that neighbourhood is found. Once the worst point of the neighbourhood is determined, then the same procedure as mentioned in section 8.3 is carried out.

8.6 Robust hybrid genetic algorithm using Taguchi techniques and Lamarckian learning during local search stage

This algorith is similar to the one mentioned in section 8.5 except that Lamarckian learning is used during the local search to guide the search towards better solutions.

The effect of the local search for a better robustness characteristic of a particular solution usually slows down the convergence of the hybrid genetic search. This effect can be viewed as a negative Baldwinian effect, where the fitness of the original solution is replaced by the fitness of the robust characteristic determined during the local search. The fitness of the robust characteristic of a particular solution is the worst fitness of the local neighbourhood.

This is precisely the inverse of ordinary Baldwinian learning where the fitness of the learned solution from the local search replaces the original fitness in the population. The fitness of the learned solution is the best fitness of the local neighbourhood.

To offset this effect and improve the convergence rate, Lamarckian learning is used in the robust hybrid genetic search. Lamarckian learning in fact negates the effect the negative Baldwinian learning.

Chapter 9

Examples of robust electronic circuit design using a robust hybrid genetic algorithm with Taguchi methods

In this chapter, a tutorial problem is discussed on the mechanism of the new robust hybrid genetic algorithm. The latter is also tested on two different problems.

9.1 Tutorial problem

This problem shows how the robust hybrid genetic algorithm works on a one dimensional problem. Consider a circuit which contains only one component that can be varied. Figure 19 shows the fitness surface of the component with respect to the circuit.

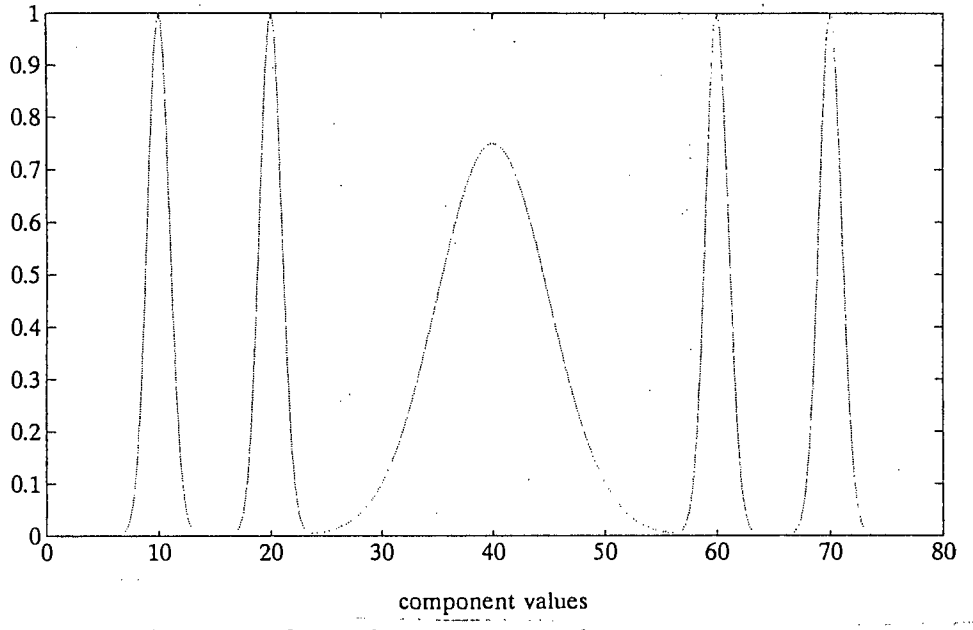


Figure 19: Fitness surface of component values.

As it can be seen from Figure 19, there are four good component values for the circuit, namely 10, 20, 60 and 70. The PBIL and CHC algorithms are able to find these four points. As components are usually manufactured with an associated tolerance, the fitness surface of the component values would change if the component is modelled with a particular tolerance. With a tolerance of 1% associated with the component value, the resulting fitness surface of the component with respect to the circuit is shown in Figure 20.

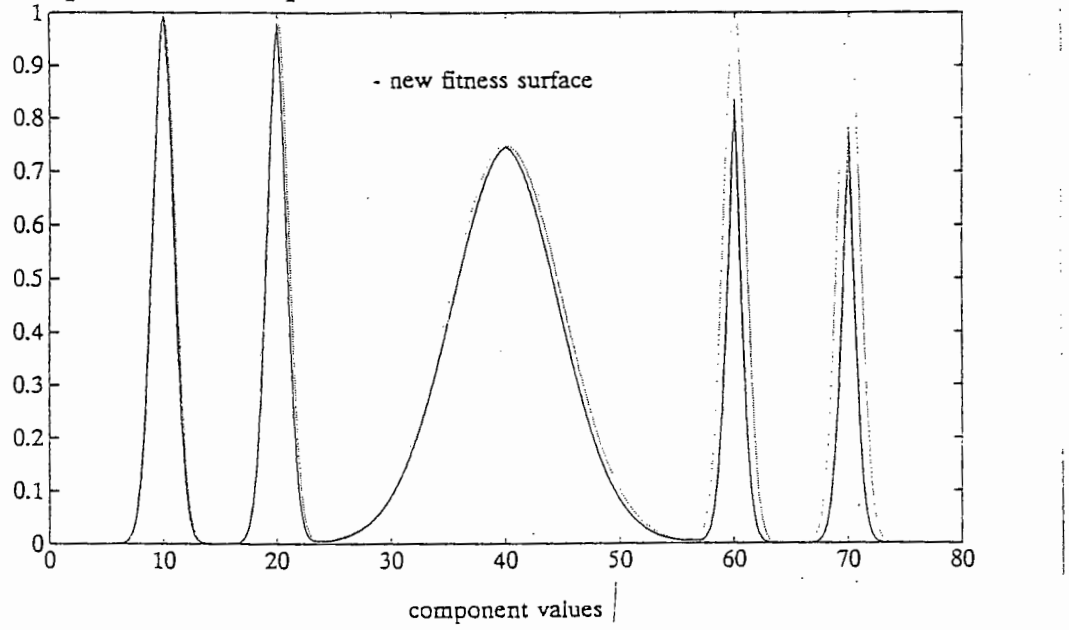


Figure 20 : Fitness surface of component values with associated tolerance of 1%

This fitness surface shows that only two component values are robust to a tolerance of 1%, namely 10 and 20. Since this problem is only in one dimension, this means that at any particular value, the neighbourhood can be tested for only two possibilities, that is either +1% or -1% of the particular value. The fitness surface shown in Figure 20 is obtained as follows:

1. Shift all component values 1% to the left of Figure 19 (forming a new fitness surface as shown in Figure 21).
2. Shift all component values 1% to the right of Figure 19 (forming another fitness surface as shown in Figure 21).

3. Compare the two new surfaces by taking the minimum at each component value. This forms the fitness surface shown in Figure 20.

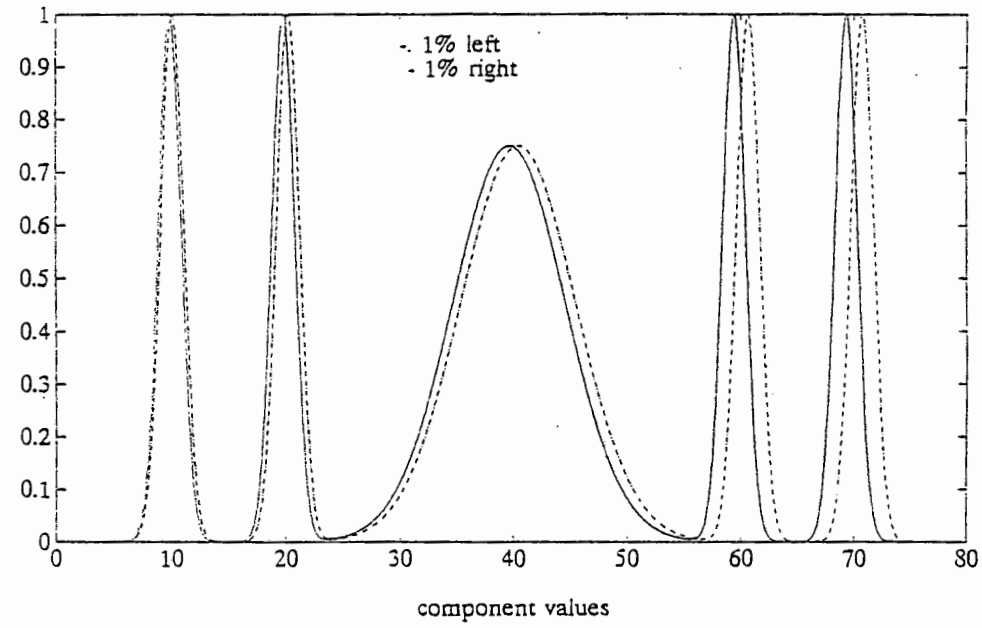


Figure 21: Shift of 1% to the left and 1% to the right

For a tolerance of 1% of the component values, the robust hybrid genetic algorithm sees the fitness surface shown in Figure 20 instead of that in Figure 19. Steps 1 and 2 form the local search (learning) in a hybrid genetic algorithm (usually done by the Taguchi techniques in several dimensions). Step 3 is the Negative Baldwinian learning.

9.2 Robust design of the seventh order Butterworth low pass filter

This problem uses the same optimization technique as in section 6.2 except that component values are not chosen from the Twelve series of preferred values. Instead, values are chosen on a continuous scale over a four decade range. A set of component values for the 7th order butterworth low pass filter (see Figure 9

) obtained from both the PBIL and CHC algorithms are tested for tolerance of 1,3,5,7 and 10% of component values.

The PBIL set of component values is :
C1= 2.02e-7F; L2= 9.82e-4H; C3= 2.84e-7F; L4=1.24e-3H; C5= 9.47e-8F; L6= 2.94e-4H; C7= 8.66e-8F

The CHC set of component values is :
C1= 9.47e-8F; L2= 1.09e-3H; C3= 2.25e-7F; L4= 9.82e-4H; C5= 7.50e-8F; L6= 5.73e-4H; C7= 8.82e-8F

Table 6 shows the worst perturbed fitness for the PBIL and CHC solutions over the range of tolerance sets. (For definition of fitness function used in this thesis, refer to section 2.3)

Tolerance	PBIL fitness	CHC fitness
0%	5.55e-6	4.19e-5
1%	2.24e5	5.07e-5
3%	1.63e7	2.33e4
5%	8.15e7	3.48e6
7%	2.30e8	1.92e7
10%	6.58e8	8.67e7

Table 6: Worst perturbed fitness for the PBIL and CHC solutions over the range of tolerance sets.

This shows that the solutions obtained from both optimizers are unrobust to tolerance of components. Circuit response plots of the values obtained by both PBIL and CHC can be found in Appendix E1.

So, a new hybrid genetic algorithm with a full factorial perturbation combinations (128 combinations for 7 components) during the local search stage is used on the 7th order Butterworth low pass filter. The hybrid PBIL and hybrid CHC algorithms are evolved over the range of tolerance. Tables 7 and 8 show the solutions obtained from the hybrid PBIL and CHC algorithms over the range of tolerance.

For all the remaining problems, the parameters of both the PBIL and that of the CHC algorithms are fixed to the following values:
PBIL : maxeval : 15 000; pop : 50; bfmax : 0.001
CHC : maxeval : 15 000; r : 0.35; pop : 50; bfmax : 0.001
where bfmax represents the worst fitness of the neighbourhood to be reached as termination criterion for both optimizers. (Negative Baldwinian Learning).

Tole-rance	C1 F	L2 H	C3 F	L4 H	C5 F	L6 H	C7 F	Worst fitness
1%	1.43e-7	9.47e-4	2.31e-7	1.04e-3	1.60e-7	7.50e-4	5.62e-8	2.79e-6
3%	1.07e-7	9.82e-4	9.31e-8	6.04e-4	1.65e-7	9.47e-4	5.52e-8	2.49e-4
5%	7.10e-8	9.31e-4	8.82e-8	7.91e-4	8.98e-8	9.82e-4	8.98e-8	7.57e-4
7%	4.87e-8	9.82e-4	1.09e-7	9.65e-4	9.47e-8	9.65e-4	9.14e-8	7.38e-4
10%	5.33e-8	9.82e-4	9.82e-8	9.82e-4	9.82e-8	1.07e-3	8.98e-8	9.31e-4

Table 7: Hybrid PBIL solutions with different tolerance sets using full factorial

Tole-rance	C1 F	L2 H	C3 F	L4 H	C5 F	L6 H	C7 F	Worst fitness
1%	4.37e-8	9.47e-4	9.65e-8	7.23e-4	8.82e-8	9.31e-4	1.26e-7	6.06e-4
3%	8.82e-8	8.20e-4	6.98e-8	7.64e-4	1.63e-7	9.65e-4	9.47e-8	2.00e-4
5%	8.82e-8	8.20e-4	6.98e-8	7.64e-4	1.63e-7	9.65e-4	9.47e-8	2.81e-4
7%	8.82e-8	8.20e-4	6.98e-8	7.64e-4	1.63e-7	9.65e-4	9.47e-8	3.95e-4
10%	9.31e-8	9.82e-4	1.60e-7	6.61e-4	7.50e-8	8.51e-4	9.47e-8	6.38e-4

Table 8: Hybrid CHC solutions with different tolerance sets using full factorial

This shows that the solutions obtained from both hybrid optimizers are robust to tolerance of components. Circuit response plots of the values obtained by both hybrid PBIL and hybrid CHC can be found in Appendix E2.

However, the problem with a full factorial search during the local search stage is the lengthy computing time to obtain a solution.

Table 9 shows the average computing time to obtain a solution for the hybrid PBIL and hybrid CHC algorithms.

Tolerance	PBIL (minutes)	CHC (minutes)
1%	1	2
3%	1.5	3
5%	5	3.5
7%	35	5.4
10%	125	7

Table 9: Average computing time to obtain a solution for the hybrid PBIL and hybrid CHC algorithms using full factorial

The Taguchi method was used during the local search stage to decrease the computing time. The hybrid PBIL and hybrid CHC algorithms are evolved over the range of tolerance. Tables 10 and 11 show the solutions obtained from the hybrid PBIL and CHC algorithms over the range of tolerance.

Tole-rance	C1 F	L2 H	C3 F	L4 H	C5 F	L6 H	C7 F	Worst fitness
1%	1.43e-7	9.47e-4	2.13e-7	1.04e-3	1.60e-7	7.50e-4	5.62e-8	2.79e-6
3%	7.77e-8	1.00e-3	1.22e-7	5.83e-4	1.57e-7	9.82e-4	5.14e-8	3.53e-4
5%	8.82e-8	9.47e-4	1.68e-7	5.94e-4	9.14e-8	8.35e-4	8.82e-8	2.90e-4
7%	9.31e-8	9.47e-4	1.68e-7	6.49e-4	9.14e-8	8.20e-4	9.82e-8	2.40e-4
10%	9.13e-8	9.82e-4	9.82e-8	7.23e-4	1.20e-7	9.65e-4	8.06e-8	5.75e-4

Table 10: Hybrid PBIL solutions with different tolerance sets using Taguchi methods

Tole-rance	C1 F	L2 H	C3 F	L4 H	C5 F	L6 H	C7 F	Worst fitness
1%	8.66e-8	8.66e-4	6.61e-8	8.06e-4	1.75e-7	1.26e-3	6.26e-8	1.51e-4
3%	9.47e-8	8.82e-4	1.38e-7	8.82e-4	1.60e-7	7.91e-4	9.65e-8	2.67e-5
5%	9.47e-8	1.38e-3	1.57e-7	1.13e-3	8.98e-8	7.91e-4	7.77e-8	3.64e-5
7%	8.82e-8	8.20e-4	6.98e-8	7.64e-4	1.63e-7	9.65e-4	9.47e-8	3.95e-4
10%	7.64e-8	8.06e-4	1.60e-7	8.06e-4	1.84e-7	9.65e-4	9.31e-8	9.30e-5

Table 11: Hybrid CHC solutions with different tolerance sets using Taguchi methods

This shows that the solutions obtained from both hybrid optimizers are robust to tolerance of components. Circuit response plots of the values obtained by both hybrid PBIL and hybrid CHC with Taguchi methods during the local stage can be found in Appendix E3.

Table 12 shows the average computing time to obtain a solution for the hybrid PBIL and hybrid CHC algorithms.

Tolerance	PBIL (minutes)	CHC (minutes)
1%	0.4	0.5
3%	1	1.7
5%	3.5	2.6
7%	9.3	4
10%	54	4.5

Table 12: Average computing time to obtain a solution for the hybrid PBIL and hybrid CHC algorithms using Taguchi methods

However, the hybrid PBIL algorithm still takes a lot of computing time to find robust solutions as compared to the CHC algorithm. So, Lamarckian learning can be applied to counter the Negative Baldwinian learning of the robust hybrid genetic algorithm in order to improve convergence in the hybrid genetic algorithms. This method does not work well on the hybrid CHC algorithm. It should be noted that this technique improves the convergence rate for the PBIL search, but deteriorates that of the CHC search.

This is because the CHC algorithm uses an elitist selection criterion, and as such, Lamarckian learning pushes the search direction too quickly into possibly unrobust solution space. In this case, the effect of Lamarckian learning overcompensate the negative Baldwinian effect. In the PBIL algorithm, the search direction is more adaptable to the solution presented by Lamarckian learning as it updates the probability vector only slightly. Hence, Lamarckian effect in the PBIL algorithm does not overcompensate the Negative Baldwinian effect.

Tables 13 and 14 show hybrid PBIL solutions with different tolerance sets using Taguchi methods and Lamarckian learning during the local stage and the computational time for the hybrid PBIL solutions respectively.

Tole-rance	C1 F	L2 H	C3 F	L4 H	C5 F	L6 H	C7 F	Worst fitness
1%	8.51e-8	7.23e-4	2.29e-7	5.42e-4	1.00e-7	7.10e-4	7.37e-8	3.27e-4
3%	4.07e-8	1.65e-3	1.07e-7	8.35e-4	9.47e-8	1.13e-3	9.14e-8	1.68e-4
5%	9.31e-8	9.47e-4	1.81e-7	7.50e-4	1.31e-7	6.61e-4	5.94e-8	2.31e-4
7%	8.35e-8	8.06e-4	9.31e-8	7.77e-4	1.24e-7	8.98e-4	9.14e-8	5.39e-4
10%	9.82e-8	1.13e-3	1.65e-7	9.31e-4	9.82e-8	9.82e-4	6.85e-8	1.32e-4

Table 13: Hybrid PBIL solutions with different tolerance sets using Taguchi methods and Lamarckian learning during the local stage.

Circuit response plots of the values obtained by the hybrid PBIL with Taguchi and Lamarckian learning during the local stage can be found in Appendix E4.

Tolerance	PBIL (minutes)
1%	0.4
3%	1.6
5%	2.5
7%	7.9
10%	22

Table 14: Average computing time to obtain a solution for the hybrid PBIL algorithm.

This clearly shows that when Lamarckian learning is included during the local search, the computing time for the hybrid PBIL algorithm is better when compared to the one with only Taguchi method.

9.3 Robust design of the 8th order normalized Butterworth low pass filter

This problem uses the same optimization technique as in section 9.2. However, values are chosen on a continuous scale over two decade range with a template specified by a 0.5 dB passband ripple with a passband edge at 1 rad/sec, and a stopband attention of -160 dB at a stopband edge of 10 rad/sec. A linear grid of twenty five frequencies is chosen in the passband together with twenty five frequencies in the stopband. A set of component values for the 8th order butterworth low pass filter (see Figure 22) obtained from both the PBIL and CHC algorithms are tested for tolerance of 1,3,5,7 and 10% of component values. It should be noted that the 8th order Butterworth filter has more than one set of component solution. This would be difficult for the evolutionary search to find a robust set of component solution.

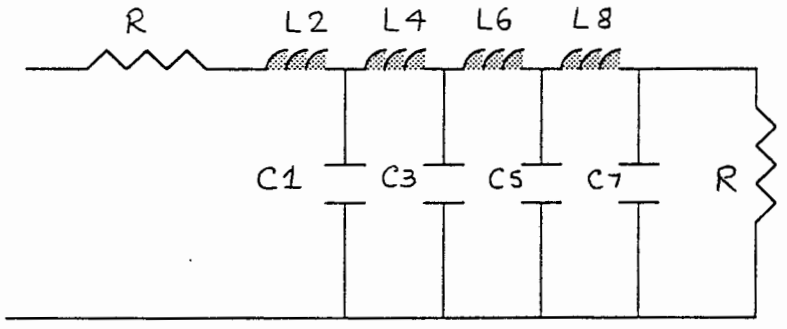


Figure 22: 8th order Butterworth low pass filter

The PBIL set of component values is :

C1= 1.83F; L2= 3.40e-1H; C3= 1.33e-1F; L4=7.10e-1H; C5= 6.61e-1F; L6= 5.73e-1H; C7= 2.15e-1F; L8= 3.43e-1H

The CHC set of component values is :

C1= 1.27F; L2= 6.26e-1H; C3= 7.43e-1F; L4= 1.14e-1H; C5= 4.74e-1F; L6= 8.66e-1H; C7= 6.79e-1F; L8= 4.66e-1H

Table 15 shows the worst perturbed fitness for the PBIL and CHC solutions over the range of tolerance sets. (For definition of fitness function used in this thesis, refer to section 2.3)

Tolerance	PBIL fitness	CHC fitness
0%	5.05e-7	4.71e-8
1%	9.9476e10	9.9493e10
3%	9.9481e10	9.9498e10
5%	9.9486e10	9.9503e10
7%	9.9491e10	9.9507e10
10%	9.9499e10	9.9514e10

Table 15: Worst perturbed fitness for the PBIL and CHC solutions over the range of tolerance sets.

This shows that the solutions obtained from both optimizers are unrobust to tolerance of components. Circuit response plots of the values obtained by both PBIL and CHC can be found in Appendix E5.

The hybrid PBIL and hybrid CHC algorithms are evolved over the range of tolerance using the best setting obtained from the previous section (hybrid PBIL with Taguchi + Lamarckian, hybrid CHC with Taguchi only). Tables 16 and 17 show the solutions obtained from the hybrid PBIL and CHC algorithms over the range of tolerance.

Tole-rance	C1 F	L2 H	C3 F	L4 H	C5 F	L6 H	C7 F	L8 H	Worst fitness
1%	7.70e-1	3.61e-1	2.27e-1	3.75e-1	6.67e-1	2.69e-1	2.19e-1	6.32e-1	6.09e-6
3%	6.55e-1	1.27e-1	6.49e-1	4.18e-1	1.45e-1	3.68e-1	2.46e-1	4.41e-1	5.42e-5
5%	7.30e-1	1.96e-1	3.34e-1	1.93e-1	2.59e-1	5.28e-1	1.89e-1	2.25e-1	8.48e-4
7%	9.47e-1	1.65e-1	2.57e-1	2.44e-1	2.09e-1	4.87e-1	1.24e-1	1.74e-1	8.54e-3

Table 16: Hybrid PBIL solutions with different tolerance sets using Taguchi methods and Lamarckian learning during the local search stage.

Tole-rance	C1 F	L2 H	C3 F	L4 H	C5 F	L6 H	C7 F	L8 H	Worst fitness
1%	7.99e-1	5.62e-1	5.67e-1	4.66e-1	6.92e-1	2.33e-1	2.41e-1	8.35e-1	1.96e-7
3%	5.88e-1	1.58e-1	2.79e-1	2.59e-1	5.28e-1	3.55e-1	2.43e-1	6.04e-1	2.50e-5
5%	5.05e-1	2.21e-1	4.91e-1	1.88e-1	2.17e-1	2.71e-1	1.32e-1	4.07e-1	2.15e-2
7%	3.46e-1	1.18e-1	4.10e-1	2.01e-1	3.92e-1	2.25e-1	1.08e-1	4.66e-1	6.19e-1

Table 17: Hybrid CHC solutions with different tolerance sets using Taguchi methods only during the local search stage.

Both algorithms could not find an appropriate solution set for the 7% tolerance values. As a result, the 10% tolerance values were not included in the tables.

Table 18 shows the average computing time to obtain a solution for the hybrid PBIL and hybrid CHC algorithms.

Tolerance	PBIL (minutes)	CHC (minutes)
1%	0.5	0.3
3%	2.7	27
5%	130	234
7%	210	270

Table 18: Computing time for hybrid PBIL and hybrid CHC algorithms.

Circuit response plots of the values obtained by both hybrid PBIL and hybrid CHC can be found in Appendices E6 and E7 respectively.

Chapter 10

Conclusions

As a result of the findings of the investigation done in this report, the following conclusions may be drawn:

10.1 Circuit optimization techniques using evolutionary methods

Both evolutionary optimizers, namely, PBIL and CHC, work very well on the circuit optimization problems.

10.2 Robust circuit design by evolutionary methods

The method of evolving robust circuit using evolutionary methods (choosing preferred values) is better than conventional circuit design where usually nearest preferred values are used in circuits.

10.3 Robust hybrid genetic algorithm with full factorial perturbation of component values in the local search stage

This new robust hybrid genetic algorithm works well to find sets of component values, for a circuit, robust to a particular tolerance. However, this method is computer intensive.

10.4 Robust hybrid genetic algorithm with Taguchi techniques in the local stage

The robust hybrid genetic algorithm with Taguchi techniques performs well in finding sets of component values insensitive to a particular tolerance. This method shows that the Taguchi techniques can be used to find the sensitivity of the neighbourhood of a particular solution using a few carefully selected experiments. This helps to decrease the computing time to find a robust set of component values. In terms of computing time, the hybrid CHC algorithm works faster than the hybrid PBIL algorithm.

10.5 Robust hybrid genetic algorithm with Taguchi techniques and Lamarckian learning in the local stage

As the neighbourhood is being investigated by the Taguchi techniques, the latter is able to predict the best and worst points in that region. The best solution, during learning, is used to guide the evolutionary search to a better fitness space - this is known as Lamarckian learning. The effect of the Lamarckian learning is to improve convergence, hence decrease the computing time to find a robust set of component values. This method improves the computing time of the hybrid PBIL algorithm but deteriorates that of the hybrid CHC algorithm.

List of References

[1] Baluja, S. 'Population-Based Incremental Learning: A method for integrating Genetic Search Based Function Optimisation and competitive learning'. Carnegie Mellon University, June 2, 1994.

[2] Baluja, S. 'An Empirical Comparison of Seven Iterative and Evolutionary Function Optimisation Heuristics'. Carnegie Mellon University, Sept. 1, 1995.

[3] Eshelman, L.J. 'The CHC Adaptive Search Algorithm: How to have safe search when engaging in nontraditional genetic recombination'. Phillips Laboratories, North American Phillips Corporation, Briarcliff manor, New York.

[4] Huelsman, L.P. 'Optimization - A Powerful Tool for Analysis and Design'. *IEEE Transactions on Circuits and Systems* - I, Vol. 40, No. 7, July 1993.

[5] Horrocks, D.H.; Spittle, M.C. 'Component Value Selection for Active Filters using Genetic Algorithms'. *Proc. of IEEE Workshop on Natural Algorithms in Signal Processing*, Chelmsford UK, 14-16 November, Vol. 1, pp 13/1 - 13/6.

[6] Horrocks, D.H.; Khalifa, Y.M.A.: 'Genetically derived filters using preferred value components'. *Proc. of IEEE colloq. on Linear Analogue Circuits and systems*, Oxford UK, Oct. 1994.

[7] Horrocks, D.H.; Khalifa, Y.M.A. 'Genetic Algorithm design of electronic analogue circuits including parasitic effects'.

[8] Bendell, A.; Disney, J.; Pridmore, W.A. *Taguchi Methods.* IFS Publications.

[9] Dehnad, K. *Quality Control, Robust Design and the Taguchi Method.* Wadsworth & Brooks/Cole Statistics/Probability Series.

[10] Logothetis, N.; Wynn, H.P. *Quality Through Design.* Oxford Science Edition.

[11] Whitley, D.; Gordon, V.S.; Mathias, K.E. 'Lamarckian Evolution, The Baldwin Effect and Function Optimization'. *International Conference on Evolutionary Computation*, 1994.

[12] Whitley, D. 'Modelling Hybrid Genetic Algorithms'. Computer Science Department, Colorado State University.

Appendices

Appendix A: PBIL algorithm

```
?BIL

A simple PBIL search function.  The function receives the following
parameters :
      b -> any random vector in the search space (used to determine
           the length of the vector and possibly to allow for variations
           of the PBIL function for different 'starting points' or PV bias.
      Fmax -> Best fitness to be achieved
      POPSIZE-> population size.
      func-> a string which contains the function name
The following values are returned:
      F       : the fitness of the best solution
      Xreal   : the real values of X - the best solution
      v       : the best solution as a bitstring
      N       : the number of function evaluations used
      fits    : the best fitnesses of each population during the search

The search function evaluates the population as a matrix instead of
evaluating one member at a time. This helps to decrease the computing
time of PBIL.


nction [F,Xreal,v,N,fits] = fpbilb4(b,Fmax,POPSIZE,func)

POPSIZE;              % Population size
= .1;                 % Learning rate
 = .005;              % Forgetting factor
rk = -1;              % mark = 1 to maximize, -1 to minimize
XEVAL = 15000;        % Maximum number of evaluations
unt = 0;              % Initialising number of evaluations
ax = mark*Fmax;       % Setting the fitness to be reached
= length(b);          % Length of the population
 = .5*ones(M,n);      % Initialising PV for population
stever = -inf;
ts = [];

ile (bestever<Fmax)&(count<MAXEVAL)
 best = -inf;
      P = rand(M,n)<PV;                          % Generating pop
      [fitness, X] = eval([func '(P)']);         % Eval pop
      fitness = mark*fitness;
      count = count + M;                         % Update counter
      P=[P fitness];                             % Append fitness to pop
      [best,indx] = max(P(:,n+1));               % Find best of current pop
      B= P(indx,1:n);                            % Find best bit string from pop
         if best>bestever                        % Updating the best ever sol
            bestever = best;                     % obtained by search
            bestX=X(indx,:);
            bestB=P(indx,1:n);
         end

      fits = [fits best];
      PV = (1-L)*PV + L*(B'*ones(1,M))';         % Biasing PV with best bitstring
      PV = PV - FF*(PV - .5);                    % of pop.

      disp(['no eval best'])
      disp([count/1000 best])
d

= mark*bestever;
eal = bestX;
= bestB;
= count;
ts;
```

Appendix B: CHC algorithm

```
:HC

:his function impliments the CHC algorithm.  The function
:eceives the following parameters:
      b       : a sample solution vector in the search space
      Fmax    : the value of the global peak
      POPSIZE : the population size M
      func    : the fitness function to be evaluated
:he function returns the following values:
      bestever: the best fitness found
      bestX   : the best solution as a vector of reals
      S       : the best solution as a bitstring
      evals   : the number of fitness evaluations used.
      fits    : the fitnesses after each population during the search
      evs     : the number of evaluations performed for fits
      numstart: the number of restarts


ıction [bestever,bestX,S,evals,fits,evs,numstart] = fchc3(b,Fmax,POPSIZE,fun

:k = -1;                              % mark = 1 to maximize, -1 to minimize
=  POPSIZE;                           % population size
=  length(b);                         % string length
=  L/4;                               % Initial Hamming-distance threshold
 = 0.35;                              % divergence rate
ınt = 0;                              % Number of function evaluations
XEVAL = 15000;                        % Max number of evaluations
ax = mark*Fmax;                       % F is -ve when minimising
(rand(M,L)>0.5);                      % Create a population
,X]  =  eval([func '(P)']);           % call func to return col vect F
= mark*F;
ınt = count + M;                      % update counter
=  [P F];                             % append fitnesses as L+1th column
est,indx] = max(P(:,L+1));            % Find best of 1st population
stever=best;
ts=[];
s=[];
estart=0;
stX=0;

ile (bestever<Fmax)&(count<MAXEVAL)
 C = selectr(P);
 Cp = recomb(C,d);
 [Cprow, Cpcol] = size(Cp);
 if Cprow>0                                    % if Cp is not empty
      [F,X] = eval([func '(Cp(:,1:L))']);      % evaluate Cp
      F = mark*F;
      count = count + Cprow;                   % update counter
      Cp = [Cp(:,1:L) F];
      [best,indx]=max(Cp(:,L+1));
         if best>bestever                      % Find best fitness in child pop
            bestever=best;
            bestX=X(indx,:);
            S=Cp(indx,1:L);
         end
 end
 Pnew = selects(P,Cp);
 if all(Pnew ==P)
    d = d-1;
 end
 if d<0
    Pnew = diverge(P,r);
    nrestart=nrestart+1;                       % Update counter for no. restart
    d = r*(1-r)*L;
```

```
    [F,X] = eval([func '(Pnew(:,1:L))']);    % evaluate Pnew only if diverge
                                             %  is used
    F = mark*F;
    count = count + M-1;
    Pnew = [Pnew(:,1:L) F];                  % append the fitness values
    [best,indx] = max(Pnew(:,L+1));          % Find best fitness in div pop
    if best>bestever
       bestever = best;
       bestX = X(indx,:);
       S = Pnew(indx,1:L);
    end
 end

 P=Pnew;                                     % P(t) becomes P(t-1)
 fits=[fits bestever];
 evs=[evs count];

 disp(['no eval best'])
 disp([count/1000 bestever])
R

stever = mark*bestever;
stX;

als = count;
:s;
s;
nstart=nrestart;
```

```
ecombine

ction Cprime = recombine(C, d)

n] = size(C);
 n-1;
p = [];

 i = 1:2:m-1
mask = xor(C(i,1:L),C(i+1,1:L));      % mask tags the differing bits
half = round(sum(mask)/2);            % hamm_dist/2
if half>d
      index=find(mask==1);
     rndx = randperm(sum(mask));
     index=index(rndx(1:half));
     C(i,index) = ~C(i,index);        % flip half the differing bits
     C(i+1,index) = ~C(i+1,index);    % which were chosen at random
     temp = [temp; C(i:i+1,1:L)];     % append to temp
  end


ime = temp;
```

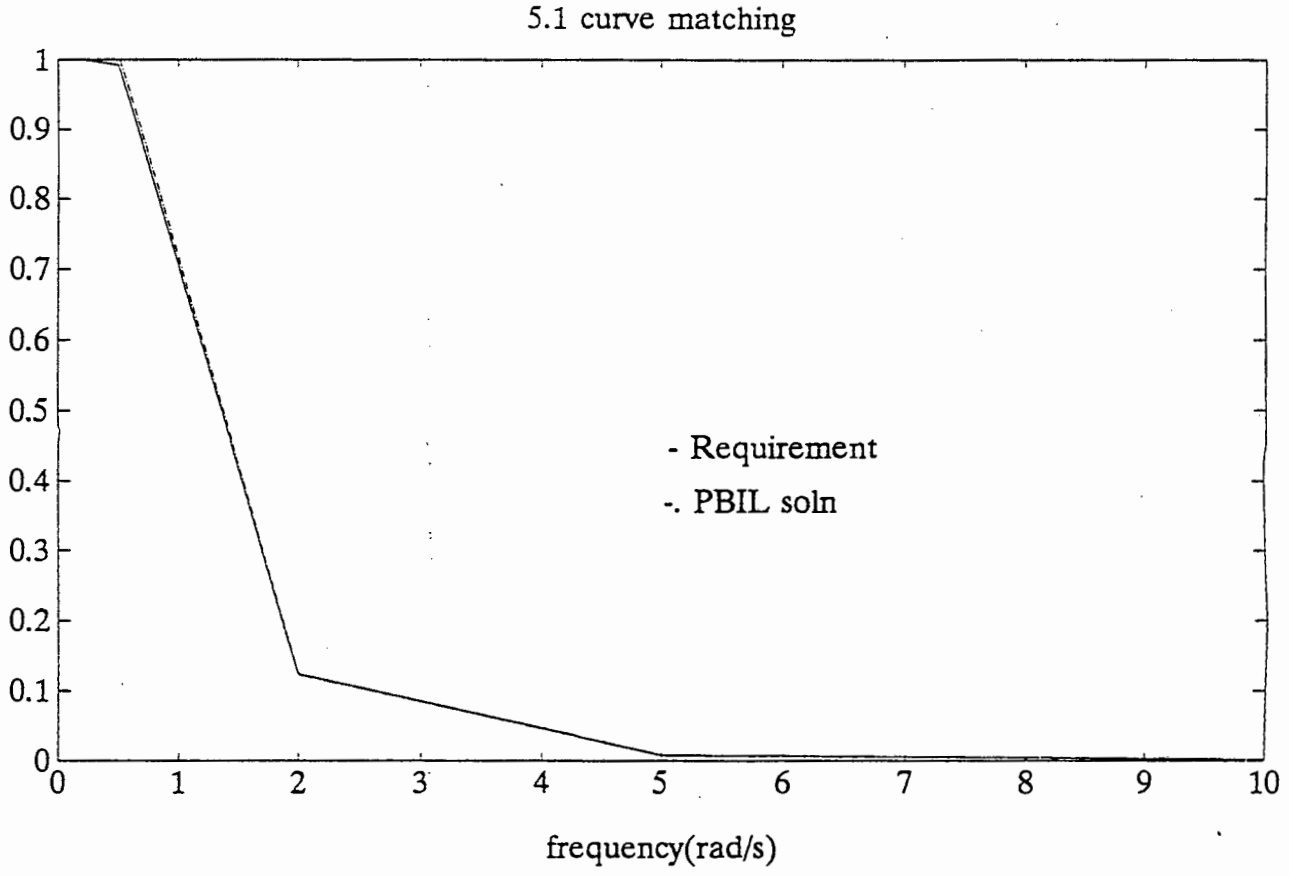
```
selectr

nction C = selectr(P)

 n] = size(P);

dex = randperm(m);     % copies all members of P to C in random order
 P(index,1:n);
```

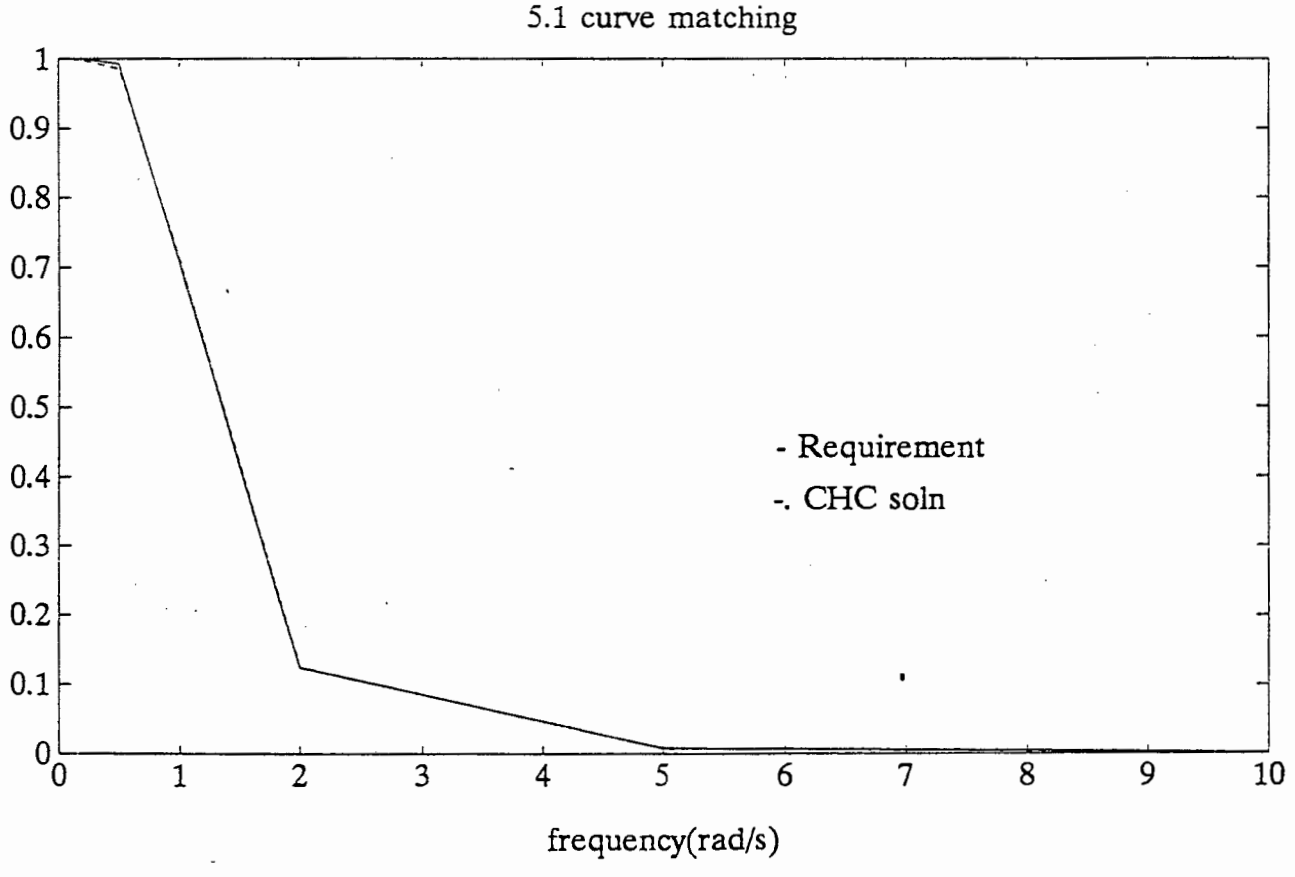
```
elects

ction Pnew = selects(P,Cprime)

n,Pn] = size(P);
n,Cn] = size(Cprime);
 = [];

 Cprime == []
 Pnew = P;
se
 Q = [Cprime;P];%Putting Cprime before P in Q allows parents to dominate
child if fitness are equal in selection procedure.
 [Qsort,Qindex] = sort(Q(:,Pn));      % sort Q in ascending order
 Q = flipud(Q(Qindex,:));             % then descending orde.
 Pnew = Q(1:Pm,:);                    % select the best M solutions
d
```

```
ıction Pnew = diverge(P,r)

:HC procedure to re-seed a stagnated population
:nserts best individual and copies with 35% of the bits mutated

  n] =size(P);
= n-1;
max, ndx] = max(P(:,n));        % find the max in P and
st = P(ndx,1:L);                % save as the best

ew = ones(m,1) * best;          % population filled with 'best'

= rand(m-1,L) < r;              % random mask with 35% ones

ew(2:m,:) = xor(M,Pnew(2:m,:));
(leaves the best solution intact and mutates 35% of the remaining bits)
```

Appendix C1: Optimization using the technique of curve matching

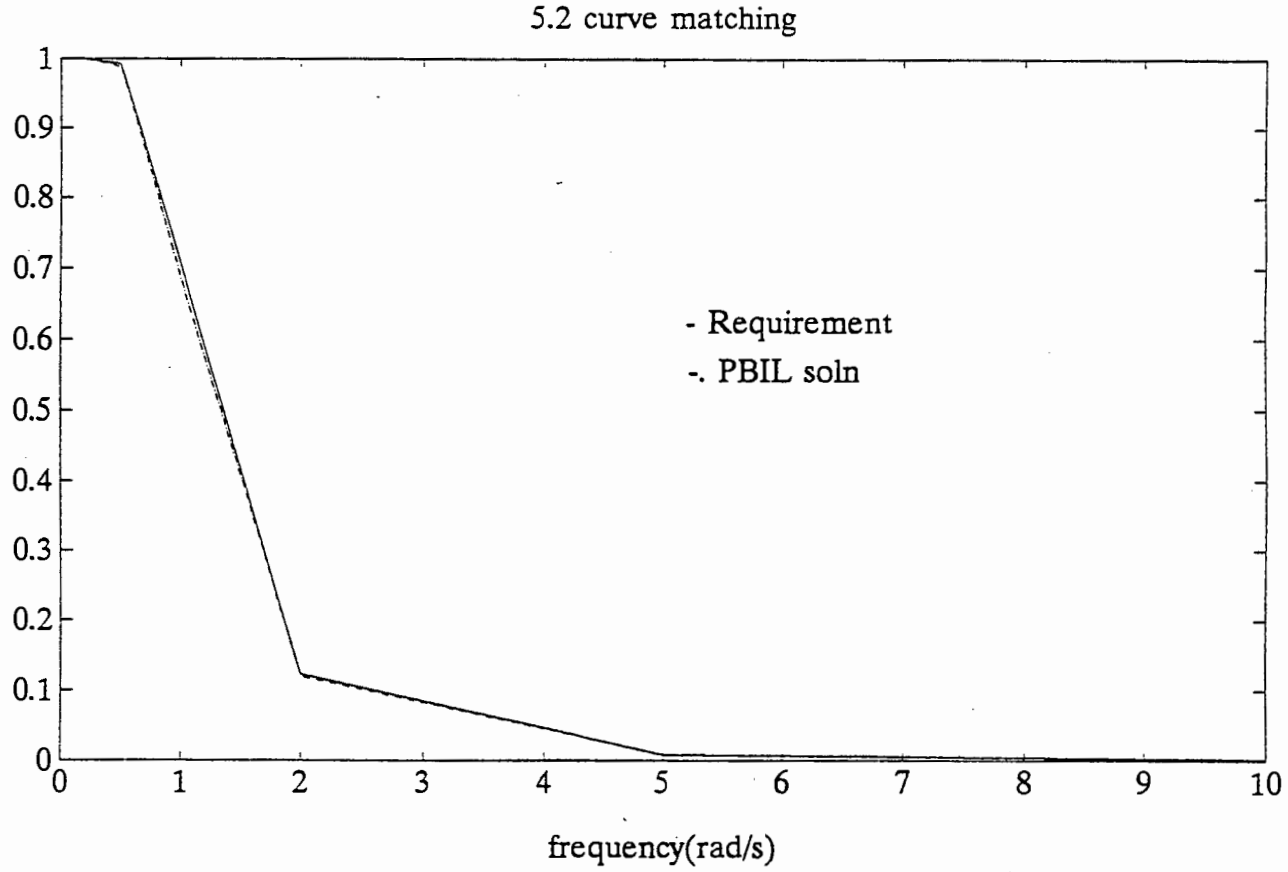
```
 This program uses the technique of curve matching.

nction [f,X]=curvm1(B)

 n]=size(B);
=n/3;
.=10*btod(B(:,1:L))/((2^L)-1);      % finding component value for x1
=10*btod(B(:,L+1:2*L))/((2^L)-1); % finding component value for x2
=10*btod(B(:,2*L+1:n))/((2^L)-1); % finding component value for x3

=[x1 x2 x3];

=[0.1 0.2 0.5 1 2 5 10]; % frequency
=[0.99999 0.99996 0.9922 0.707 0.124 0.008 0.001]; % requirement
=(r'*ones(1,m))';          % Resizing matrix r
=length(w);
=(w'*ones(1,m))';          % Resizing matrix w
=x1 .*x2;
=(z1*ones(1,l1));          % Resizing matrix z1
=x1 .*x2 .*x3;
=(z2*ones(1,l1));          % Resizing matrix z2
=x1+x3;
=(z3*ones(1,l1));          % Resizing matrix z3

 Voltage transfer function for particular example

=abs(1 ./(1 - ((w .^2) .*(z1)) + ((i*w) .*(z3 - ((w .^2) .*(z2))))));

=sum((r-T)' .^2)';    % fitness of component values generated.
```

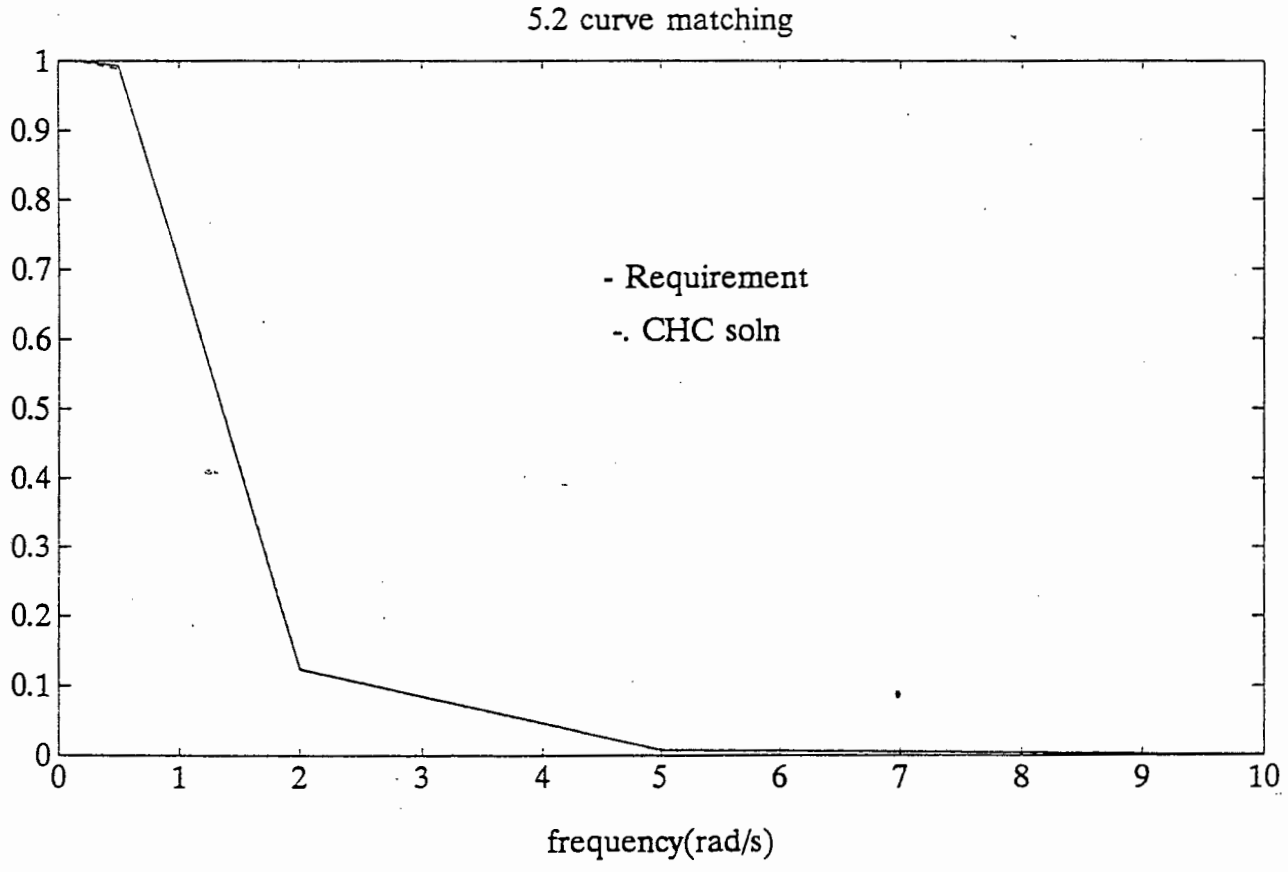

5.1 curve matching
- Requirement
-. PBIL soln
frequency(rad/s)


5.1 curve matching
- Requirement
-. CHC soln
frequency(rad/s)

Appendix C2: Optimization using the technique of nonlinear equation solving

```
This program uses the technique of nonlinear equation solving.

unction [f,X]=curvm2(B)

n n]=size(B);
=n/3;
1=10*btod(B(:,1:L))/((2^L)-1);      % finding component value for x1
2=10*btod(B(:,L+1:2*L))/((2^L)-1); % finding component value for x2
3=10*btod(B(:,2*L+1:n))/((2^L)-1); % finding component value for x3

=[x1 x2 x3];

=[1 2 2];                % Requirement
=(r'*ones(1,m))';        % Resizing matrix r


Set of nonlinear equations

1=1 ./(x1 .*x2 .*x3);
2=(x1+x3) ./(x1 .*x2 .*x3);
3=1 ./x3;

=[g1 g2 g3];

=sum((r-G)' .^2)';     % fitness of component values generated.
```

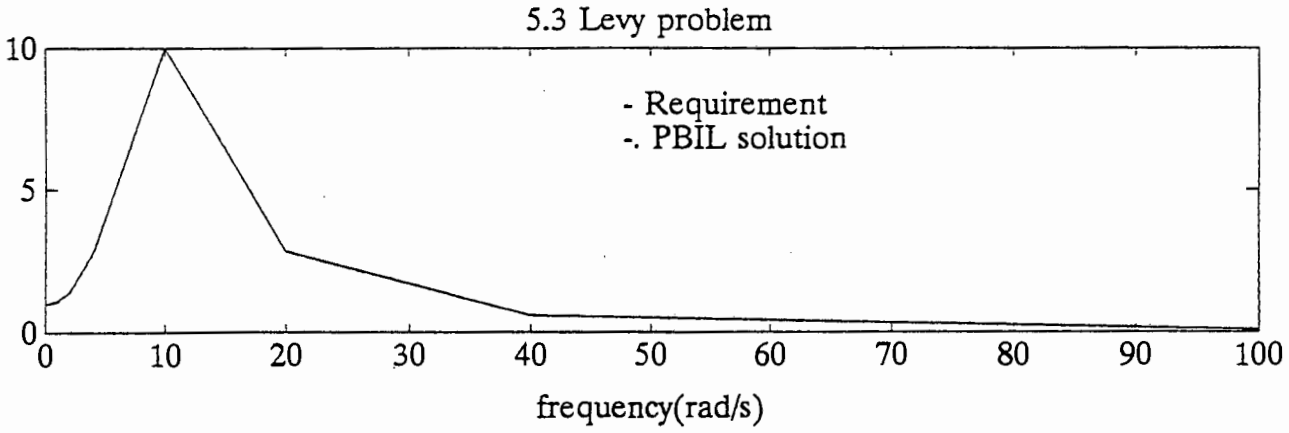

5.2 curve matching
- Requirement
-. PBIL soln
frequency(rad/s)

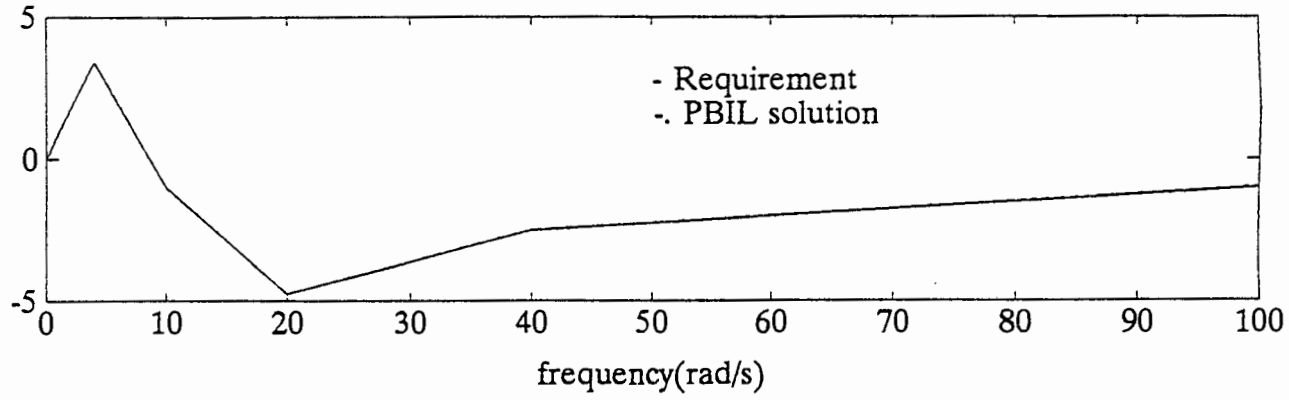

5.2 curve matching
- Requirement
-. CHC soln
frequency(rad/s)

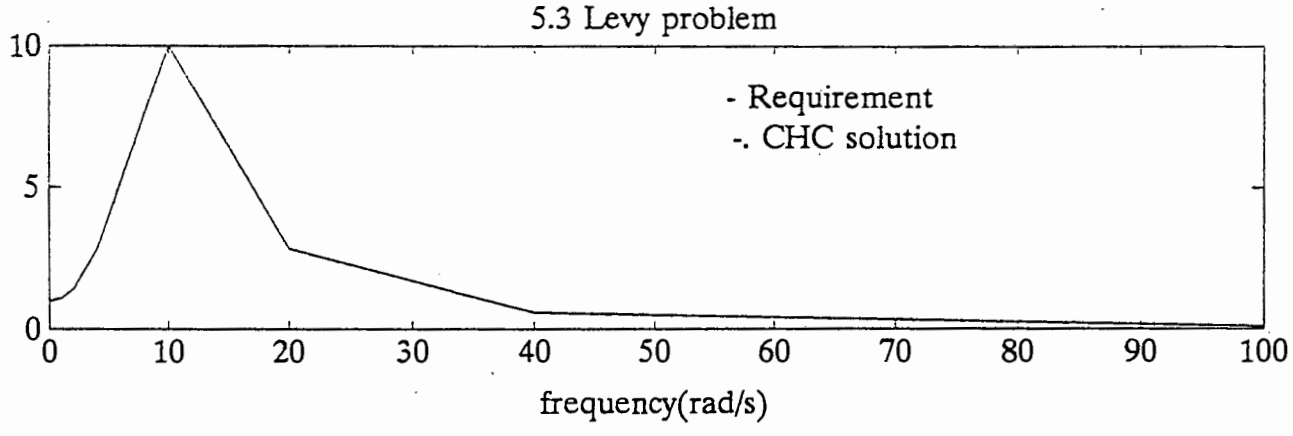
Appendix C3: Optimization using the technique of curve matching of the real and imaginary part of a second order rational polynomial function

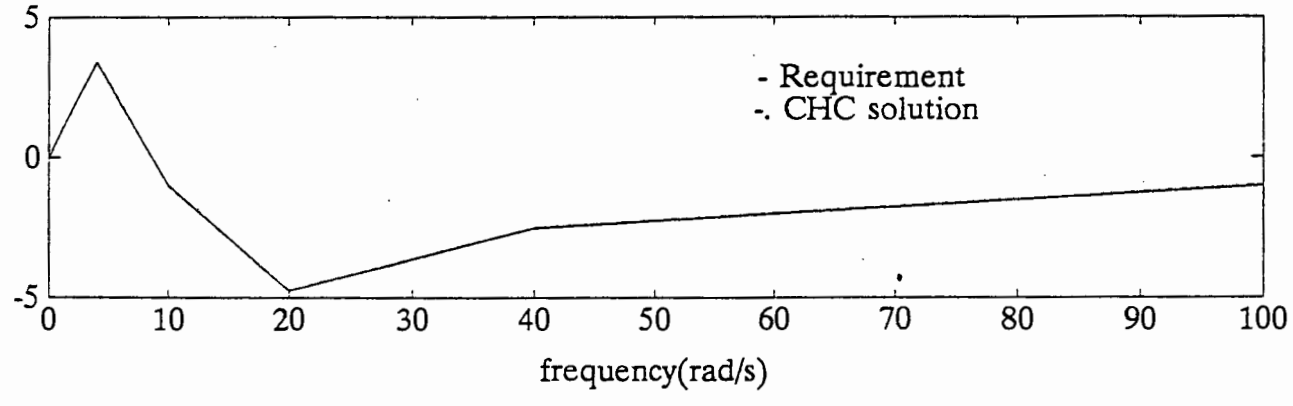
This program is from rational polynomial LSE approximation of complex functions. It uses the Levy problem for curve matching of the real and imaginary part on a general 2nd order rational polynomial function.

```
nction [f,X]=cmlev(B)

 n]=size(B);
n/5;

:[0 0.1 0.2 0.5 0.7 1 2 4 7 10 20 40 70 100];
:((1+i*w) ./(1+(i*w) ./10+((i*w) ./10) .^2));  % Levy's function
'=real(F);                                     % Real part of levy's func
-F=(rF'*ones(1,m))';
'=imag(F);                                  % Imaginary part of levy's func
-F=(iF'*ones(1,m))';

-=btod(B(:,1:L))/((2^L)-1);        % Finding 1st coefficient
.=btod(B(:,L+1:2*L))/((2^L)-1);    % Finding 2nd coefficient
:=btod(B(:,2*L+1:3*L))/((2^L)-1); % Finding 3rd coefficient
.=btod(B(:,3*L+1:4*L))/((2^L)-1); % Finding 4th coefficient
:=btod(B(:,4*L+1:n))/((2^L)-1);    % Finding 5th coefficient

=[a0 a1 a2 b1 b2];

=((a0*ones(1,14)+a1*(i*w)+a2*((i*w) .^2)) ./(ones(m,14)+b1*(i*w)+ ...
)2*((i*w) .^2)));  % General rational polynomial fitting Levy's function.
=real(Z);
=imag(Z);

=(sum((reF'-rZ') .^2)+sum((ieF'-iZ') .^2))';   % fitness function
```


5.3 Levy problem
- Requirement
-. PBIL solution
frequency(rad/s)


- Requirement
-. PBIL solution
frequency(rad/s)

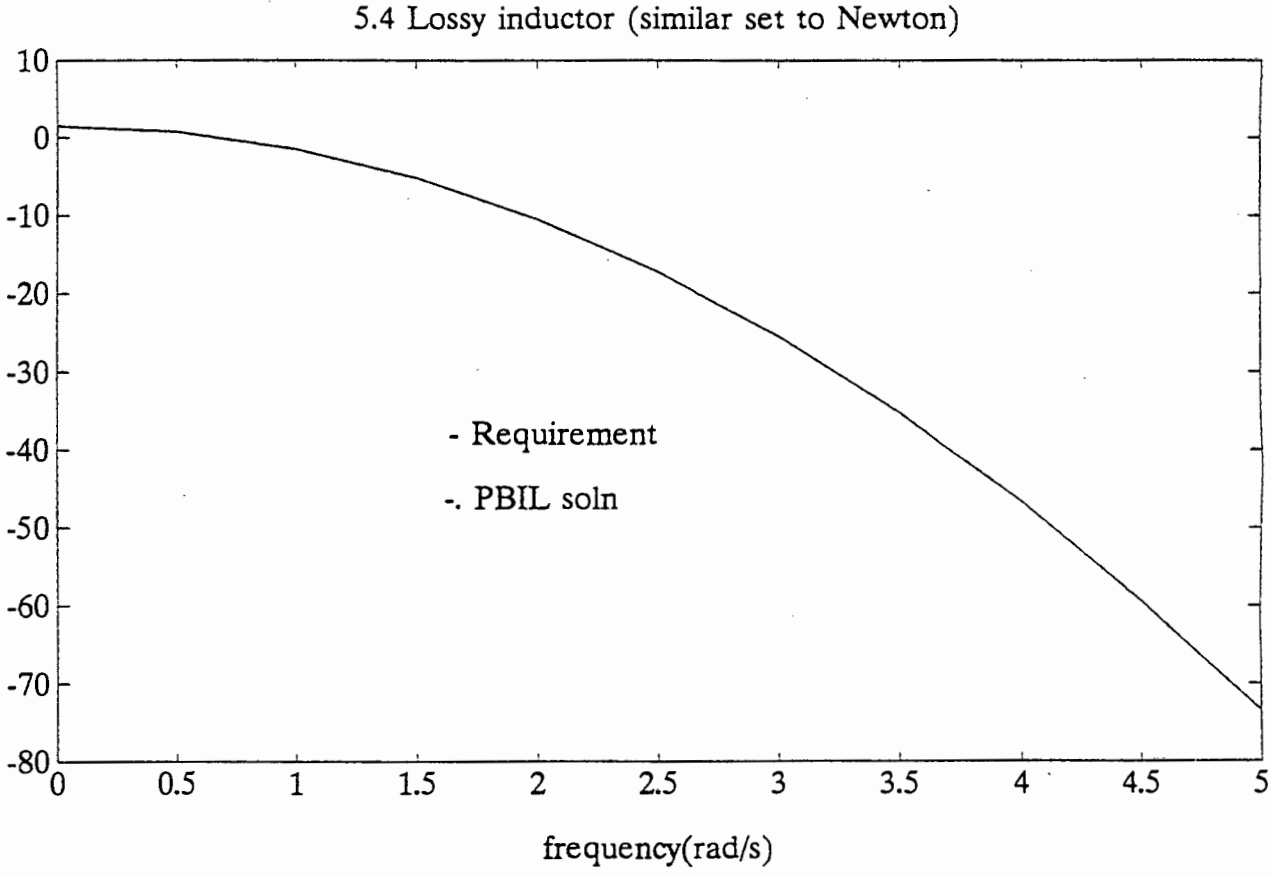

5.3 Levy problem
- Requirement
-. CHC solution
frequency(rad/s)

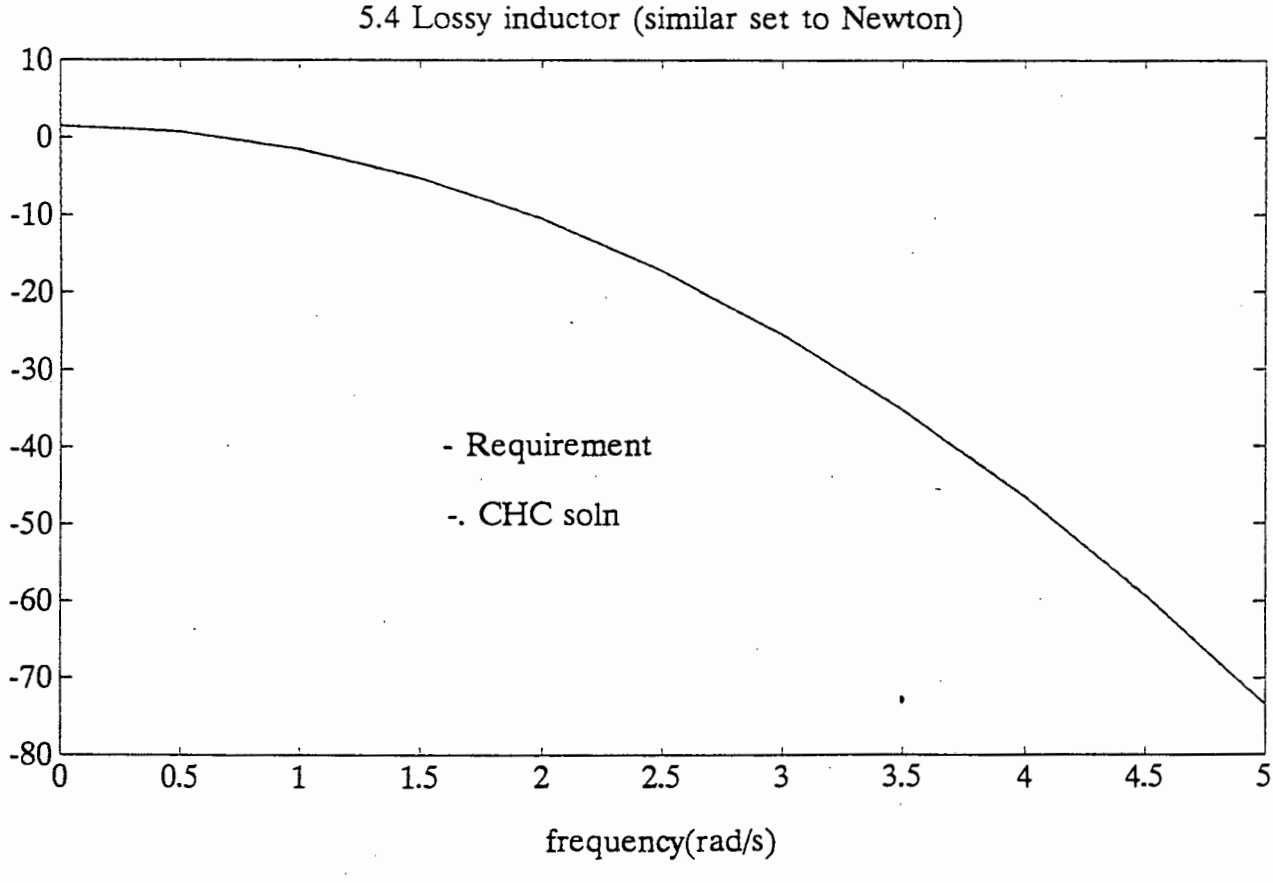

- Requirement
-. CHC solution
frequency(rad/s)

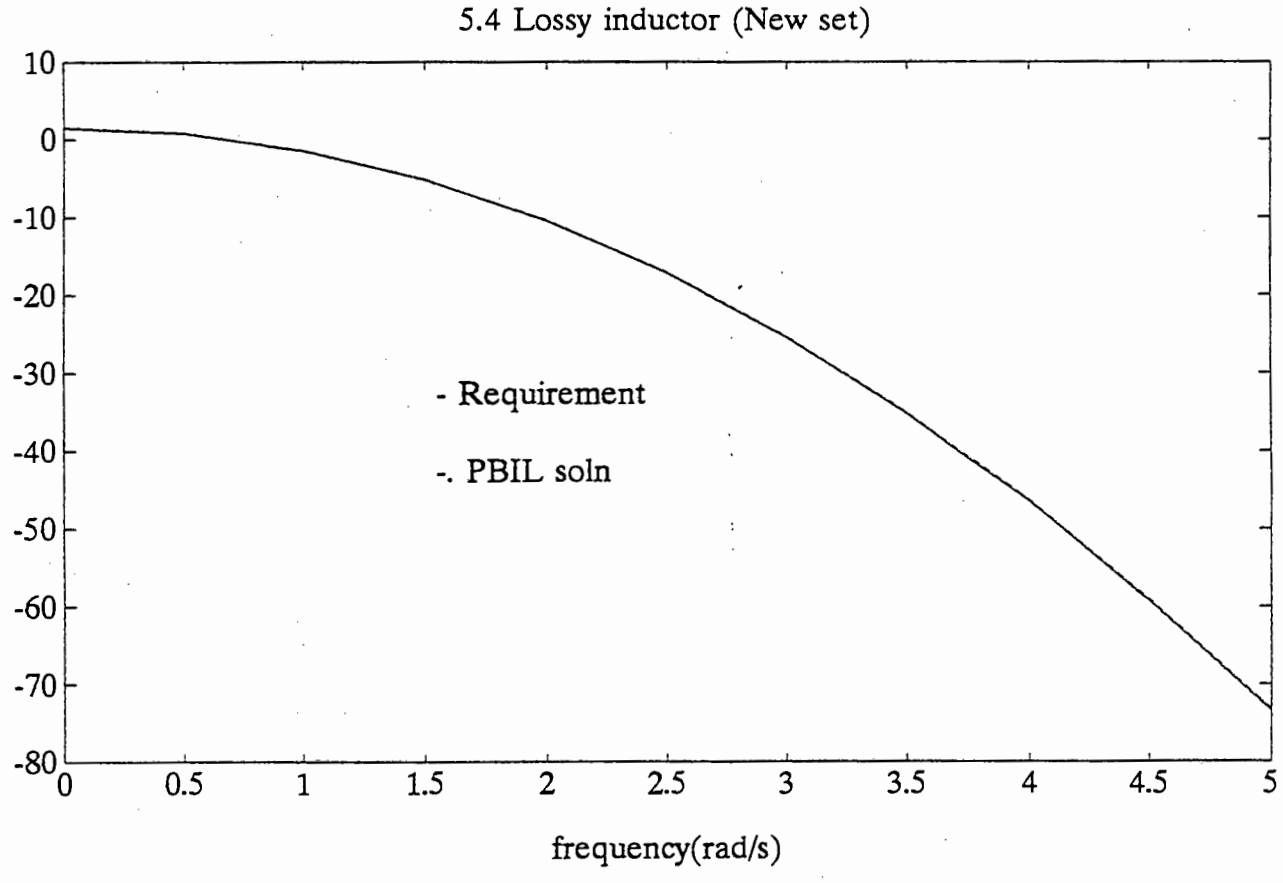
Appendix C4: Optimization of an electronic circuit containing a lossy inductor

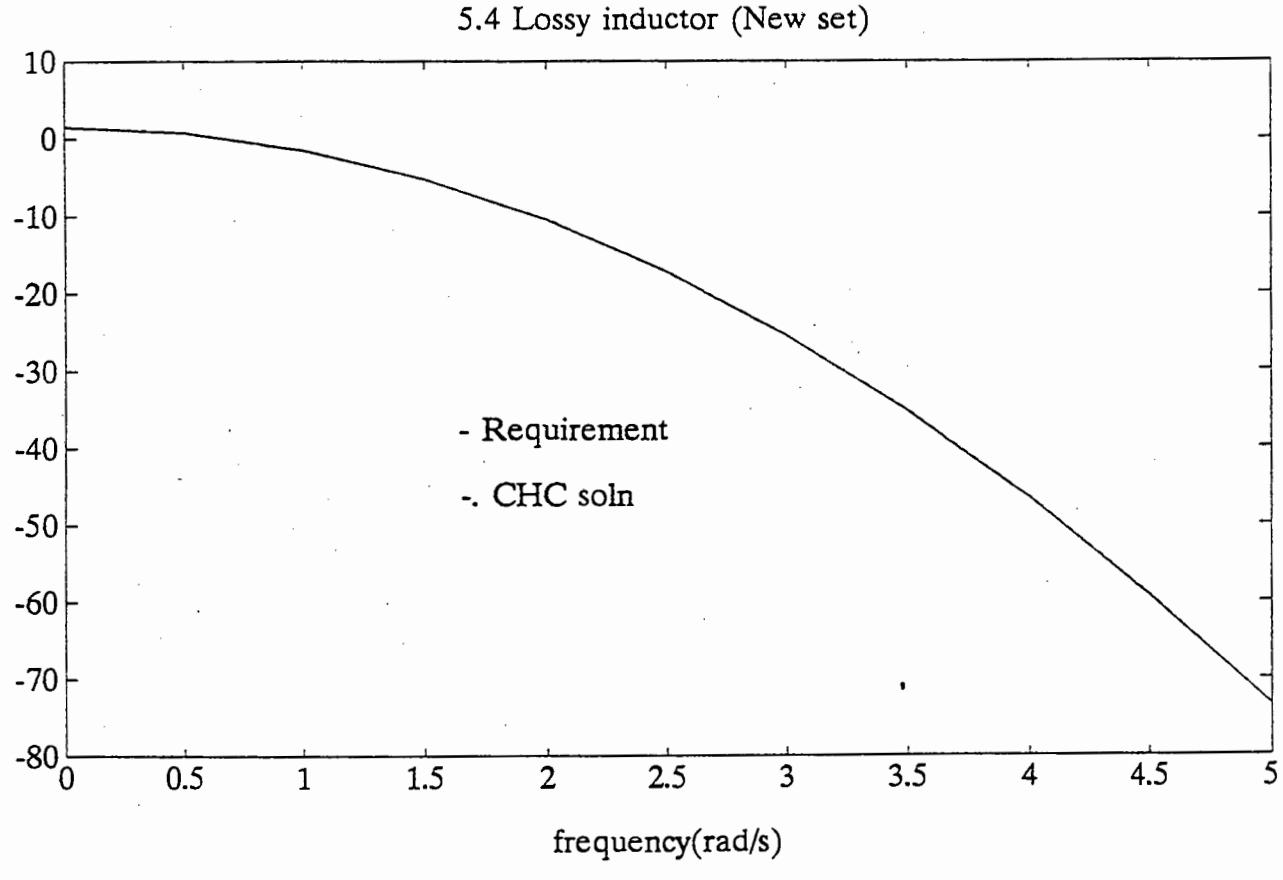
This program simulates a low pass filter with a lossy inductor. This problem is difficult to solve as it has to take into account the loss in the inductor which is prescribed as a relation of r=L/10. So, to solve this problem in the usual way, Newton's technique has to be applied. This requires a lot of mathematical computation by hand, and some approximation.

```
unction [f,X]=loind(B)

m n]=size(B);
=n/4;

.=1;
.=10*btod(B(:,1:L))/((2^L)-1);          % Finding component value L1
.=10*btod(B(:,L+1:2*L))/((2^L)-1);      % Finding component value C1
=10*btod(B(:,2*L+1:3*L))/((2^L)-1);     % Finding component value C2
=10*btod(B(:,3*L+1:n))/((2^L)-1);       % Finding component value R2
=L1/10; %Loss in inductor

=[1.5 3 3 1.5];   % Requirements for low pass filter with lossy inductor

=(R'*ones(1,m))'; % Resizing matrix R


 Coefficients of circuit

.=R1 .*L1 .*C1 .*C2;
=(L1 .*((R1 ./R2) .*C1+C2))+(r .*R1 .*C1 .*C2);
=((L1 ./R2)+(R1 .*(C1+C2))+(r .*(C2+(R1 ./R2) .*C1)));
=1+(R1 ./R2)+(r ./R2);
=[g1 g2 g3 g4];

=[L1 C1 C2 R2 G];

=((G(:,1:4)-R(:,1:4)) .^2)';   % fitness function
=sum(f)';
```


5.4 Lossy inductor (similar set to Newton)
10
0
-10
-20
-30
-40
-50
-60
-70
-80
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
- Requirement
-. PBIL soln
frequency(rad/s)


5.4 Lossy inductor (similar set to Newton)
10
0
-10
-20
-30
-40
-50
-60
-70
-80
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
- Requirement
-. CHC soln
frequency(rad/s)


5.4 Lossy inductor (New set)
10
0
-10
-20
-30
-40
-50
-60
-70
-80
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
- Requirement
-. PBIL soln
frequency(rad/s)


5.4 Lossy inductor (New set)
10
0
-10
-20
-30
-40
-50
-60
-70
-80
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
- Requirement
-. CHC soln
frequency(rad/s)

```
Component value selection for active filters, using the E12 series. The
problem is to find suitable component values for 6 resistors and 2
capacitors on a state variable filter with low-pass frequency response. This
problem is from the paper ' Component value selection for Active filters
using Genetic Algorithms' by D.H. Horrocks & M.C. Spittle.

nction [f,X]=cvs(B)

 = 1591.55;        % cut-off frequency
   = 1.41421;      % selectivity factor

 = [10 12 15 18 22 27 33 39 47 56 68 82];  % E12 series
.= [2 3 4 5];       % Decade range for resistors
= [-10 -9 -8 -7];  % Decade range for capacitors

 n]=size(B);
=n/8;

.=btod(B(:,3:1));
=btod(B(:,1+3:2*1));
=btod(B(:,2*1+3:3*1));
=btod(B(:,3*1+3:4*1));
=btod(B(:,4*1+3:5*1));
=btod(B(:,5*1+3:6*1));
=btod(B(:,6*1+3:7*1));
=btod(B(:,7*1+3:n));

.A=btod(B(:,1:2));
A=btod(B(:,1+1:1+2));
A=btod(B(:,2*1+1:2*1+2));
A=btod(B(:,3*1+1:3*1+2));
A=btod(B(:,4*1+1:4*1+2));
A=btod(B(:,5*1+1:5*1+2));
A=btod(B(:,6*1+1:6*1+2));
A=btod(B(:,7*1+1:7*1+2));

=[B1 B2 B3 B4 B5 B6 B7 B8];                % Pointer for E12 series
=[B1A B2A B3A B4A B5A B6A B7A B8A];        % Pointer for 'power of'

inte12= any(B'>11);
int12=find(pointe12==0);
oint12=find(pointe12==1);
eval=B(point12,:);
Aeval=BA(point12,:);
eval=B(npoint12,:);

   Xneval = ones(length(npoint12),1+4);
   fneval = 1e15*ones(1,(length(npoint12)));

if length(point12)>0
   R1 = S( Beval(:,1) +1 ) .*10 .^( S1 ( BAeval(:,1) +1 ));
   R2 = S( Beval(:,2) +1 ) .*10 .^( S1 ( BAeval(:,2) +1 ));
   R3 = S( Beval(:,3) +1 ) .*10 .^( S1 ( BAeval(:,3) +1 ));
   R4 = S( Beval(:,4) +1 ) .*10 .^( S1 ( BAeval(:,4) +1 ));
   R5 = S( Beval(:,5) +1 ) .*10 .^( S1 ( BAeval(:,5) +1 ));
   R6 = S( Beval(:,6) +1 ) .*10 .^( S1 ( BAeval(:,6) +1 ));
   C1 = S( Beval(:,7) +1 ) .*10 .^( S2 ( BAeval(:,7) +1 ));
   C2 = S( Beval(:,8) +1 ) .*10 .^( S2 ( BAeval(:,8) +1 ));

   W1 = sqrt( (R4 ./R3) .*(1 ./(C1 .*C2 .*R5 .*R6)) );
   Q1 = ( ( (R3 .*(R1+R2)) ./(R1 .*(R3+R4)) ) .*( sqrt ( (C1 .*R4 .*R5) ...
   ./(C2 .*R3 .*R6) )  ));

   DW = abs(W0-W1);
```

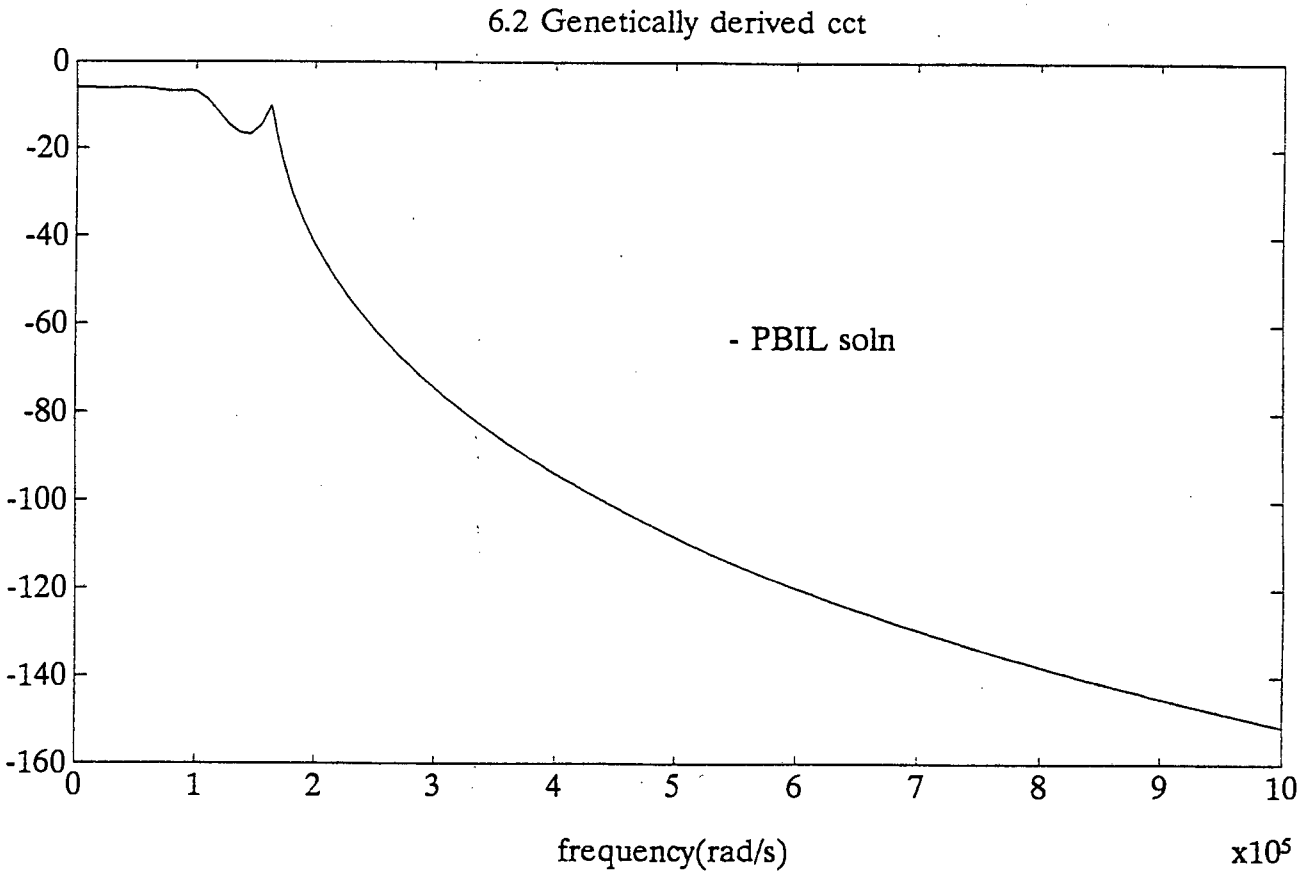
```
  DQ = abs(Q-Q1);

  err = 0.5*( DW/W0 ) + 0.5*( DQ/Q ); % Calculating error function

  Xeval=[R1' R2' R3' R4' R5' R6' C1' C2' W1' Q1'];
  X=ones(m,l+4);
  X(point12,:)=Xeval;

  feval = err;
  f=ones(1,m);
  f(point12)=feval;
end

  f(npoint12)=fneval;
  f=f';
```

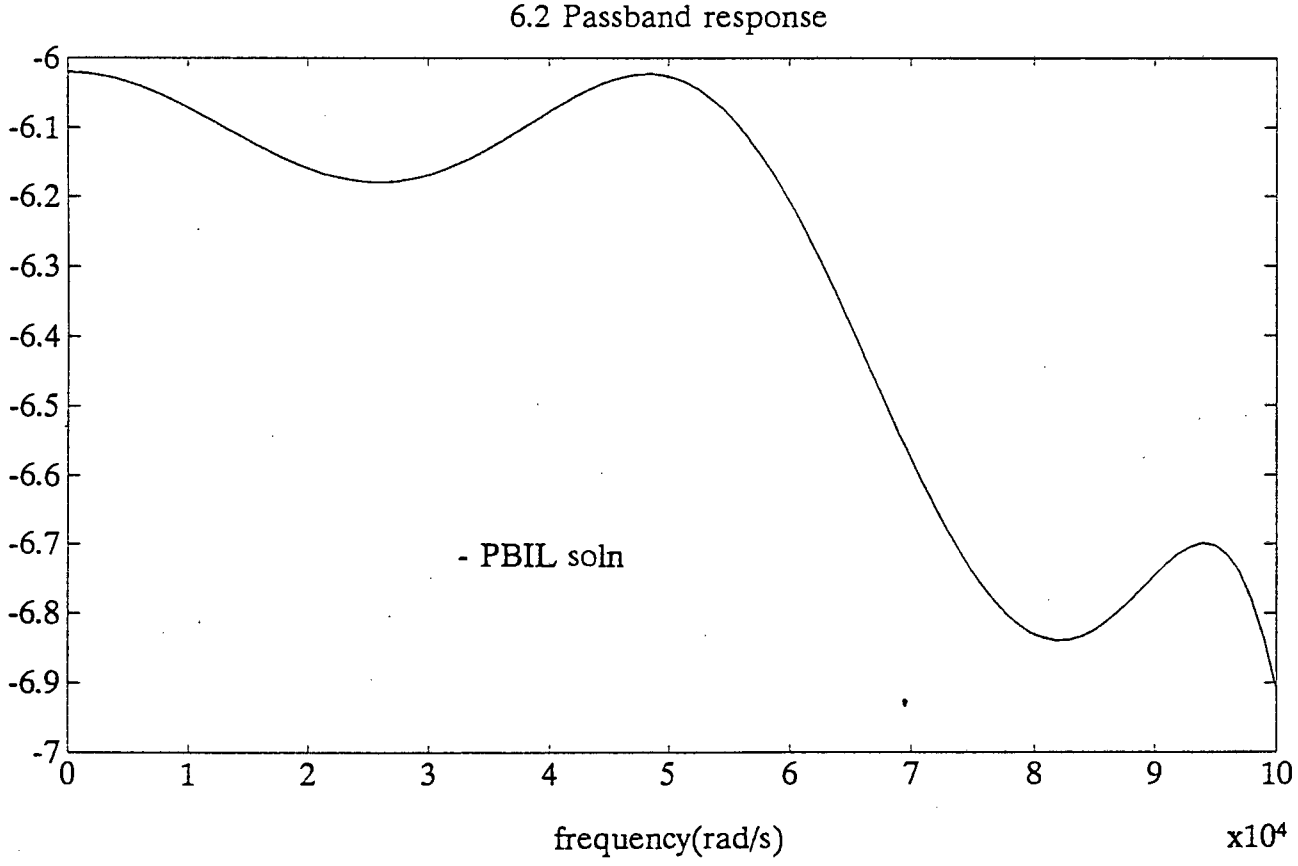
Appendix D2: Genetically derived filter circuit using preferred component values

'Genetically derived filter circuits using prefered component values' by D.H. Horrocks & Y.M.A. Khalifa. This paper tackles the problem of an all-pole low pass LC chebychev filter with 1 dB pass band ripple,pass band edge at 1e5 rad/sec and stop band attention of -150 dB at st.p band edge of 1e6 rad/sec.

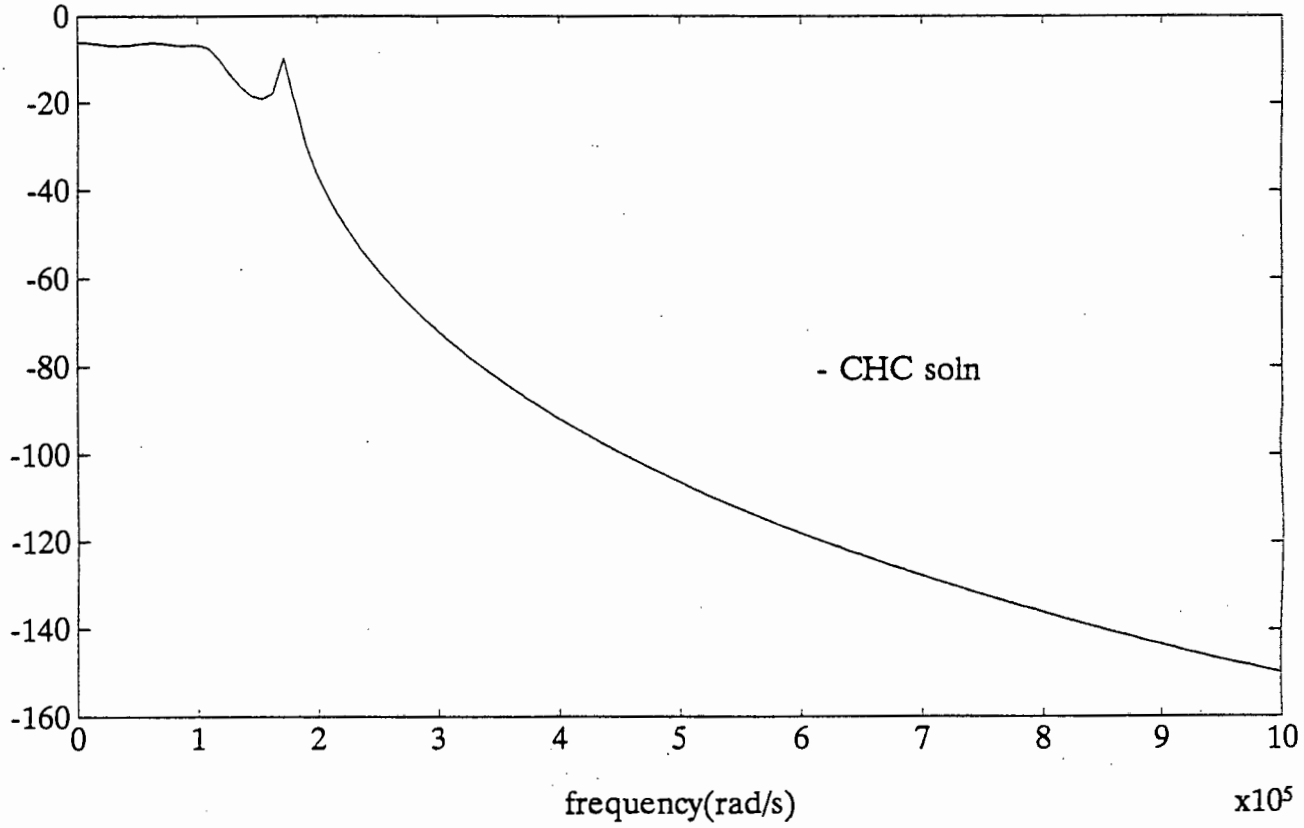
```
unction [f,X]=gdfc(B)

 m n]=size(B);
.=n/7;

%1=btod(B(:,3:1));
%2=btod(B(:,1+3:2*1));
%3=btod(B(:,2*1+3:3*1));
%4=btod(B(:,3*1+3:4*1));
%5=btod(B(:,4*1+3:5*1));
%6=btod(B(:,5*1+3:6*1));
%7=btod(B(:,6*1+3:7*1));

%1A=btod(B(:,1:2));
%2A=btod(B(:,1+1:1+2));
%3A=btod(B(:,2*1+1:2*1+2));
%4A=btod(B(:,3*1+1:3*1+2));
%5A=btod(B(:,4*1+1:4*1+2));
%6A=btod(B(:,5*1+1:5*1+2));
%7A=btod(B(:,6*1+1:6*1+2));

%=[B1 B2 B3 B4 B5 B6 B7];                % Pointer for E12
%A=[B1A B2A B3A B4A B5A B6A B7A];        % Pointer for 'Power of'

 = [10 12 15 18 22 27 33 39 47 56 68 82];      % E12 series
1= [-5 -4 -3 -2];                              % Decade range for inductors
2= [-8 -7 -6 -5];                              % Decade range for capacitors

ointe12=any(B'>11);
oint12=find(pointe12==0);
point12=find(pointe12==1);
eval=B(point12,:);
Aeval=BA(point12,:);
neval=B(npoint12,:);

neval=ones(length(npoint12),1+3);
 = Xneval;
neval=1e15*ones(1,(length(npoint12)));
e=length(point12);

f le>0
  R1 = 100*ones(1,le);
  R2 = 100*ones(1,le);
  L2 = S( Beval(:,1) +1 ) .*10 .^( S1 ( BAeval(:,1) +1 ));
  L4 = S( Beval(:,2) +1 ) .*10 .^( S1 ( BAeval(:,2) +1 ));
  L6 = S( Beval(:,3) +1 ) .*10 .^( S1 ( BAeval(:,3) +1 ));
  C1 = S( Beval(:,4) +1 ) .*10 .^( S2 ( BAeval(:,4) +1 ));
  C3 = S( Beval(:,5) +1 ) .*10 .^( S2 ( BAeval(:,5) +1 ));
  C5 = S( Beval(:,6) +1 ) .*10 .^( S2 ( BAeval(:,6) +1 ));
  C7 = S( Beval(:,7) +1 ) .*10 .^( S2 ( BAeval(:,7) +1 ));

  Xeval = [L2' L4' L6' C1' C3' C5' C7'];

  R1 = (R1'*ones(1,200));
  R2 = (R2'*ones(1,200));
  L2 = (L2'*ones(1,200));
  L4 = (L4'*ones(1,200));
  L6 = (L6'*ones(1,200));
```

```
C1 = (C1'*ones(1,200));
C3 = (C3'*ones(1,200));
C5 = (C5'*ones(1,200));
C7 = (C7'*ones(1,200));

w1=linspace(0.1,1e5);
w2=linspace(1e5,1e6,101);
w0=[w1 w2(2:101)];
s=i*w0;
s=(s'*ones(1,le))';

Xa22=((1 ./R1)+(s .*C1)+(1 ./(s .*L2)));
Xa33=((1 ./(s .*L2))+(s .*C3)+(1 ./(s .*L4)));
Xa44=((1 ./(s .*L4))+(s .*C5)+(1 ./(s .*L6)));
Xa55=((1 ./(s .*L6))+(s .*C7)+(1 ./R2));

Xb13=-(((1 ./R1) .*(1 ./(s .*L2))) ./Xa22);
Xb33=Xa33-(((1 ./(s .*L2)) .^2) ./Xa22);

Xc14=(((1 ./(s .*L4)) .*Xb13) ./Xb33);
Xc44=Xa44-(((1 ./(s .*L4)) .^2) ./Xb33);

Xd51=(((1 ./(s .*L6)) .*Xc14) ./Xc44);
Xd55=Xa55-(((1 ./(s .*L6)) .^2) ./Xc44);

Ampl = -Xd51 ./ Xd55;

Calculating passband error above -6dB

pba=(abs(Ampl(:,1:100))-0.501187*ones(le,100))';
pba1=pba .*(pba>0);
pband_ab=sum((pba1) .^2);

Calculating passband error below -7dB

pbb=(abs(Ampl(:,1:100))-0.446684*ones(le,100))';
pbb1=pbb .*(pbb<0);
pband_bl=sum((pbb1) .^2);

Calculating total error in passband

pband_err=pband_ab+pband_bl;

Calculating stopband error above -150dB at w=1e6 rad/s

psb=(abs(Ampl(:,200))-3.162278e-8*ones(le,1))';
psb1=psb .*(psb>0);
sband_err=((psb1) .^2);

err=1e10*pband_err+0.5e10*sband_err;

Xeval=[Xeval pband_err' sband_err'];
X=ones(m,l+3);
X(point12,:)=Xeval;

feval = err;
f=ones(1,m);
f(point12)=feval;
d
f(npoint12)=fneval;
f=f';
```

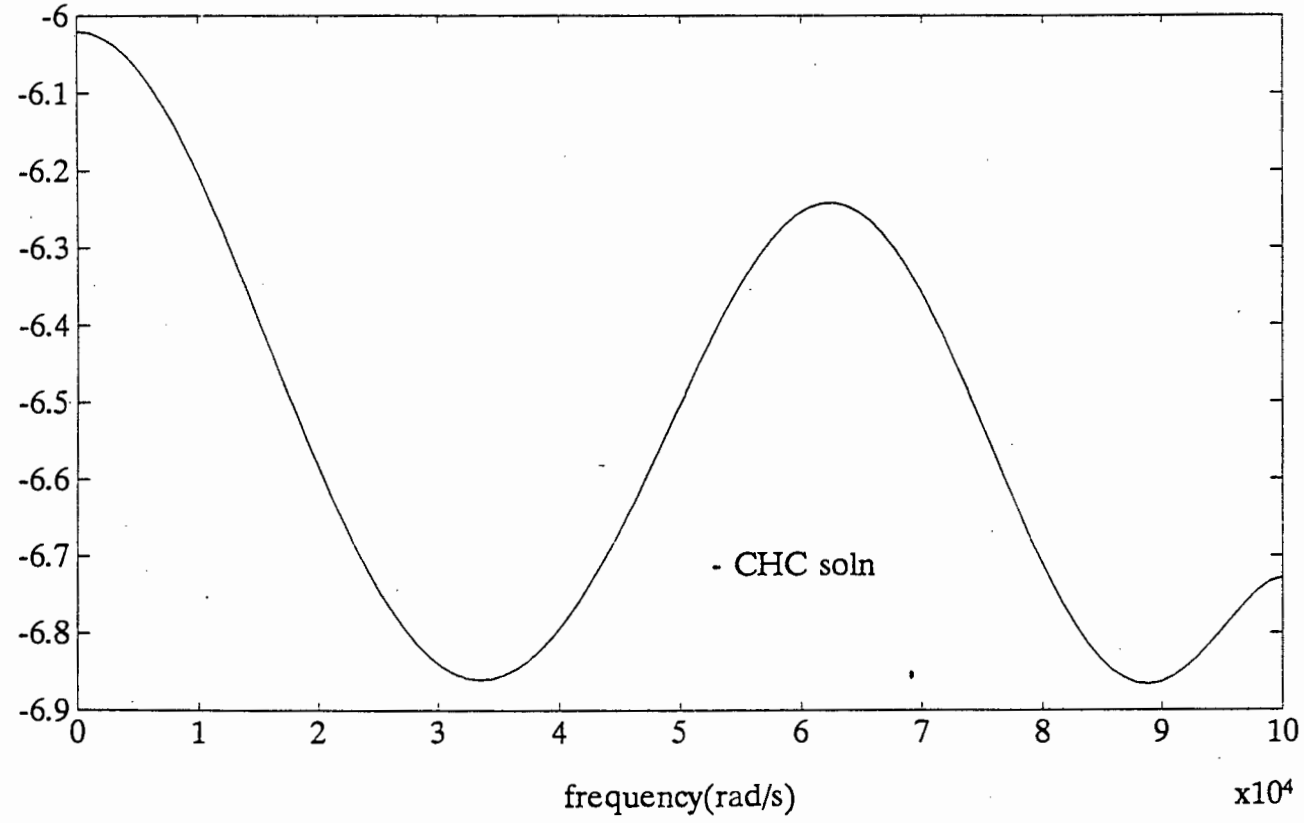

6.2 Genetically derived cct
0
-20
-40
-60
-80
-100
-120
-140
-160
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^5

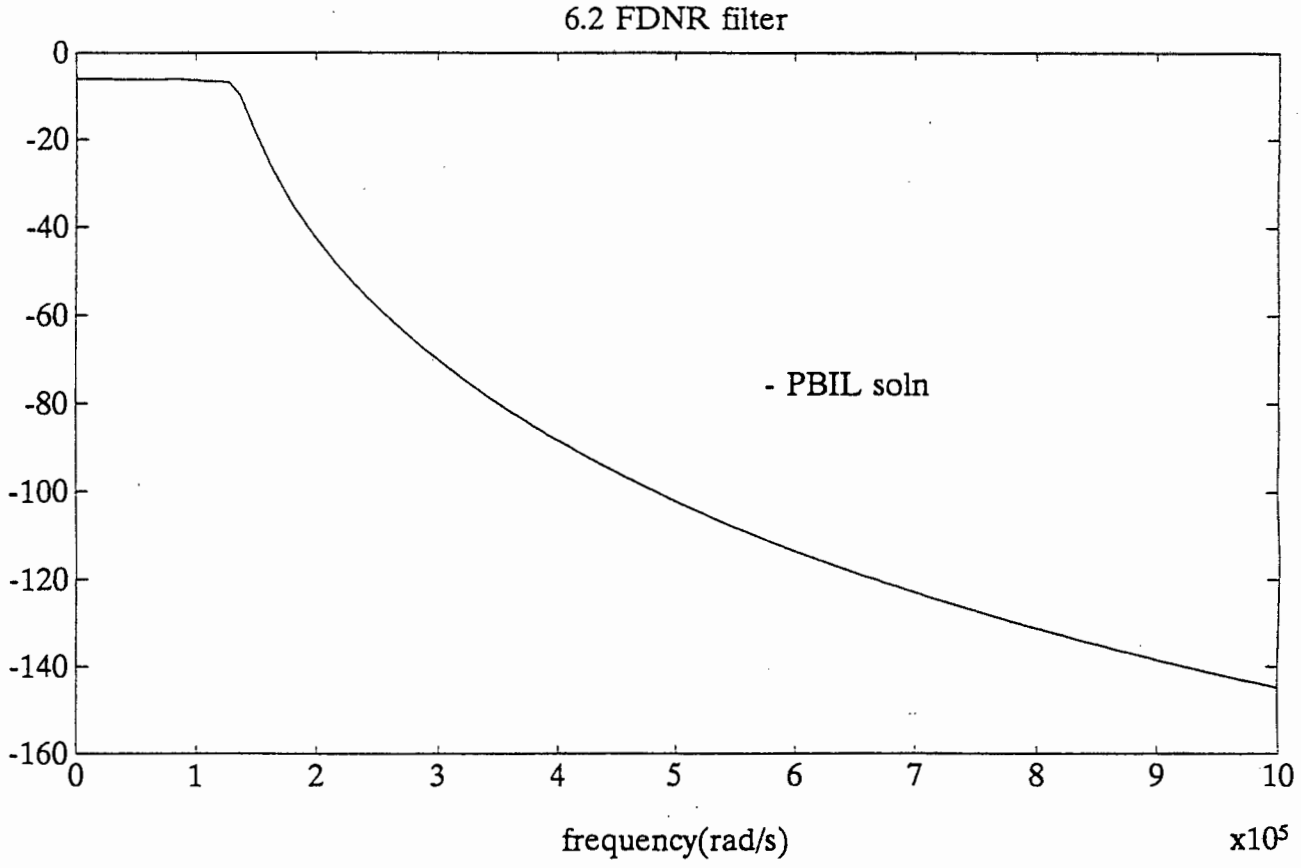

6.2 Passband response
-6
-6.1
-6.2
-6.3
-6.4
-6.5
-6.6
-6.7
-6.8
-6.9
-7
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^4

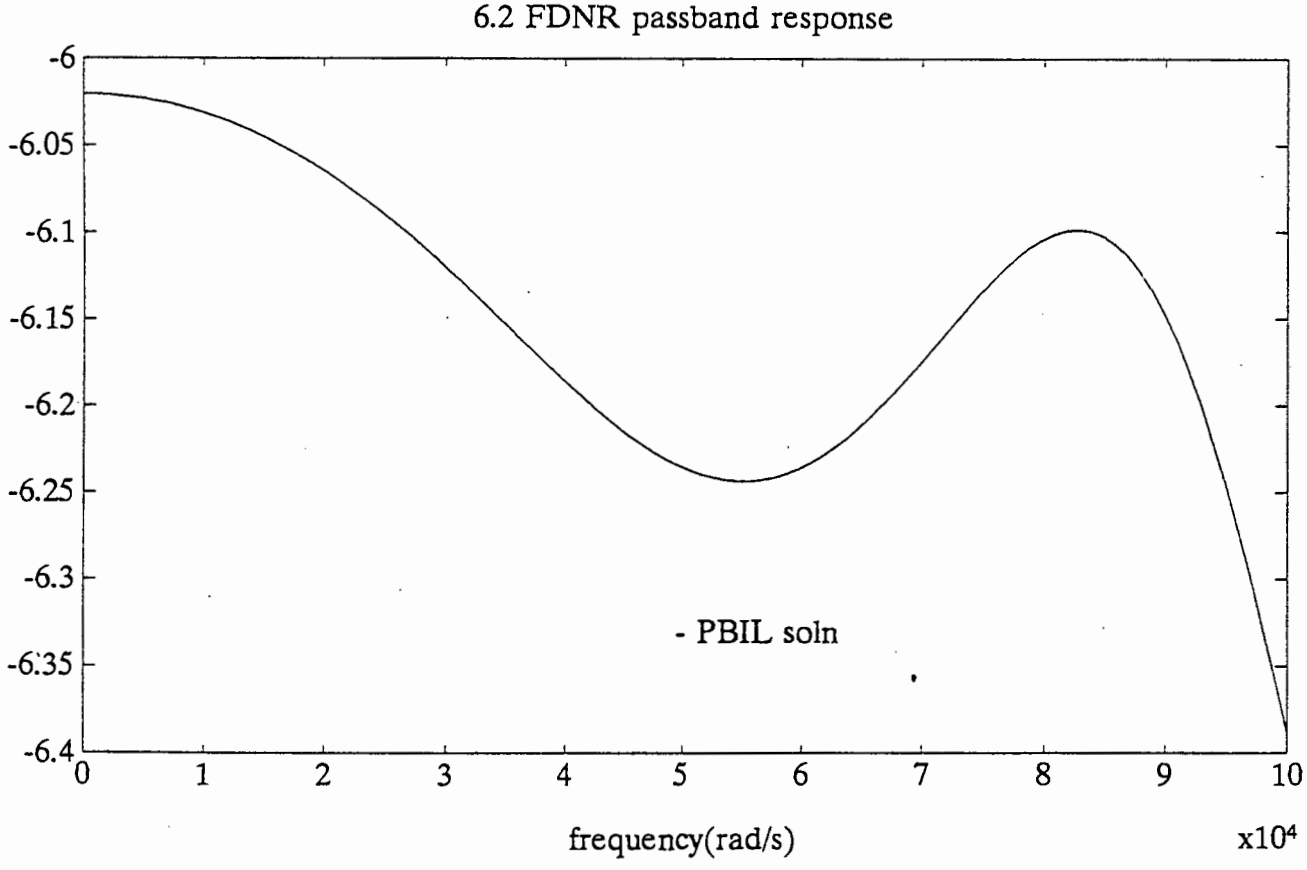
6.2 Genetically derived cct

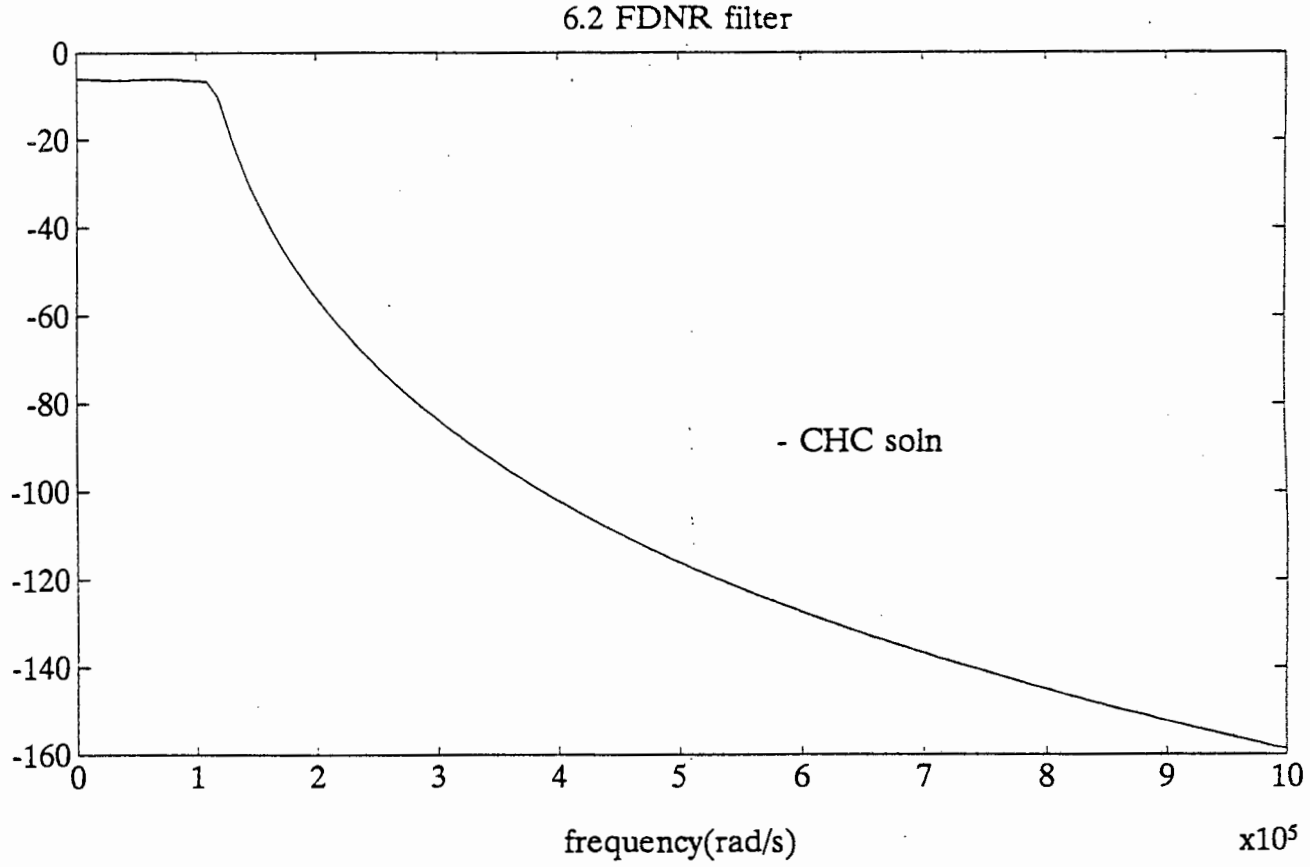


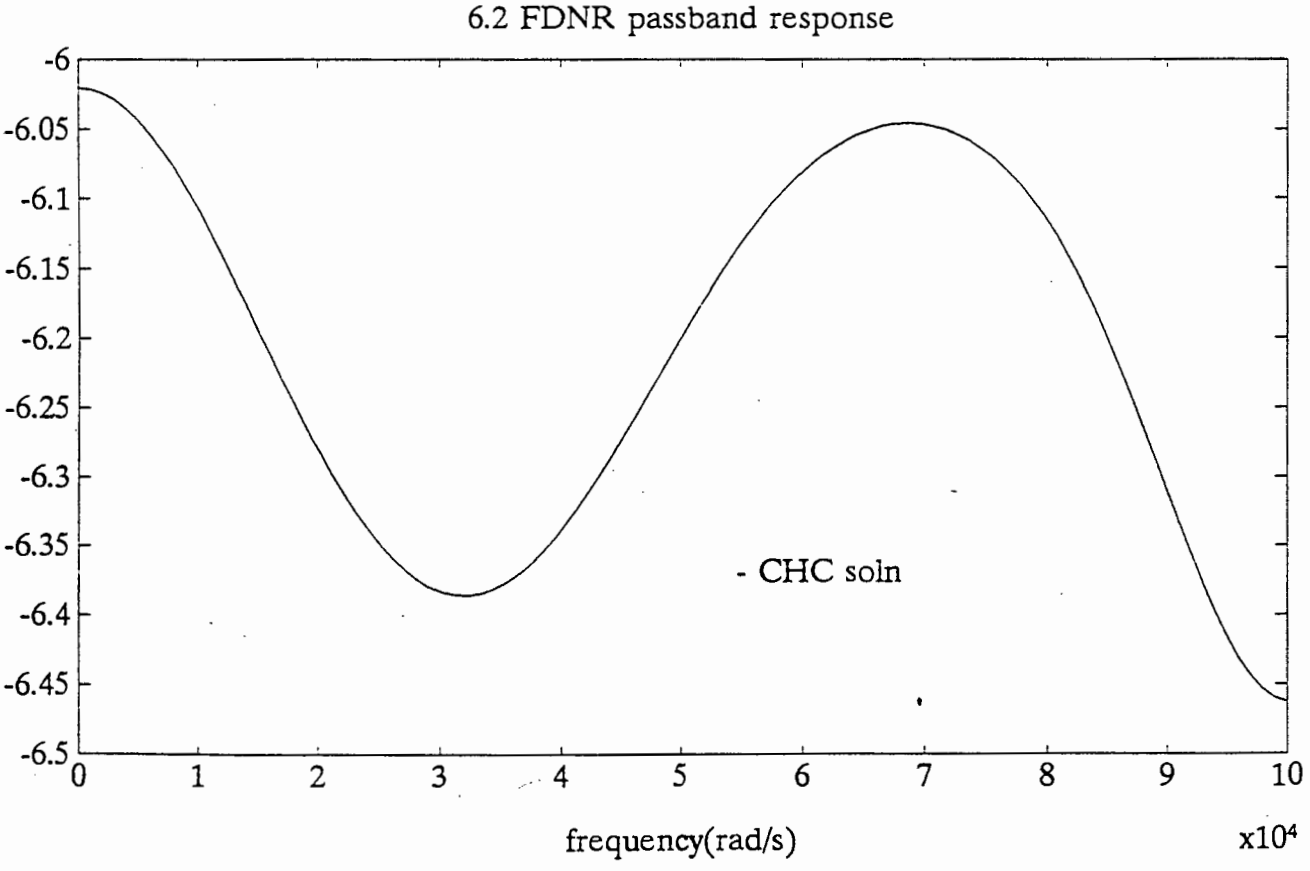
6.2 Passband response




6.2 FDNR filter
0
-20
-40
-60
-80
-100
-120
-140
-160
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

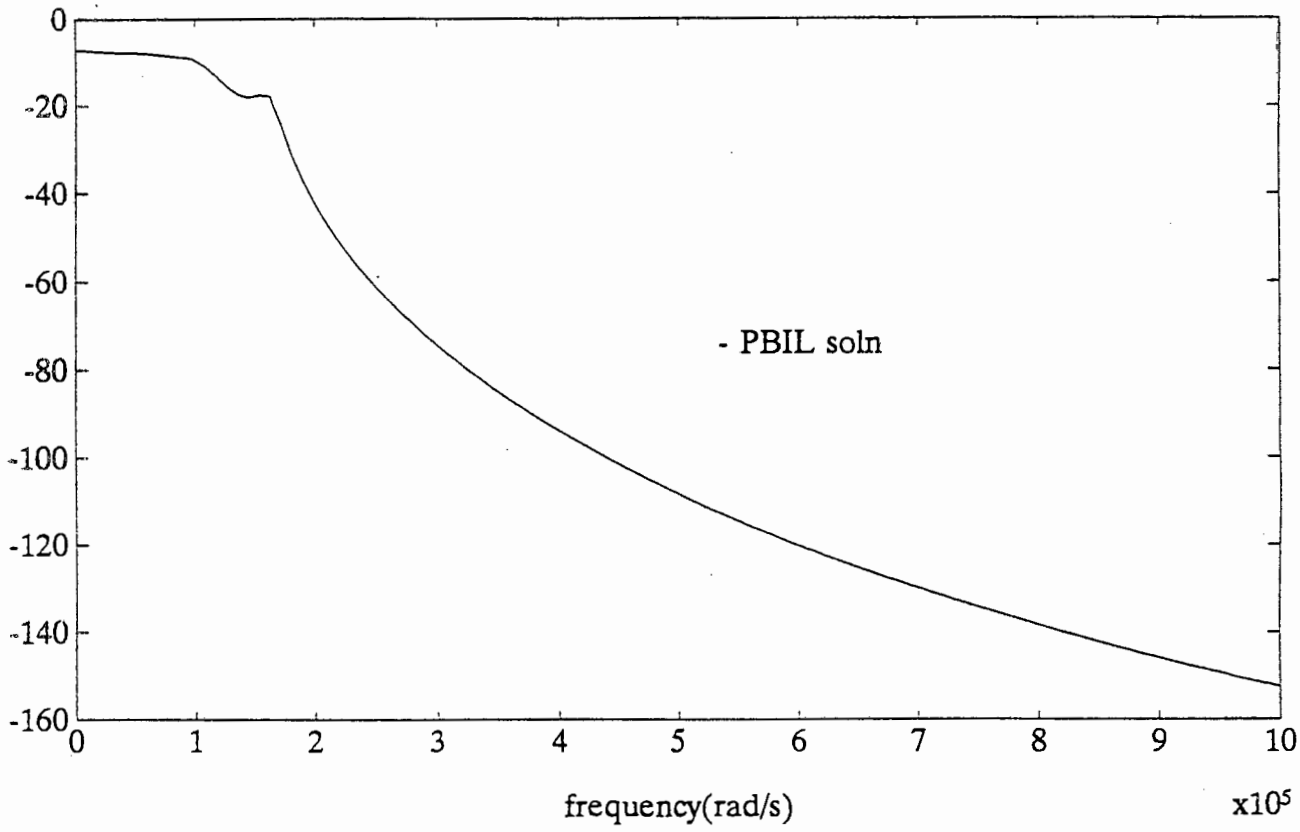

6.2 FDNR passband response
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
-6.4
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4


6.2 FDNR filter
0
-20
-40
-60
-80
-100
-120
-140
-160
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

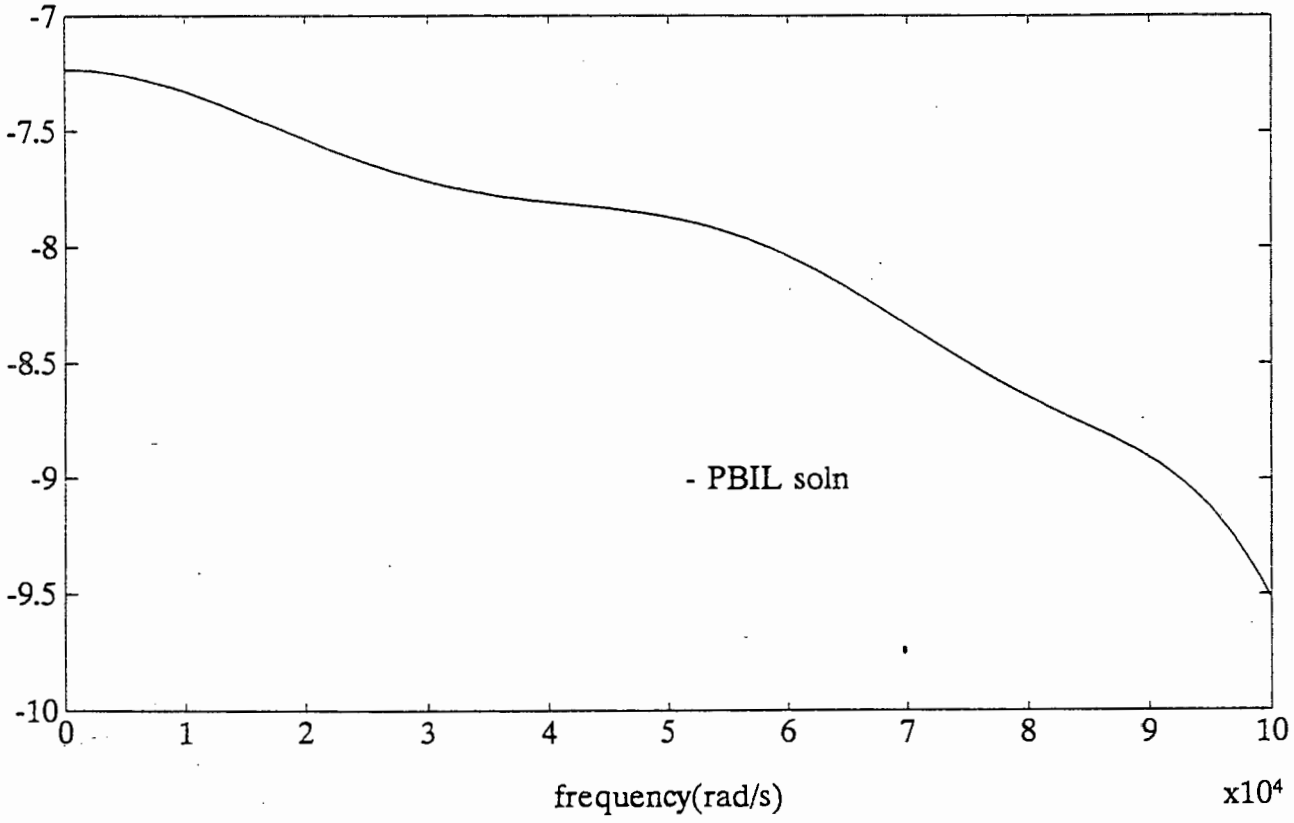

6.2 FDNR passband response
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
-6.4
-6.45
-6.5
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

Appendix D3: Genetically derived filter circuit modelling parasitic effects using preferred component values

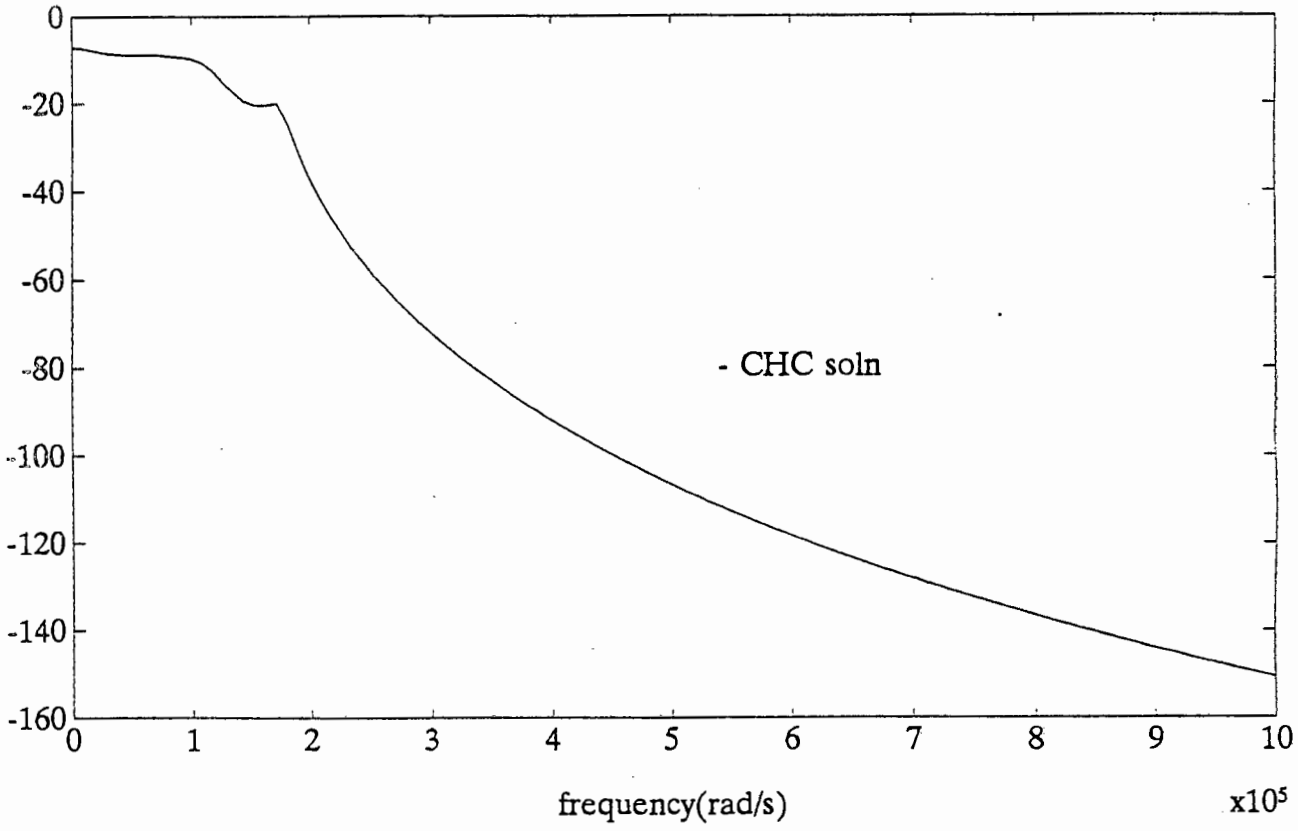
6.3 Optimizer solution(6.2) on cct containing parasitic effects



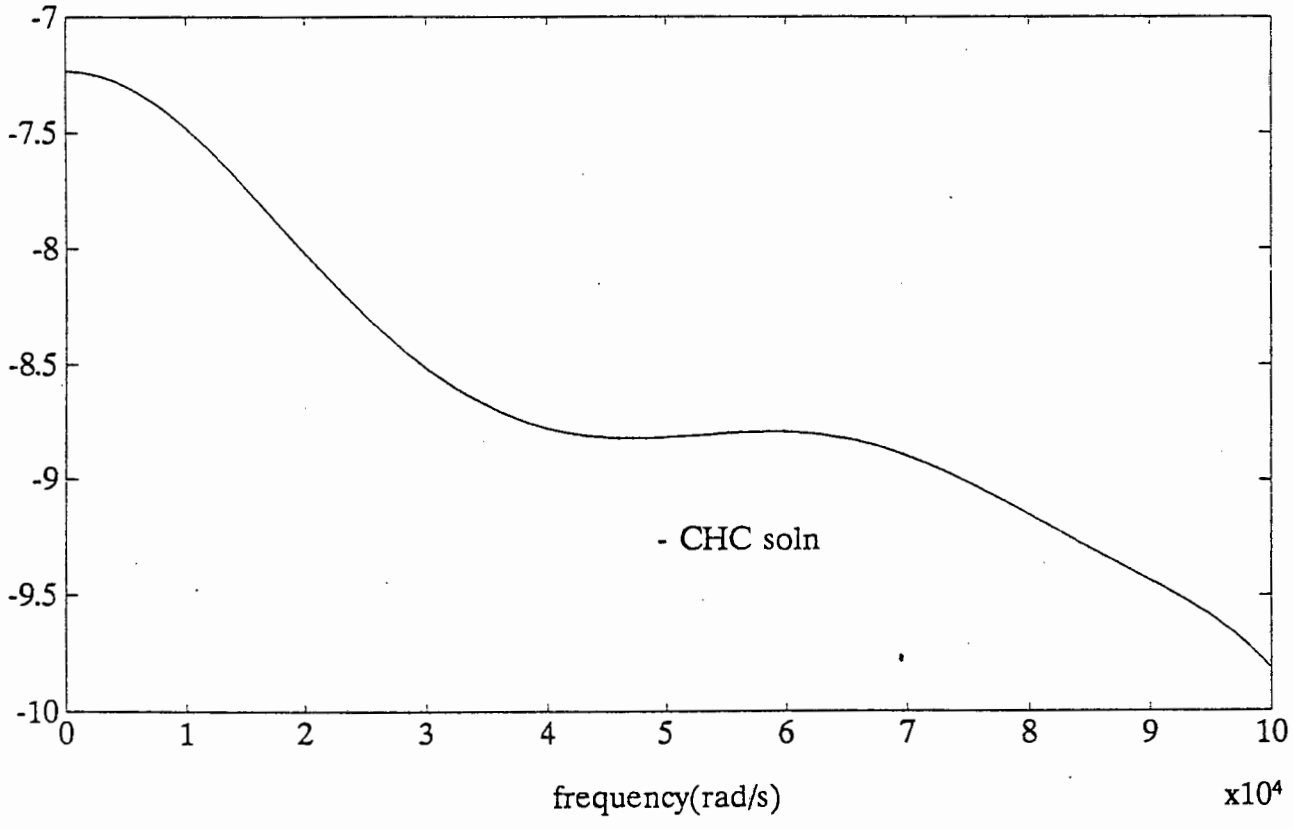
6.3 Passband response degraded



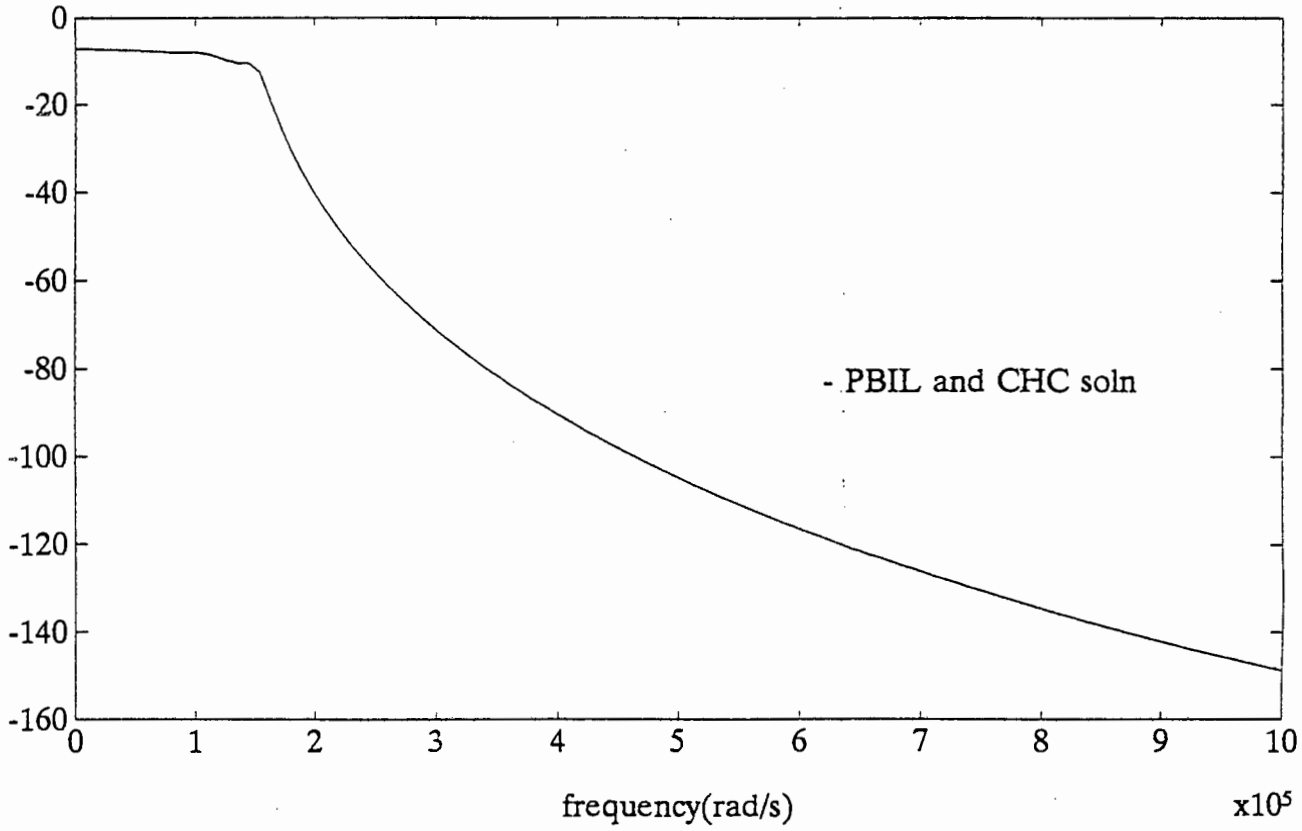
6.3 Optimizer solution(6.2) on cct containing parasitic effects



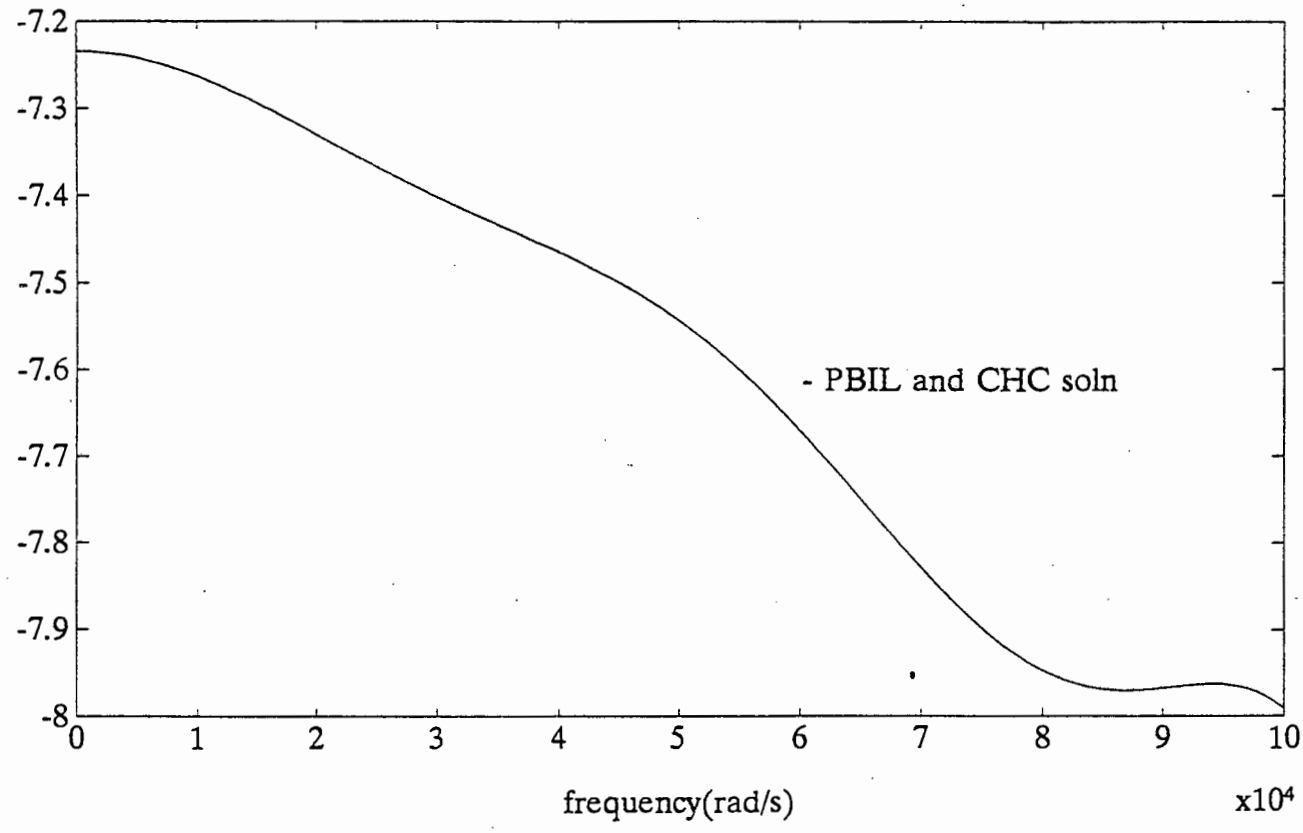
6.3 Passband response degraded



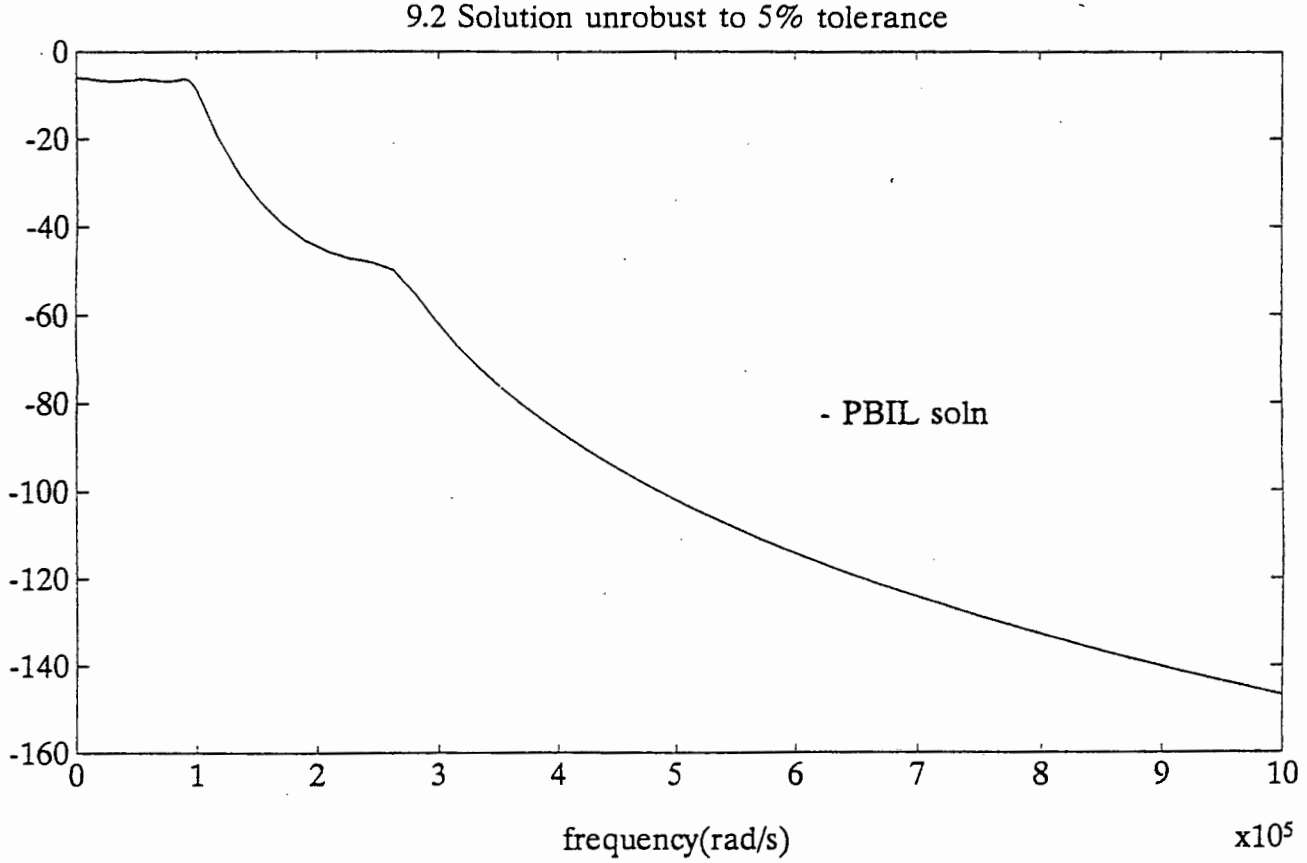
6.3 Modelling parasitic effects

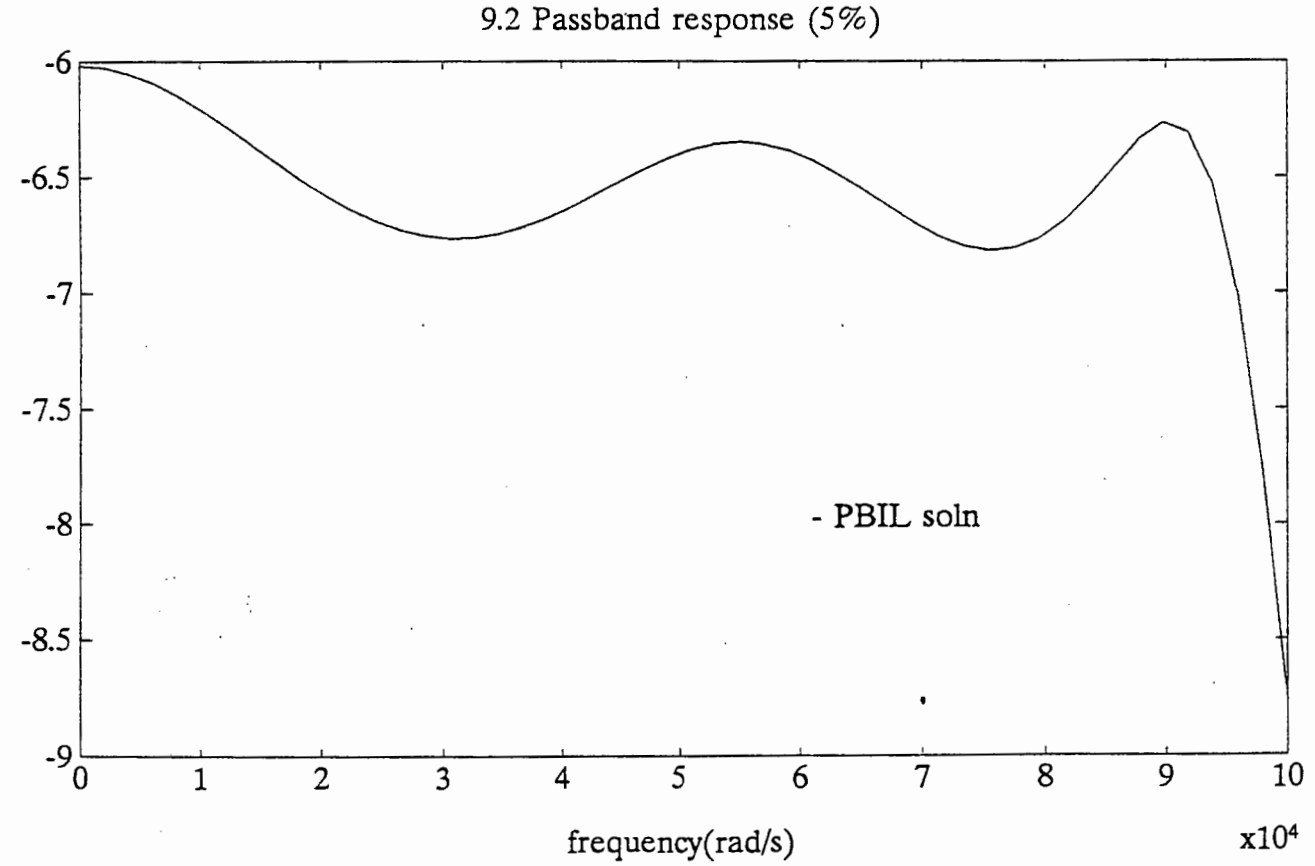


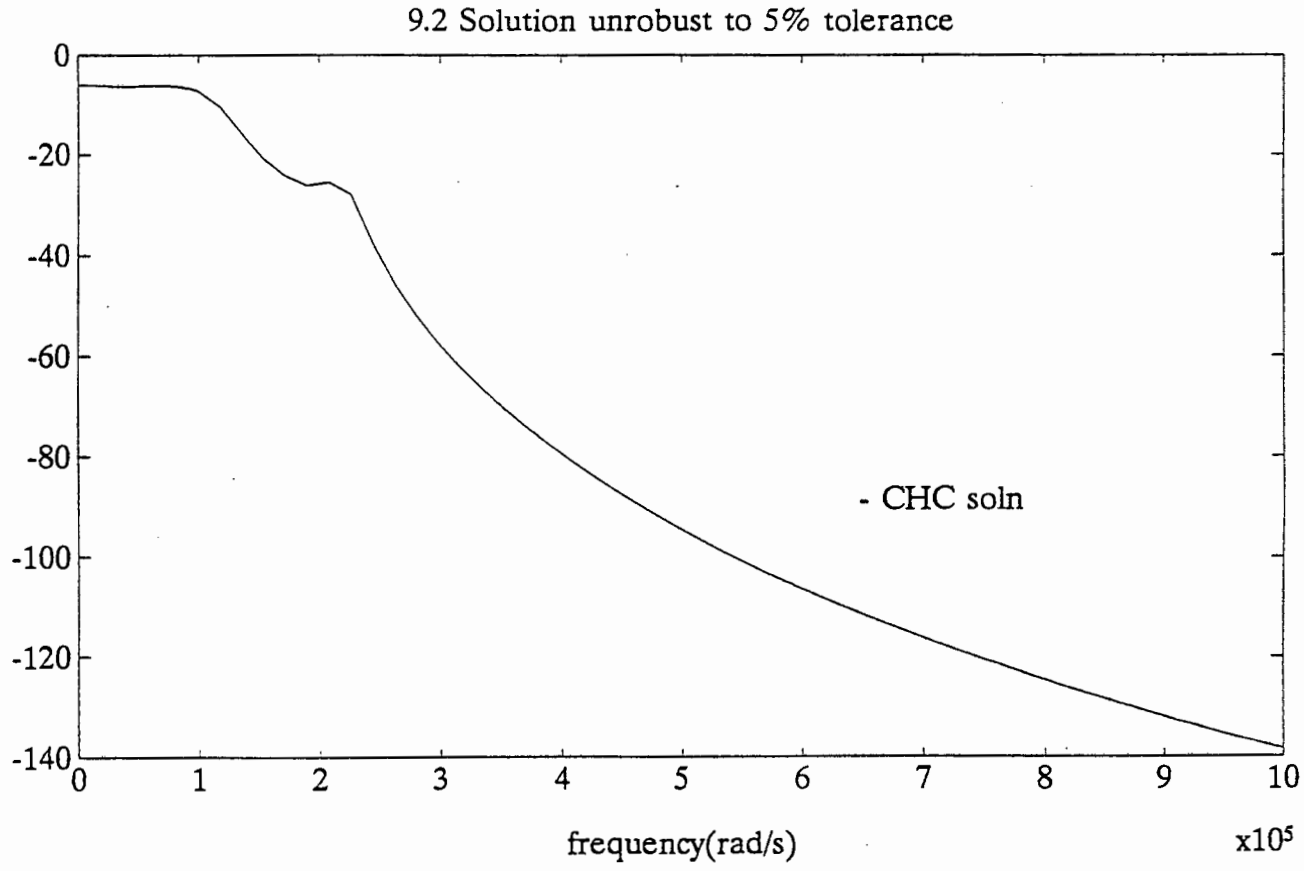
6.3 Passband response

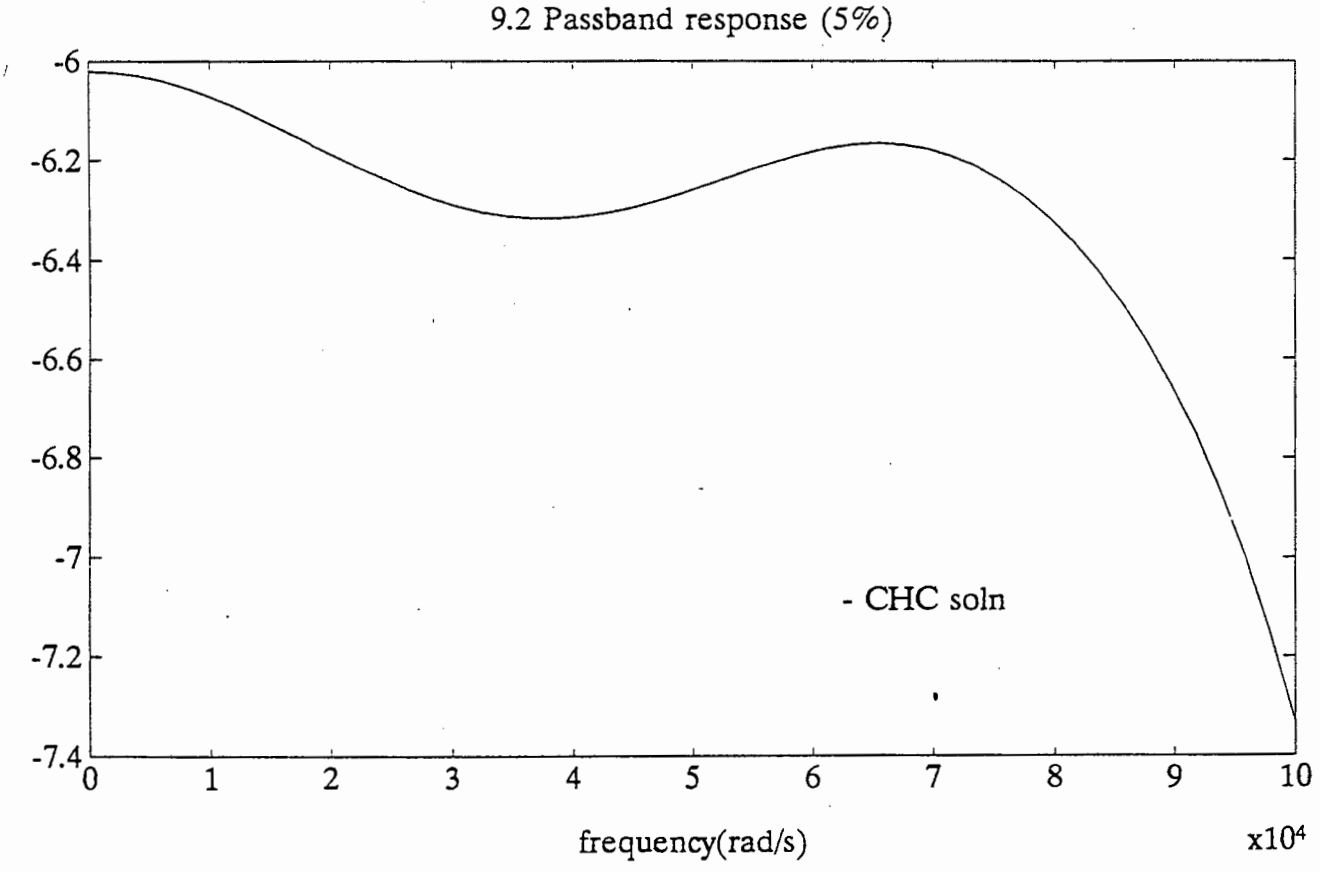


Appendix E1: Study of the robustness characteristic of optimizer solutions on the 7th order butterworth filter

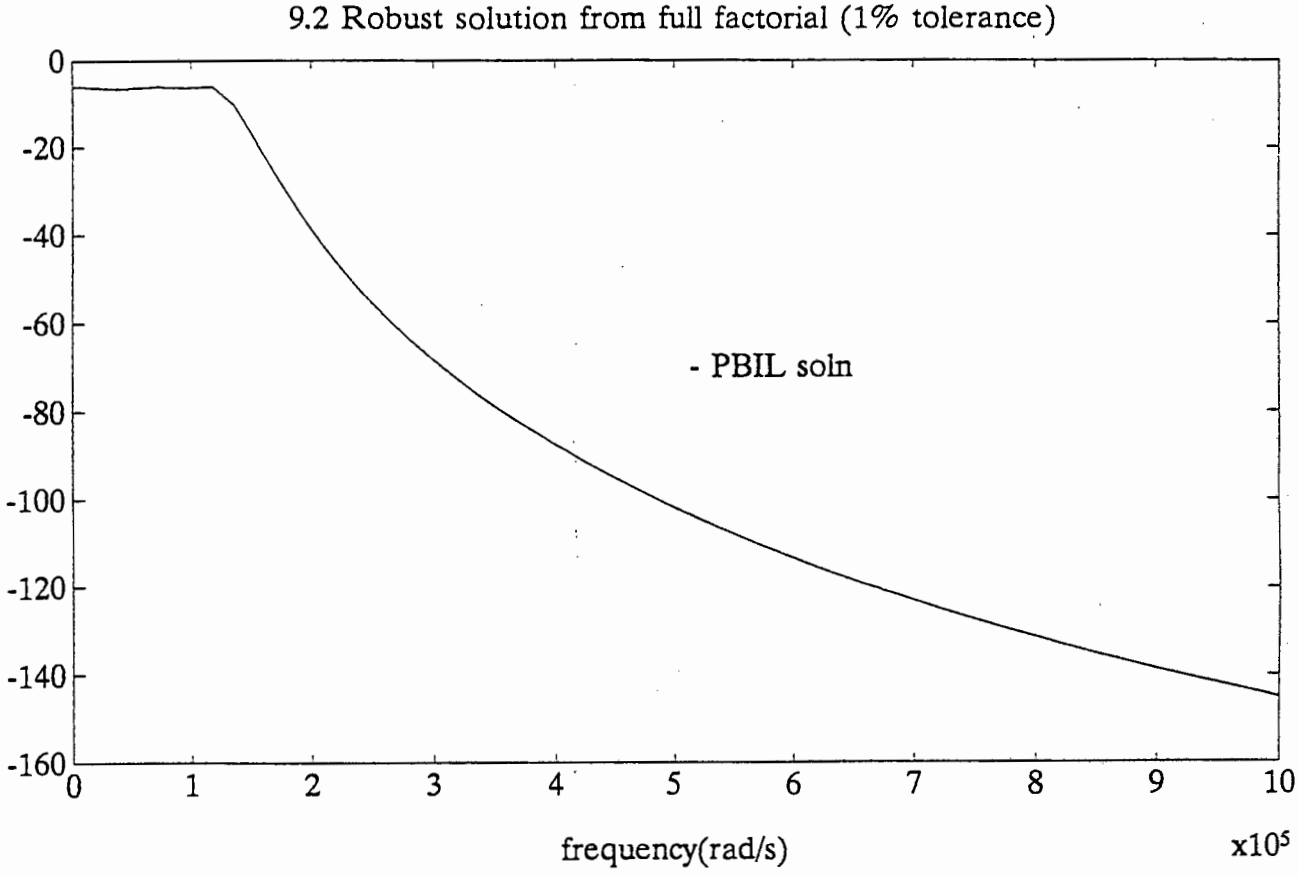

9.2 Solution unrobust to 5% tolerance
0
-20
-40
-60
-80
-100
-120
-140
-160
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

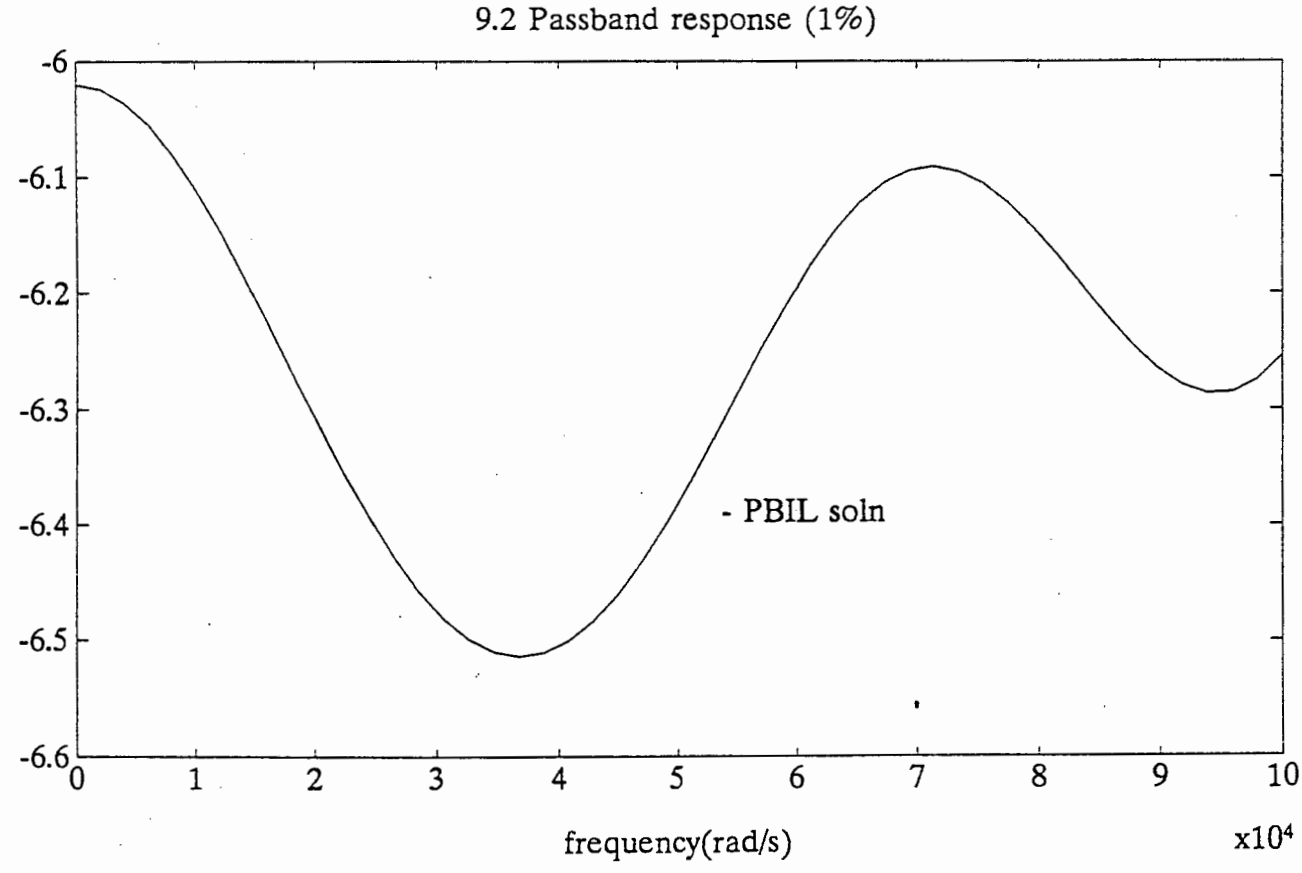

9.2 Passband response (5%)
-6
-6.5
-7
-7.5
-8
-8.5
-9
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

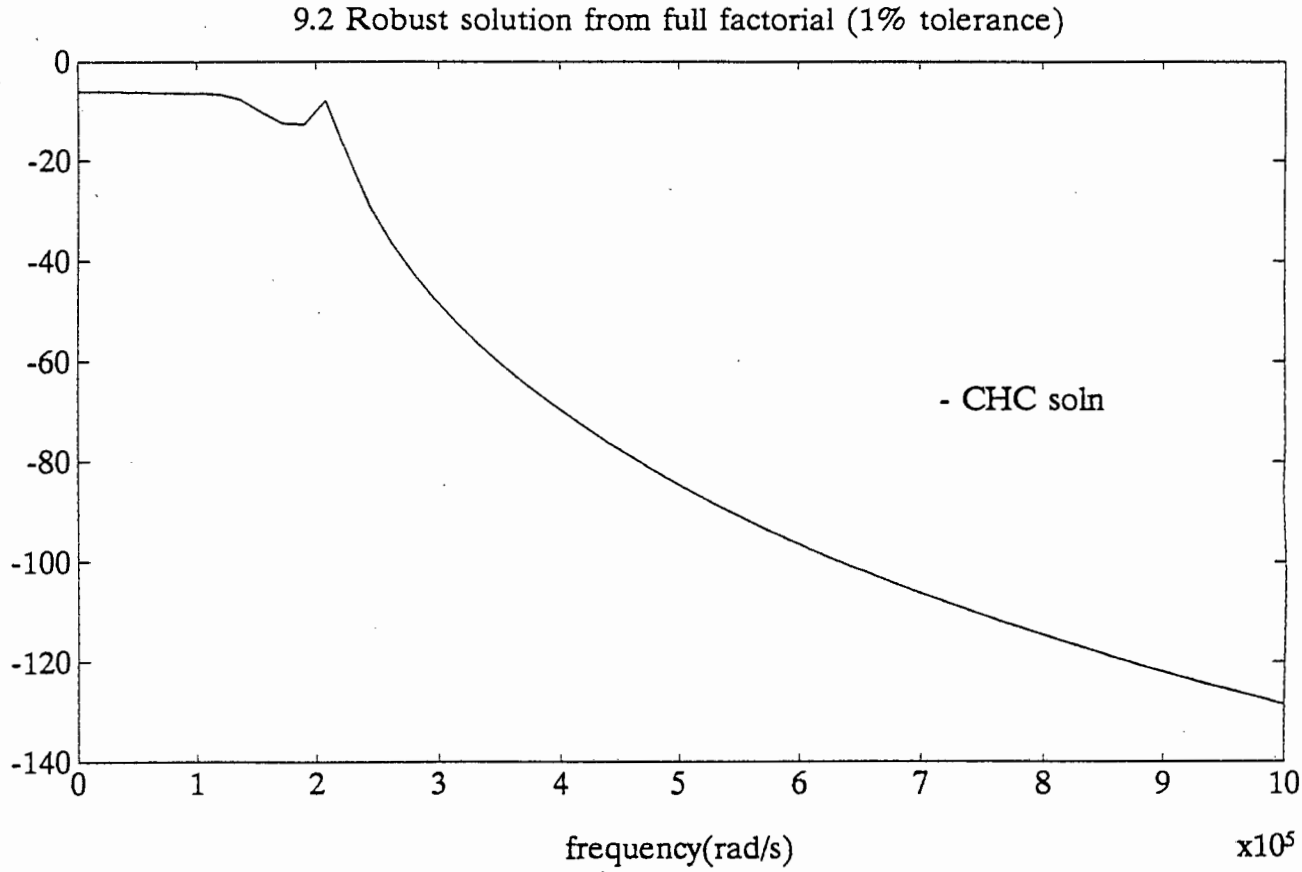

9.2 Solution unrobust to 5% tolerance
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

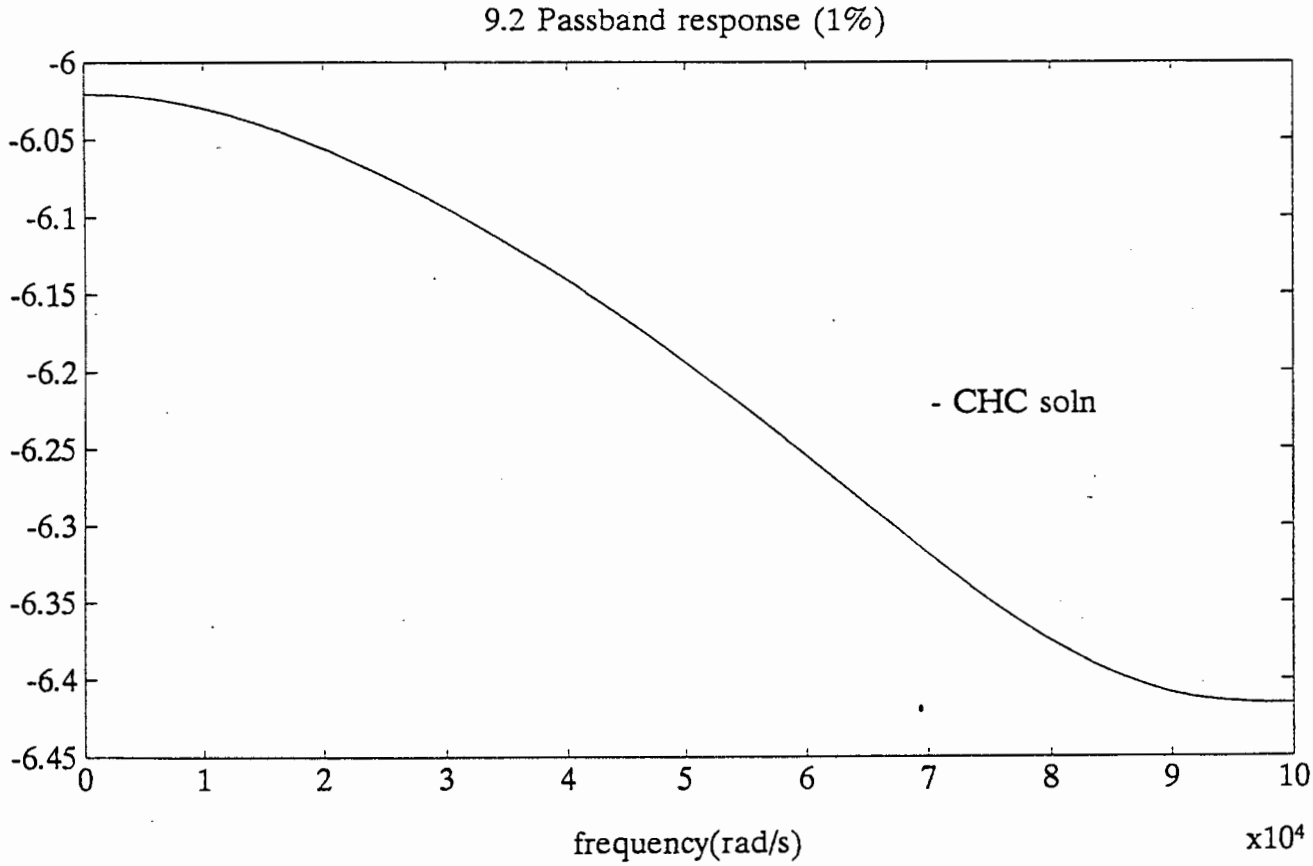

9.2 Passband response (5%)
-6
-6.2
-6.4
-6.6
-6.8
-7
-7.2
-7.4
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

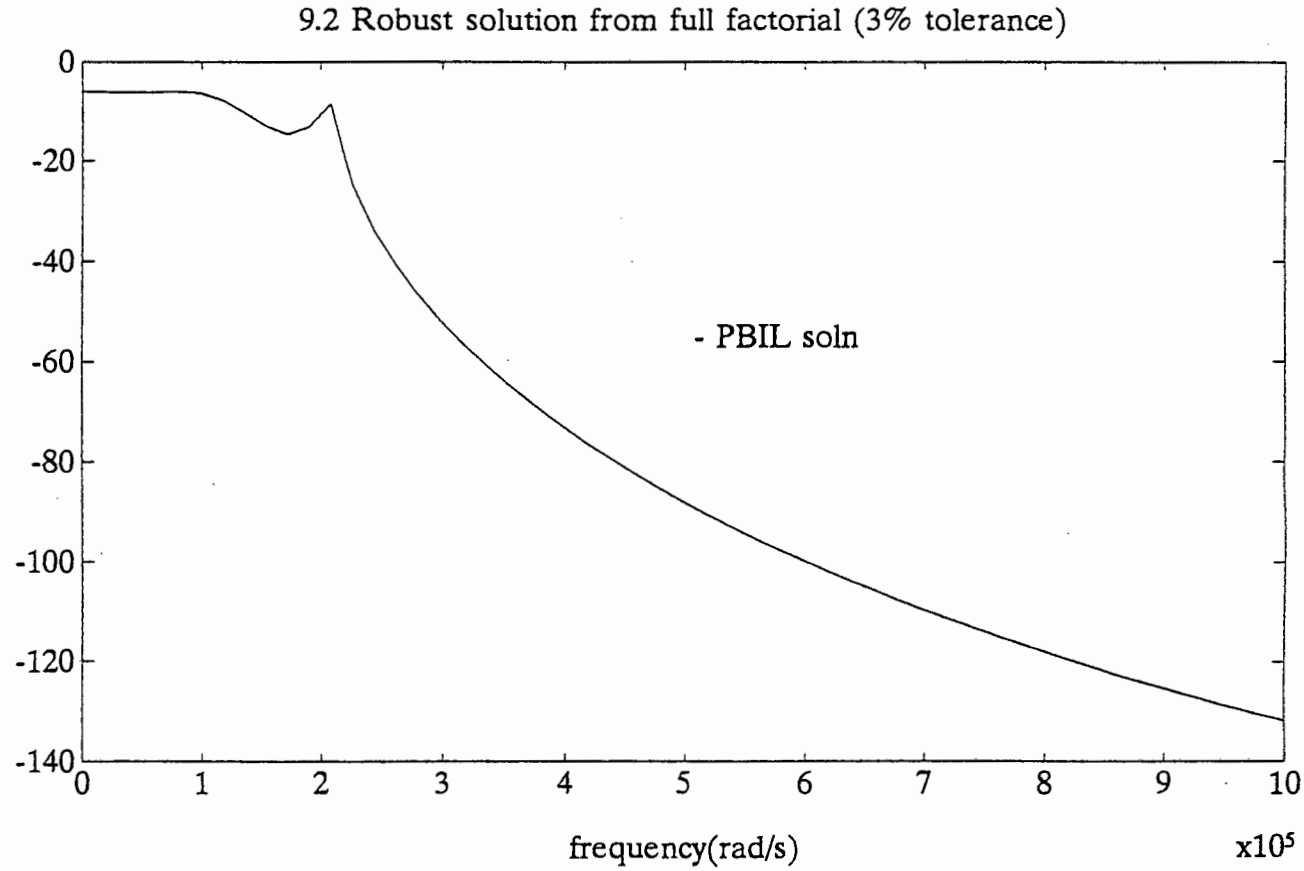
Appendix E2: Hybrid PBIL and hybrid CHC using full factorial search on the 7th order butterworth filter

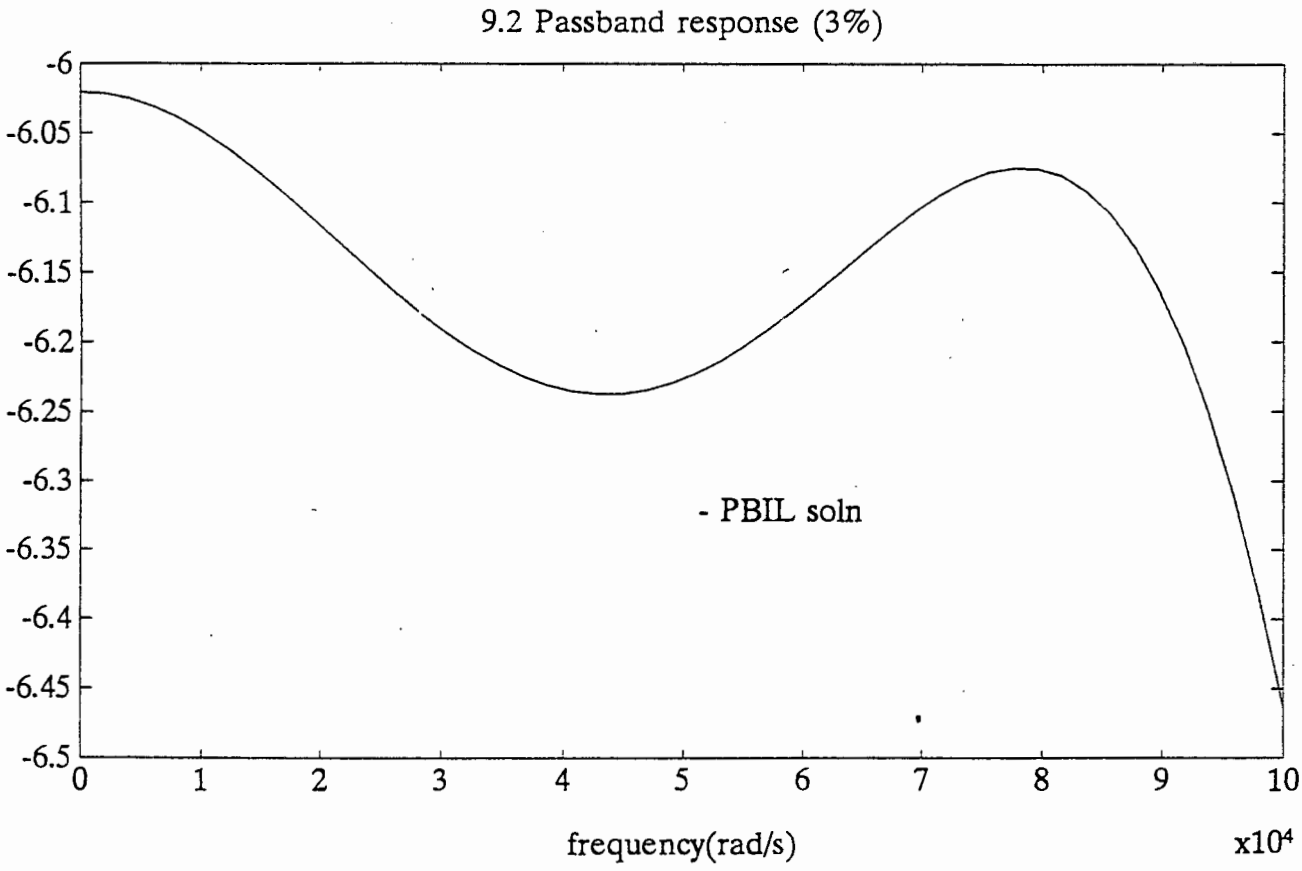

9.2 Robust solution from full factorial (1% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
-160
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^5

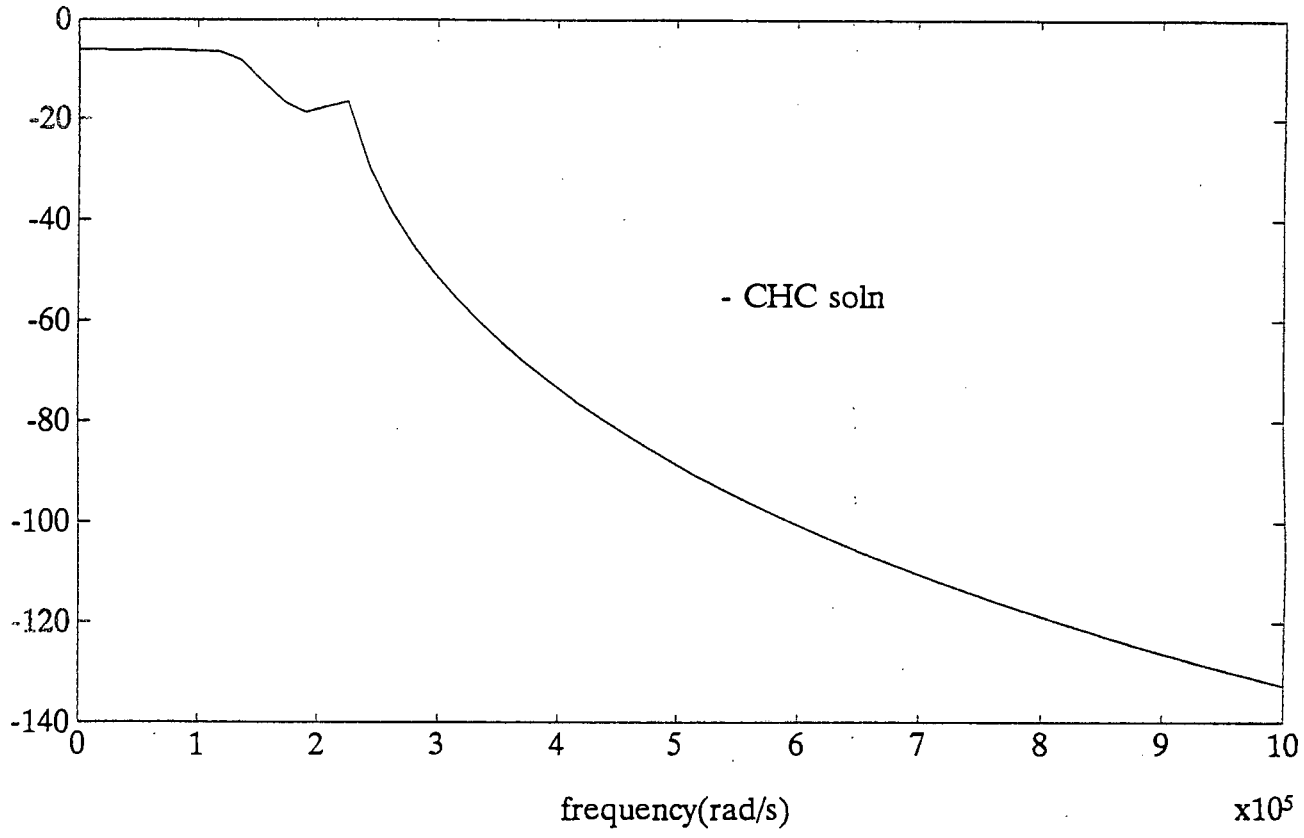

9.2 Passband response (1%)
-6
-6.1
-6.2
-6.3
-6.4
-6.5
-6.6
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^4

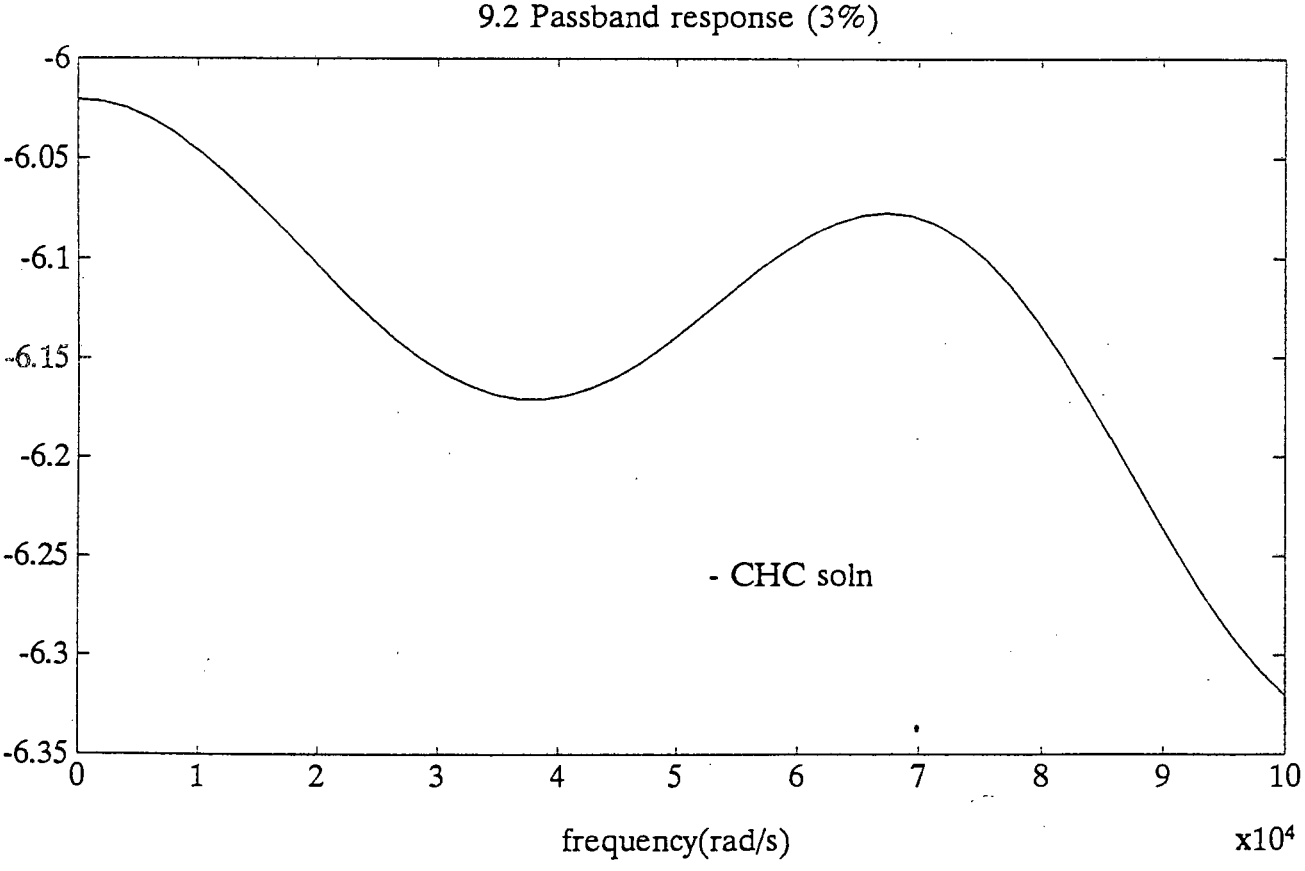

9.2 Robust solution from full factorial (1% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

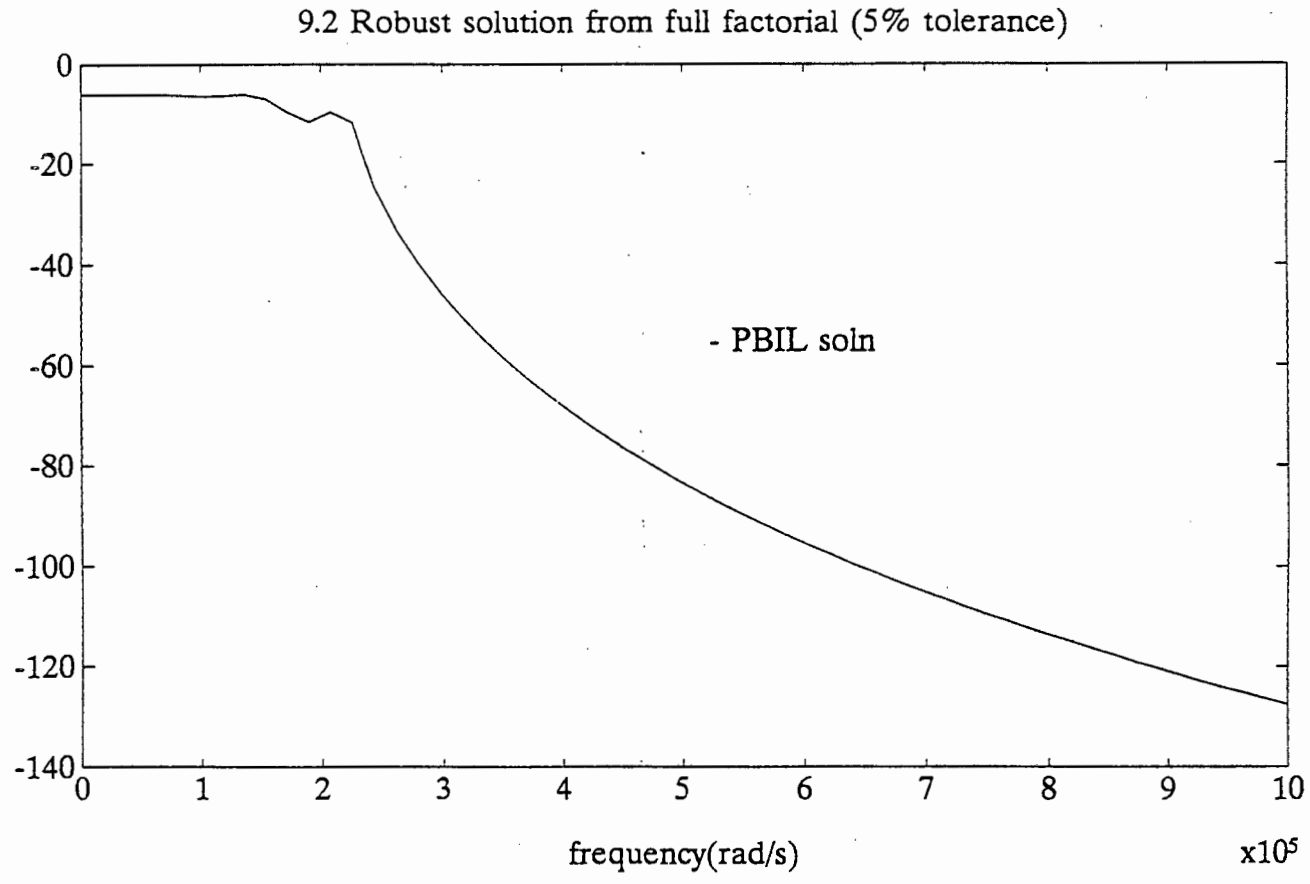

9.2 Passband response (1%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
-6.4
-6.45
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

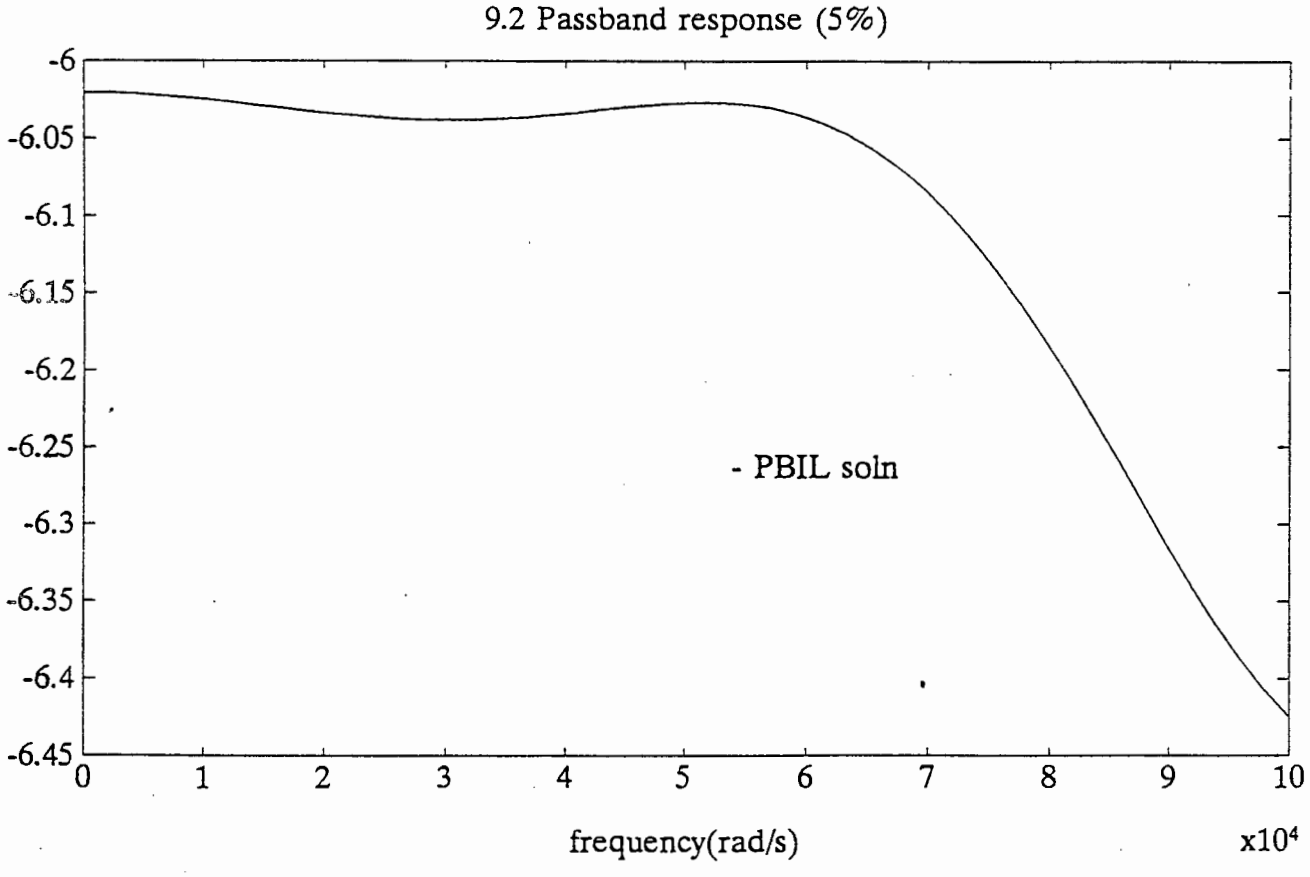

9.2 Robust solution from full factorial (3% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^5

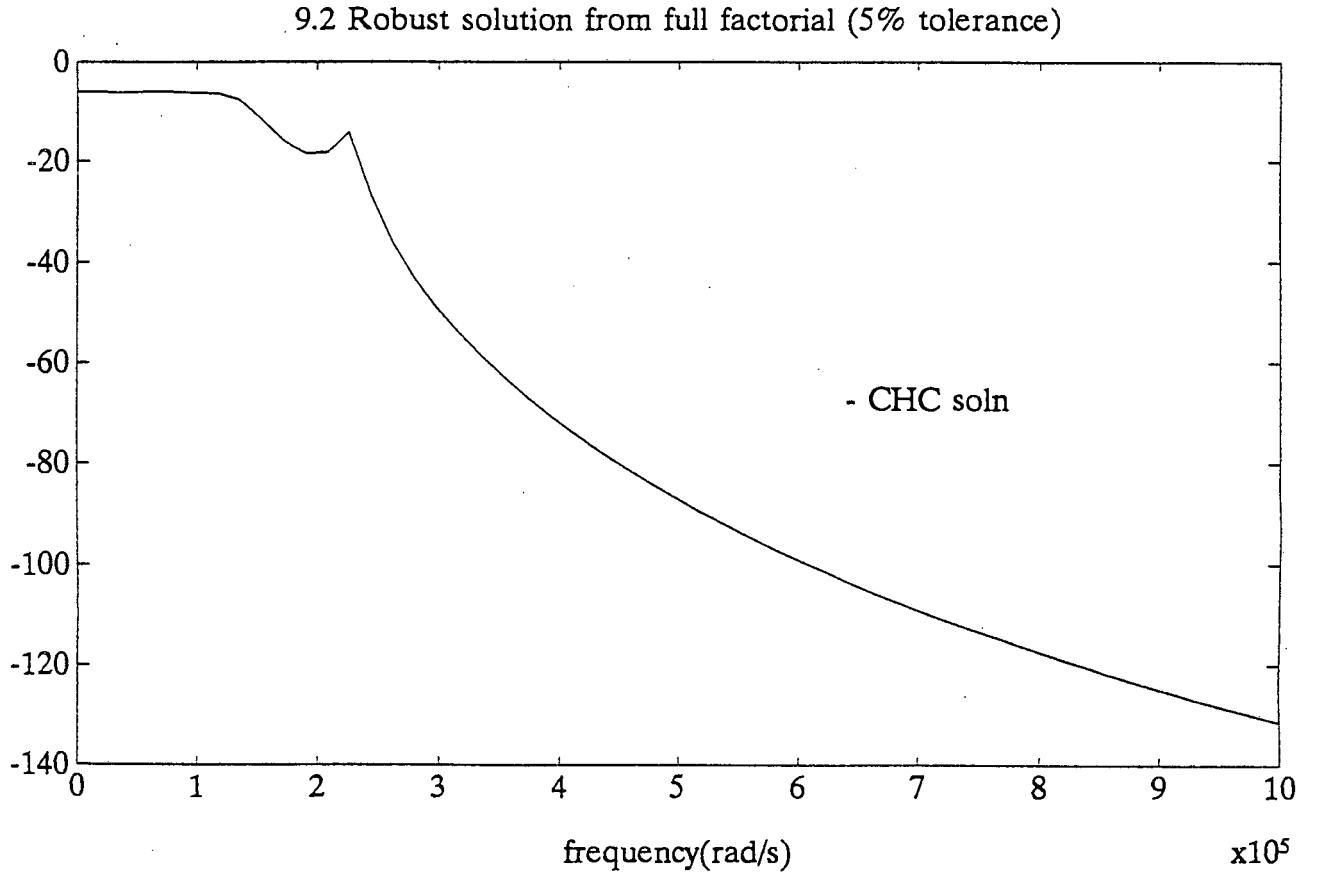

9.2 Passband response (3%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
-6.4
-6.45
-6.5
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^4

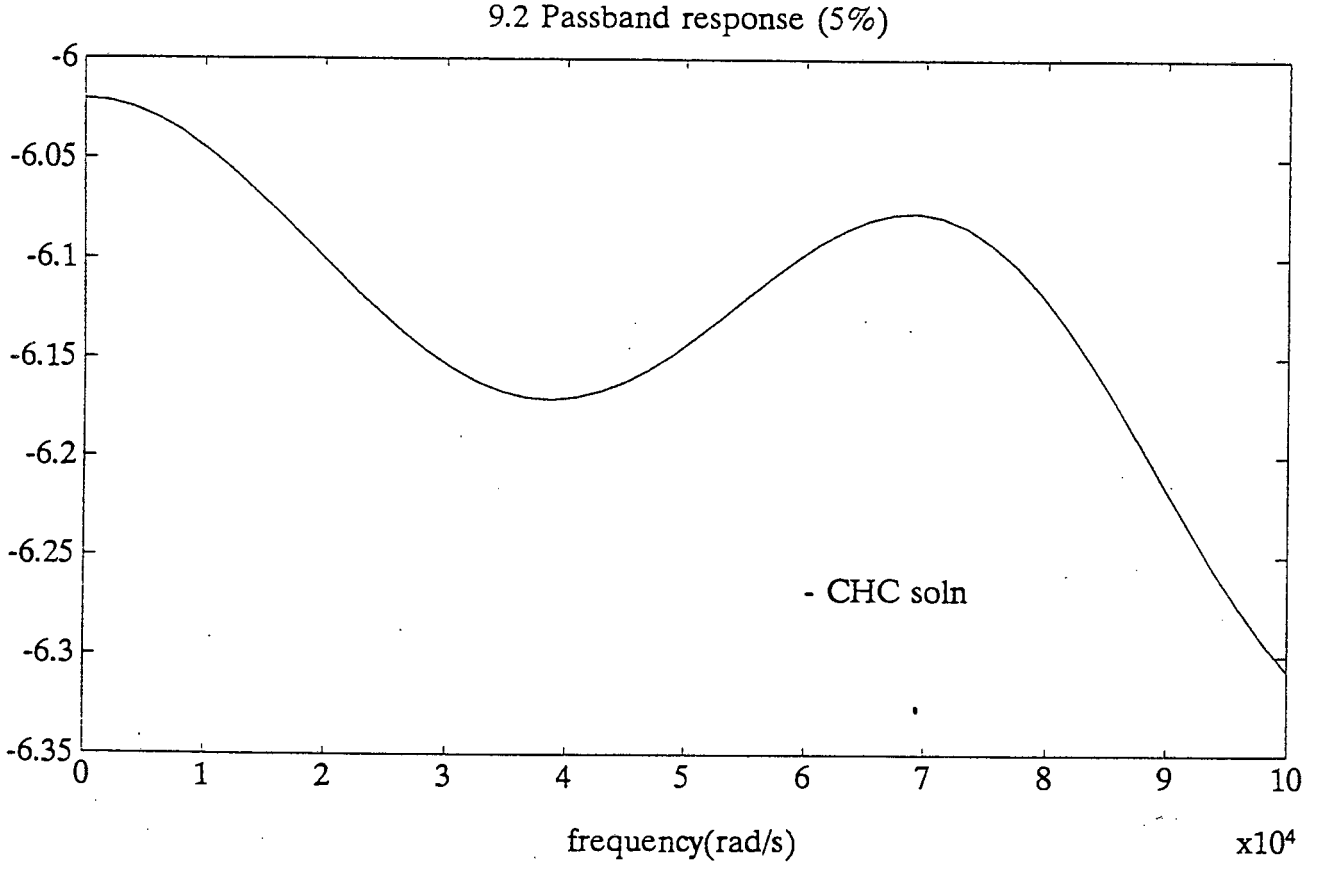

9.2 Robust solution from full factorial (3% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

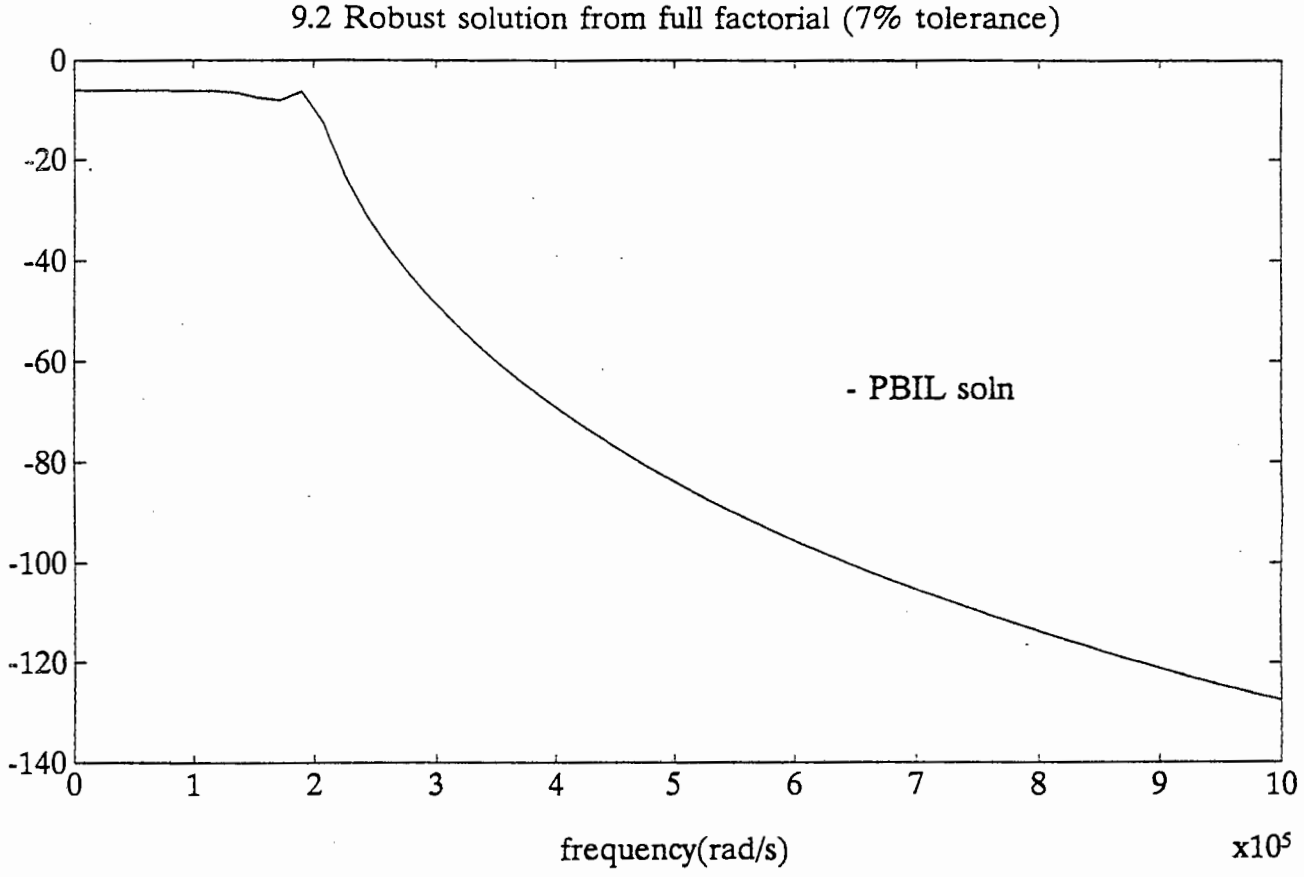

9.2 Passband response (3%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

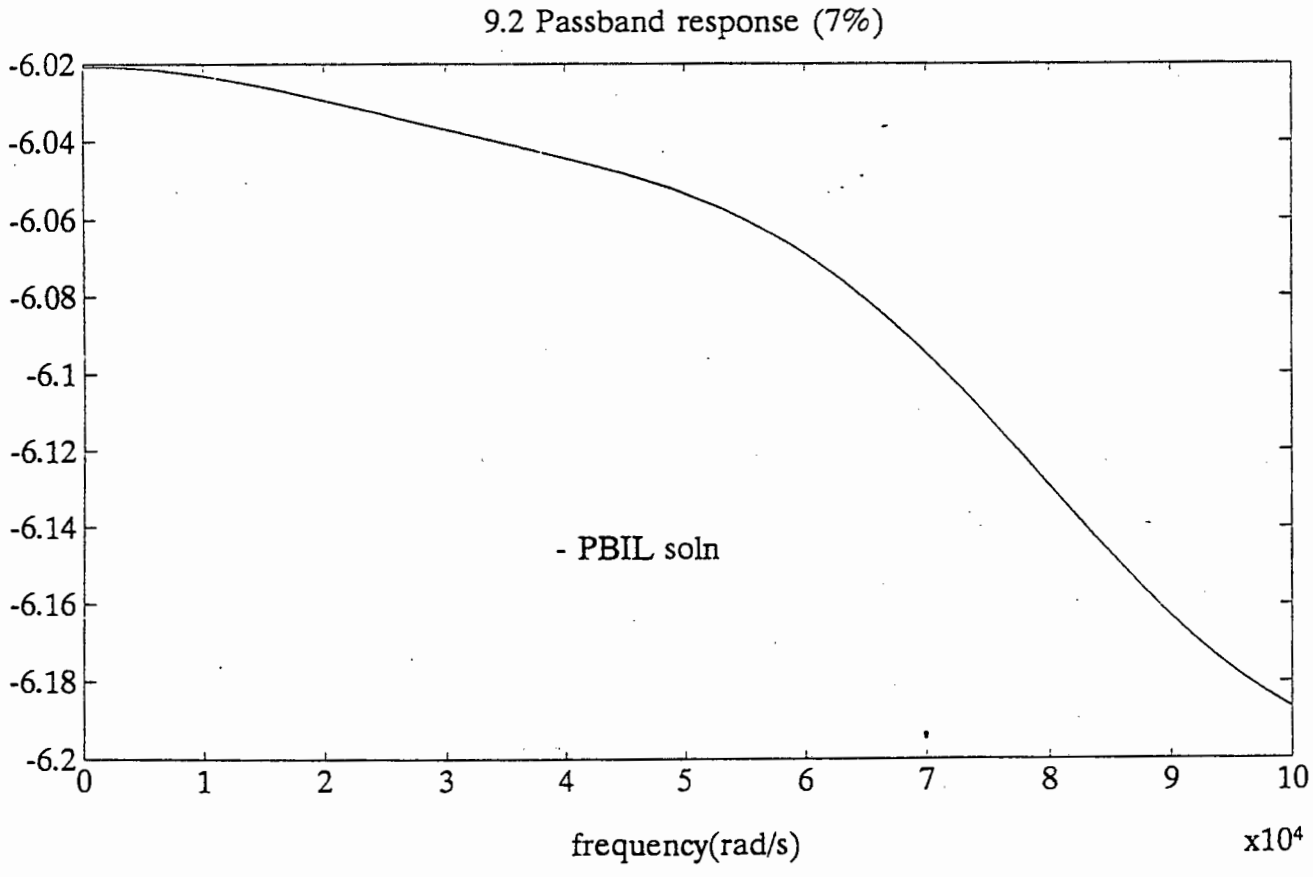

9.2 Robust solution from full factorial (5% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^5

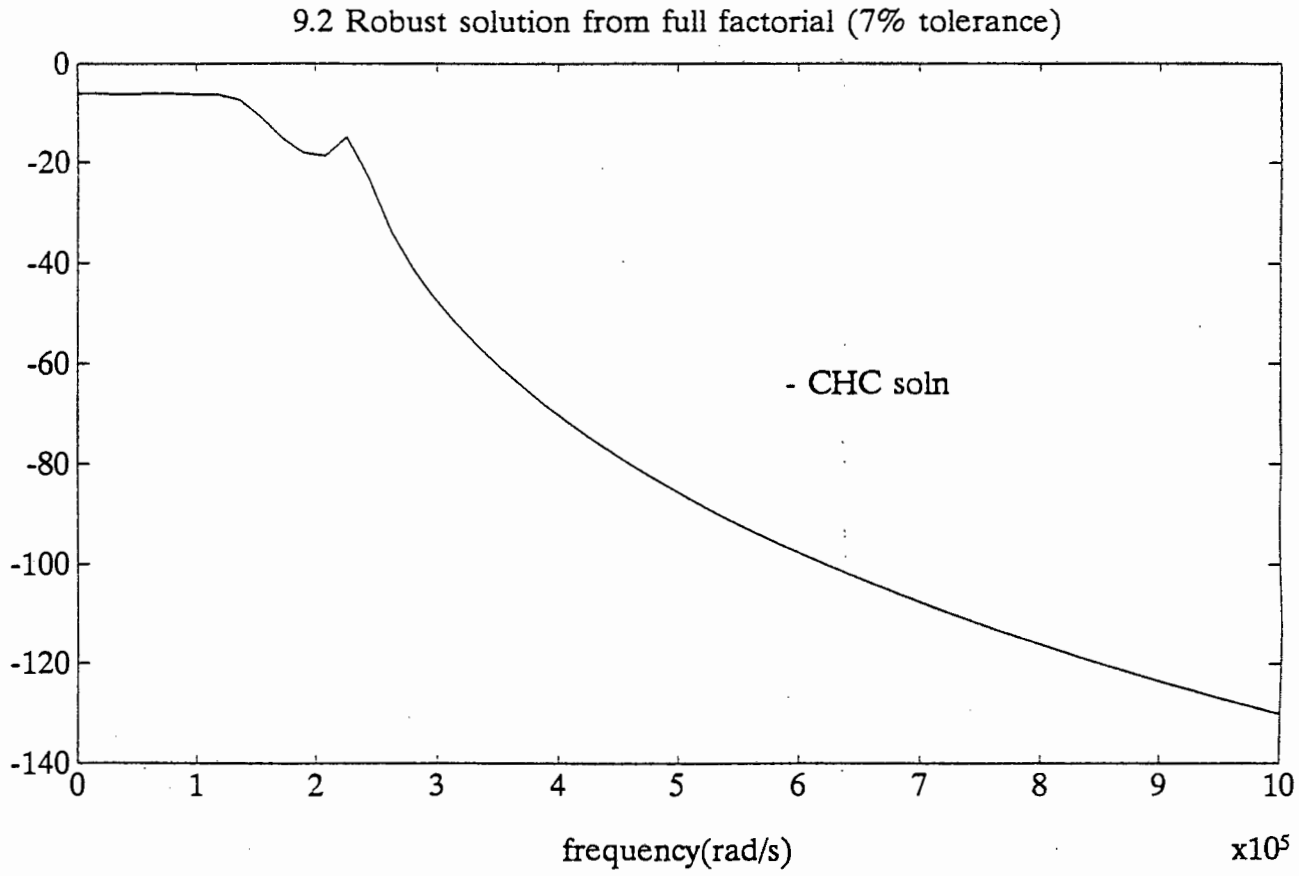

9.2 Passband response (5%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
-6.4
-6.45
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^4

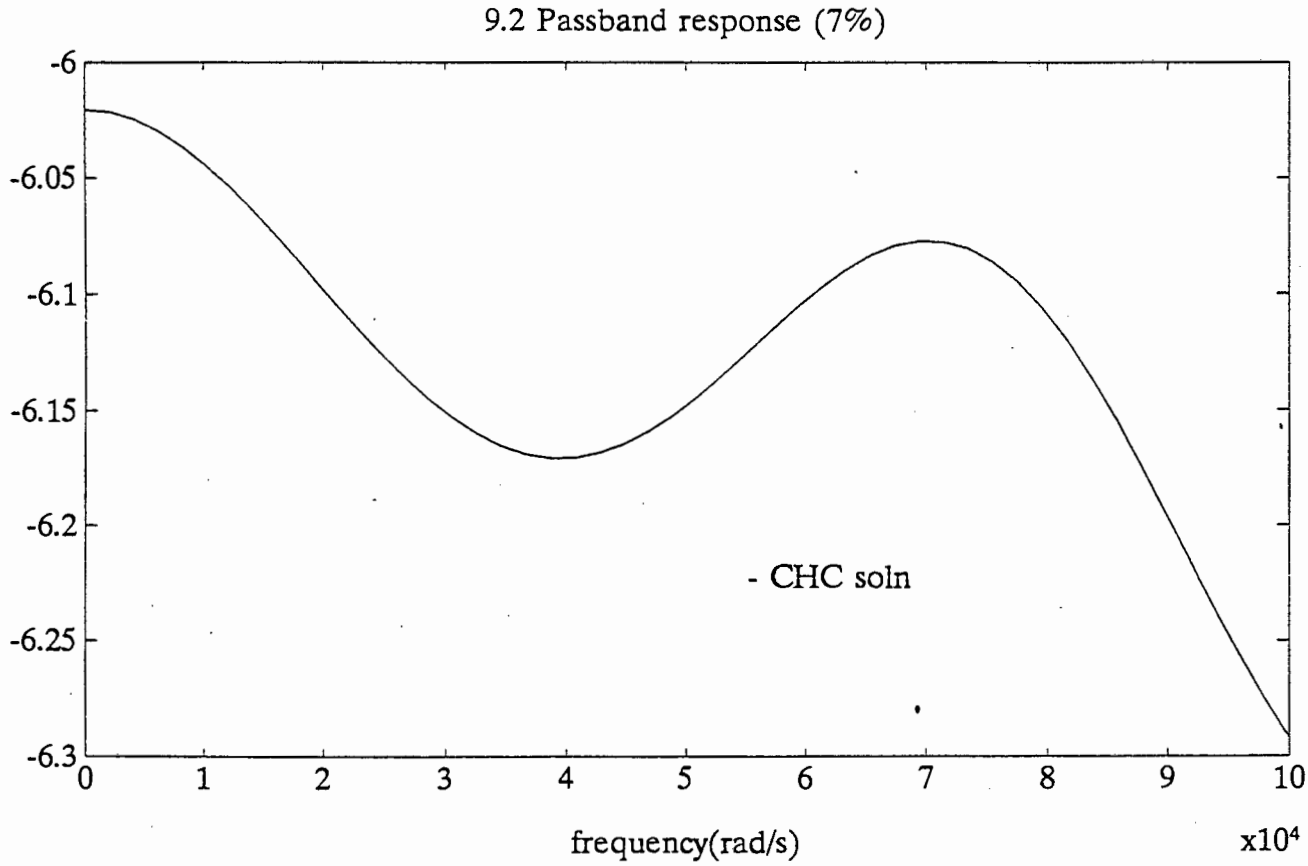

9.2 Robust solution from full factorial (5% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

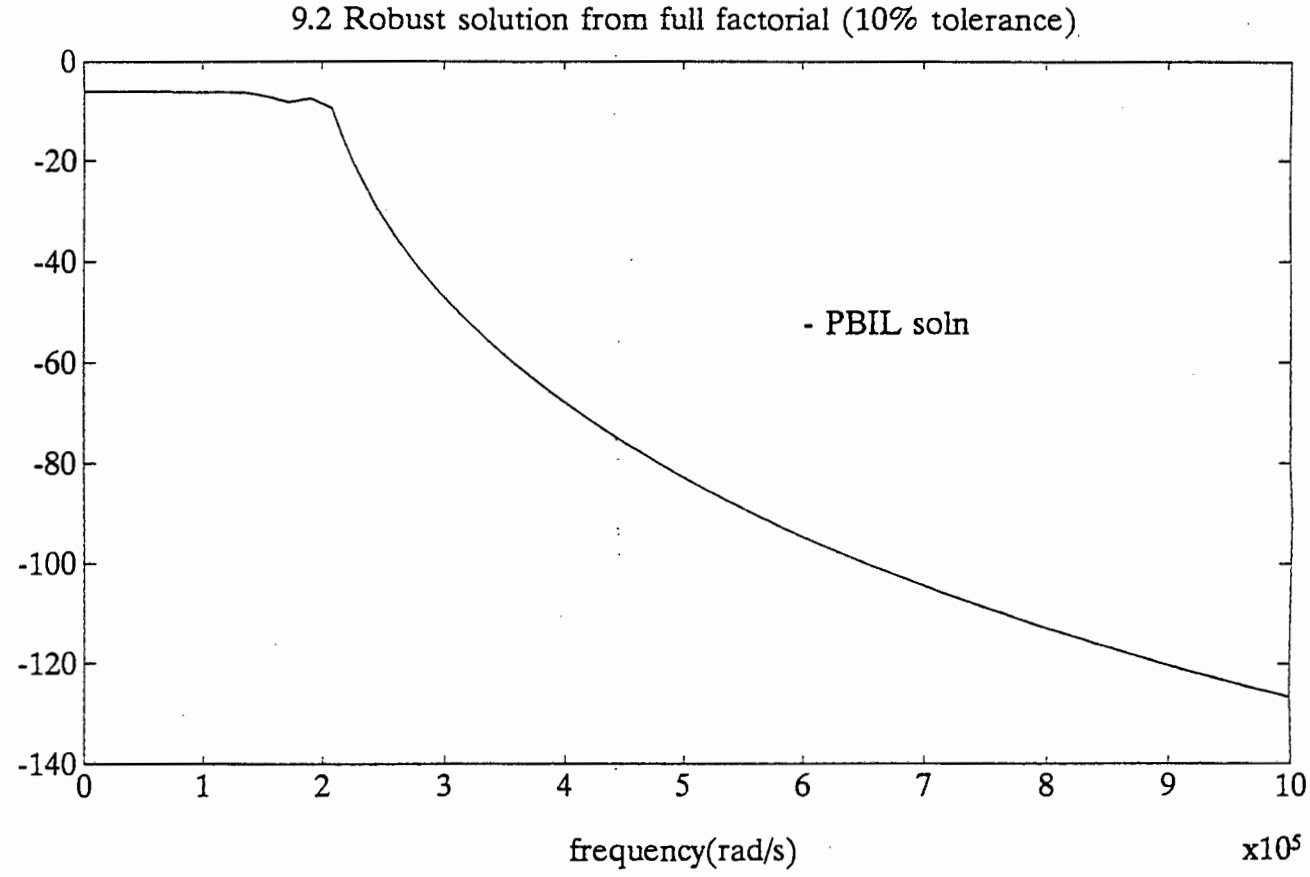

9.2 Passband response (5%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

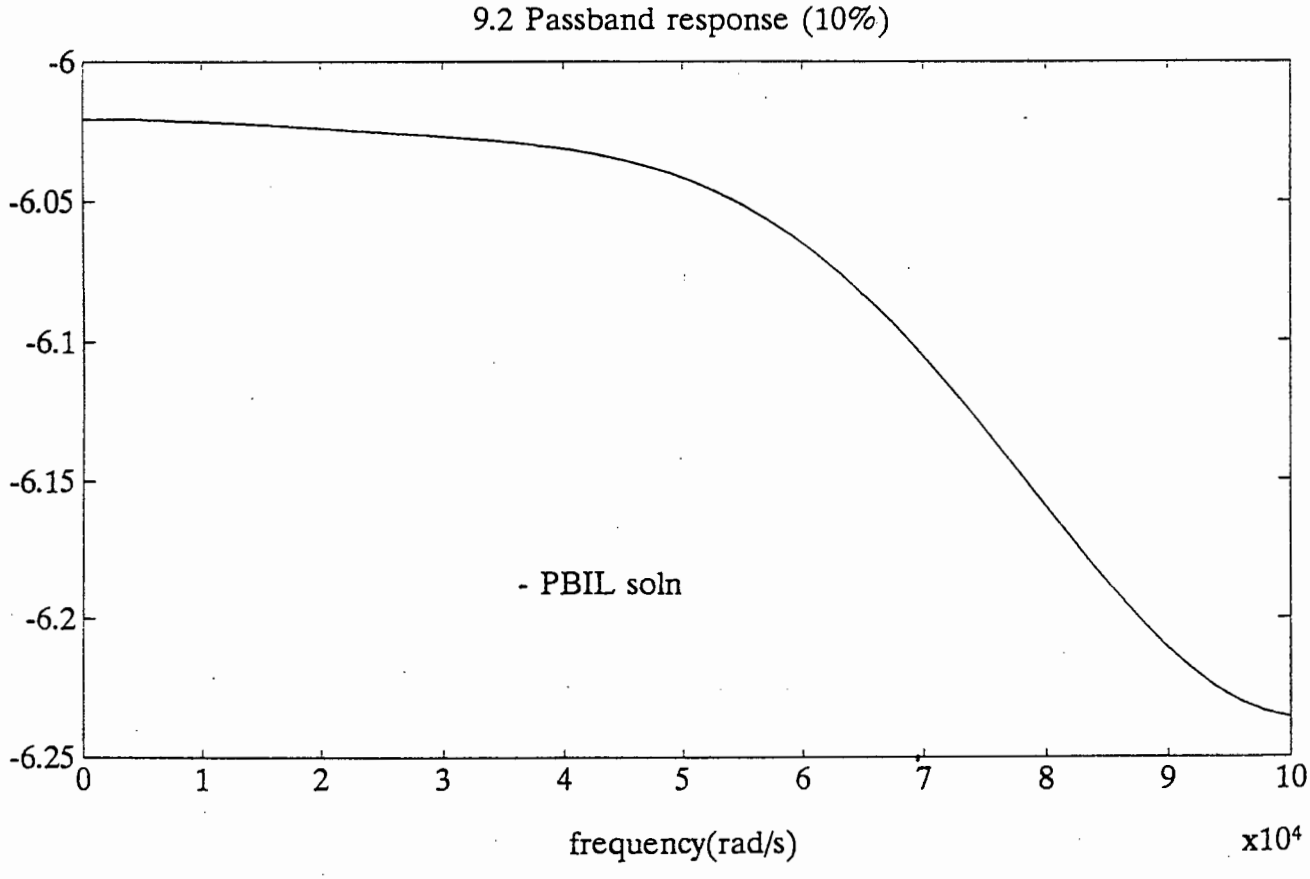

9.2 Robust solution from full factorial (7% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^5

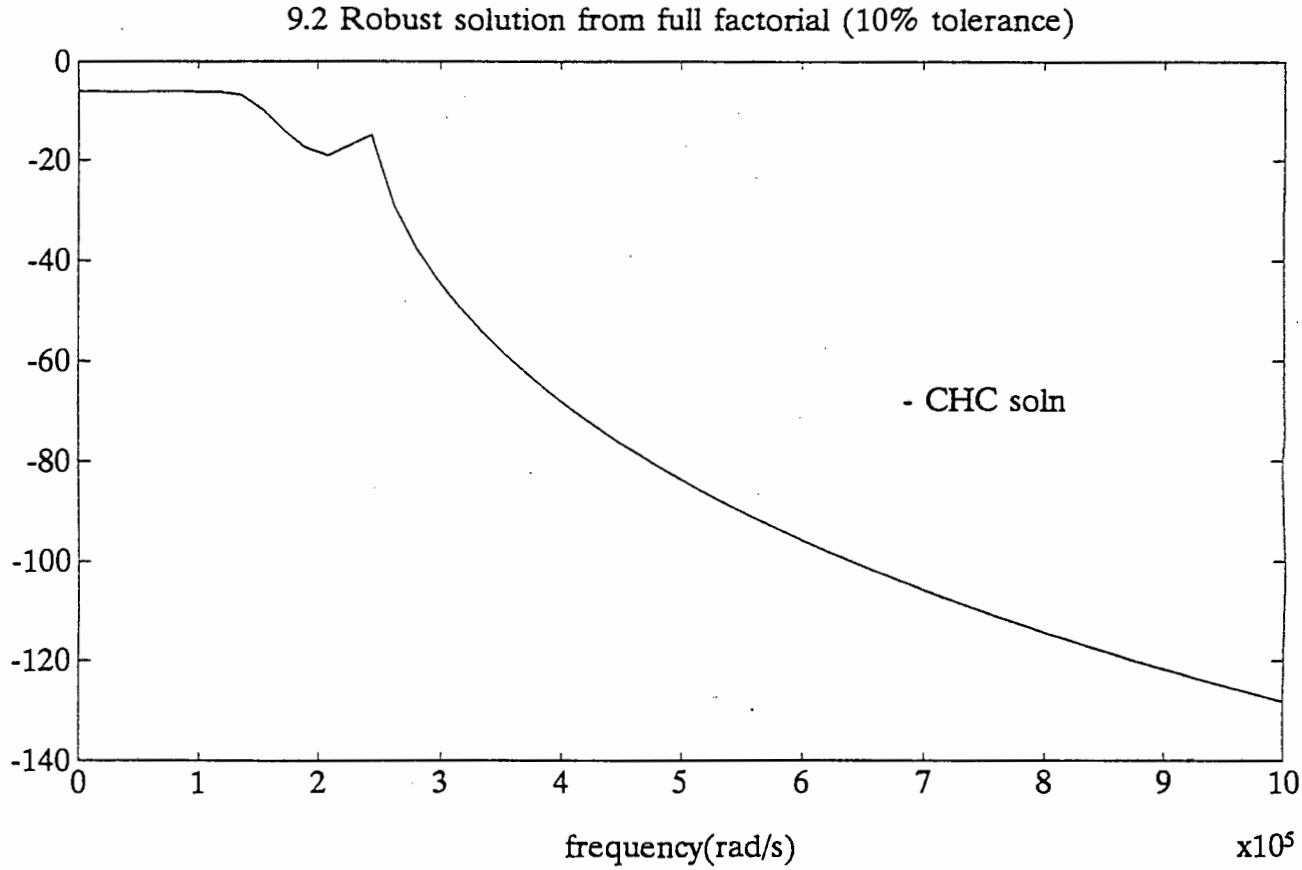

9.2 Passband response (7%)
-6.02
-6.04
-6.06
-6.08
-6.1
-6.12
-6.14
-6.16
-6.18
-6.2
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^4

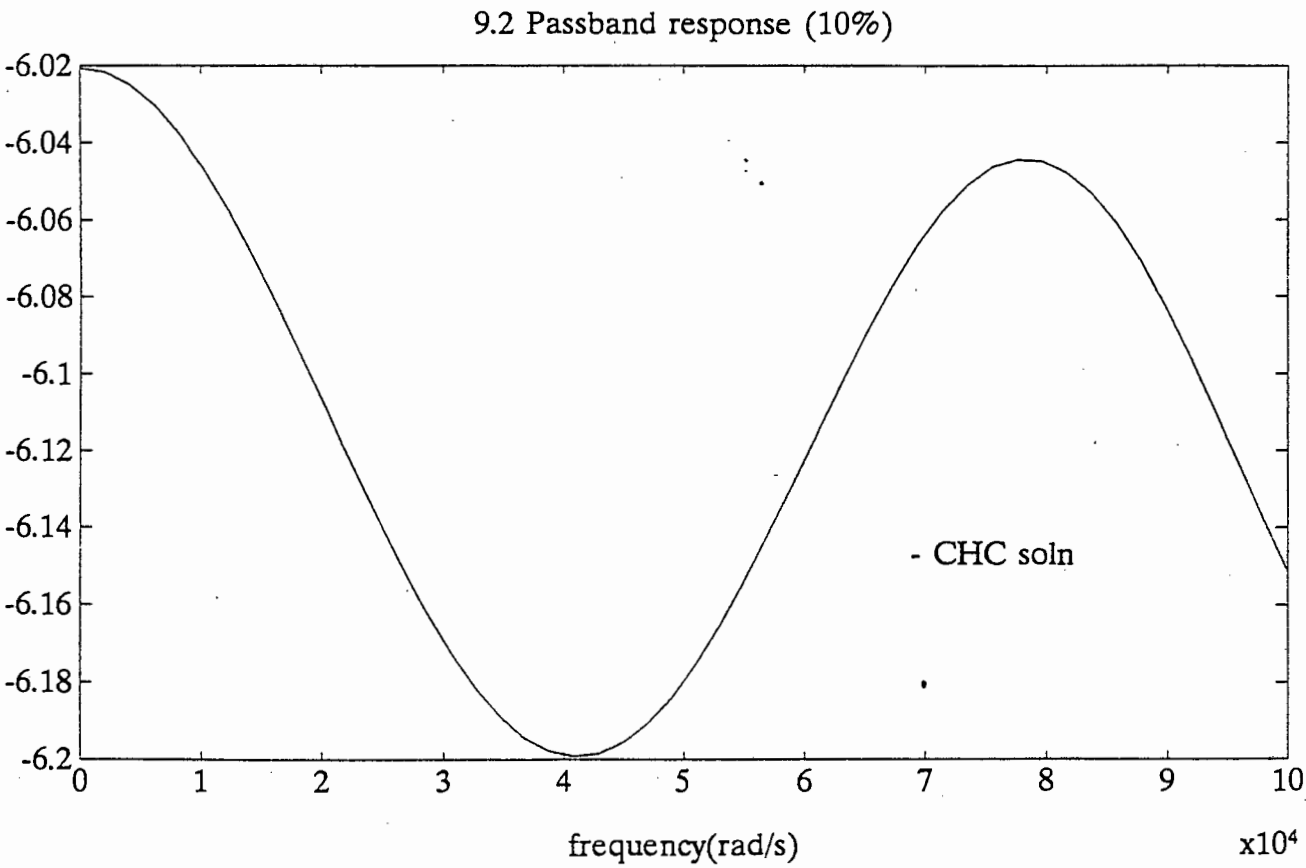

9.2 Robust solution from full factorial (7% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5


9.2 Passband response (7%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4


9.2 Robust solution from full factorial (10% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^5


9.2 Passband response (10%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
- PBIL soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^4


9.2 Robust solution from full factorial (10% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5


9.2 Passband response (10%)
-6.02
-6.04
-6.06
-6.08
-6.1
-6.12
-6.14
-6.16
-6.18
-6.2
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

Appendix E3: Hybrid PBIL and hybrid CHC using Taguchi method on the 7th order butterworth filter

```
'Genetically derived filter circuits using prefered component values' by
D.H. Horrocks & Y.M.A. Khalifa. This paper tackles the problem of an
all-pole low pass LC chebychev filter with 1 dB pass band ripple,pass band
edge at 1e5 rad/sec and stop band attention of -150 dB at stop band edge of
1e6 rad/sec.

unction [f,X]=rdp(B)

m n]=size(B);
=n/7;

1=0.001953125*btod(B(:,1:l));
2=0.001953125*btod(B(:,l+1:2*l));
3=0.001953125*btod(B(:,2*l+1:3*l));
4=0.001953125*btod(B(:,3*l+1:4*l));
5=0.001953125*btod(B(:,4*l+1:5*l));
6=0.001953125*btod(B(:,5*l+1:6*l));
7=0.001953125*btod(B(:,6*l+1:7*l));

Decade range for inductors (-6 to -2)
Decade range for capacitors (-10 to -6)

Resizing the component values onto the log. scale eg.[(-2- -6)*B1+-6]
A=[(4*B1-6) (4*B2-6) (4*B3-6) (4*B4-10) (4*B5-10) (4*B6-10) (4*B7-10)];

 R1 = 100*ones(1,m);
 R2 = 100*ones(1,m);
 L2 = 10 .^BA(:,1);
 L4 = 10 .^BA(:,2);
 L6 = 10 .^BA(:,3);
 C1 = 10 .^BA(:,4);
 C3 = 10 .^BA(:,5);
 C5 = 10 .^BA(:,6);
 C7 = 10 .^BA(:,7);

 X = [L2 L4 L6 C1 C3 C5 C7];

 err=campl(X);
 f=err';
```

```
function file used to calculate error of network for section 9.2
 function err=campl(c)

 [m n]=size(c);

 % Component values

 R1=100*ones(1,m);
 R2=100*ones(1,m);
 L2=c(:,1);
 L4=c(:,2);
 L6=c(:,3);
 C1=c(:,4);
 C3=c(:,5);
 C5=c(:,6);
 C7=c(:,7);

 w1=linspace(0.1,1e5,50);
 [p1 q1]=size(w1);
 w2=linspace(1e5,1e6,51);
 w0=[w1 w2(2:51)];
 [p q]=size(w0);
 s=i*w0;
 s=(s'*ones(1,m))';

 R1 = (R1'*ones(1,q));
 R2 = (R2'*ones(1,q));
 L2 = (L2*ones(1,q));
 L4 = (L4*ones(1,q));
 L6 = (L6*ones(1,q));
 C1 = (C1*ones(1,q));
 C3 = (C3*ones(1,q));
 C5 = (C5*ones(1,q));
 C7 = (C7*ones(1,q));

 % Symbolic representation of the NAM of the circuit

 Xa22=((1 ./R1)+(s .*C1)+(1 ./(s .*L2)));
 Xa33=((1 ./(s .*L2))+(s .*C3)+(1 ./(s .*L4)));
 Xa44=((1 ./(s .*L4))+(s .*C5)+(1 ./(s .*L6)));
 Xa55=((1 ./(s .*L6))+(s .*C7)+(1 ./R2));

 Xb13=-(((1 ./R1) .*(1 ./(s .*L2))) ./Xa22);
 Xb33=Xa33-(((1 ./(s .*L2)) .^2) ./Xa22);

 Xc14=(((1 ./(s .*L4)) .*Xb13) ./Xb33);
 Xc44=Xa44-(((1 ./(s .*L4)) .^2) ./Xb33);

 Xd51=(((1 ./(s .*L6)) .*Xc14) ./Xc44);
 Xd55=Xa55-(((1 ./(s .*L6)) .^2) ./Xc44);

 Ampl = -Xd51 ./ Xd55;

Calculating passband error above -6dB

 pba=(abs(Ampl(:,1:q1))-0.501187*ones(m,q1))';
 pba1=pba .*(pba>0);
 pband_ab=sum((pba1) .^2);

Calculating passband error below -7dB

 pbb=(abs(Ampl(:,1:q1))-0.446684*ones(m,q1))';
 pbb1=pbb .*(pbb<0);
 pband_bl=sum((pbb1) .^2);
```

```
Calculating total error in passband

 pband_err=pband_ab+pband_bl;

Calculating stopband error above -150dB at w=1e6 rad/s

 psb=(abs(Ampl(:,q))-3.162278e-8*ones(m,1))';
 psbl=psb .*(psb>0);
 sband_err=((psbl) .^2);

 err=1e10*pband_err+0.5e10*sband_err;
```

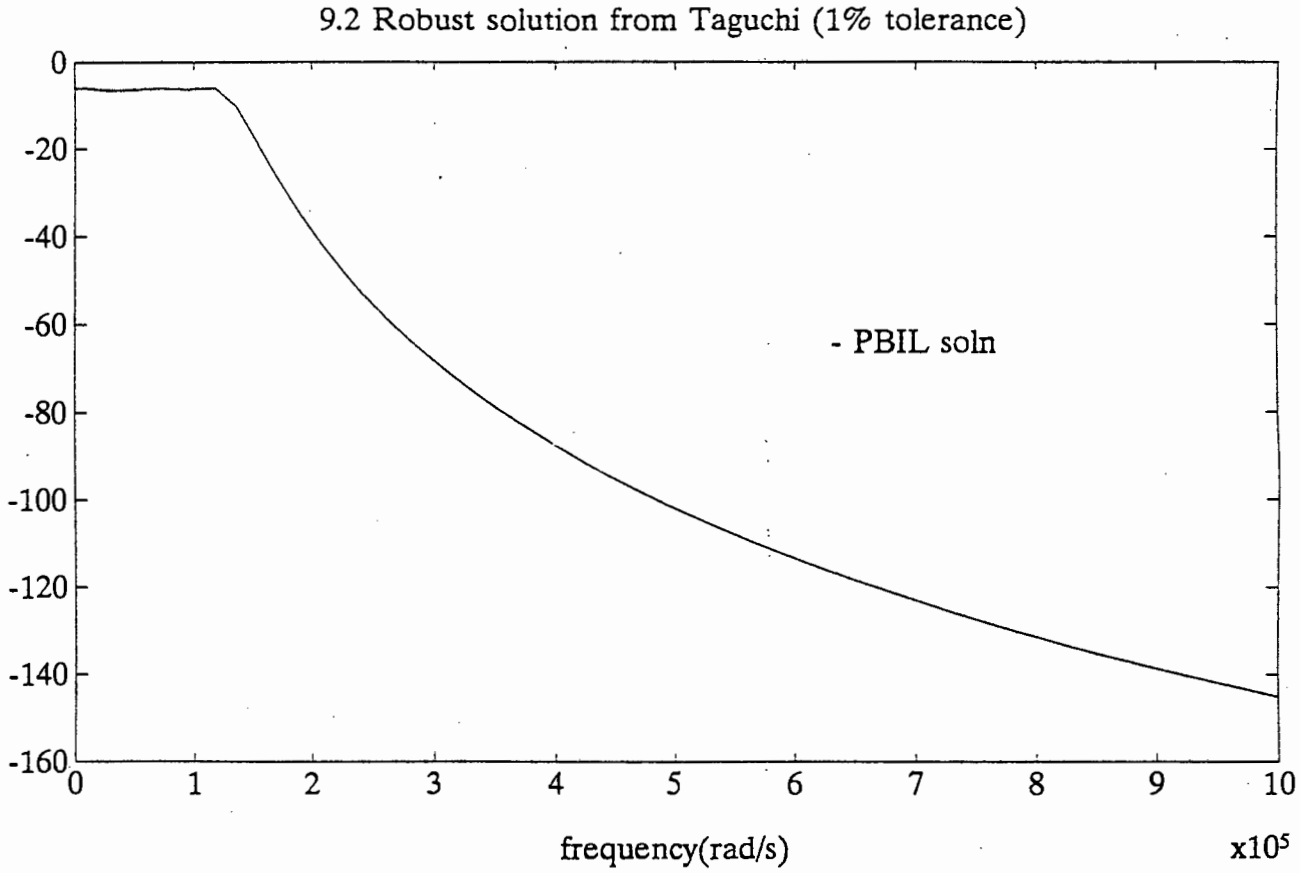
```
 Using population for robust analysis
 input
 X    -> unsorted real valued population
 F    -> unsorted fitnesses
 pert -> % tolerance
 output
 indx -> pointer of best soln from unsorted population
 indy -> pointer of best soln from sorted population
 bwfitness -> Robust fitness of population

unction [indx,indy,bwfitness] = spop(X,F,pert)

m n]=size(X);

un=8;      % no of combinations required by the Taguchi methods.
evels=2;   % no of levels required by the orthogonal matrix

 Sorting population in descending order.

F1 in]=sort(F);              % Sorting fitnesses in ascending order
2=flipud(F1);
2(m+1)=1e16;
=flipud(in);                 % indexing fitnesses in descending order
1=X(j,:);                    % Sorting real valued pop in descending order


 Initialisation

2=zeros(run*m,n);
rr=zeros(1,run*m);
eanerr=zeros(1,m);
ev=zeros(2,n);
ev1=zeros(1,n);
rrv=zeros(1,m+1);
rrp=zeros(1,m);
=1;
rrv(1,k)=inf;


hile errv(1,k)>=abs(F2(k))

f k<m+1

 Perturbing population with combination of worse cases ( from Taguchi )

2(run*(k-1)+1:run*k,:)=pertba(X1(k,:),pert);


 Evaluating network ( circuit )

rr(1,run*(k-1)+1:run*k)=campl(X2(run*(k-1)+1:run*k,:));
eanerr(1,k)=mean(err(1,run*(k-1)+1:run*k));


 Calculating level effects of each variable

ev=cfac(err(1,run*(k-1)+1:run*k),meanerr(1,k));


 Calculating the worst case combination determined by the Taguchi methods

ev1=lev - ( (mean(lev))'*ones(1,2) )';
le i]=max(lev1);
1=i;
```

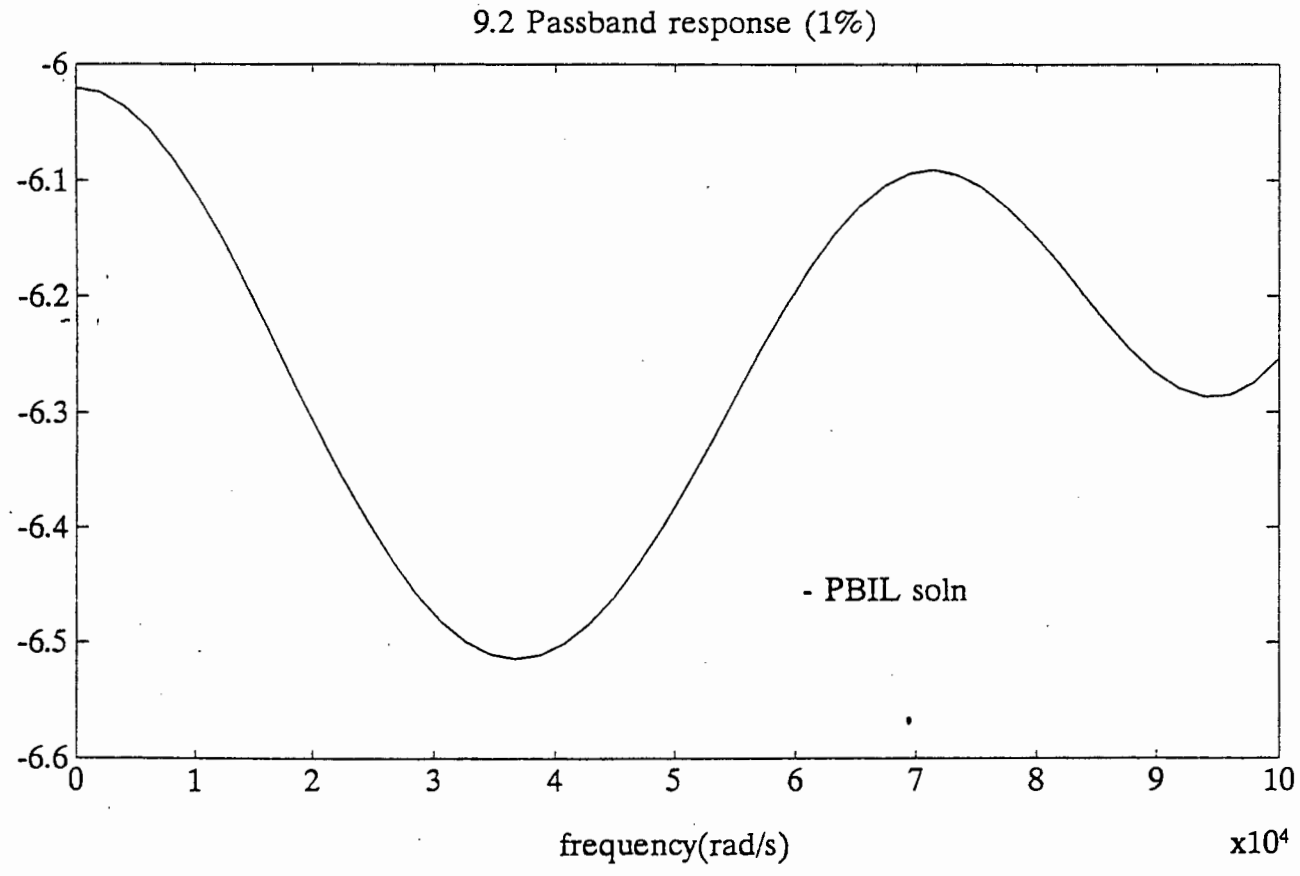
```
 Worst case combination ( in 1 / -1 format)

d=find(i1==2);
1(sd)=-ones(1,length(sd));
1=i1*pert;


 Validation experiment

3=X1(k,:)+X1(k,:) .*i1;
rrv(1,k)=campl(X3);


 Predicted experiment

2=i+[0:2:12];                          % 12 because 2 levels & 7 variables.
rrp(1,k)=meanerr(1,k)+sum(lev(i2));


 Testing the additivity principle

f abs(errv(1,k)-errp(1,k))/errv(1,k) > 0.05
4=pertba1(X1(k,:),pert);          % If additivity fails, do an exhaustive
rrv1=campl(X4);                   % search
rrv(1,k)=max(errv1);
nd

nd
=k+1;
rrv(1,k)=errv(1,k-1);

nd


 Finding robust soln from X2, hence finding the best design from the worst
 case for population X

bwfitness,indy]=min(errv(1:k-1));
wfitness=-bwfitness;


 Finding pointer of best soln for unsorted population

ndx=j(indy);
```

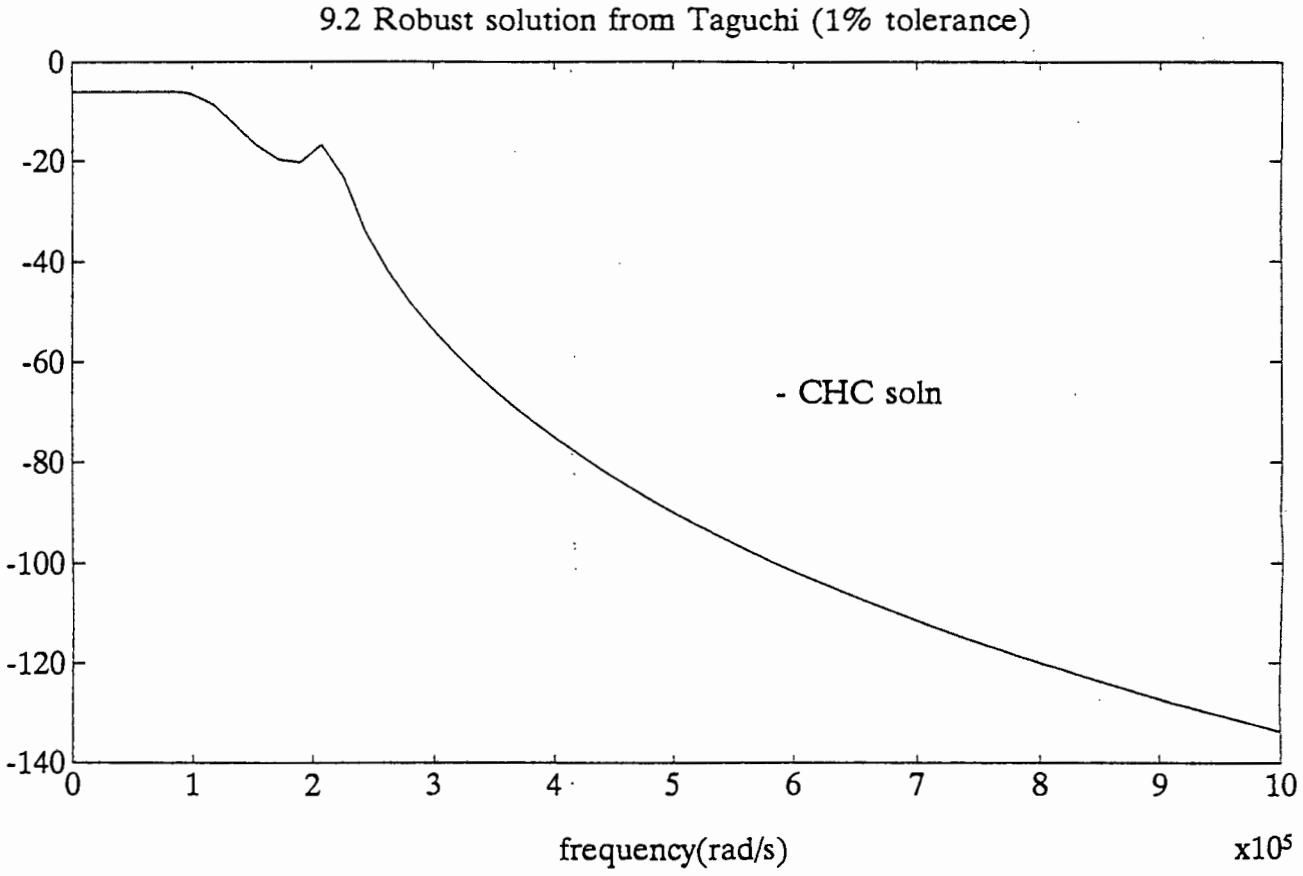
```
 Perturbation of the component values are done on whole population.
 input
 P     -> unsorted real valued member of population
 pert -> % tolerance
 output
 P2    -> unsorted real valued member of pop perturbed by m combinations
          from Taguchi orthogonal matrix.

unction P2 = pertba(P,pert)

m n]=size(P);

=omx;
m1 n1]=size(B);
 =B*pert;
-1=(P'*ones(1,m1))';
-2=P1+P1 .*B;       % Perturbing optimal solution for all cases
```

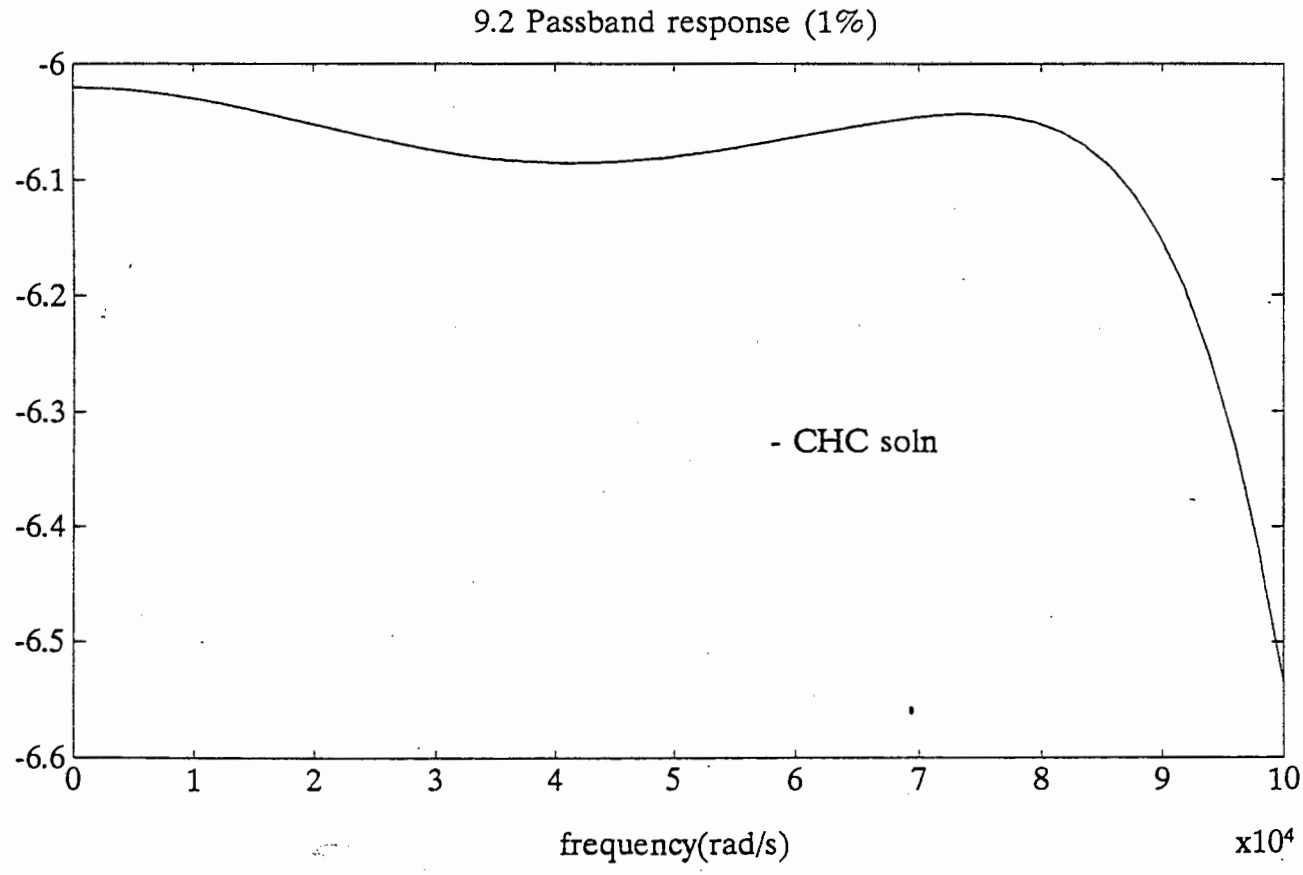
```
 File containing a Taguchi orthogonal mx to test for tolerance in
 neighbourhood region. For our problem, no. of degrees of freedom =
 7*(2-1)+1 =8
 7      -> no. of variables
 (2-1) -> 2-level design variable, interested in 1 comparison only
 1      -> overall mean
 omx obtained from the book 'intro to quality eng.' by Genich. Taguchi
 p182 ( L8 2^7 ).

unction B=omx()

=[ 1  1  1  1  1  1  1;
   1  1  1 -1 -1 -1 -1;
   1 -1 -1  1  1 -1 -1;
   1 -1 -1 -1 -1  1  1;
  -1  1 -1  1 -1  1 -1;
  -1  1 -1 -1  1 -1  1;
  -1 -1  1  1 -1 -1  1;
  -1 -1  1 -1  1  1 -1];
```

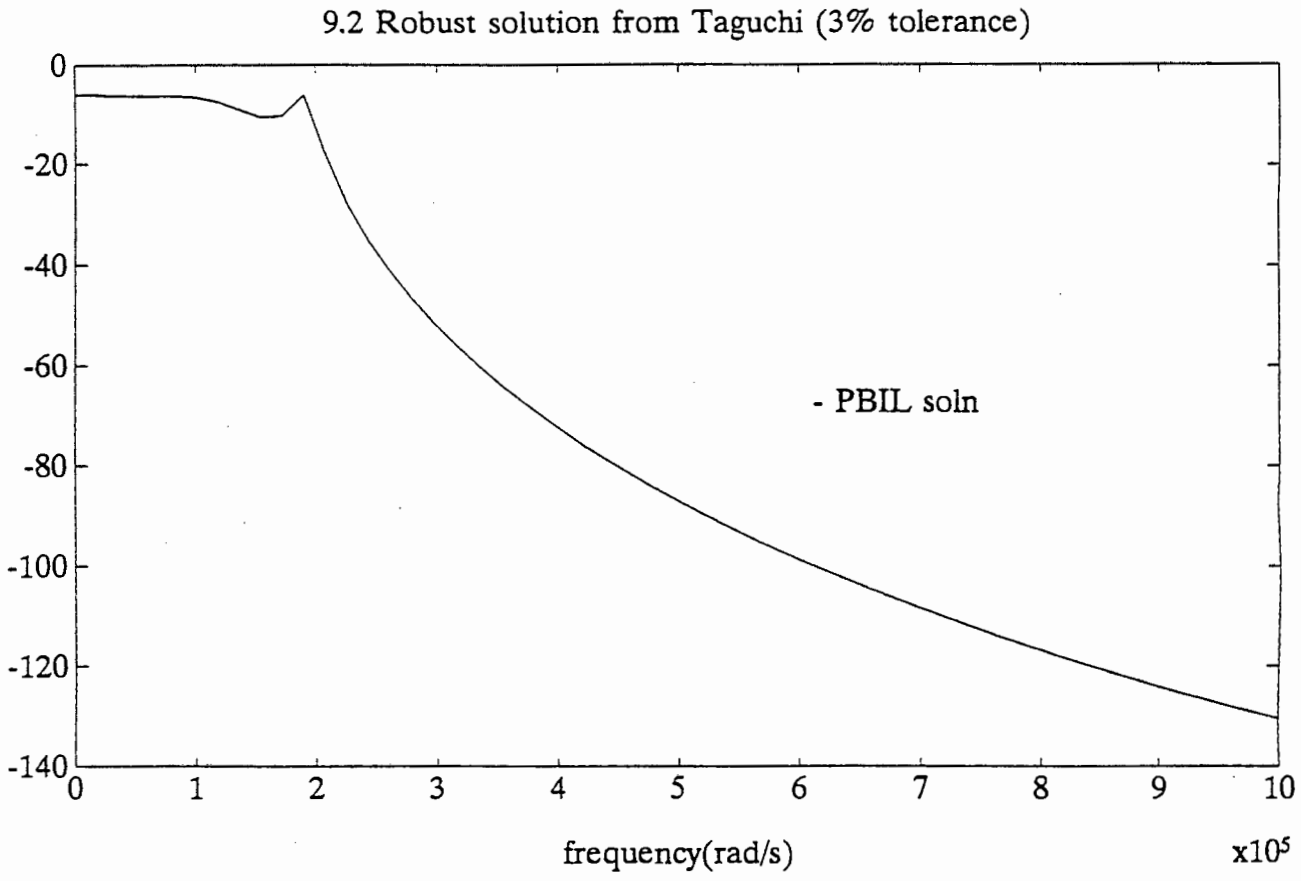
```
 file calculating the factor effects for all variables from Taguchi
 orthogonal matrix theory.
 input
 err     -> Result for member of population with all required Taguchi's run
 meanerr -> mean result of err for each member of population
 output
 lev     -> matrix output of factor effects for all variables with rows
            being variables and columns being levels

unction lev=cfac(err,meanerr)

ar=7; % No of components used
=omx; % Taguchi orth mx.
rr=(err'*ones(1,var));
rr1=(B>0) .*err;
ev(1,:)=(sum(err1) ./sum(B>0))-meanerr; % Calc for level 1 ( > 0 )
rr2=(B<0) .*err;
ev(2,:)=(sum(err2) ./sum(B<0))-meanerr; % Calc for level 2 ( < 0 )
```

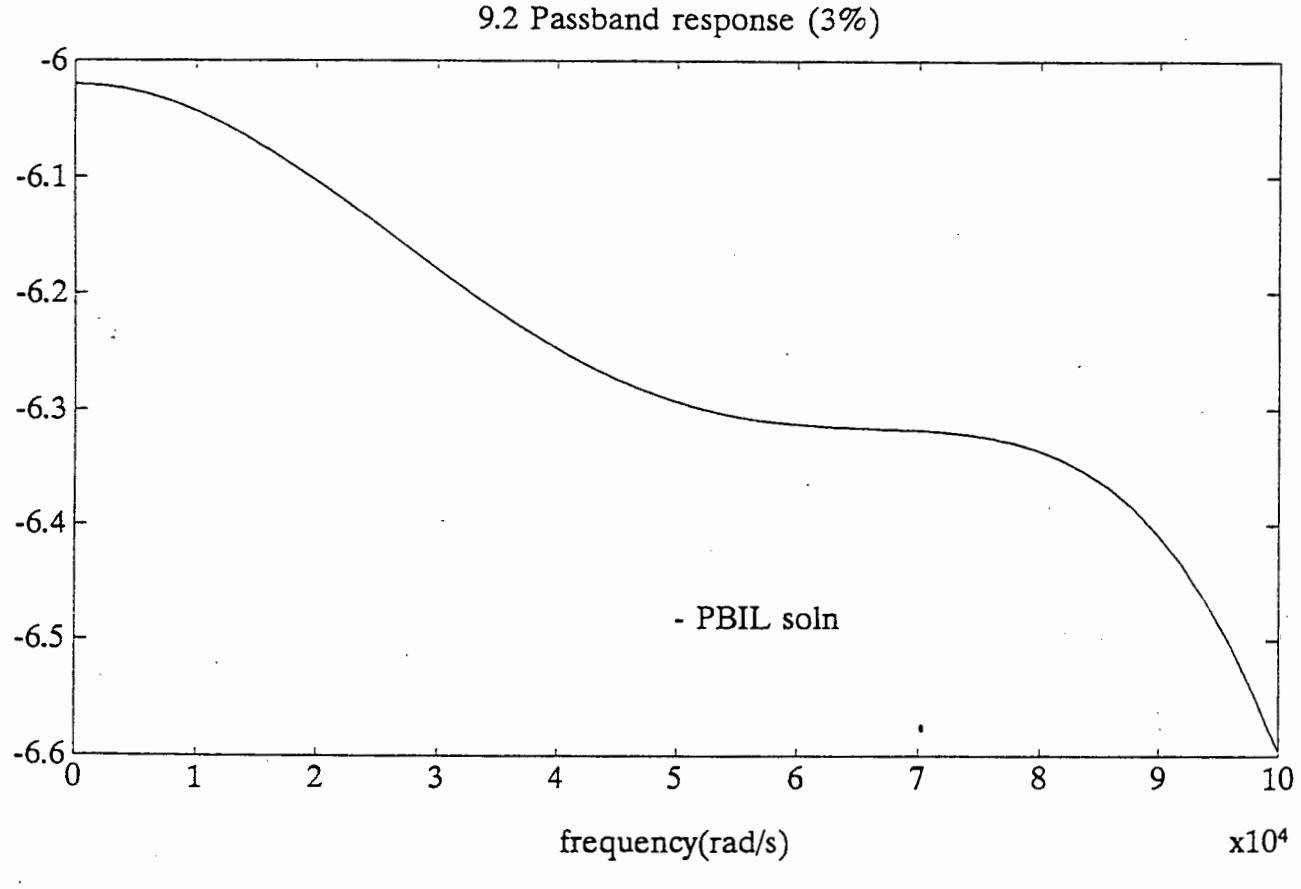

9.2 Robust solution from Taguchi (1% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
-160
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

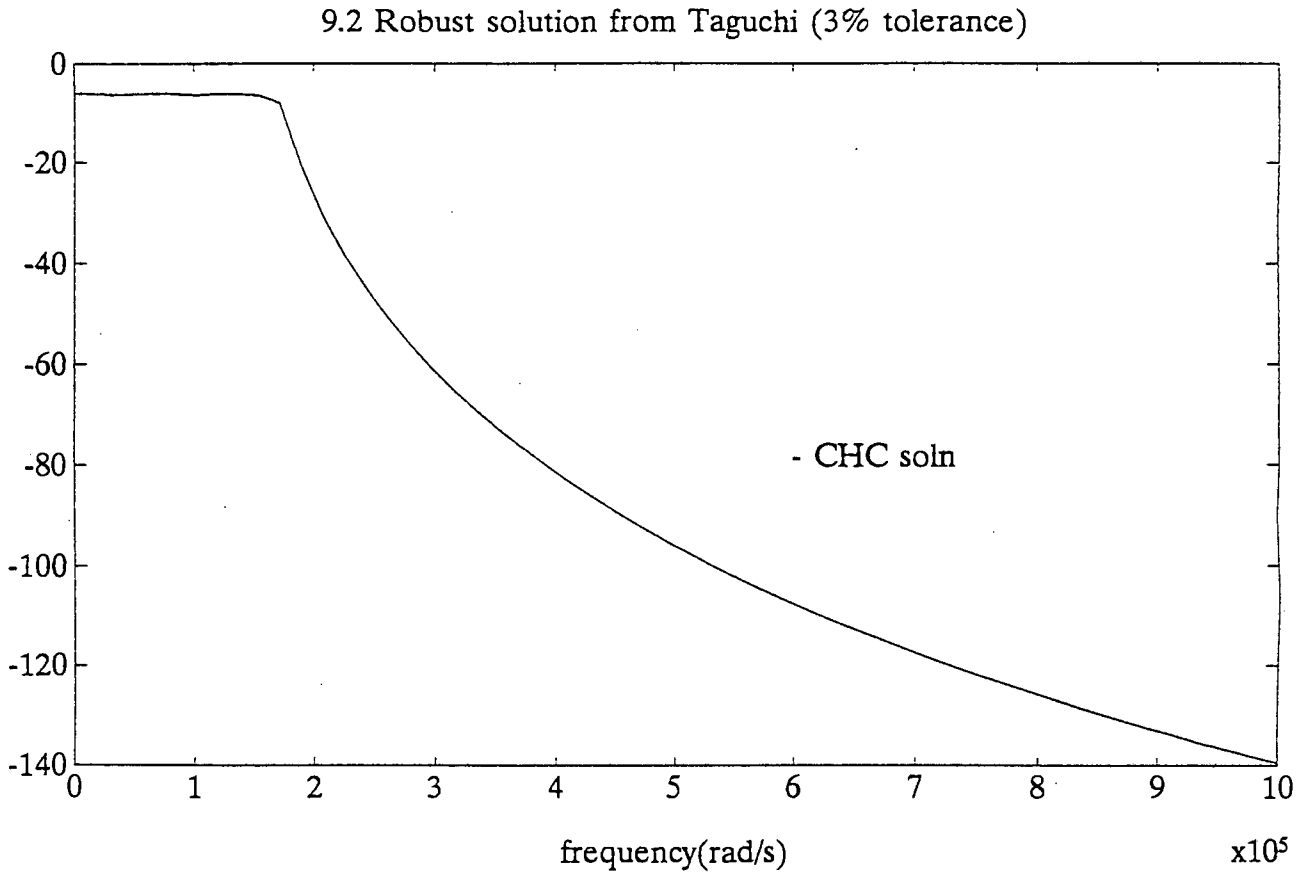

9.2 Passband response (1%)
-6
-6.1
-6.2
-6.3
-6.4
-6.5
-6.6
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

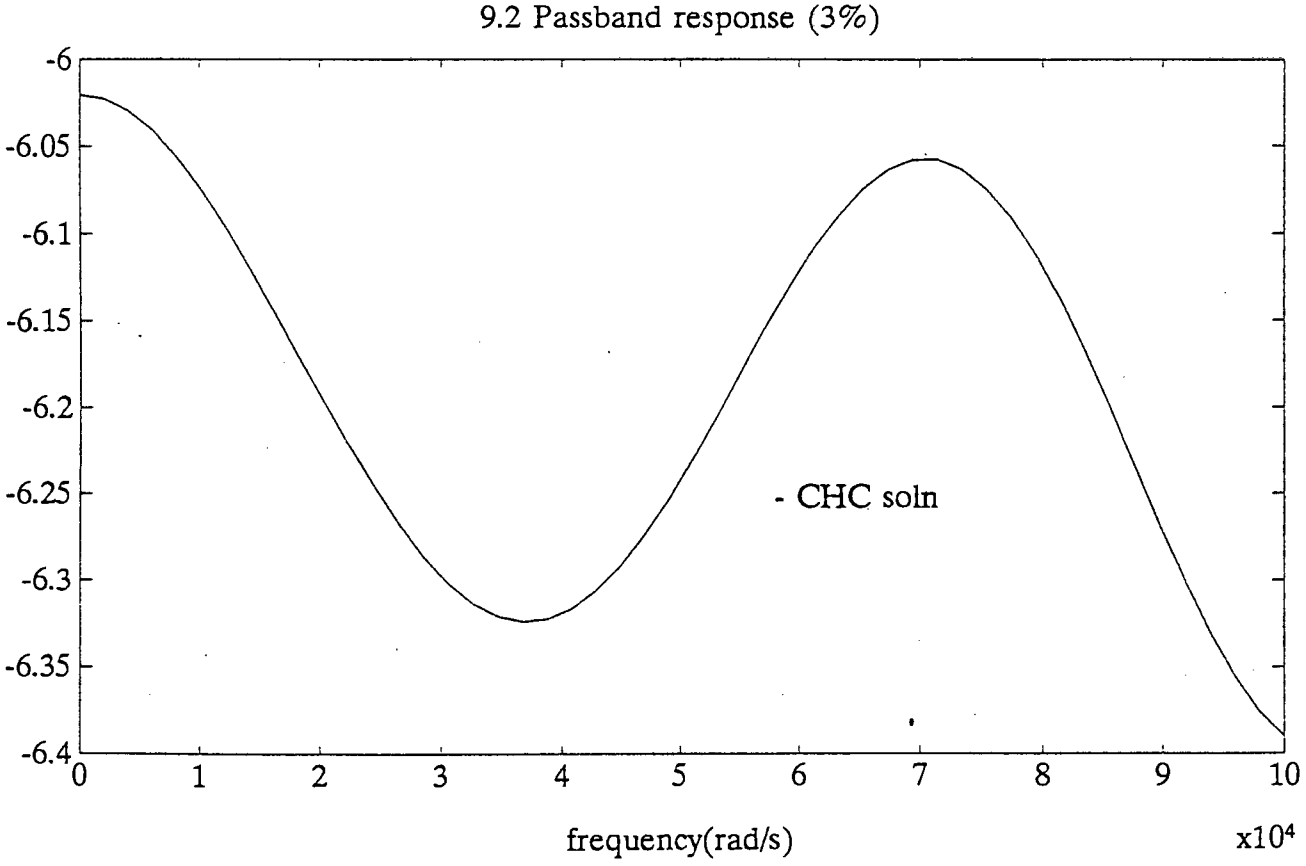

9.2 Robust solution from Taguchi (1% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

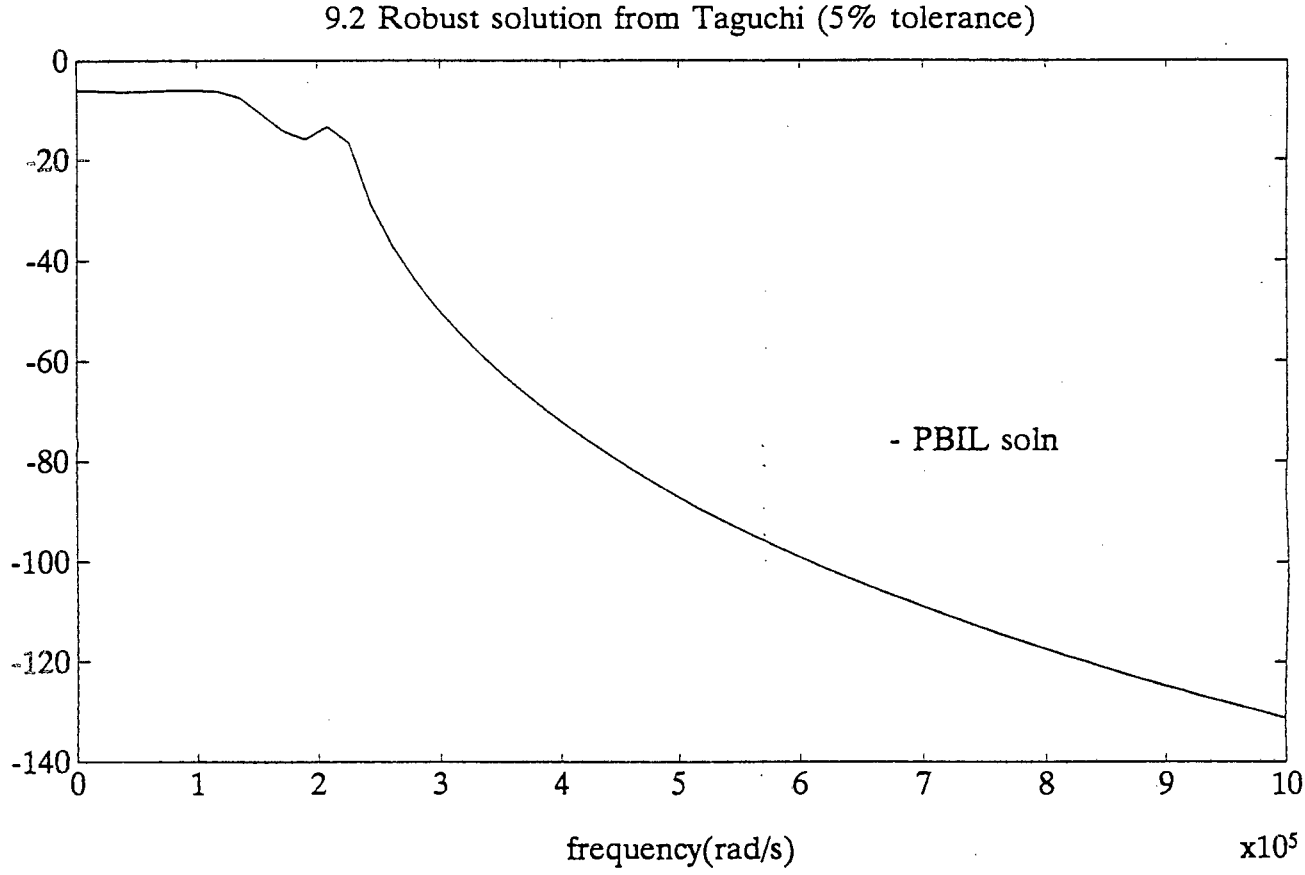

9.2 Passband response (1%)
-6
-6.1
-6.2
-6.3
-6.4
-6.5
-6.6
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

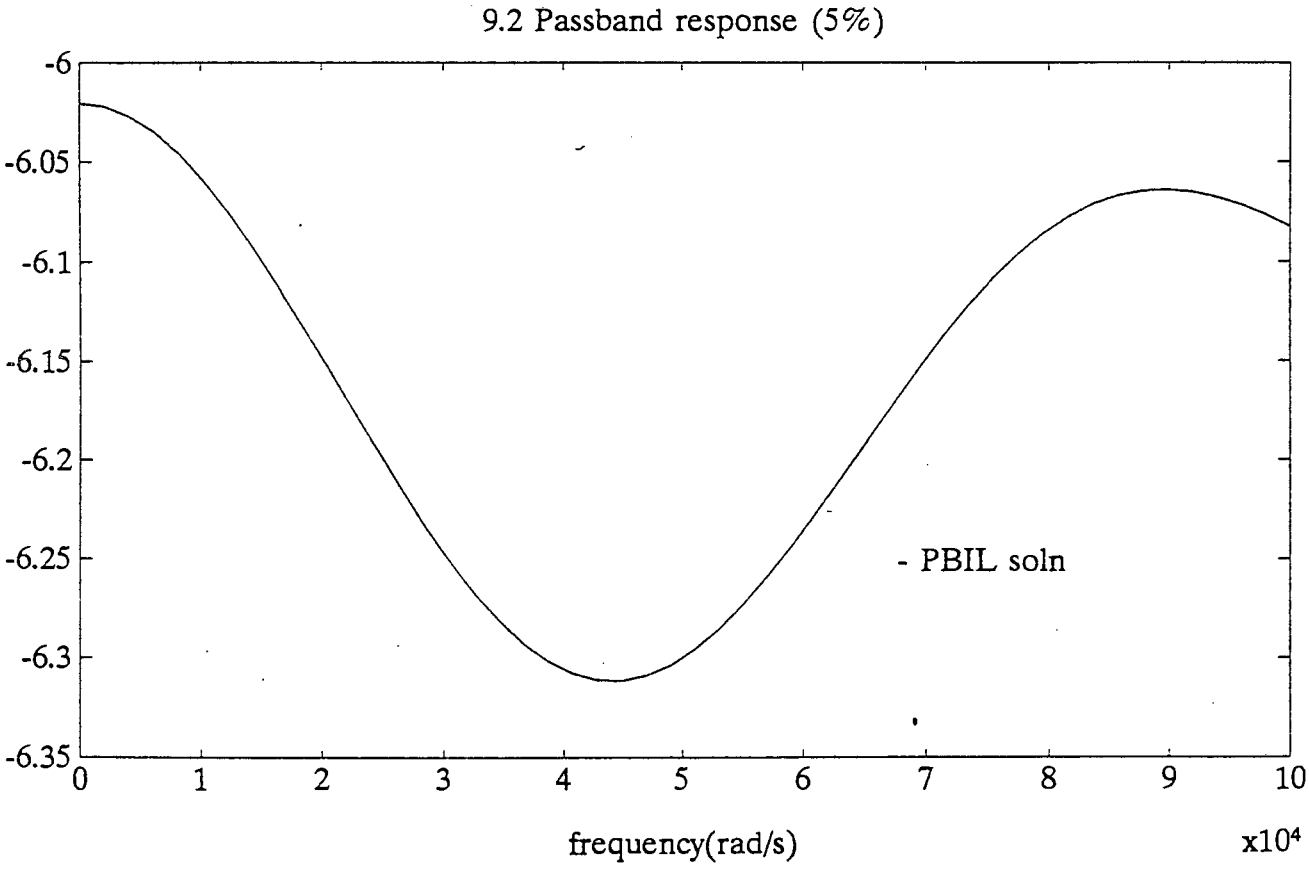

9.2 Robust solution from Taguchi (3% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

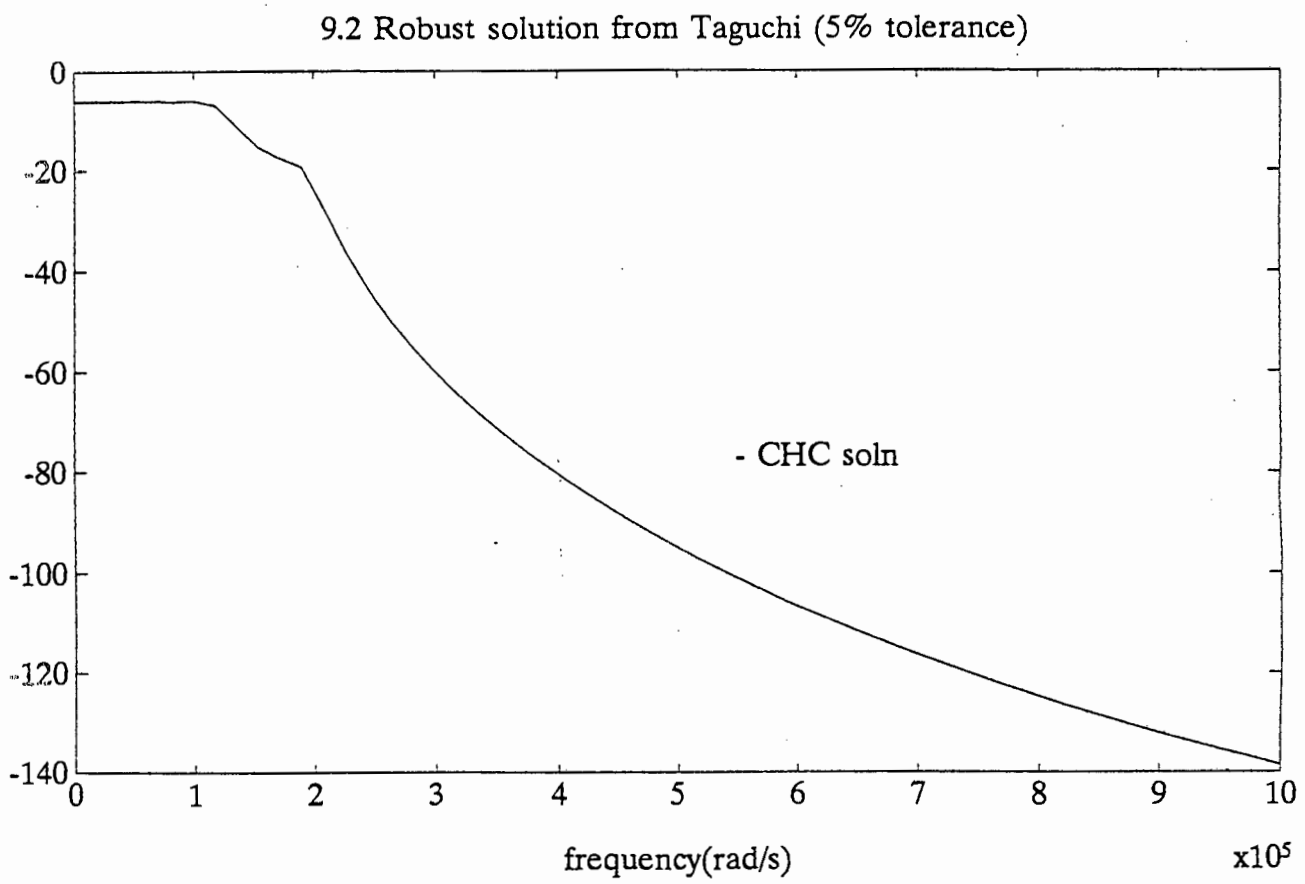

9.2 Passband response (3%)
-6
-6.1
-6.2
-6.3
-6.4
-6.5
-6.6
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

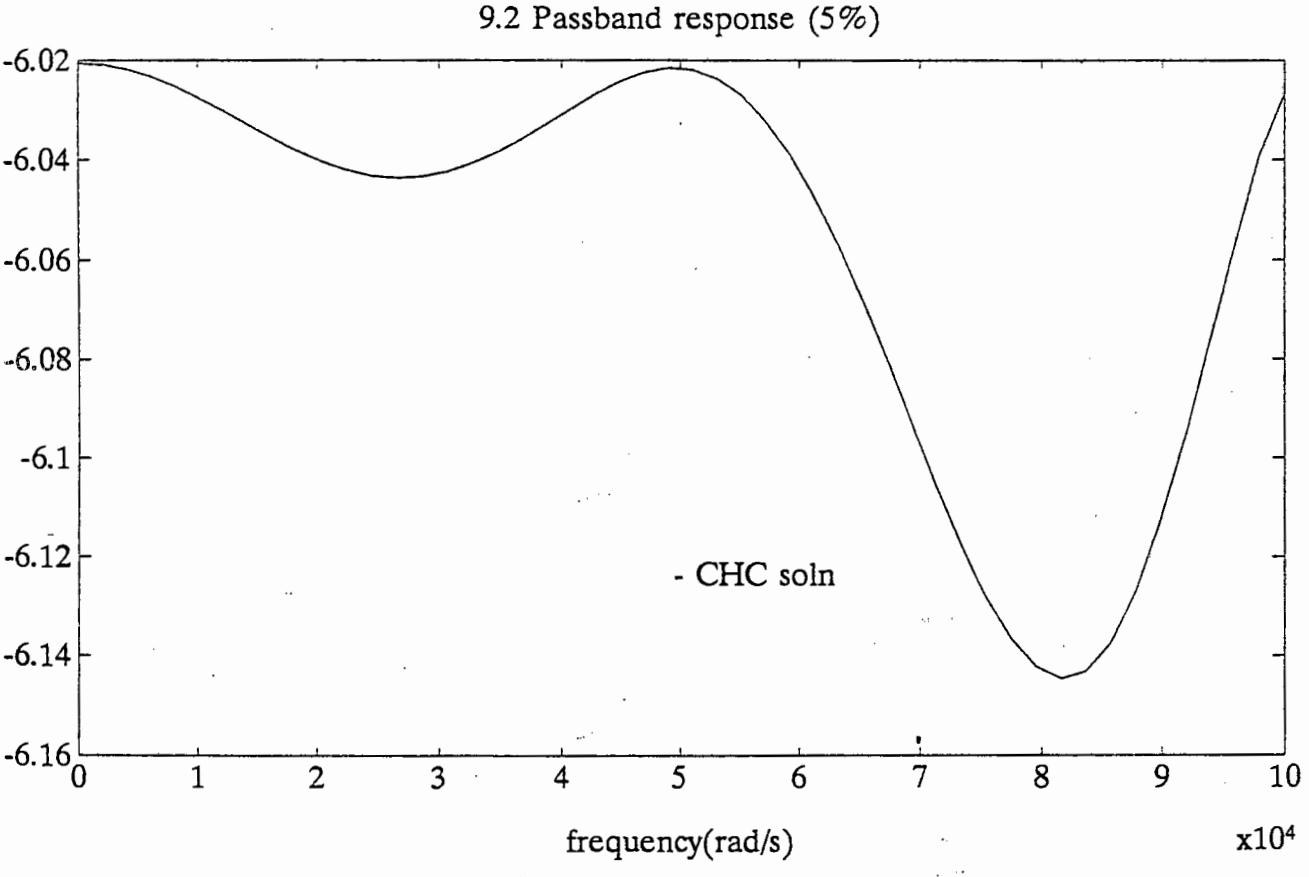

9.2 Robust solution from Taguchi (3% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

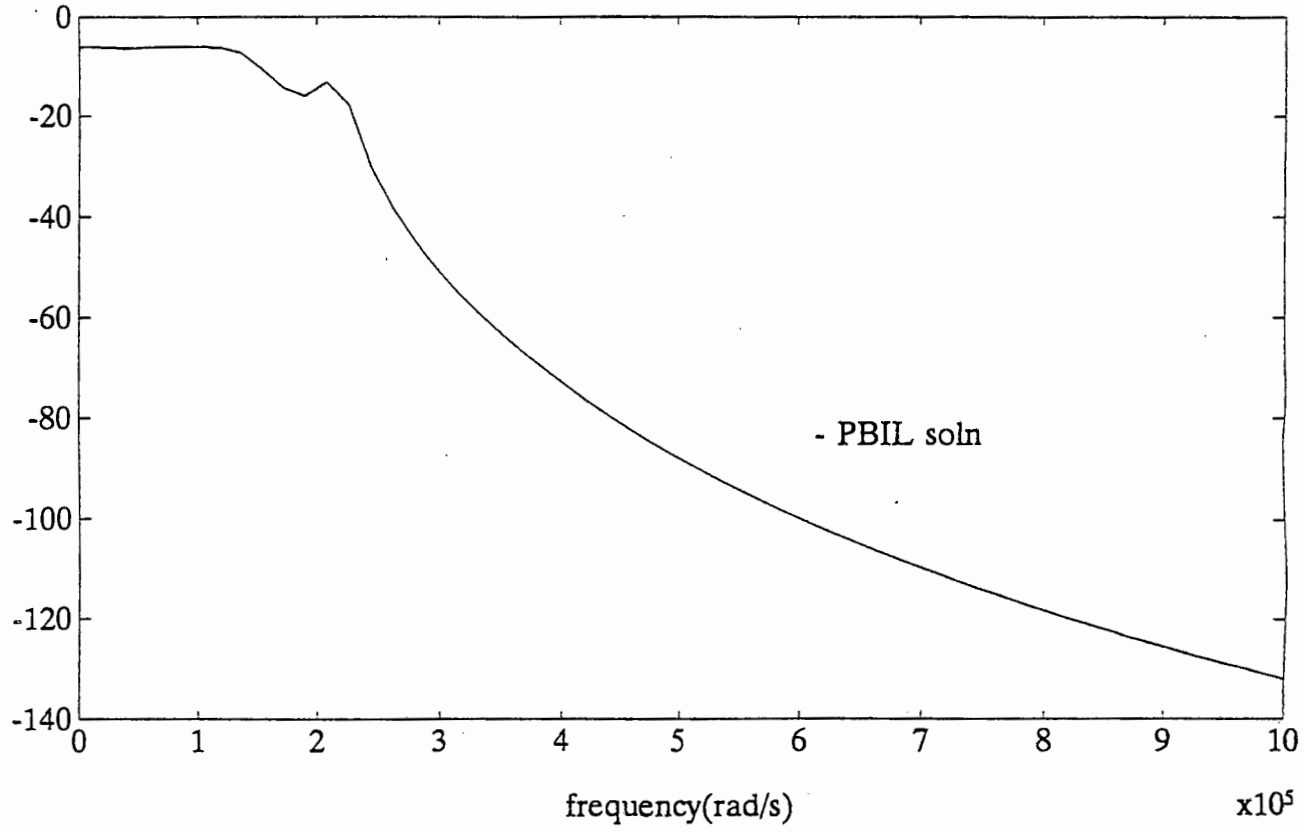

9.2 Passband response (3%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
-6.4
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

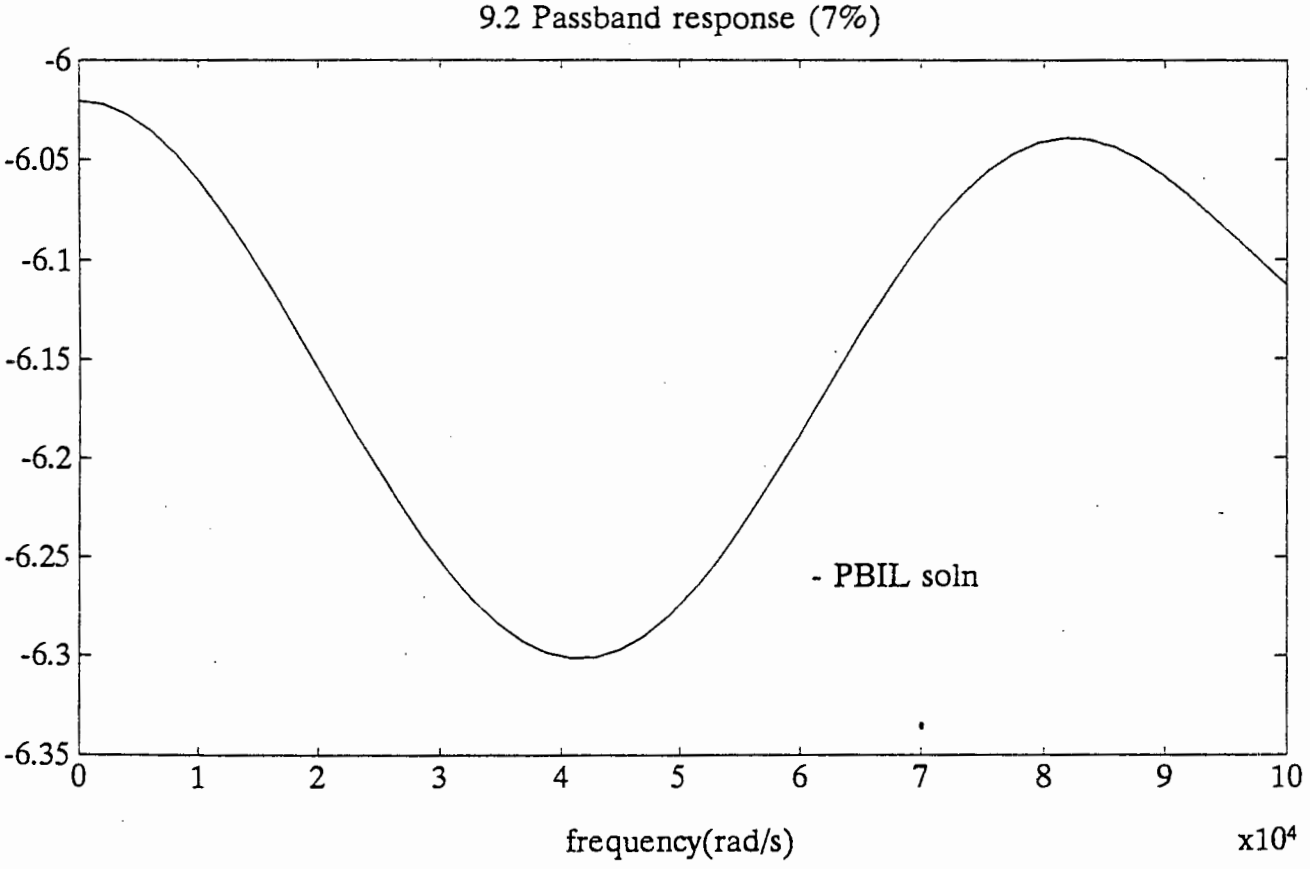

9.2 Robust solution from Taguchi (5% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

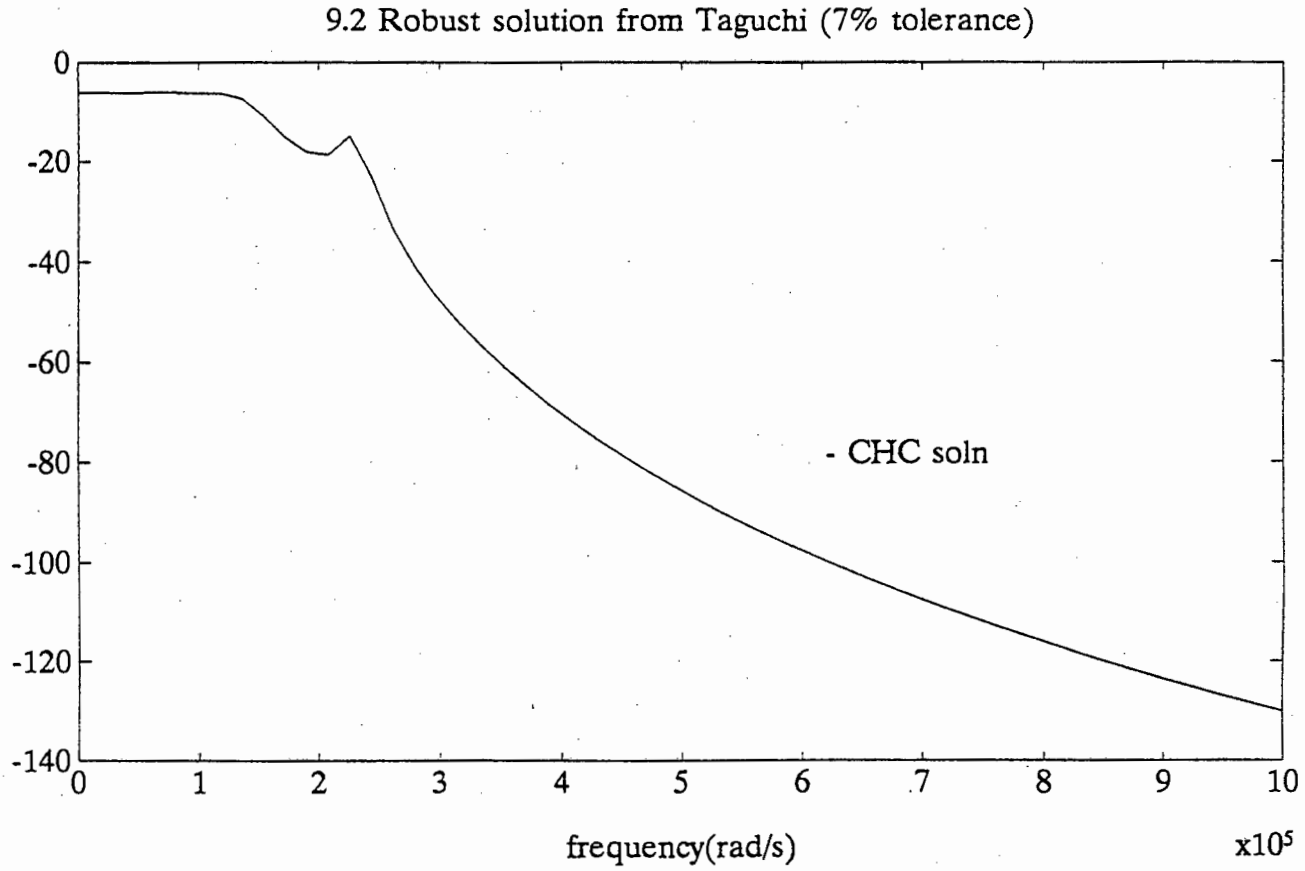

9.2 Passband response (5%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

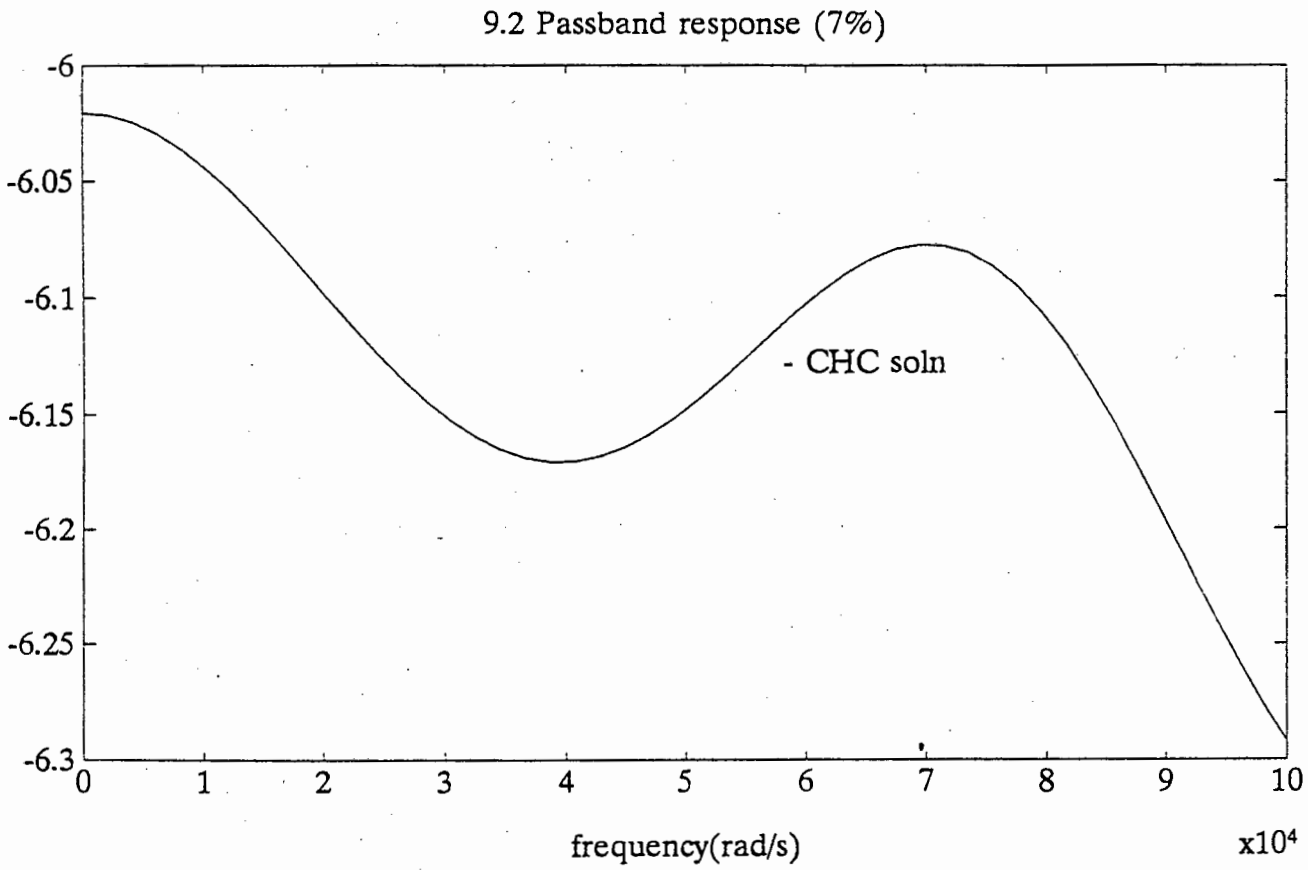

9.2 Robust solution from Taguchi (5% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
- CHC soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^5

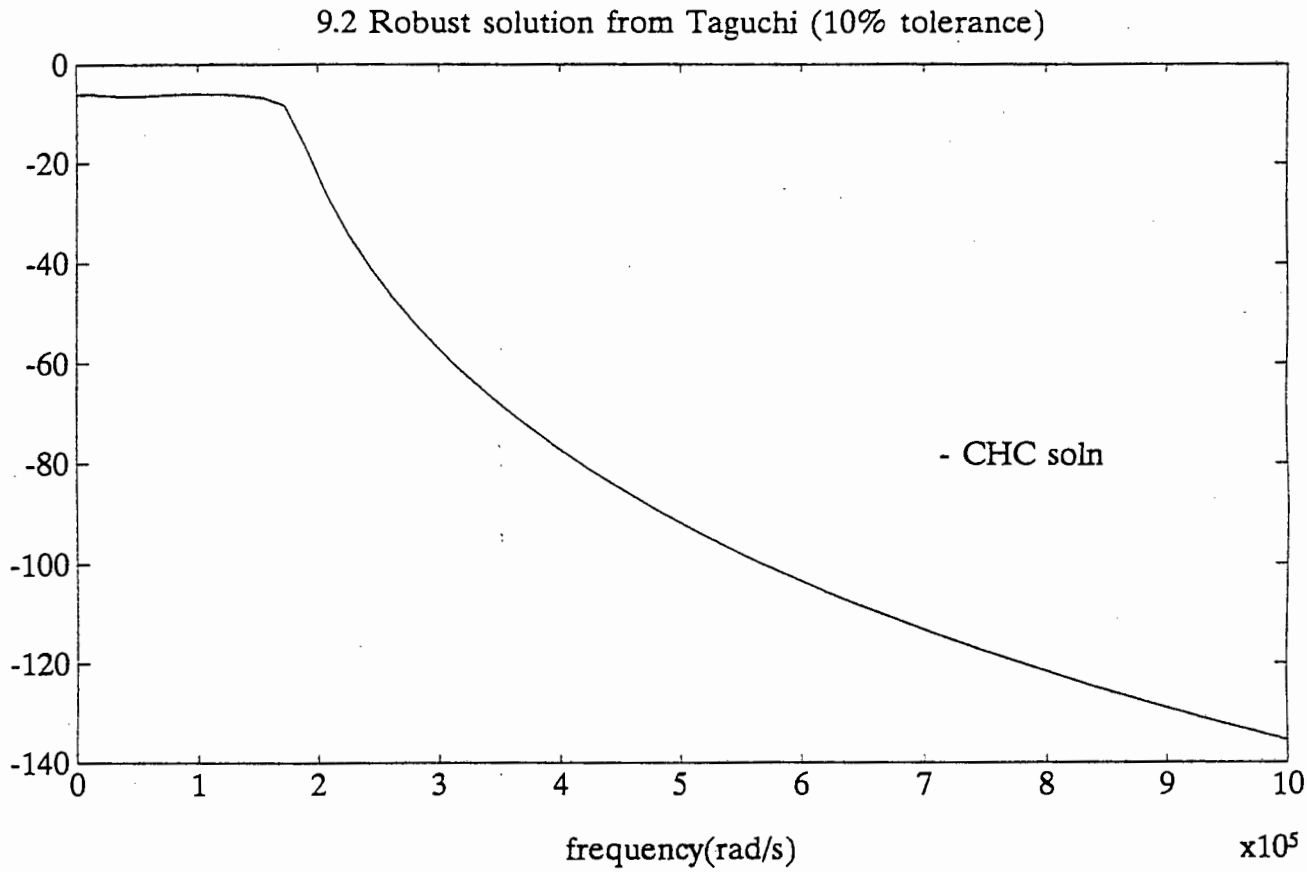

9.2 Passband response (5%)
-6.02
-6.04
-6.06
-6.08
-6.1
-6.12
-6.14
-6.16
- CHC soln
0
1
2
3
4
5
6
7
8
9
10
frequency(rad/s)
x10^4

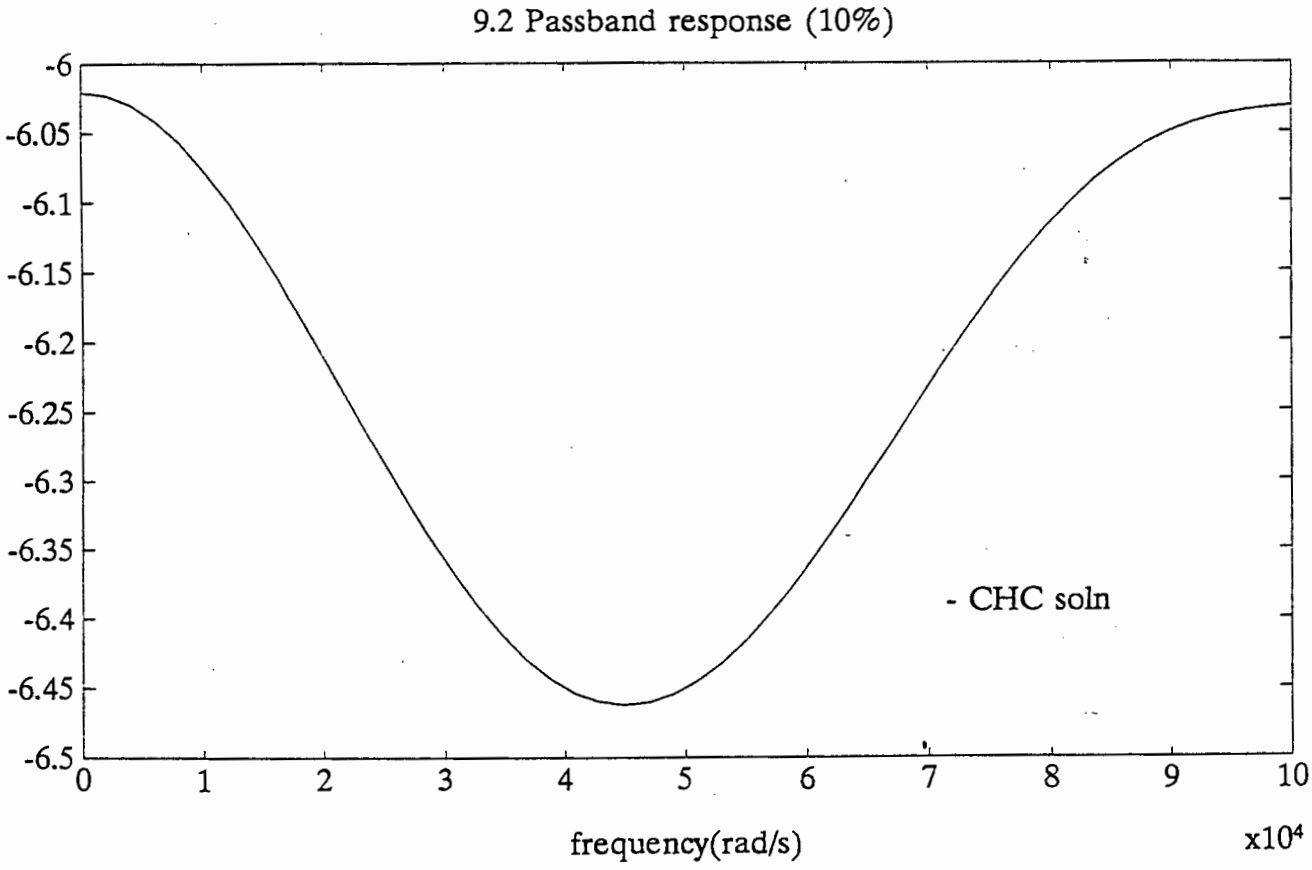

9.2 Robust solution from Taguchi (7% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5


9.2 Passband response (7%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

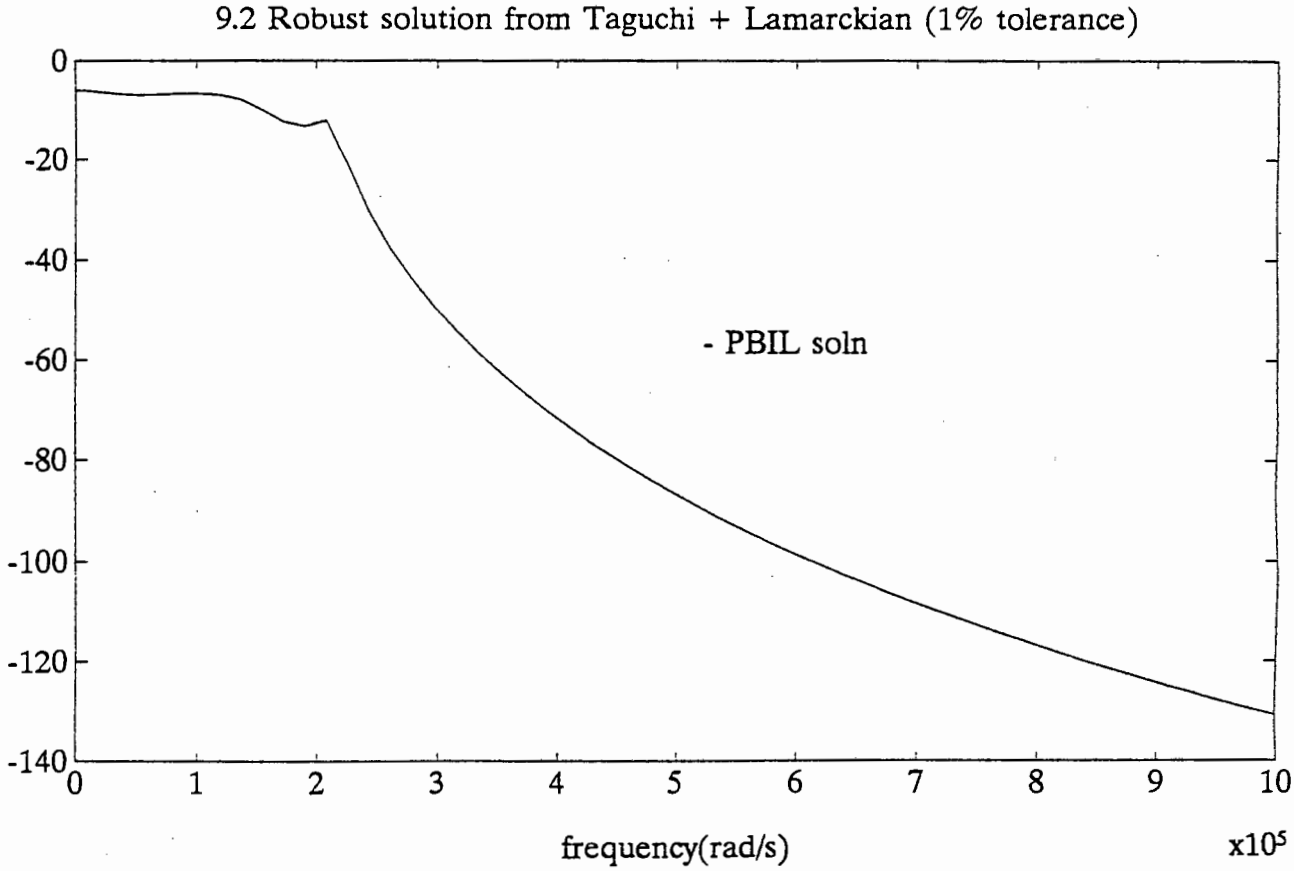

9.2 Robust solution from Taguchi (7% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

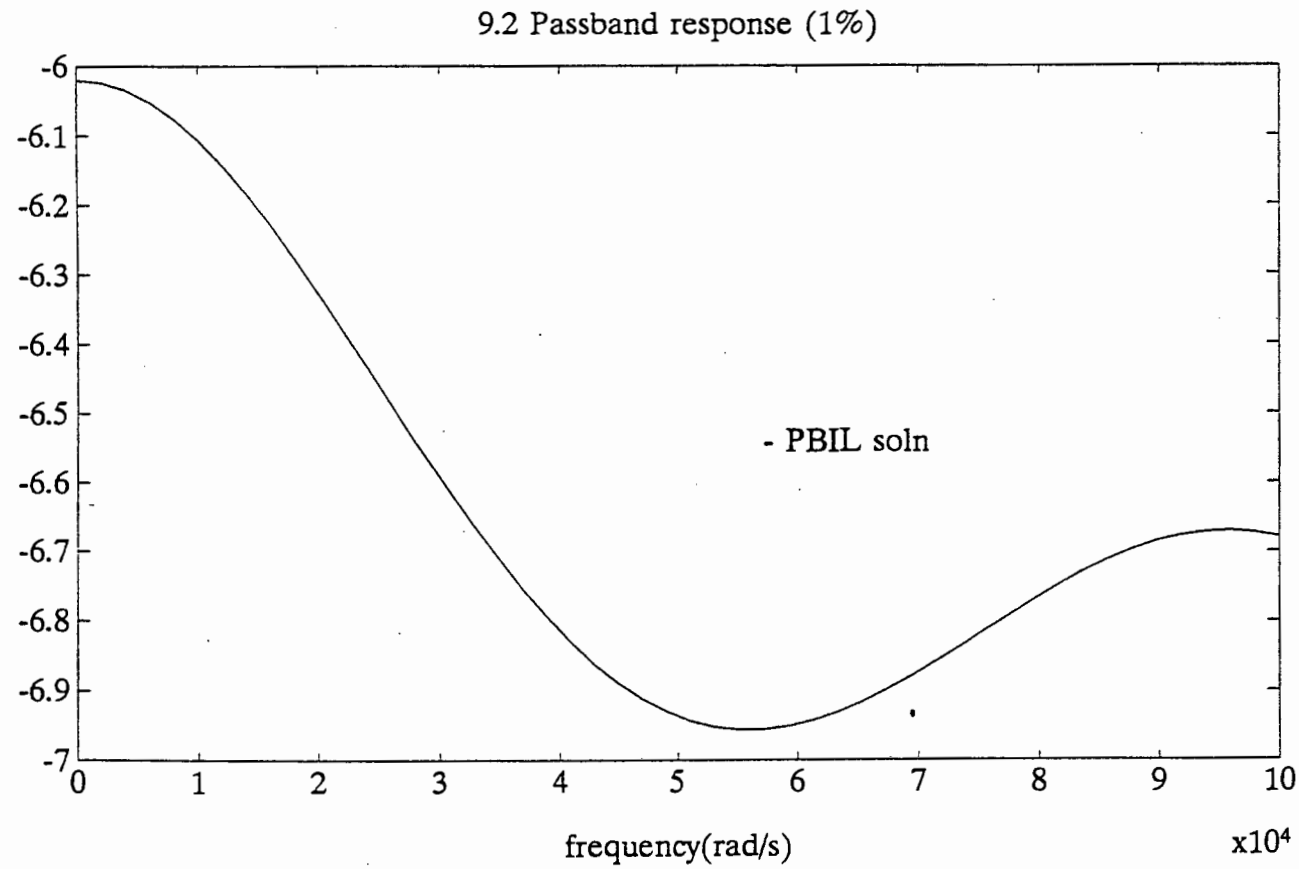

9.2 Passband response (7%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

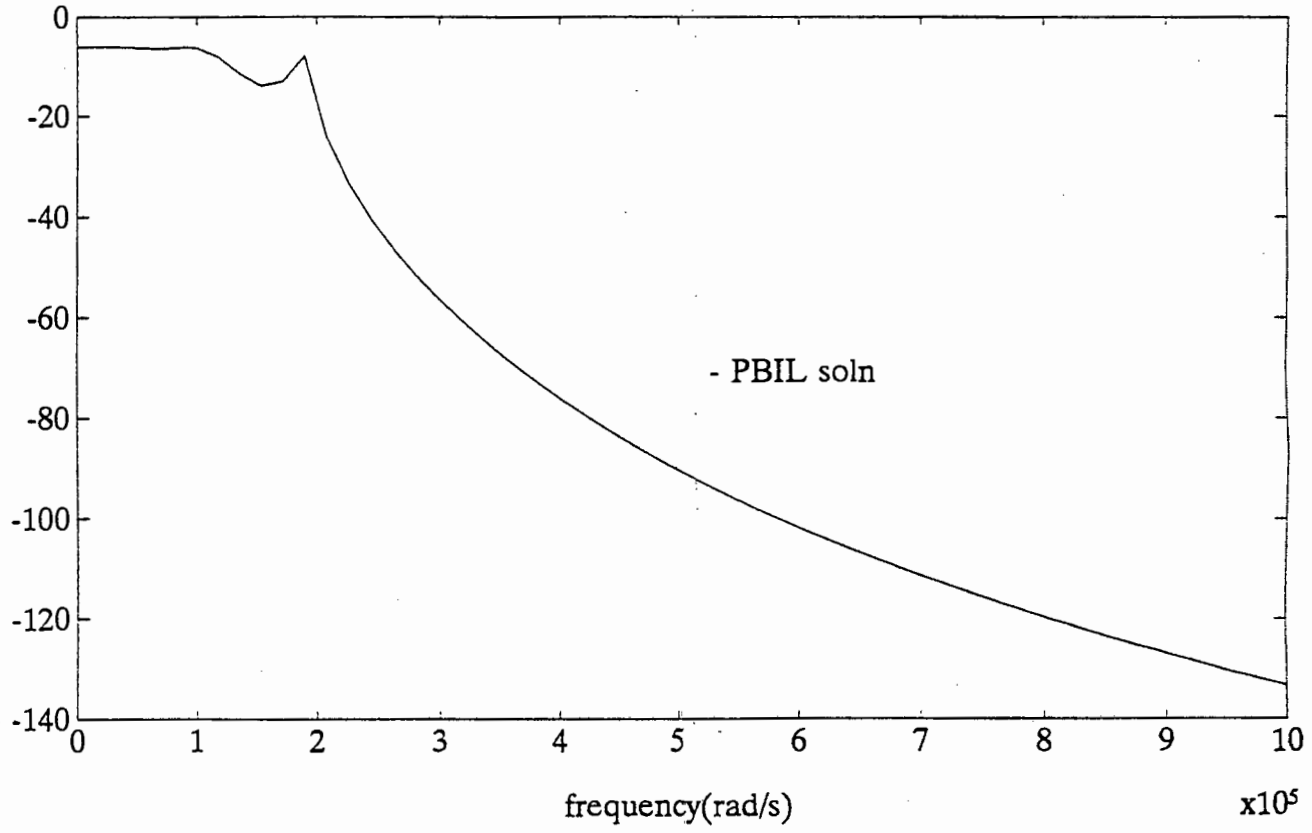

9.2 Robust solution from Taguchi (10% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

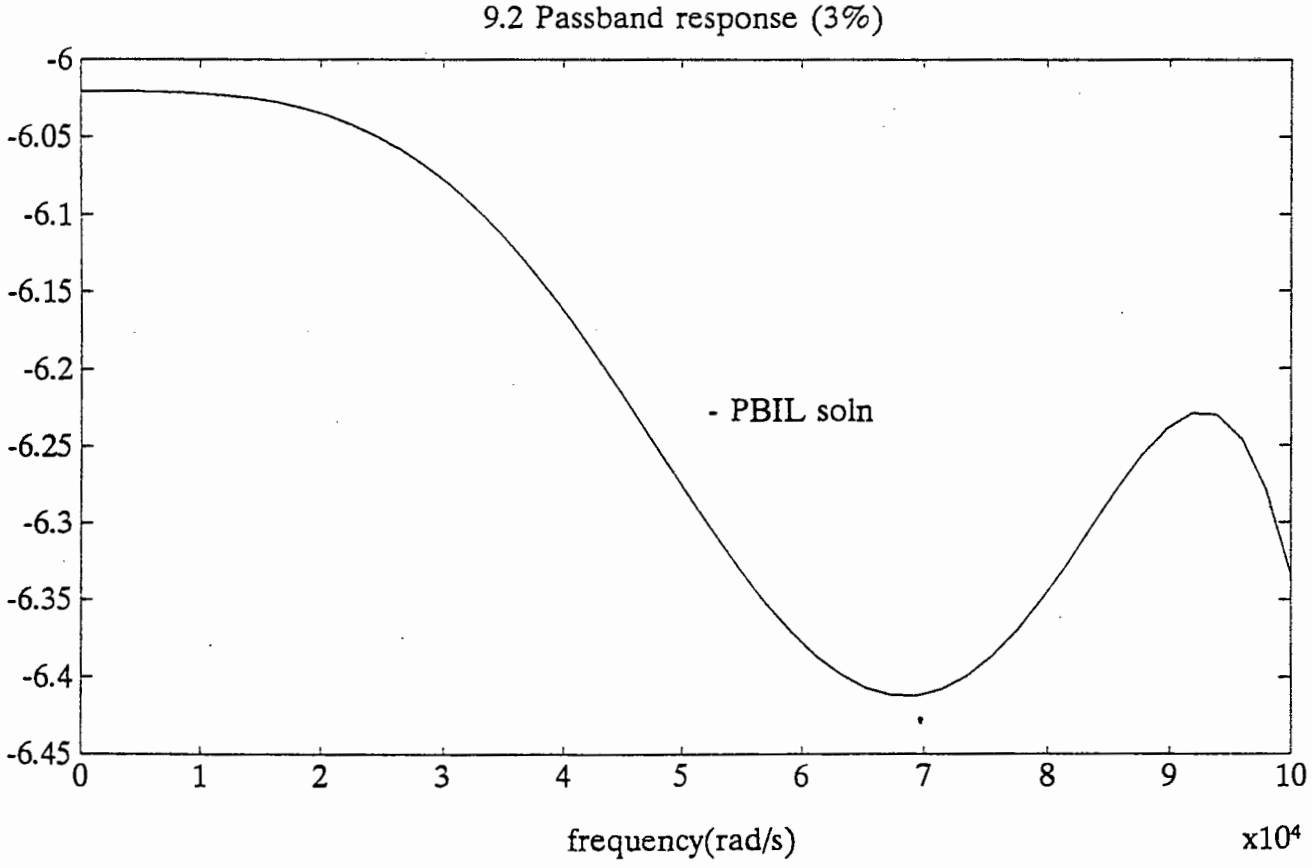

9.2 Passband response (10%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
-6.4
-6.45
-6.5
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

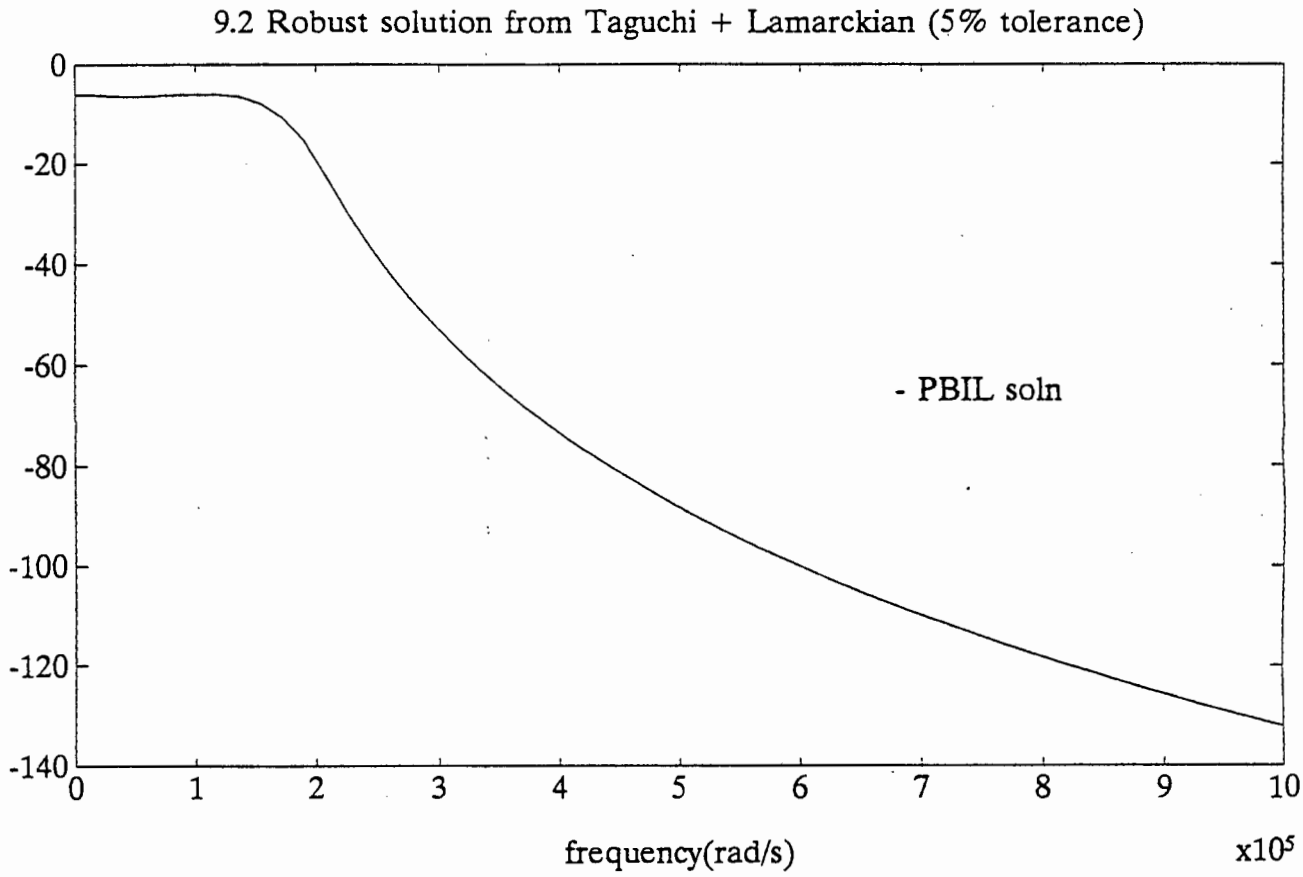
Appendix E4: Hybrid PBIL using Taguchi and Lamarckian learning on the 7th order butterworth filter

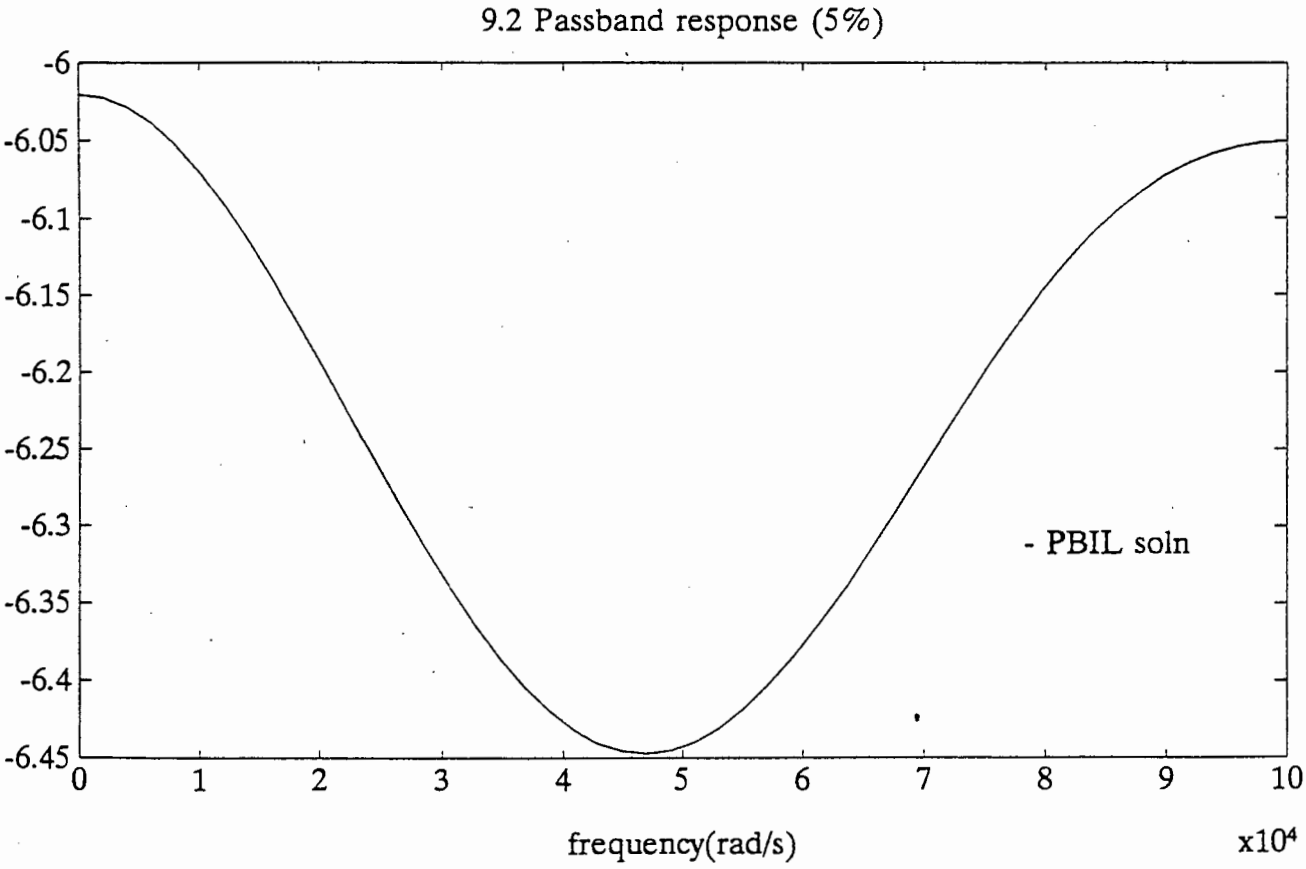

9.2 Robust solution from Taguchi + Lamarckian (1% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

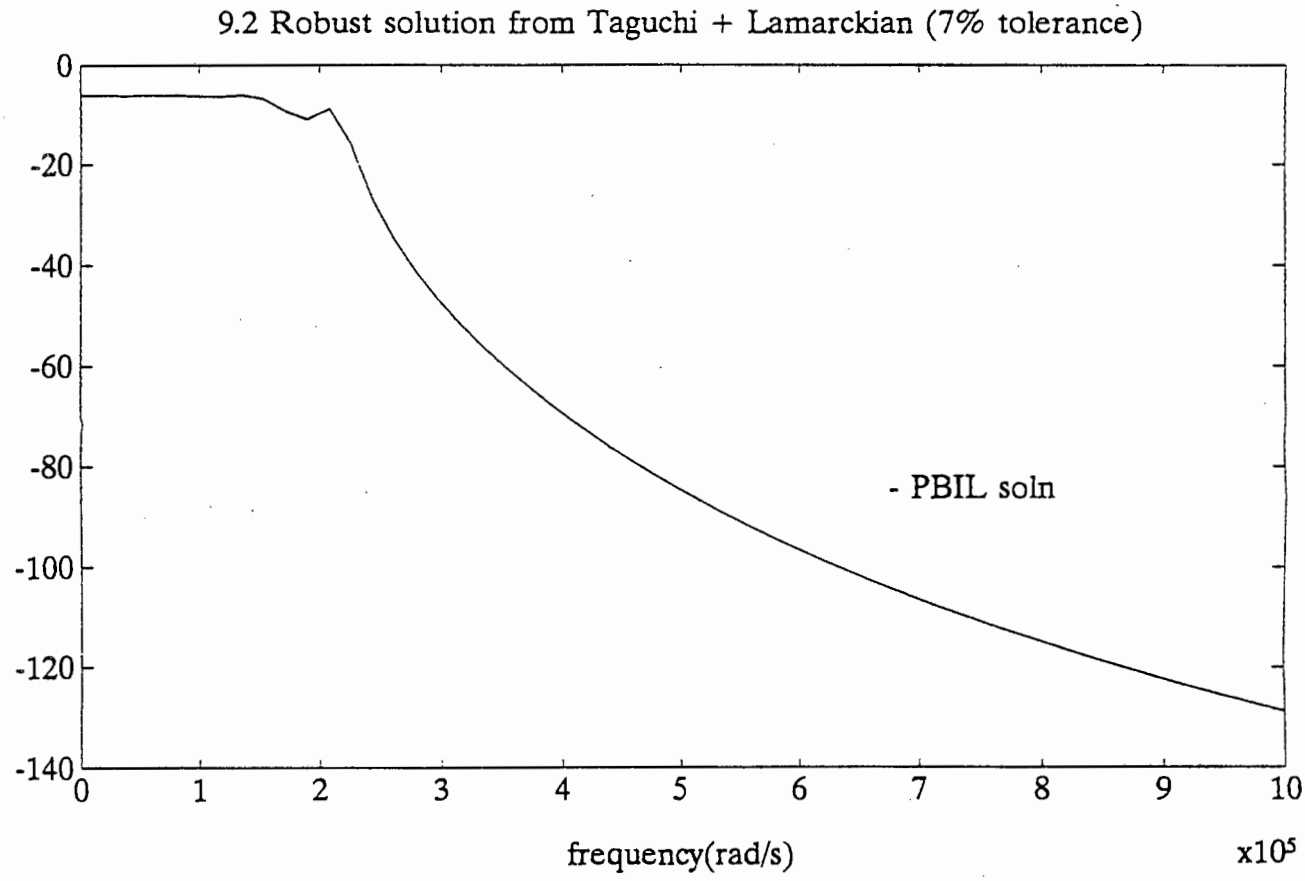

9.2 Passband response (1%)
-6
-6.1
-6.2
-6.3
-6.4
-6.5
-6.6
-6.7
-6.8
-6.9
-7
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

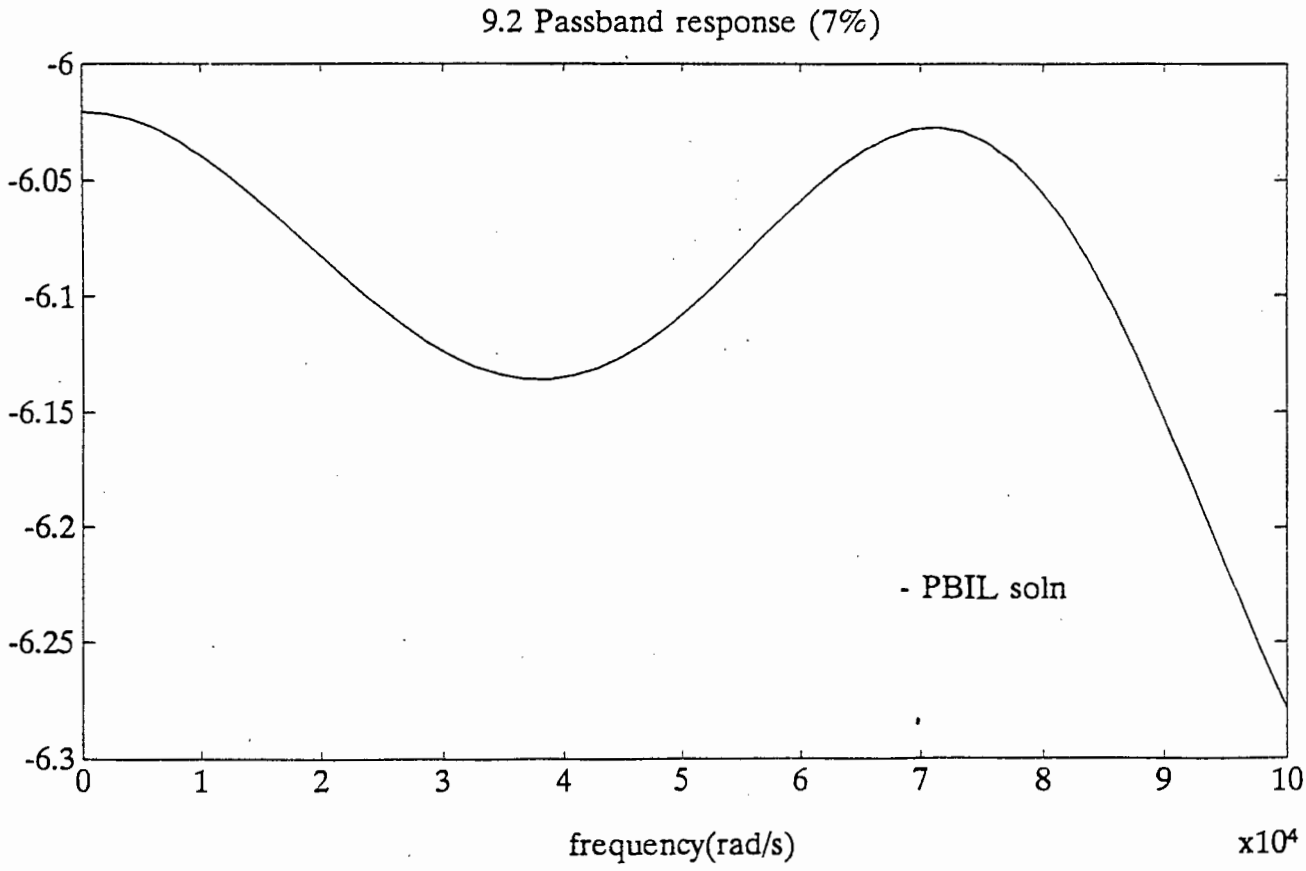

9.2 Robust solution from Taguchi + Lamarckian (3% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

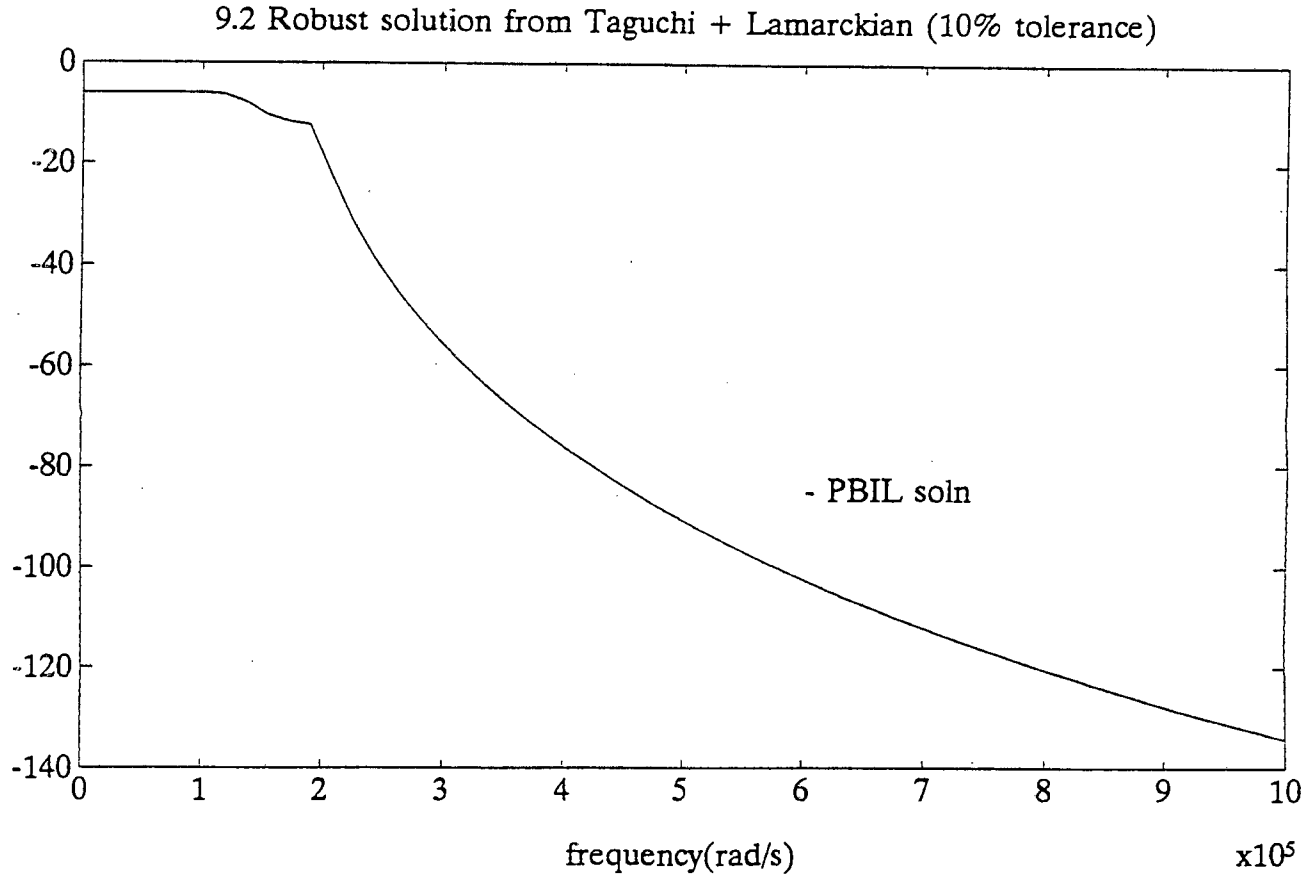

9.2 Passband response (3%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
-6.4
-6.45
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

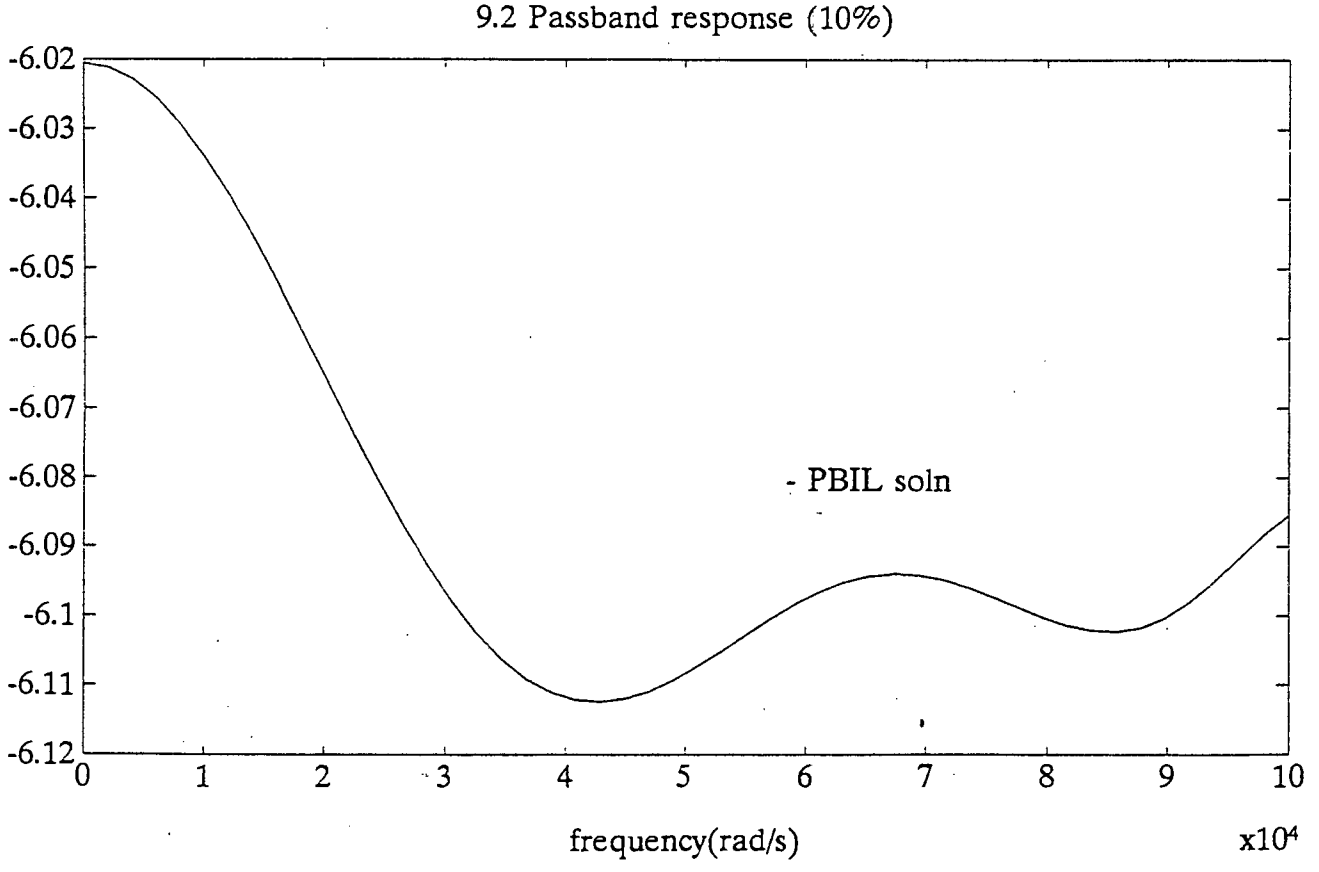

9.2 Robust solution from Taguchi + Lamarckian (5% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5


9.2 Passband response (5%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
-6.35
-6.4
-6.45
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

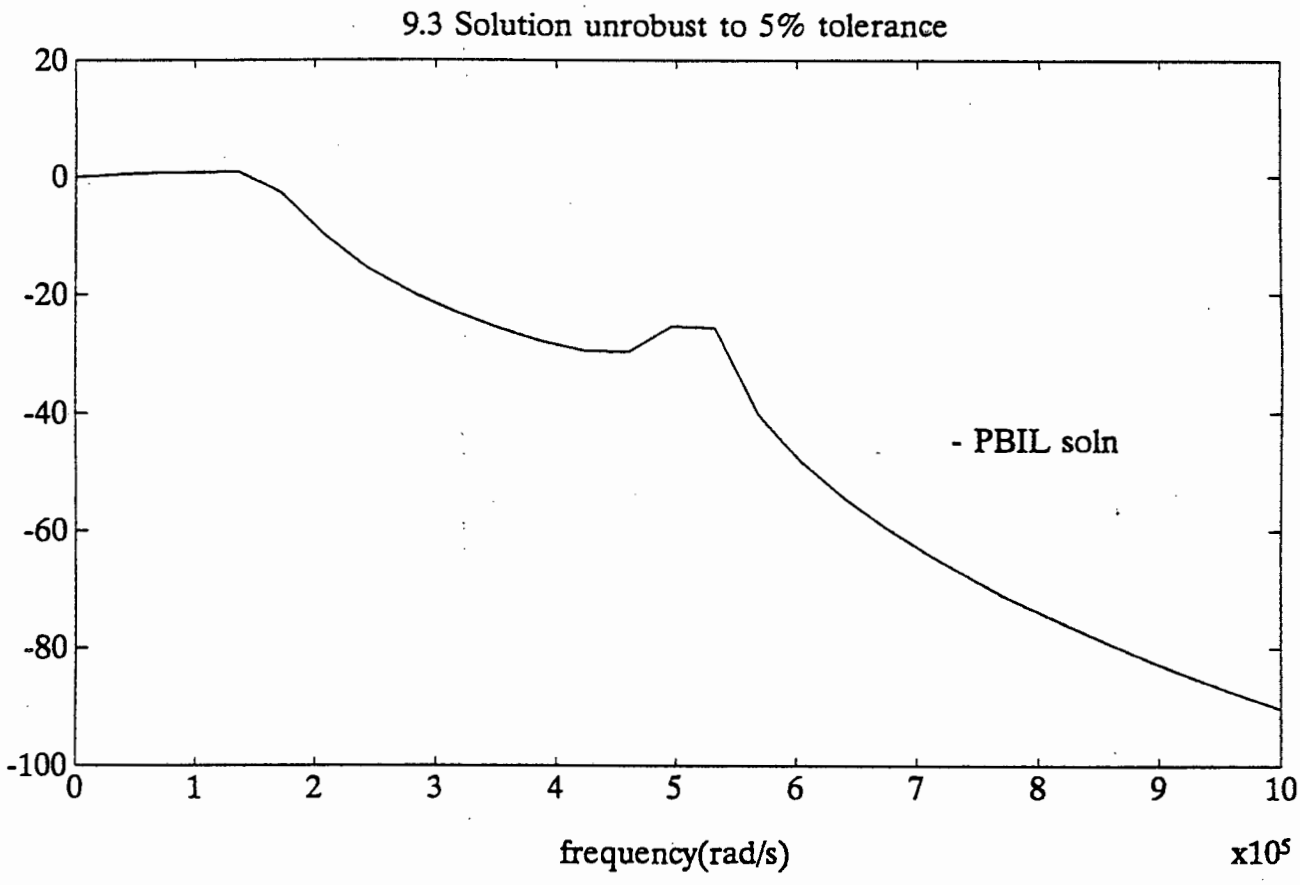

9.2 Robust solution from Taguchi + Lamarckian (7% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

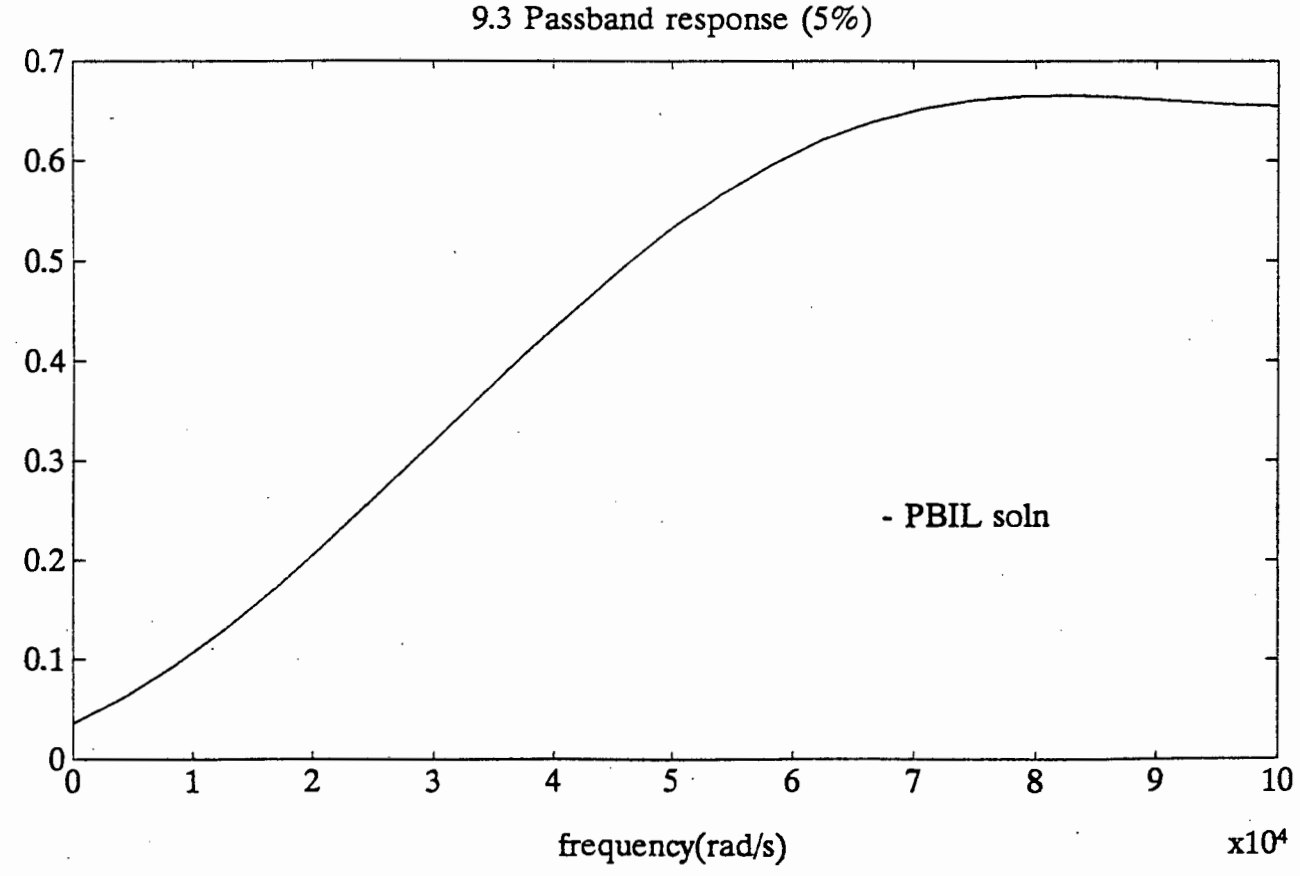

9.2 Passband response (7%)
-6
-6.05
-6.1
-6.15
-6.2
-6.25
-6.3
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

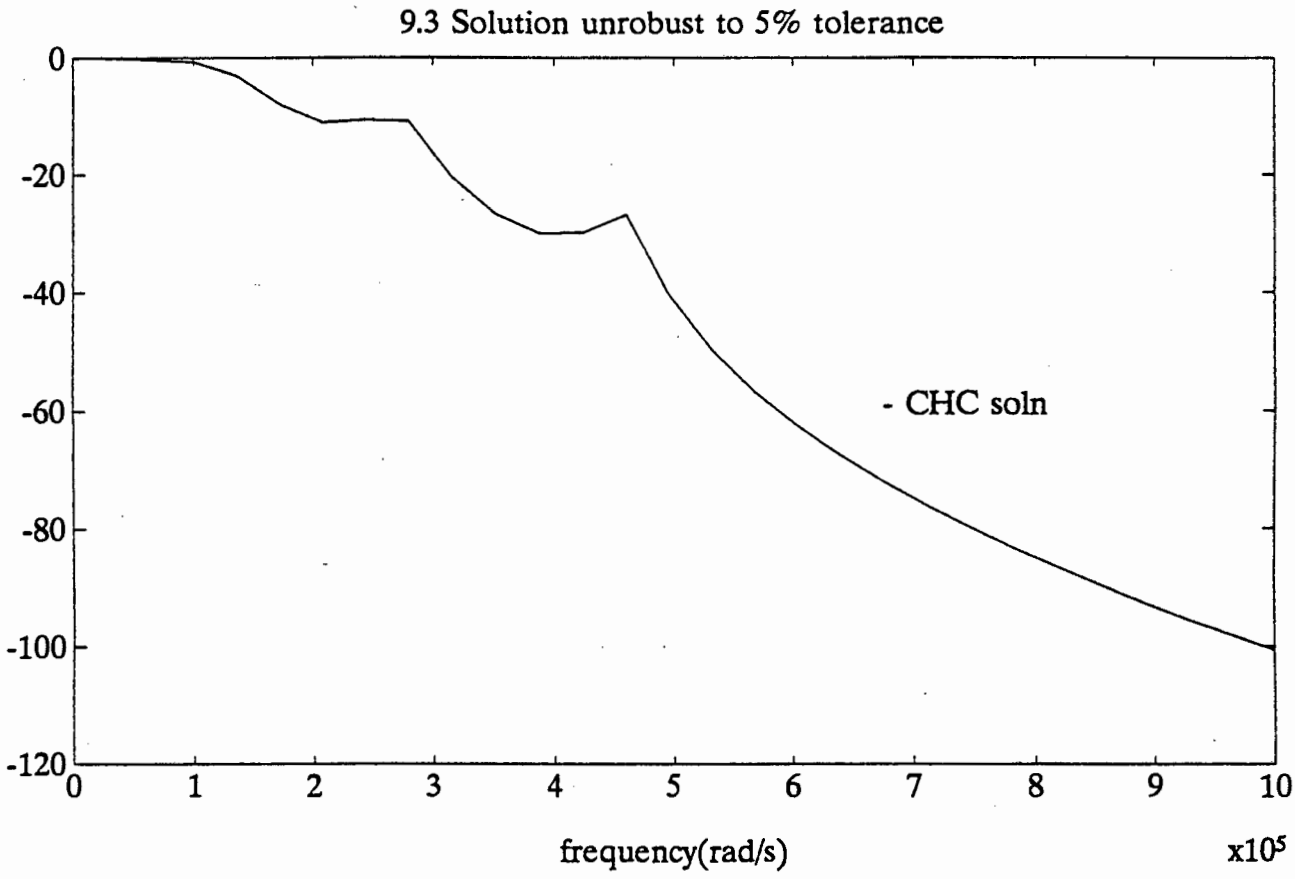

9.2 Robust solution from Taguchi + Lamarckian (10% tolerance)
0
-20
-40
-60
-80
-100
-120
-140
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

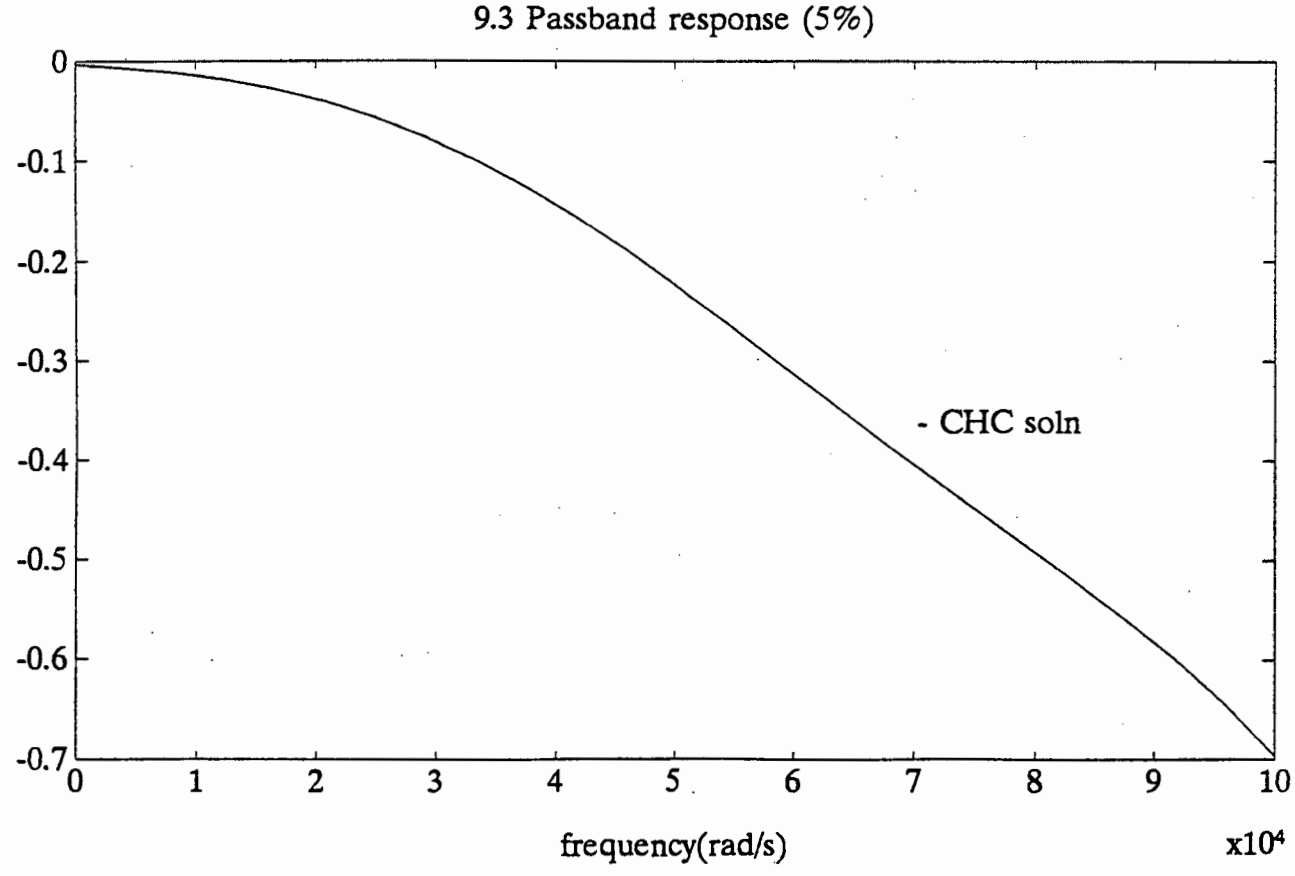

9.2 Passband response (10%)
-6.02
-6.03
-6.04
-6.05
-6.06
-6.07
-6.08
-6.09
-6.1
-6.11
-6.12
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

Appendix E5: Study of the robustness characteristic of optimizer solutions on the 8th order butterworth filter

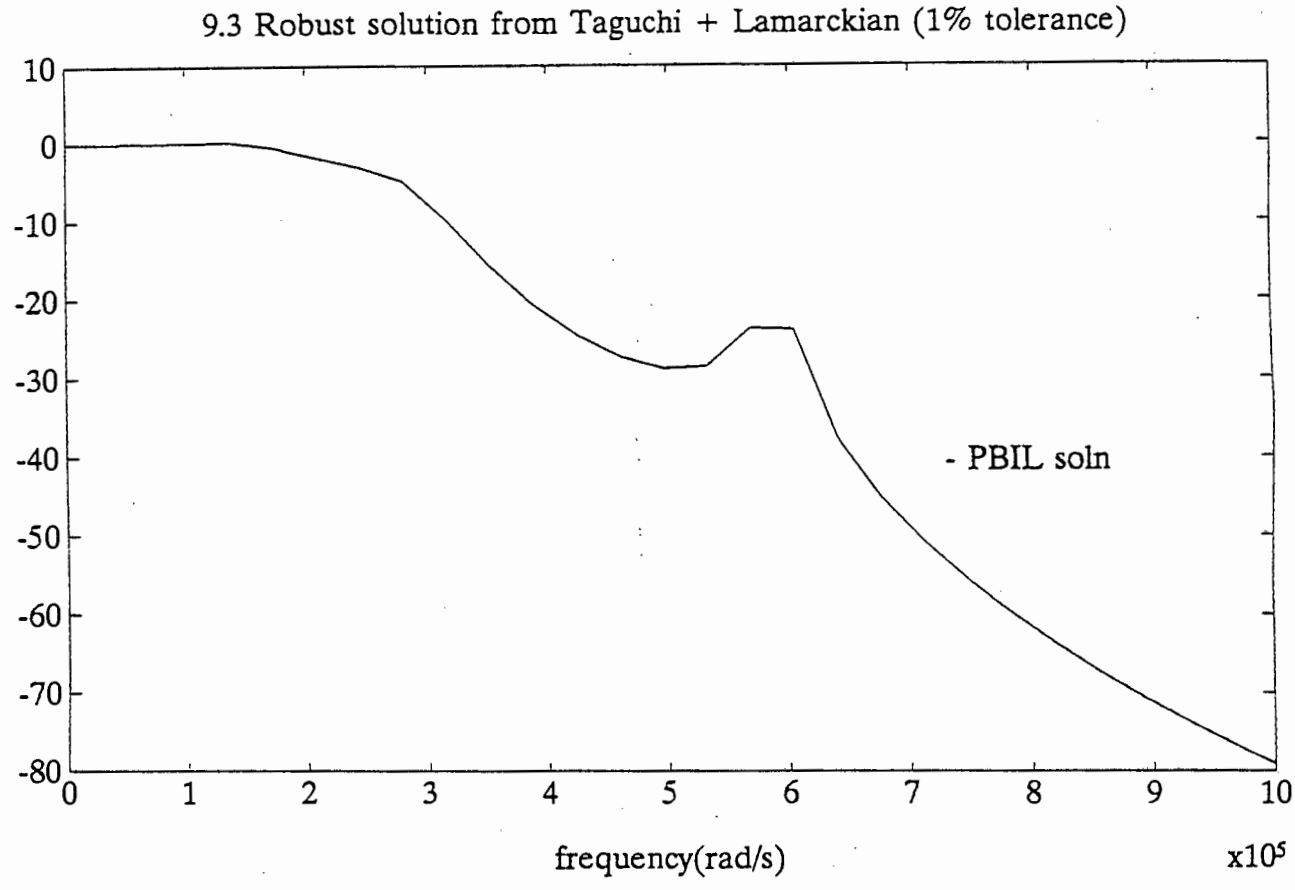

9.3 Solution unrobust to 5% tolerance
20
0
-20
-40
-60
-80
-100
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

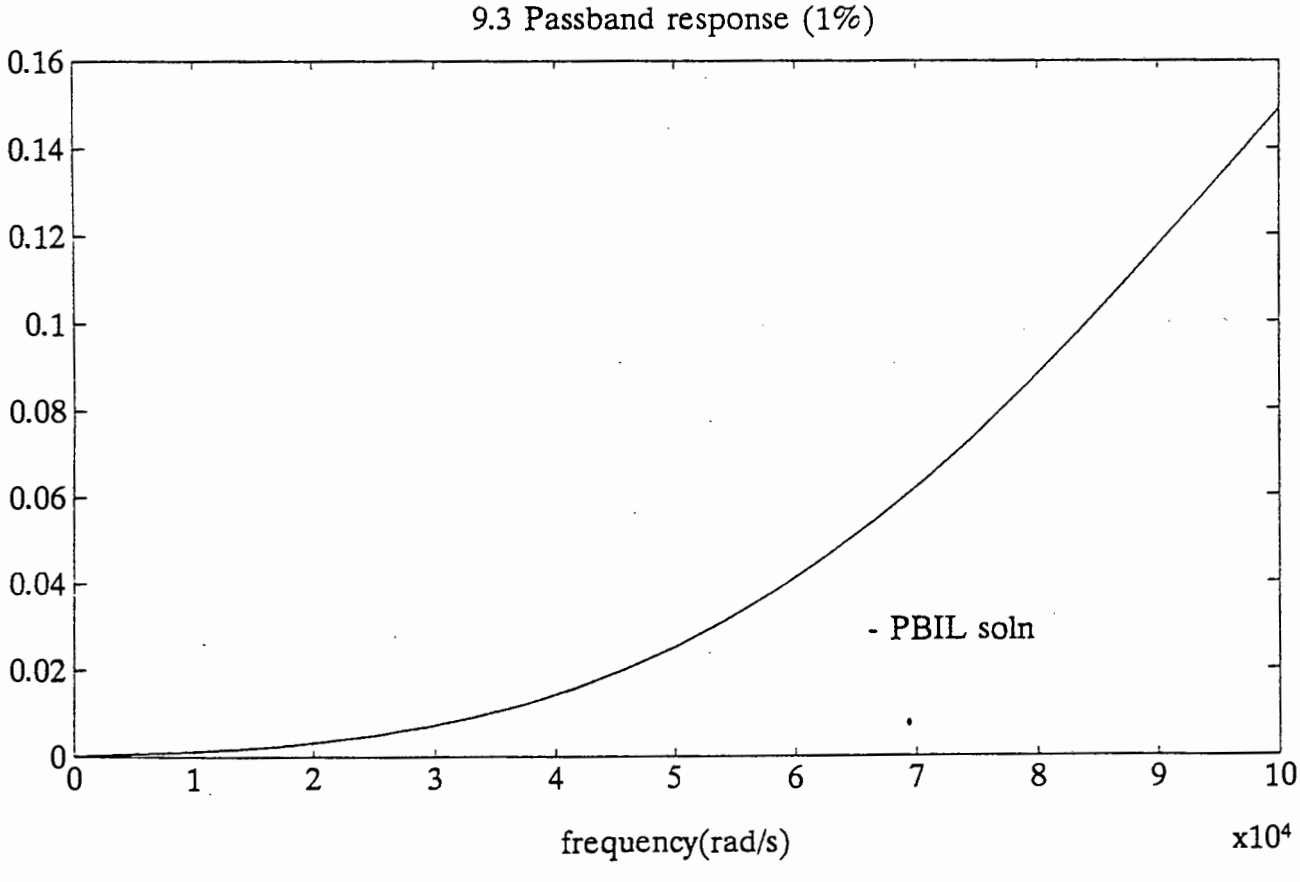

9.3 Passband response (5%)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

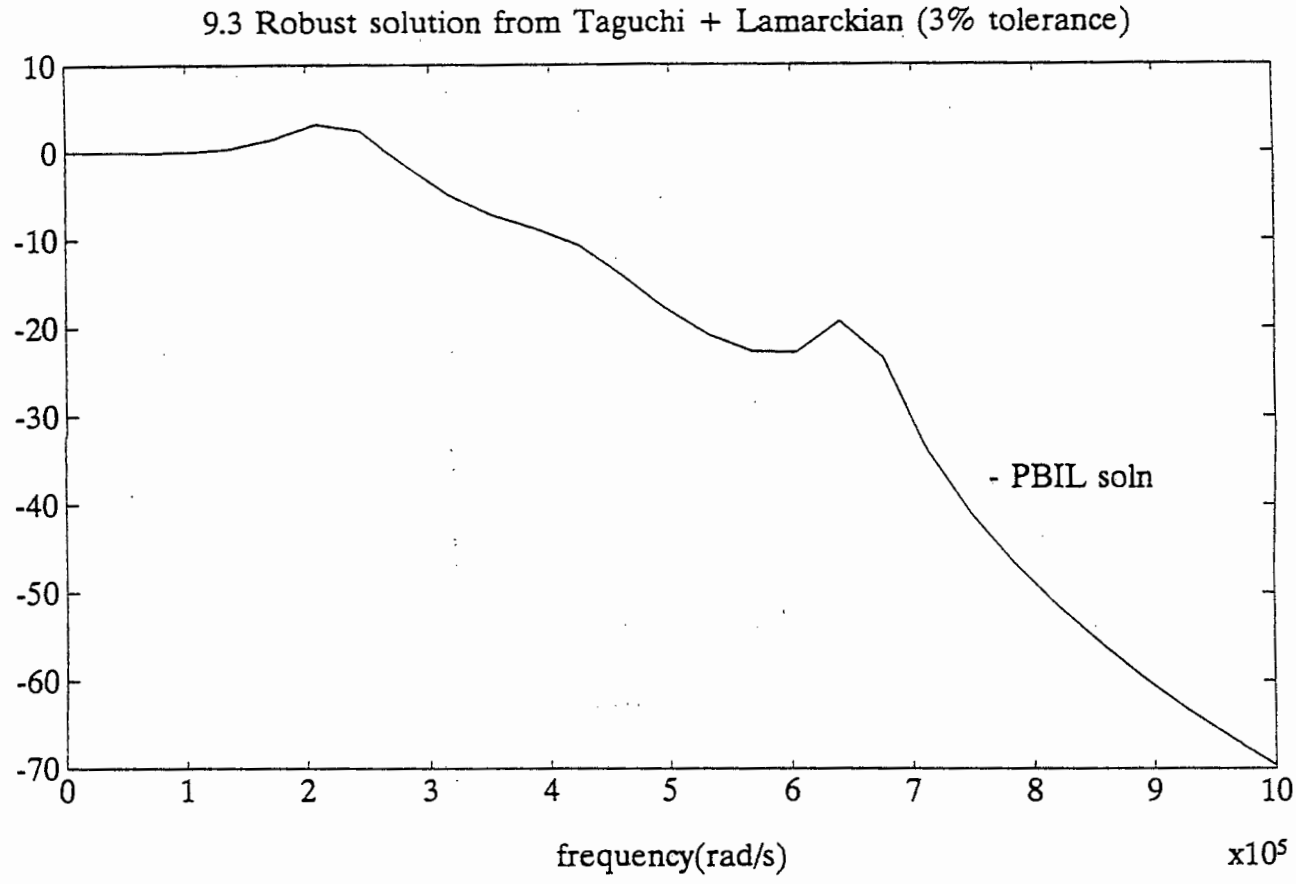

9.3 Solution unrobust to 5% tolerance
0
-20
-40
-60
-80
-100
-120
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5

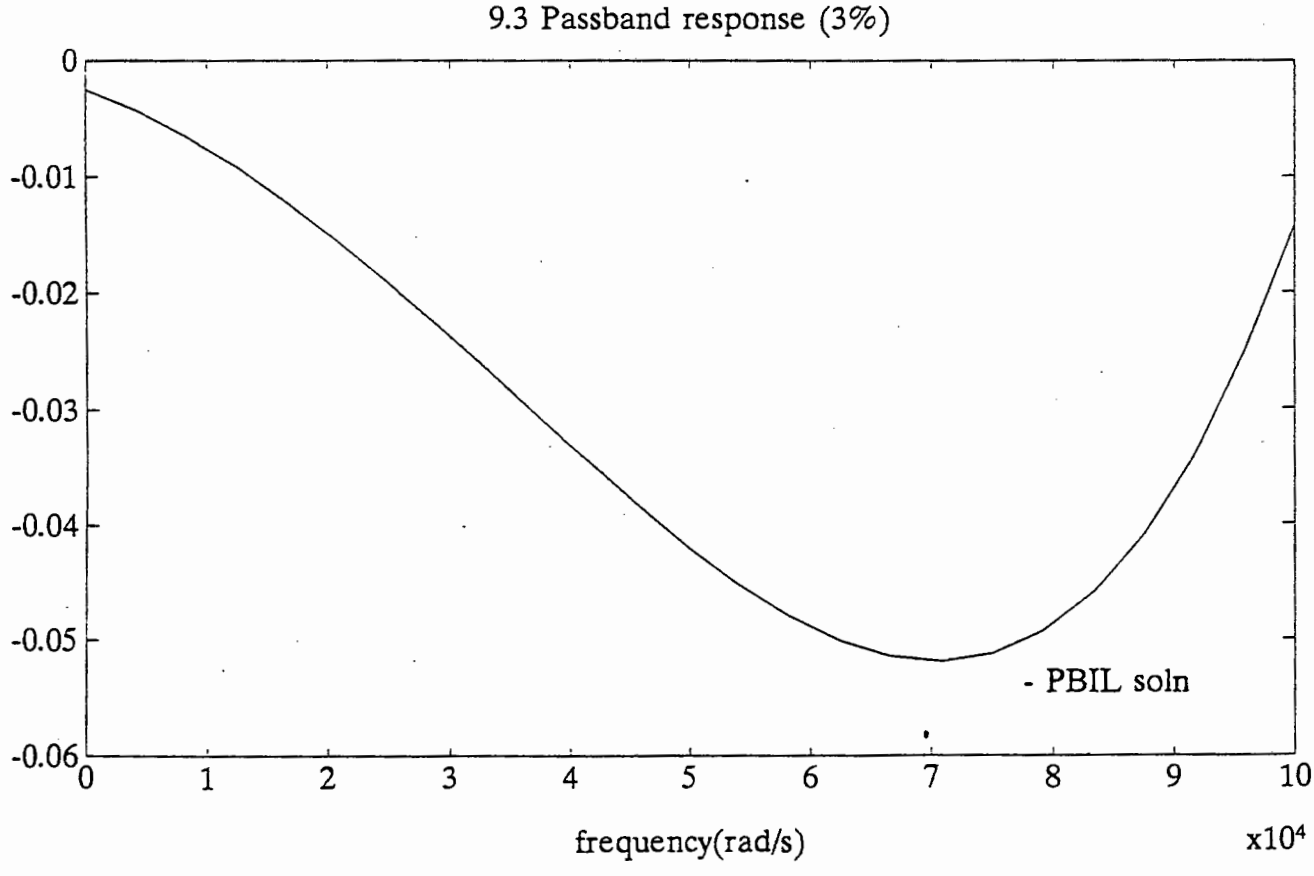

9.3 Passband response (5%)
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4

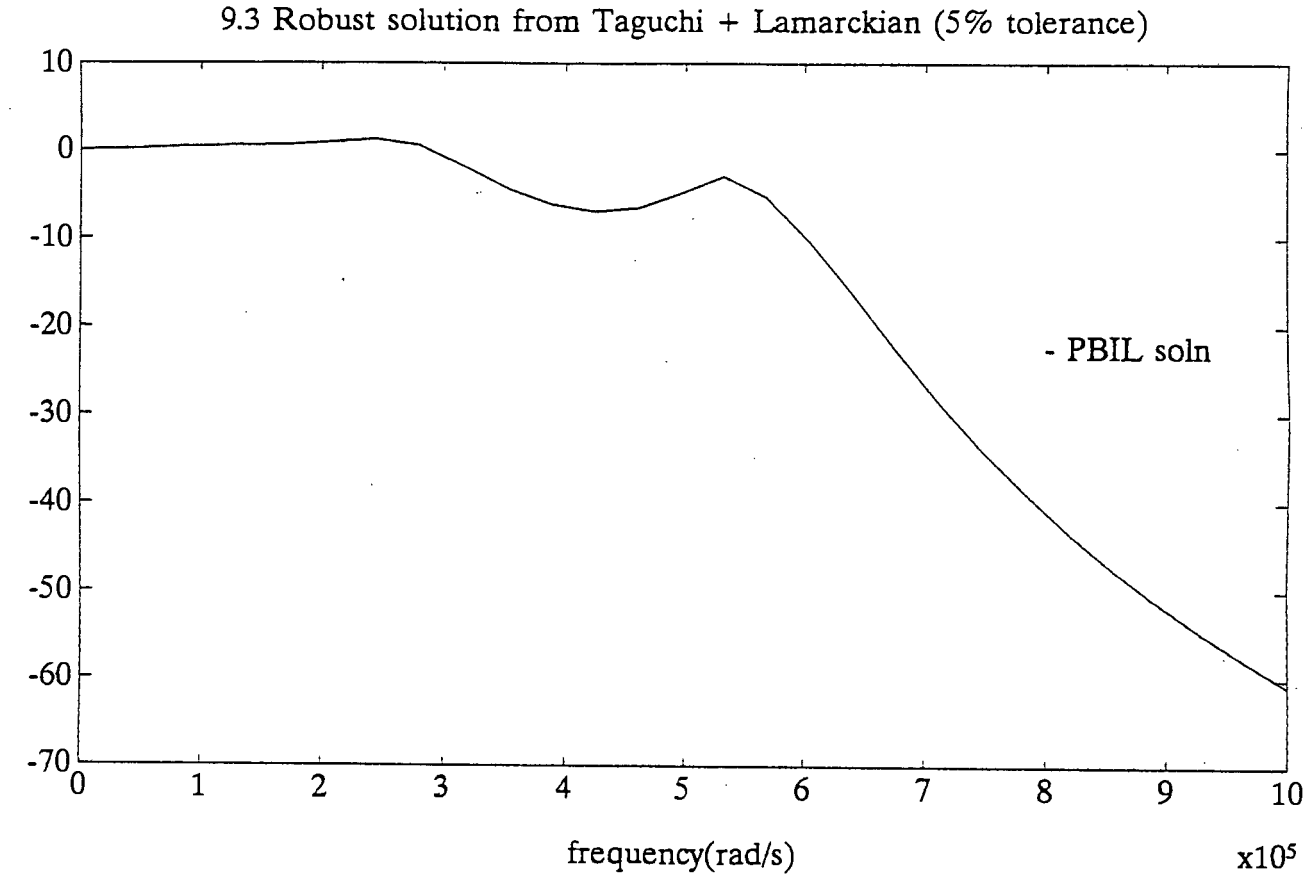
Appendix E6: Hybrid PBIL using Taguchi and Lamarckian learning on the 8th order butterworth filter

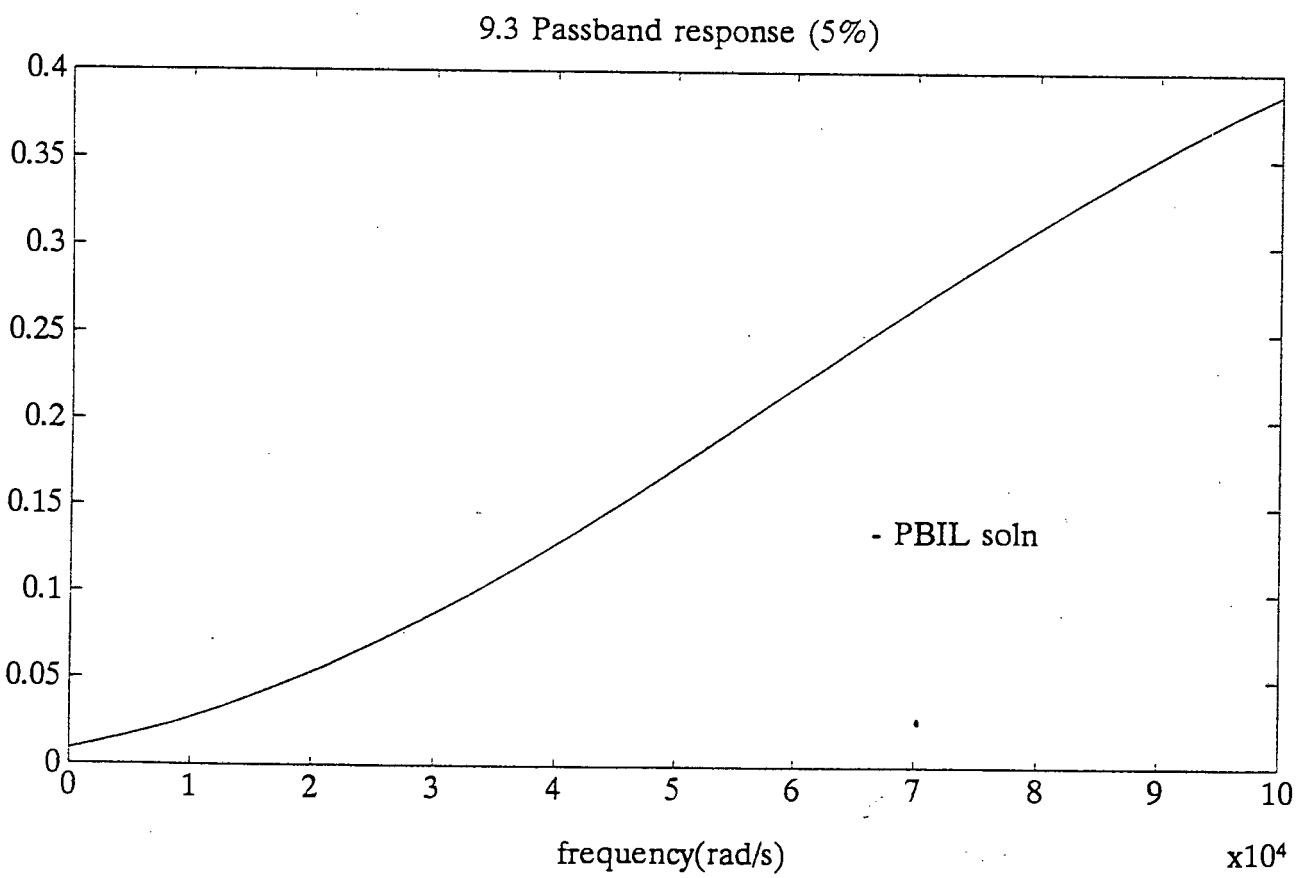

9.3 Robust solution from Taguchi + Lamarckian (1% tolerance)
10
0
-10
-20
-30
-40
-50
-60
-70
-80
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

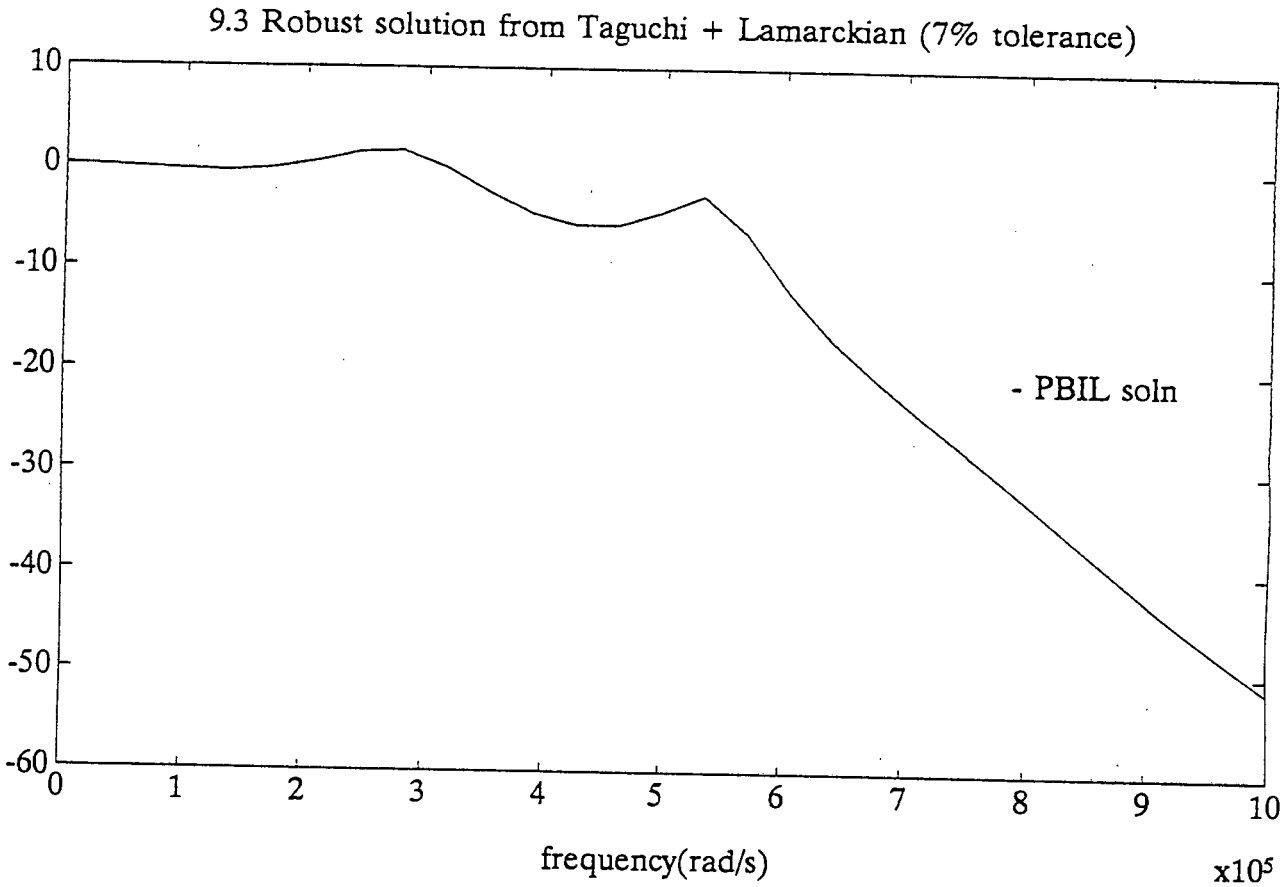

9.3 Passband response (1%)
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

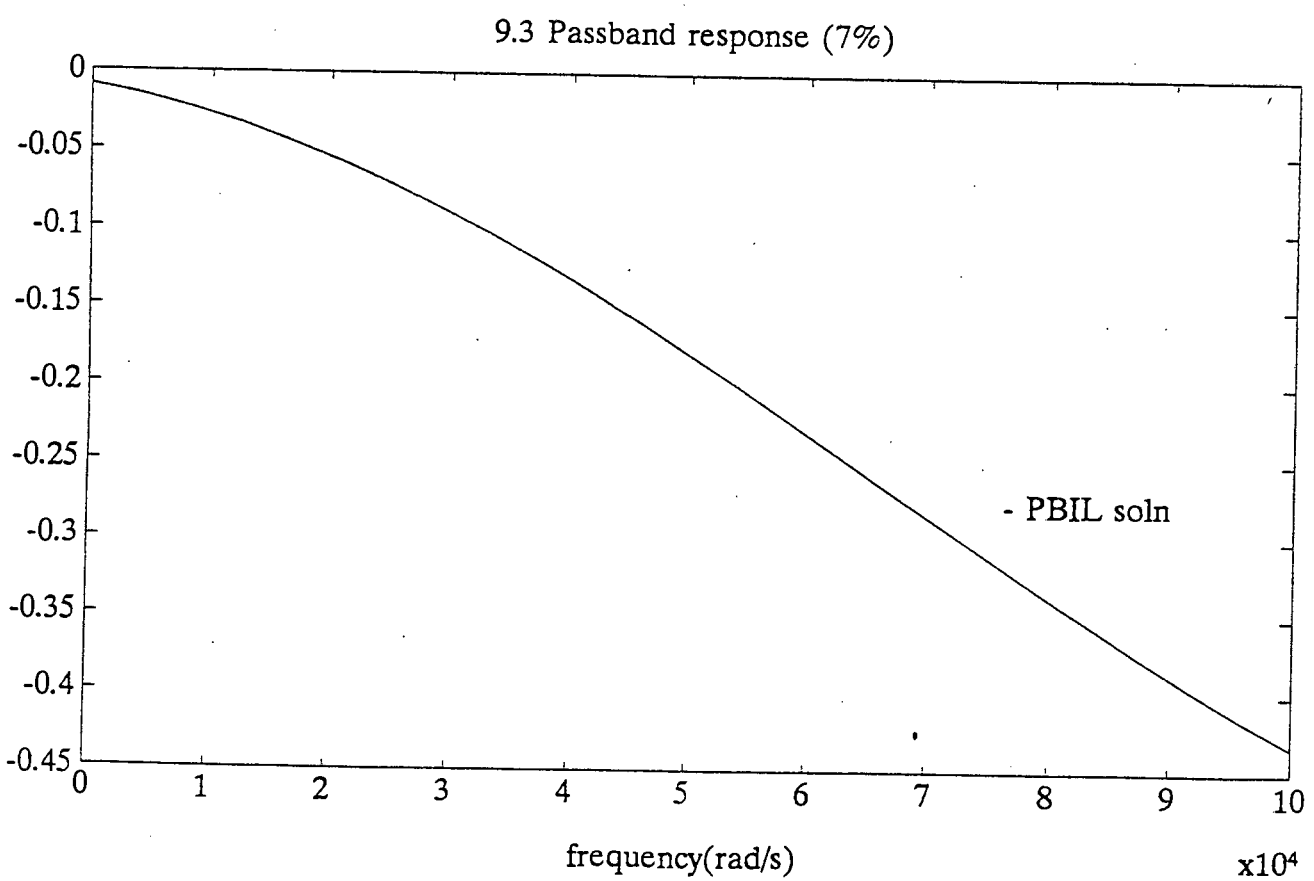

9.3 Robust solution from Taguchi + Lamarckian (3% tolerance)
10
0
-10
-20
-30
-40
-50
-60
-70
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5


9.3 Passband response (3%)
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4


9.3 Robust solution from Taguchi + Lamarckian (5% tolerance)
10
0
-10
-20
-30
-40
-50
-60
-70
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

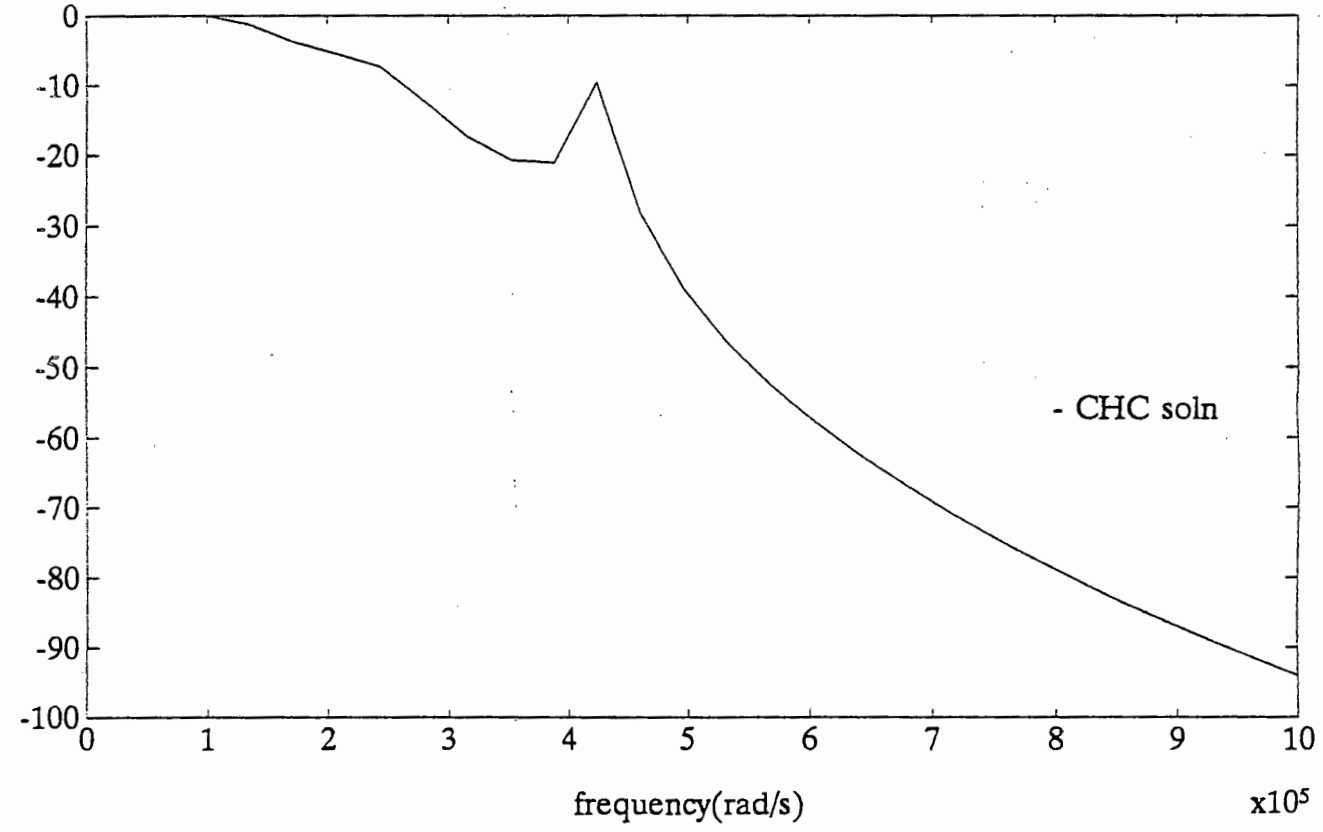

9.3 Passband response (5%)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4


9.3 Robust solution from Taguchi + Lamarckian (7% tolerance)
10
0
-10
-20
-30
-40
-50
-60
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^5

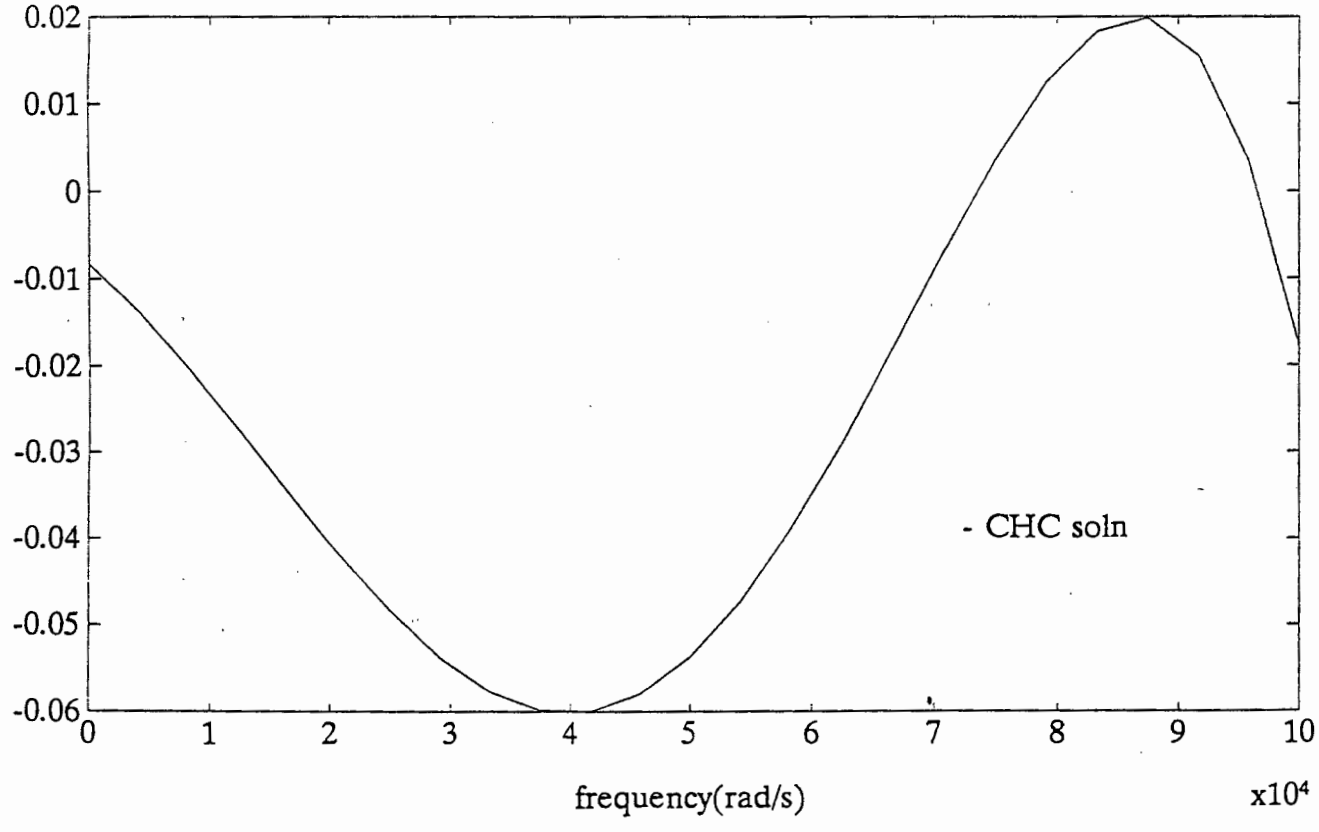

9.3 Passband response (7%)
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
-0.4
-0.45
0
1
2
3
4
5
6
7
8
9
10
- PBIL soln
frequency(rad/s)
x10^4

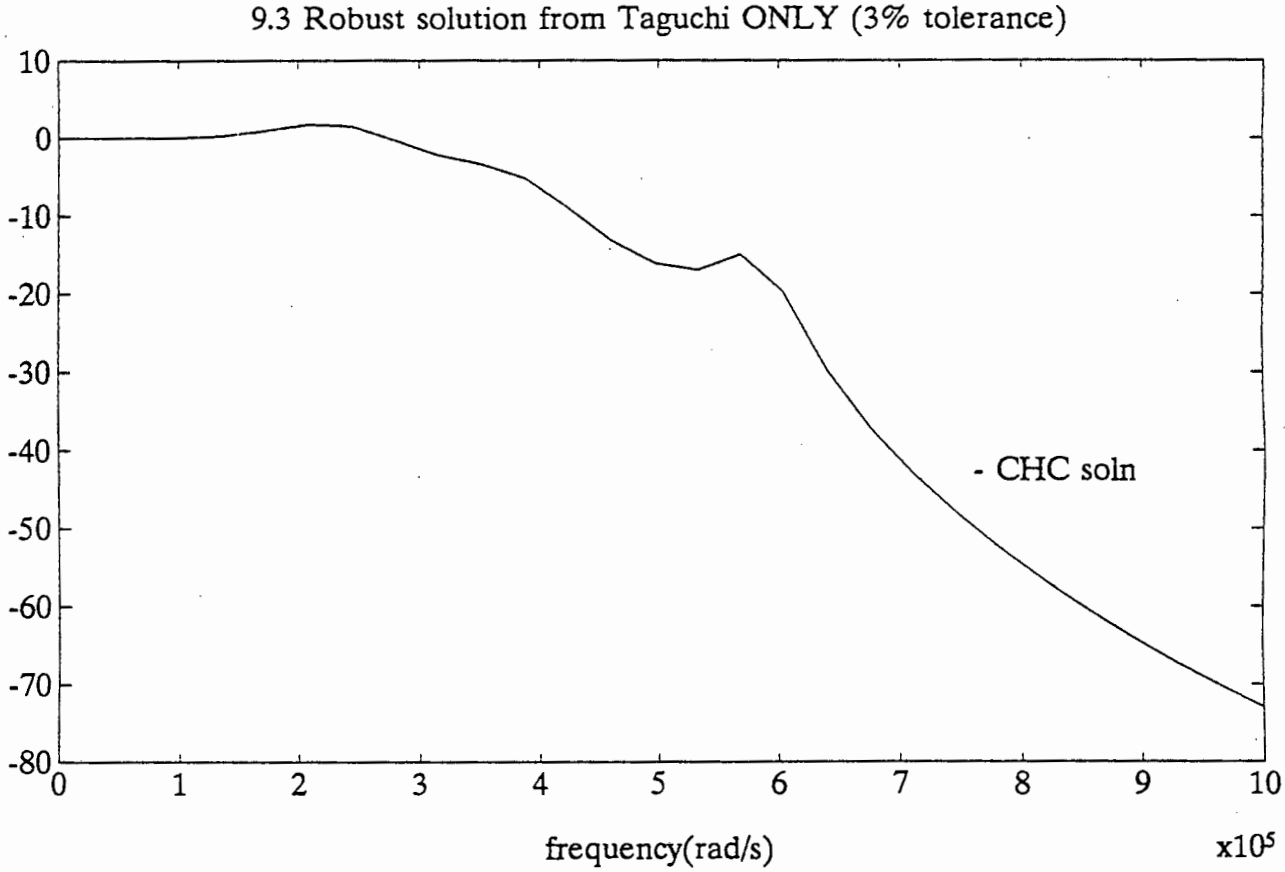
Appendix E7: Hybrid CHC using Taguchi method on the 8th order butterworth filter

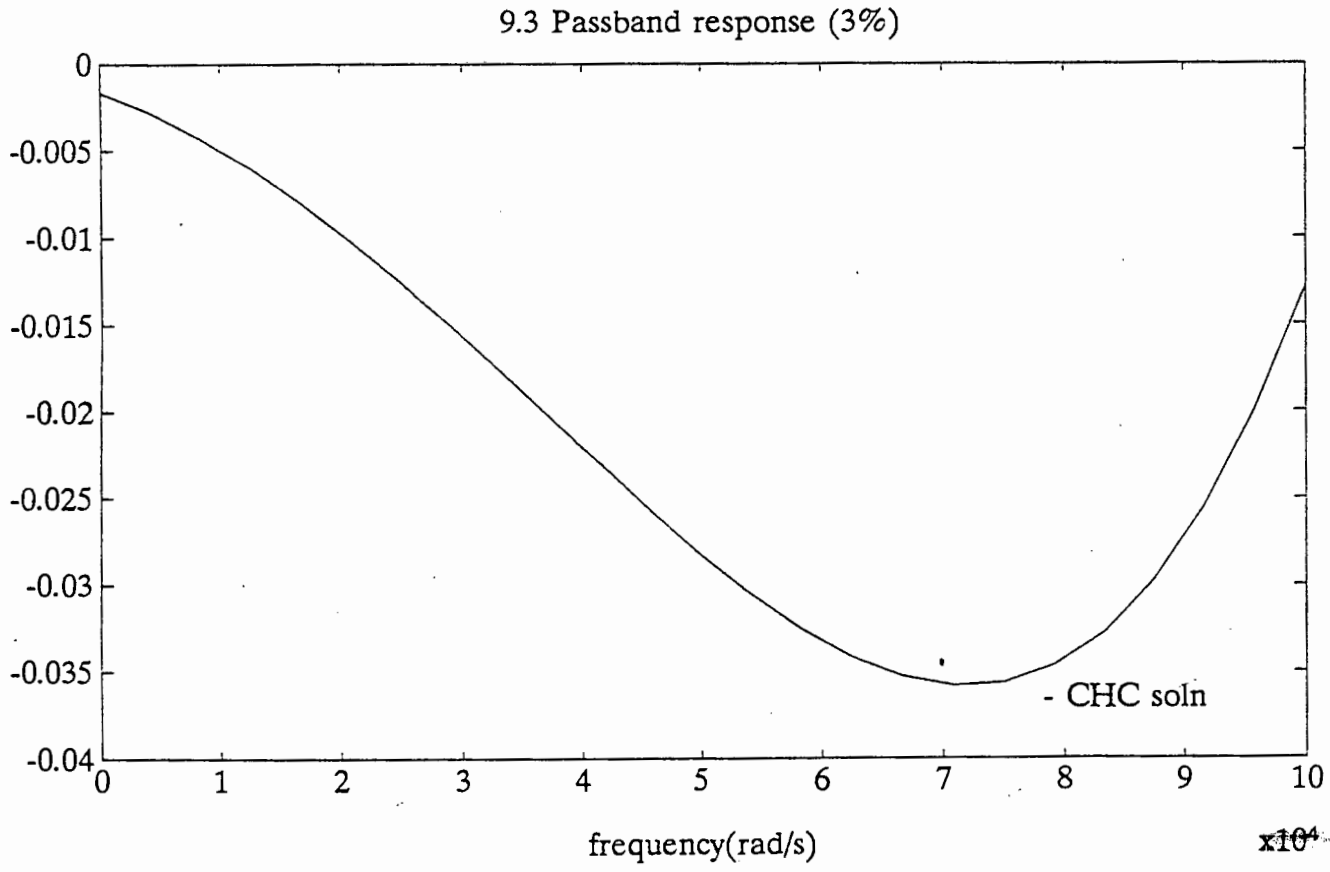
9.3 Robust solution from Taguchi ONLY ~~+ Lagrangian~~ (1% tolerance)

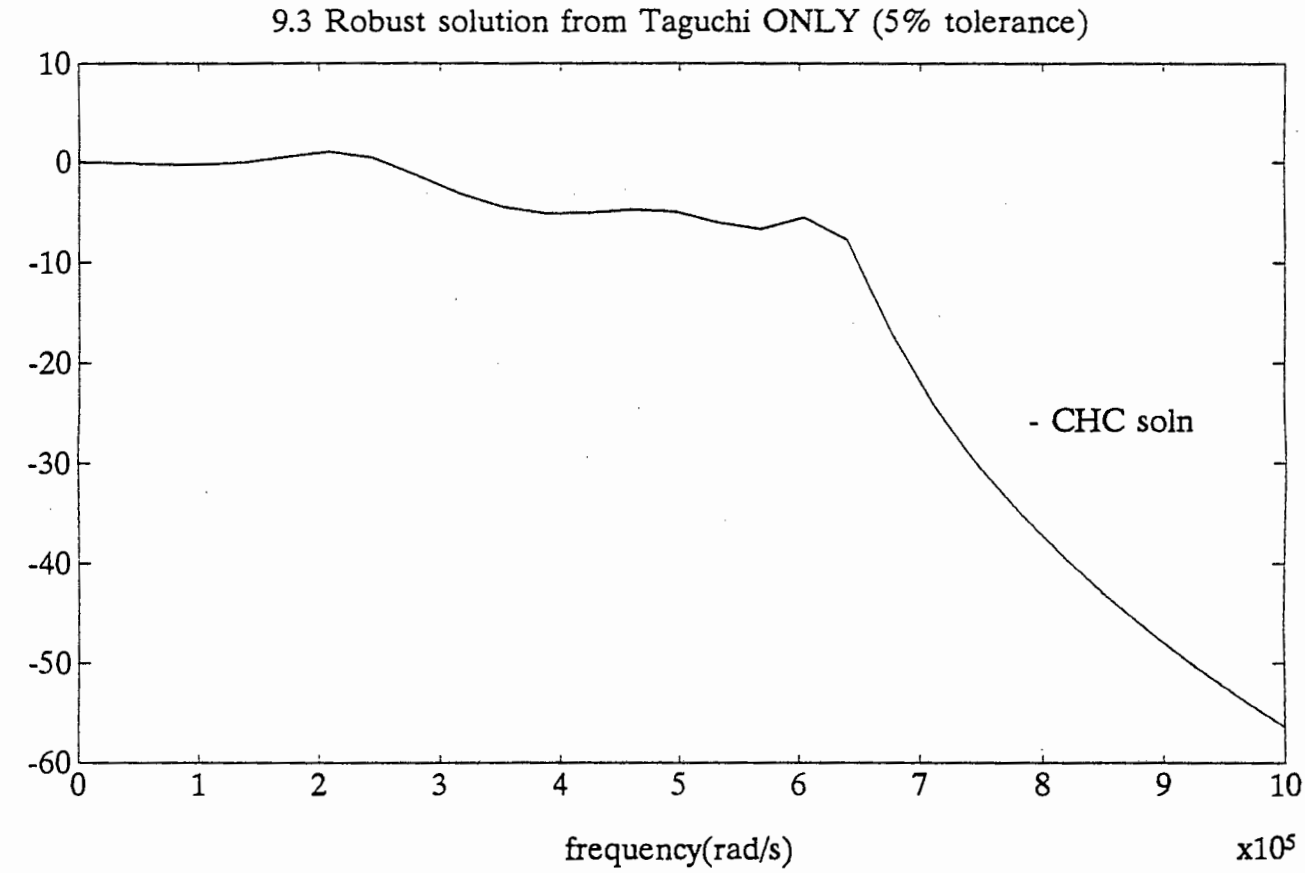


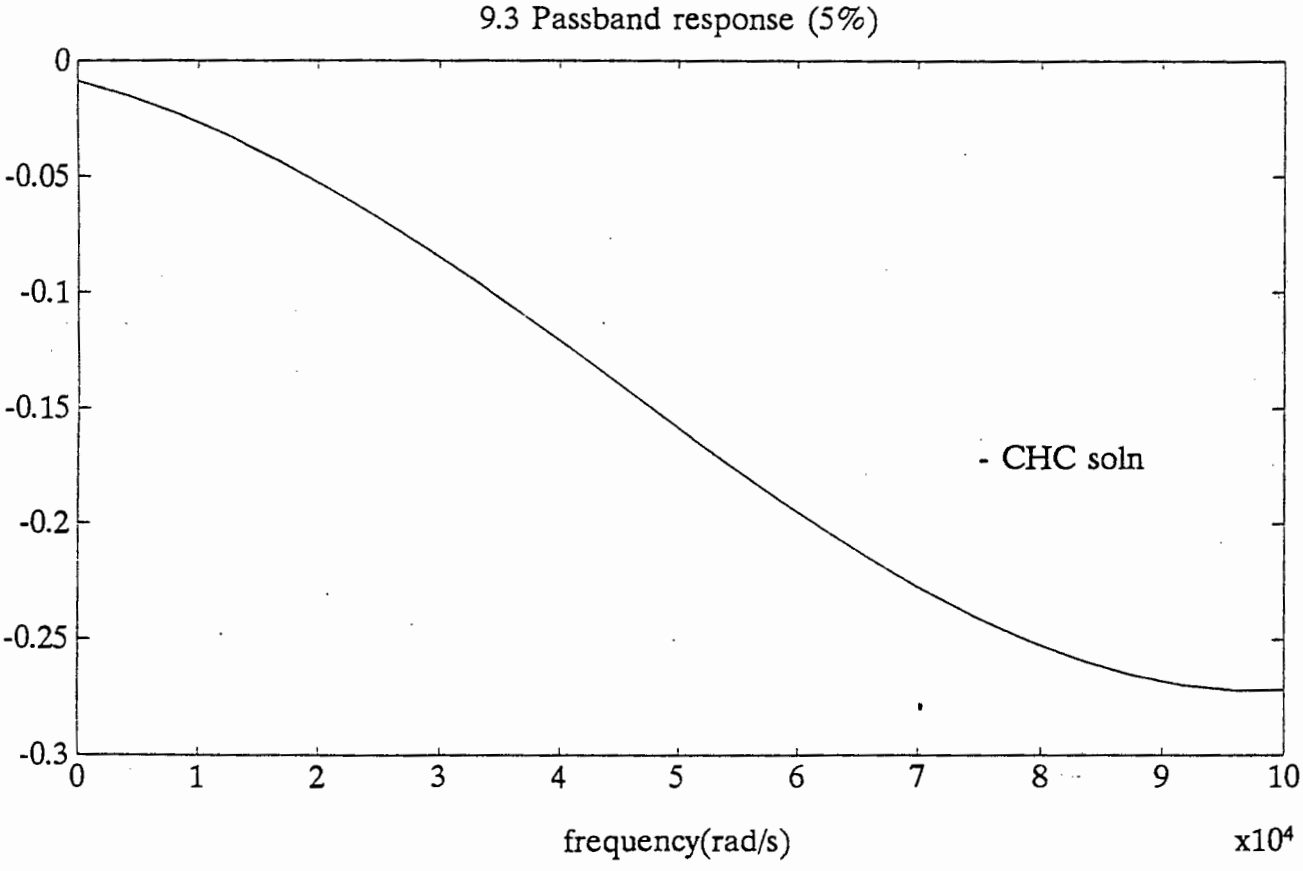
9.3 Passband response (1%)

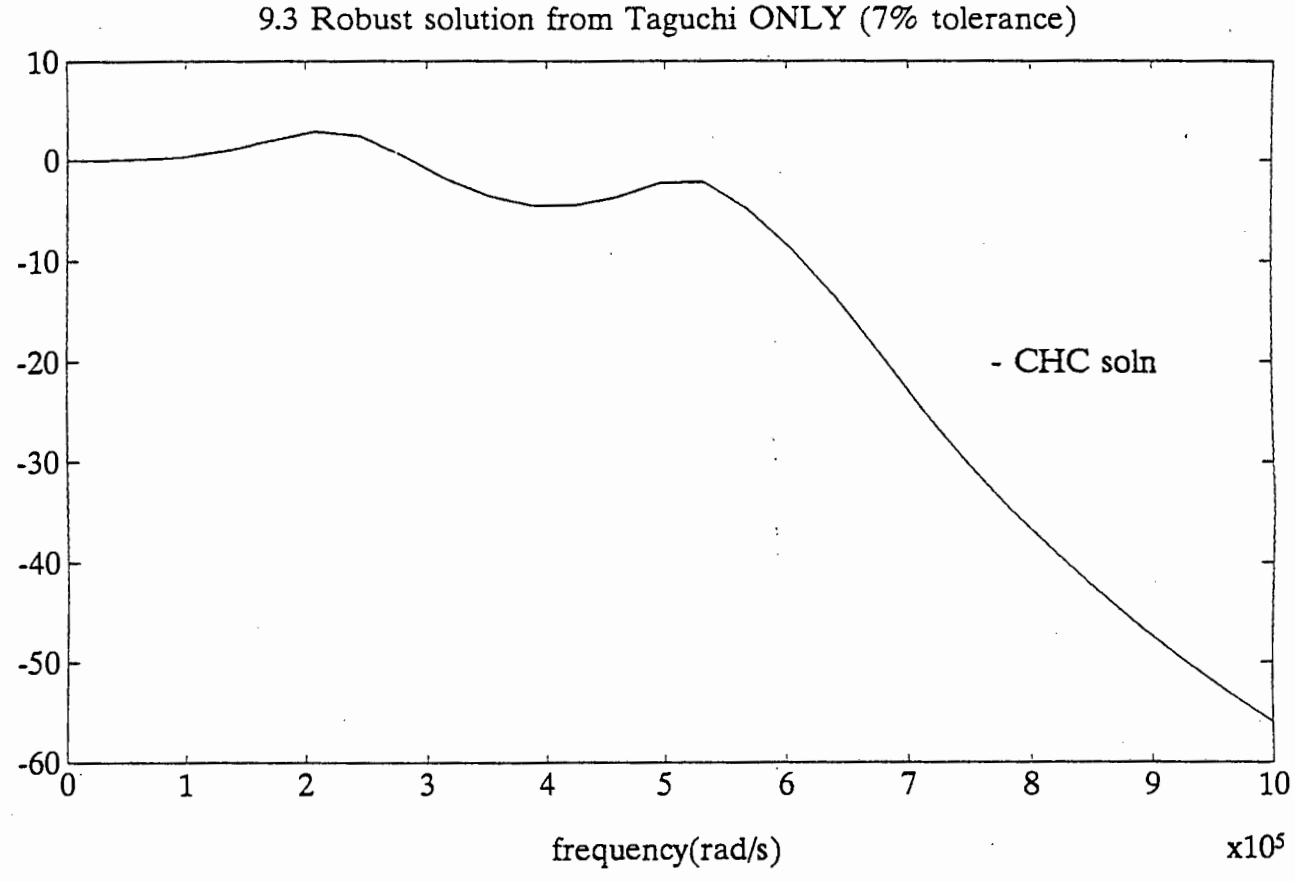


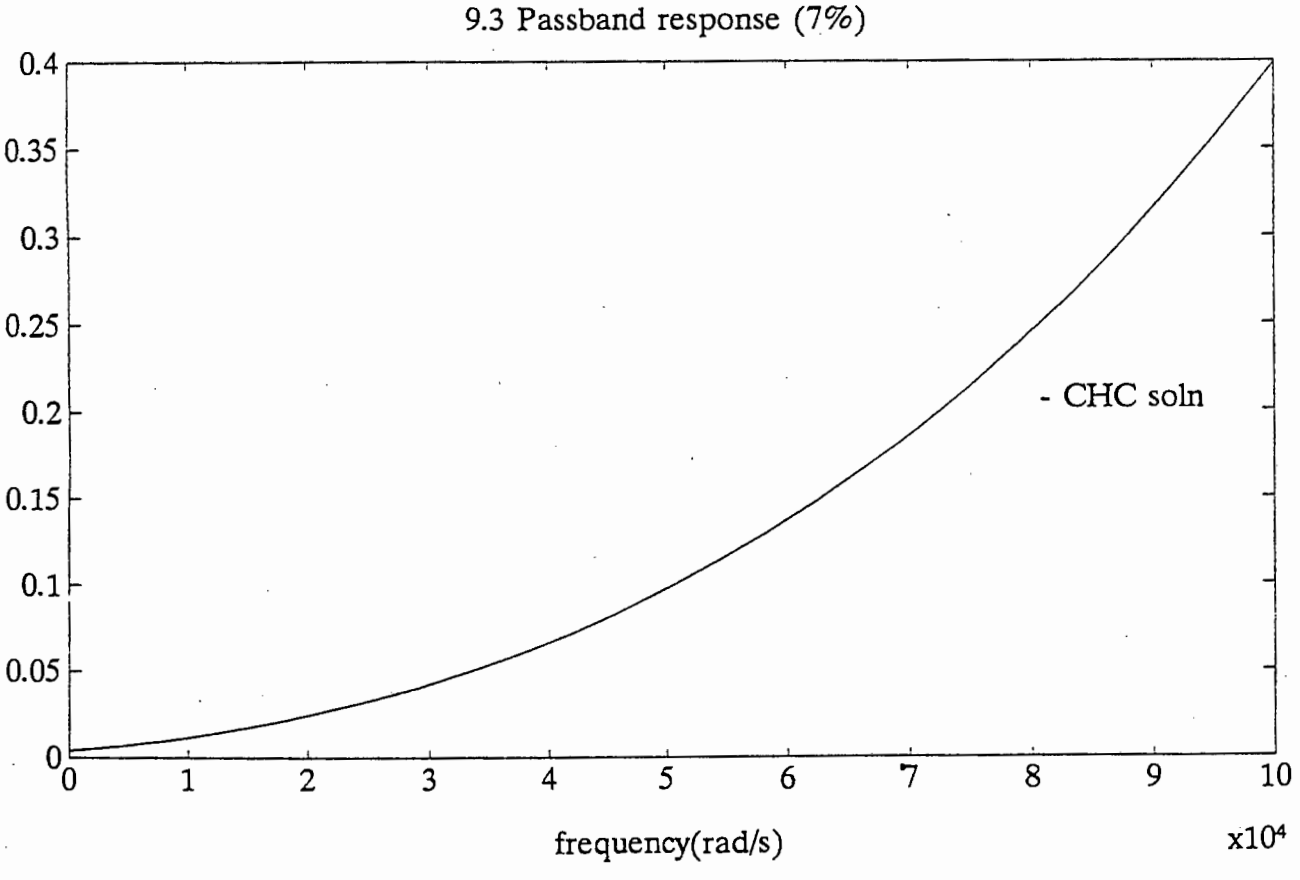

9.3 Robust solution from Taguchi ONLY (3% tolerance)
10
0
-10
-20
-30
-40
-50
-60
-70
-80
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5


9.3 Passband response (3%)
0
-0.005
-0.01
-0.015
-0.02
-0.025
-0.03
-0.035
-0.04
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4


9.3 Robust solution from Taguchi ONLY (5% tolerance)
10
0
-10
-20
-30
-40
-50
-60
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5


9.3 Passband response (5%)
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4


9.3 Robust solution from Taguchi ONLY (7% tolerance)
10
0
-10
-20
-30
-40
-50
-60
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^5


9.3 Passband response (7%)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
1
2
3
4
5
6
7
8
9
10
- CHC soln
frequency(rad/s)
x10^4