Genetic Algorithms for Filter Design

ENGR 90: Senior Design Project

Christine McGinn
Advisor: Professor Erik Cheever

May 7, 2016

1 Abstract

This project investigated the use of genetic algorithms to design several types of analog filters. This process involved "mathematically describing" the correct filter behavior in the fitness function so that the genetic algorithm could identify the correct transfer function coefficients to fit this mathematical description. For the Butterworth, Type I Chebyshev, and Bessel filters, the genetic algorithm was successful in finding transfer functions whose behavior closely resembled that of the desired filter. Another filter, the Equiripple Error filter, which has an equiripple group delay in the pass band, was considered, but the genetic algorithm was not successful in finding a transfer function which followed the desired characteristics.

2 Introduction

Genetic algorithms are a type of optimization algorithm that uses the process of natural selection as a template. A figure demonstrating the general flow of the algorithm is shown below. For biological problems, each individual in the initial population has a string of genes, which can be represented by bits, numerical values or even characters. First, these individual's genes are evaluated to determine how fit they are, that is how likely they are to survive natural selection. Three process then occur: mutation, crossover, and reproduction. Reproduction is when the individuals combine to create the new population, while giving the offspring more traits from the fitter individual. In crossover, the absolute fittest individuals are copied directly into the next generation. Finally, in mutation, some of the genes of the individuals in the next generation are randomly changed.

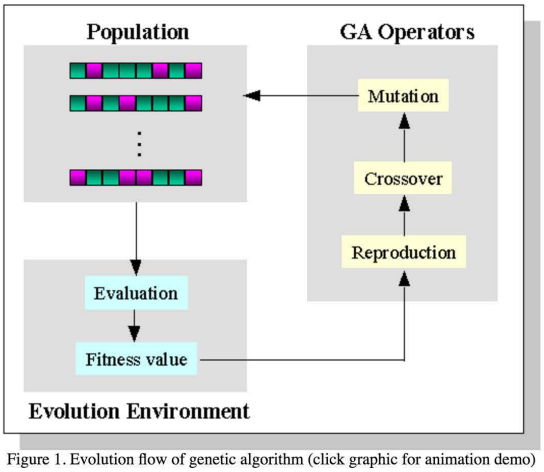

Figure 1. Evolution flow of genetic algorithm (click graphic for animation demo)

Figure 1: Flow chart describing genetic algorithms. Image source: http://www.ewh.ieee.org/soc/es/May2001/14/Begin.html.

For this project, each individual in the population is a separate transfer function and the genes are the coefficients of the transfer function. For example for the following the transfer function

$$H = \frac{10}{s^2 + 5s + 10}$$

the individual would be a one dimensional array, $(10\ 1\ 5\ 10)$. Reproduction involves averages between individuals in this case, but mutation and crossover work much the same as before.

This application of the genetic algorithm was applied to four different filters in this project: Butterworth, Type I Chebyshev, Bessel, and Equiripple Error filters in this project. These filters have different merits, hence their inclusion in this project. The Butterworth filter is maximally flat through the passband, which leads to all frequencies in the passband being attenuated only very slightly. Type I Chebyshev has much more attenuation in the stopband than the Butterworth filter, but this comes at the price of ripples in the passband. This attenuation gets even higher with a higher order filter, but again the price is more ripples in the passband. The Bessel filter has many of the same characteristics as the Butterworth filter, but it also has a maximally flat group delay which preserves the waveform of the incident signals. Finally, the Equiripple Error filter has ripples in the passband of the group delay, which corrects for some of the problems associated with the Bessel filter, such as its slow attenuation. The Equiripple Error filter is relatively uncommon and only used for very specialized audio applications so information about it can be hard to find. However, the others are fairly ubiquitous which led to success in applying genetic algorithms to their design.

3 Theory

The various filters that were implemented in this project have several defining characteristics and mathematical representations. The second order Butterworth filter has a derived transfer function of

$$H = \frac{\omega_0^2}{s^2 + \sqrt{2}\omega_0 s + \omega_0^2}$$

where ω_0 is the cutoff frequency in rad/s. The Butterworth filter has a maximally flat gain overall. For the Butterworth filter, the cutoff frequency has a magnitude of -3 dB, and at 0 rad/s, the filter has a magnitude of 0 dB. The Bode plot of the derived Butterworth filter with a cutoff frequency of 100 Hz is shown below.

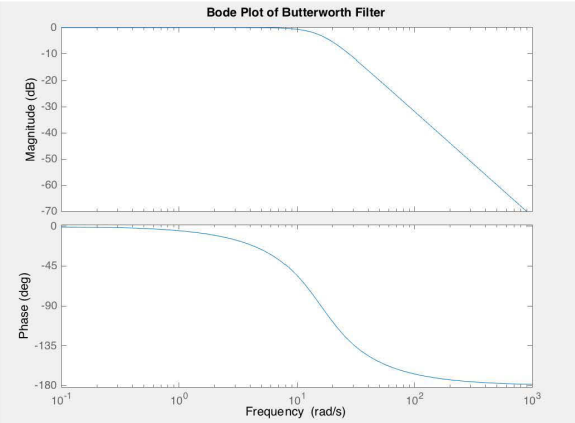

Figure 2: Designed second order Butterworth filter with cutoff frequency of 100 Hz

The Type I Chebyshev filter has several ripples in the passband, which leads to the desirable quality of more attenuation in the stopband. For a second order filter, there is only one ripple. Mathematically, at 0 rad/s and the cutoff frequency, the magnitude must be -3dB, but that in the passband, there also must be a point where the magnitude hits 0dB. The Bode plot of this filter is shown below.

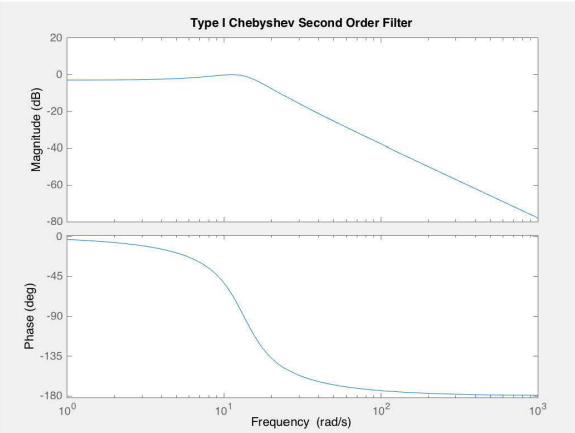

Figure 3: Designed second order Chebyshev Type I filter with cutoff frequency of 100 Hz

The third order Type I Chebyshev filter has two ripples in the passband. At 0 rad/s, the magnitude is 0 dB, and at the cutoff frequency the magnitude is -3 dB. The magnitude reaches 0 dB and -3dB at one point each in the passband to cause the ripples. The attenuation in the stopband is even higher for a third order filter as can be seen in the bode plot below.

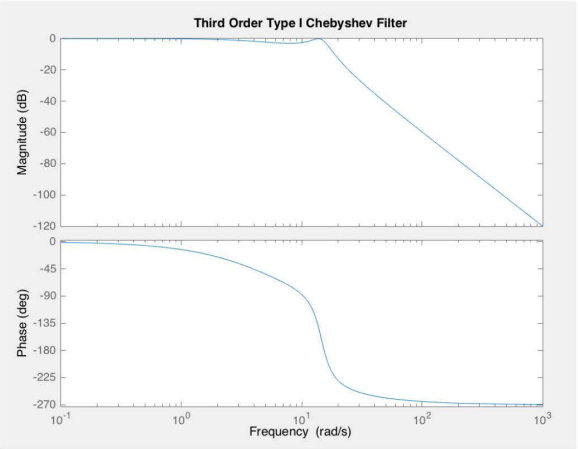

Figure 4: Designed third order Chebyshev Type I filter with cutoff frequency of 100 Hz

The Bessel filter has a bode plot very similar to that of a Butterworth filter, but some of the defining characteristics are different. At the desired cutoff frequency, the Bessel bode plot does not have a magnitude of -3 dB like the Butterworth filter. The third order filter does have a phase of -135 degrees at the cutoff frequency like many other third order filters. Its distinguishing characteristic is its maximally flat group delay. The group delay τ_g is defined as

$$\tau_g = -\frac{d\phi}{d\omega}$$

where ϕ is the phase of the filter. The bode plot of the ideal Bessel filter is shown below, as well as its group delay.

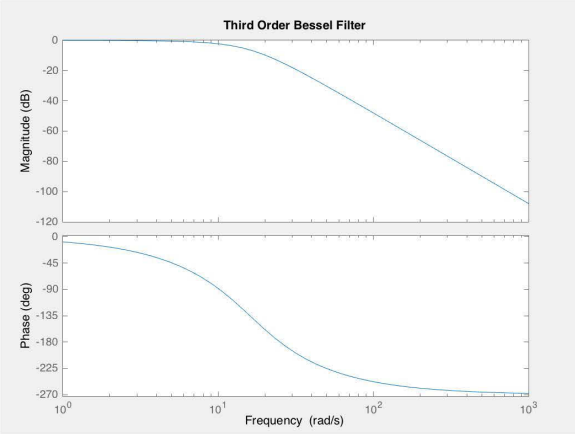

Figure 5: Designed third order Bessel filter with cutoff frequency of 100 Hz

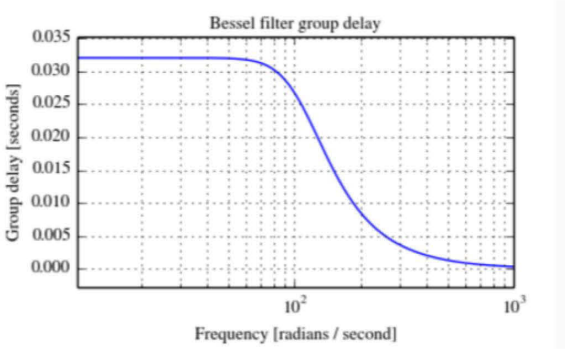

Figure 6: Group delay for Bessel filter. Image source: http://docs.sc ipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.bessel.html.

The group delay of the Equiripple Error filter has some ripples in the passband, which is its defining characteristic. The plot below shows the group delay for a phase ripple of 0.5 degrees for varying filter orders.

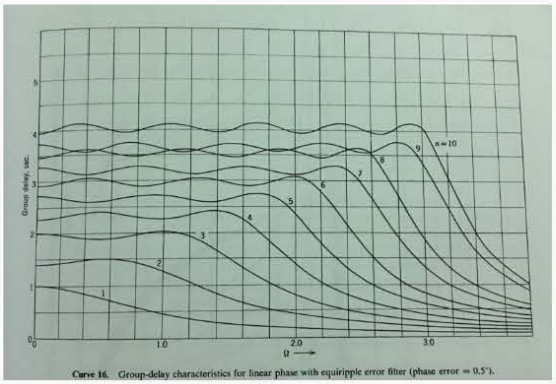

Figure 7: Group delay for Equiripple Error Filter. Image source: Zverev, Anatol I. Handbook of Filter Synthesis. New York: Wiley, 1967.

4 Methods

For this project, MATLAB was used to run the genetic algorithm and design the filters. MATLAB's genetic algorithm function was used because it is convenient and robust. For reproduction, MATLAB uses a weighted average of the two individuals with the fitter individual holding the higher weight. MATLAB's stall conditions were very important as the default stall condition would only halt the algorithm when the fitness value of the fittest individual came within 1×10^{-4} for many generations. These tolerances were much smaller than necessary for this project. MATLAB's genetic algorithm function also allows highly nonlinear fitness function and constraints, which was necessary for designing filters. Counterintuitively, MATLAB's genetic algorithm function considers the fittest individuals to be those with the lowest fitness function, and thus the final result will minimize the fitness function. To use MATLAB's genetic algorithm function with constraints, two other functions were needed: one containing the fitness function which creates the fitness value for each individual and one containing the constraint. For each type of filter used in this project, both were needed. The fitness function helped to specify the filter's behavior such as monotonically decreasing gain in the case of the Butterworth filter or ripples in the passband for the Chebyshev filter. The constraint function helped to describe points that the gain or phase must hit, such as -3dB gain for the cutoff frequency and 0 dB gain at 0 rad/s for the Butterworth filter. Each function was different for each filter to obtain the desired filter behavior.

For the Butterworth filter, the only condition in the fitness function was that every point had to be less than the previous point, thus leading to a maximally flat gain. For every point that was not less than the previous point, the fitness value was increased by one, so that a maximally flat gain gives a fitness value of zero, the minimum possible value. The constraint function held two constraints, that the gain at the cutoff frequency had to be -3 dB and 0 dB at 0 rad/s.

For the second order Type I Chebyshev filter, the number of conditions increased for the fitness function to create a passband ripple of R_p. For this project, the passband ripple chosen was -3 dB. In the passband, it was required that the gain have a point within 0.1 dB of 0dB. If this did not happen, a high penalty was added to the fitness value. Additionally, in the passband, the gain must be constantly decreasing and have the highest slope possible. For any point that was not decreasing, one was again added to the fitness value, and more was added in proportion to the slope so that a less steep slope added more to the fitness value. The constraint function required that at 0 rad/s

and the cutoff frequency, the gain was -R_p dB. For the third order Type I Chebyshev filter, the fitness function had similar conditions as the second order filter overall, but some changes had to be made in the passband to reflect the extra rippling. In the passband, it was required that the gain hit 0 dB and -R_p within a 0.1 dB range. The stopband characteristics remained exactly the same from the second order filter though to ensure high attenuation in the stopband. The constraint had to be changed as it now needed to require that at 0 rad/s there is a gain of 0 dB and a gain of -R_p dB at the cutoff frequency.

The third order Bessel filter requires a maximally flat group delay so in the fitness function, the group delay must be calculated and then evaluated to make sure each point is less than the previous one. Again, for each point that was increasing, one was added to the fitness value. Since the group delay is the negative derivative of the filter's phase, the negative slope between every two points was calculated to approximate the continuous derivative. The constraints for this case were that at 0 rad/s, the gain must be 0 dB and at the cutoff frequency, the phase had to be -135 degrees.

For the Equiripple Error filter, general characteristics were difficult to find so the code was less general. The code was very similar to the Chebyshev code, but applied to the group delay instead of filter gain. Following the plot of the group delay shown in the theory section for a second order filter with a 0.5 degree phase ripple, the values of the bounds of the ripple were approximated and hard coded into the fitness function. Since Zverev, one of the only sources of information about this filter, provided very little information about the phase and magnitude plots of the Equiripple Error filter, it was difficult to add constraints which aid in finding the correct values, such as in the case of the Bessel filter. In this case then the only constraint was that at 0 rad/s, the gain is 0 dB.

For designing the above filters using genetic algorithms, an initial population close to the derived values was added to help the code converge faster. For the Butterworth filter, no initial population was needed, but for the second order Chebyshev filter, an initial population that was thirty percent less than the derived values as well as the Butterworth coefficients were used before good results could be found without an initial population. For the third order Chebyshev filter, an initial population thirty percent less than the derived values was also tried. For the Bessel filter, an initial population ten percent less than the derived values was the only option that gave a result in a reasonable run time. With so little information about the Equiripple Error filter, there was not a good option for an initial population. In order to ensure that the generated filters are stable, a restriction that the denominator values must be positive was added.

5 Results

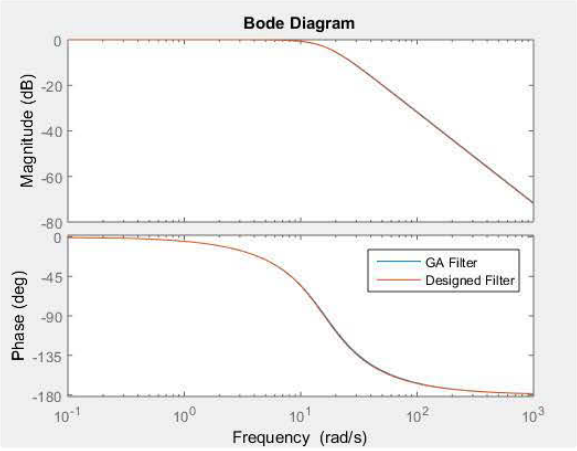

Figure 8: Bode plot for genetic algorithm's result for a second order Butterworth filter with cutoff frequency 100 Hz

```
Hfinal =

             9.101
  -------------------------------
  0.03476 s^2 + 0.8076 s + 9.106

Continuous-time transfer function.


zero =

   -0.0054


cutoff =

   -3.0000


tf1 =

          253.3
  ---------------------
  s^2 + 22.51 s + 253.3

Continuous-time transfer function.
```

Figure 9: Values for genetic algorithm's result for a second order Butterworth filter with cutoff frequency 100 Hz with no initial population

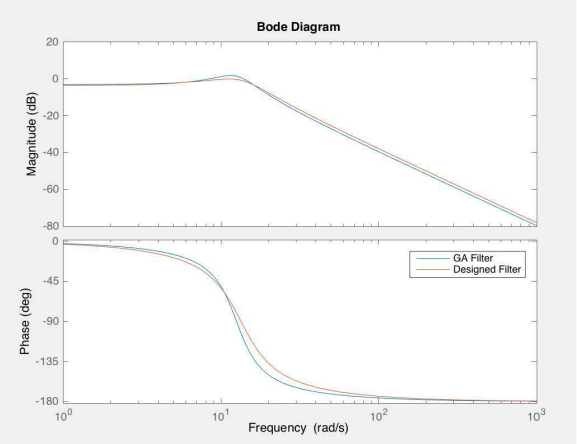

Figure 10: Bode plot for genetic algorithm's result for a second order Type I Chebyshev filter with cutoff frequency 100 Hz with an initial population 30 % from the derived values

```
           86.3
  ---------------------------
  0.825 s^2 + 5.755 s + 129.7

Continuous-time transfer function.


zero =

   -3.4882


cutoff =

   -2.9468

>> tf1

tf1 =

          127
  ----------------------
  s^2 + 10.26 s + 179.3
```

Figure 11: Values for genetic algorithm's result for a second order Type I Chebyshev filter with cutoff frequency 100 Hz with an initial population 30 % from the derived values

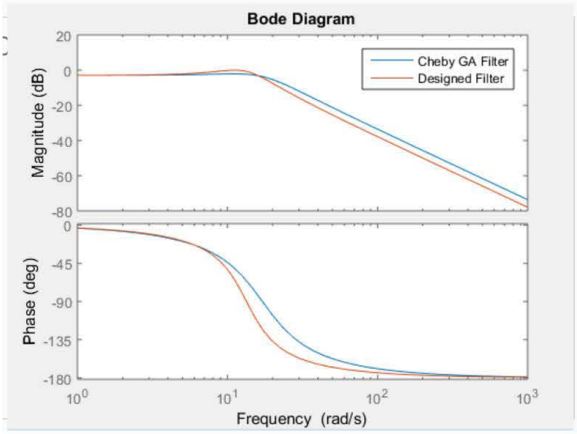

Figure 12: Bode plot for genetic algorithm's result for a second order Type I Chebyshev filter with cutoff frequency 100 Hz using the second order Butterworth values as an initial population

```
Hfinal =

        207.8
  --------------------
  s^2 + 18.4 s + 291.9

Continuous-time transfer function.


zero =

   -2.9400


cutoff =

   -3.0524

>> bode(Hfinal,tf(b,a))
>> legend('Cheby GA Filter', 'Designed Filter')
>> tf(b,a)

ans =

          127
  ---------------------
  s^2 + 10.26 s + 179.3

Continuous-time transfer function.
```

Figure 13: Values for genetic algorithm's result for a second order Type I Chebyshev filter with cutoff frequency 100 Hz using the second order Butterworth values as an initial population

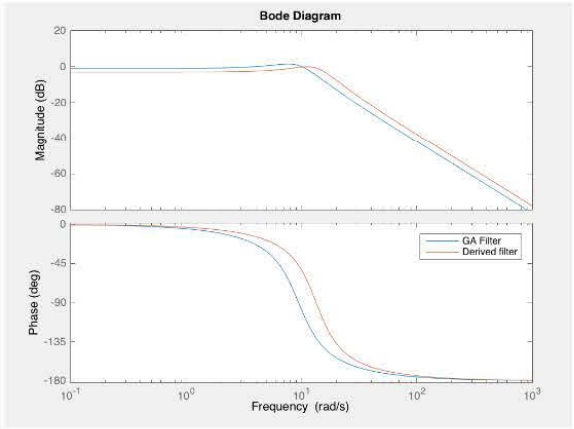

Figure 14: Bode plot for genetic algorithm's result for a second order Type I Chebyshev filter with no initial population used

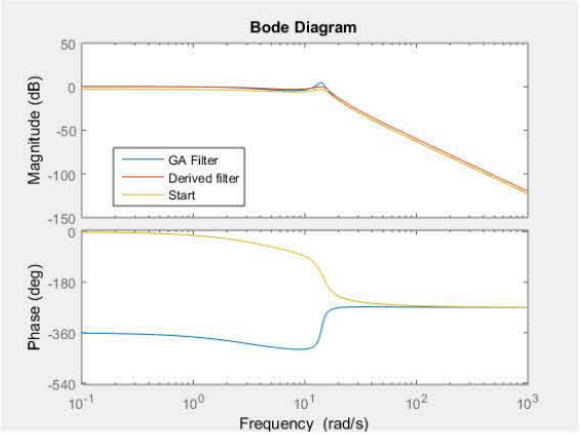

Figure 15: Bode plot for genetic algorithm's result for a third order Type I Chebyshev filter with cutoff frequency 100 Hz with an initial population 30 % from the derived values

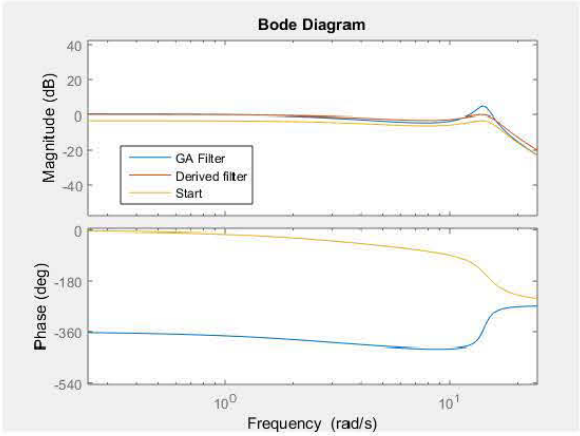

Figure 16: Passband of bode plot for genetic algorithm's result for a third order Type I Chebyshev filter with cutoff frequency 100 Hz with an initial population 30 % from the derived values

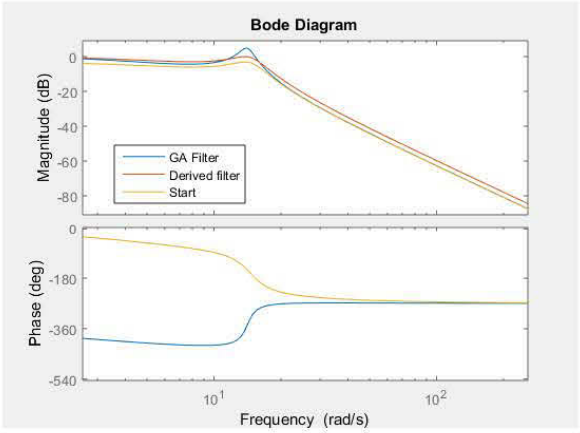

Figure 17: Stopband of bode plot for genetic algorithm's result for a third order Type I Chebyshev filter with cutoff frequency 100 Hz with an initial population 30 % from the derived values

```
Hfinal =

                720.4
  ---------------------------------
  s^3 + 1.482 s^2 + 192.7 s + 696.4

Continuous-time transfer function.


zero =

   -0.0056


cutoff =

   -2.9947
```

Figure 18: Values for genetic algorithm's result for a third order Type I Chebyshev filter with cutoff frequency 100 Hz with an initial population 30 % from the derived transfer function $1010/(s^3 + 9.505s^2 + 235.2s + 1010)$

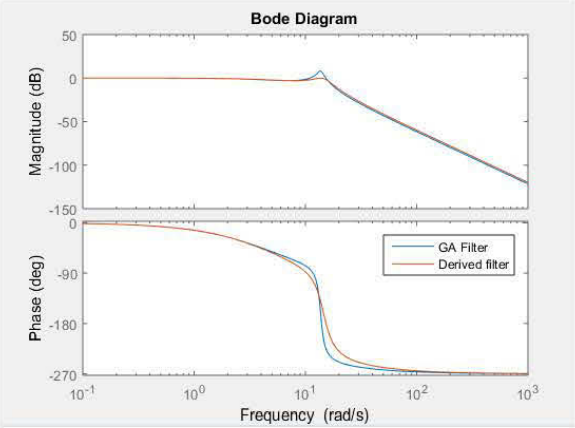

Figure 19: Bode plot for genetic algorithm's result for a third order Type I Chebyshev filter with no initial population used

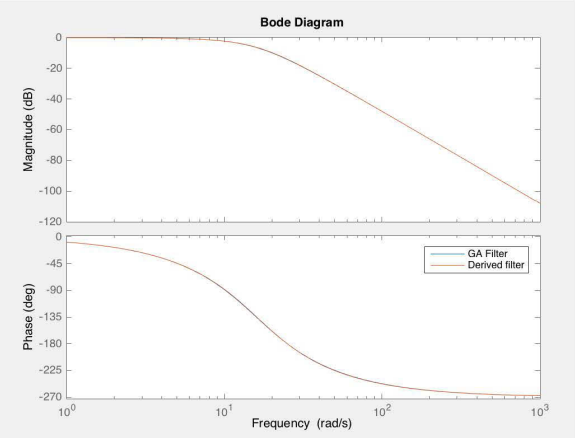

Figure 20: Bode plot for genetic algorithm's result for a third order Bessel filter with cutoff frequency 100 Hz with an initial population 10 % from the derived values

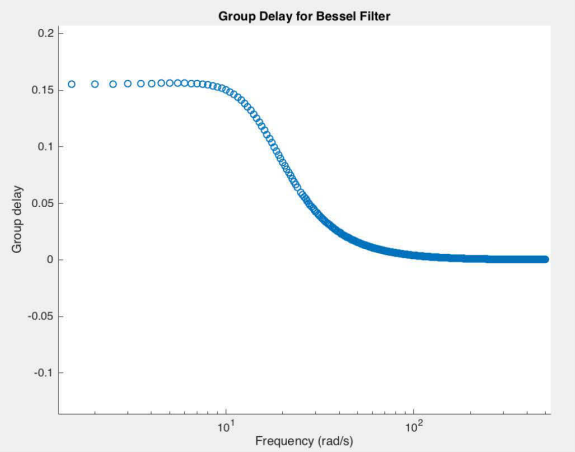

Figure 21: Group delay for genetic algorithm's result for a third order Bessel filter with cutoff frequency 100 Hz with an initial population 10 % from the derived values

```
Hfinal =

                3629
  -----------------------------------
  0.9 s^3 + 35.35 s^2 + 562.5 s + 3629

Continuous-time transfer function.


zero =

   -0.0198


angle2 =

 -135.0066

>> tf1

tf1 =

               4031
  --------------------------------
  s^3 + 38.72 s^2 + 624.7 s + 4031

Continuous-time transfer function.
```

Figure 22: Values for genetic algorithm's result for a third order Bessel filter with cutoff frequency 100 Hz with an initial population 10 % from the derived values

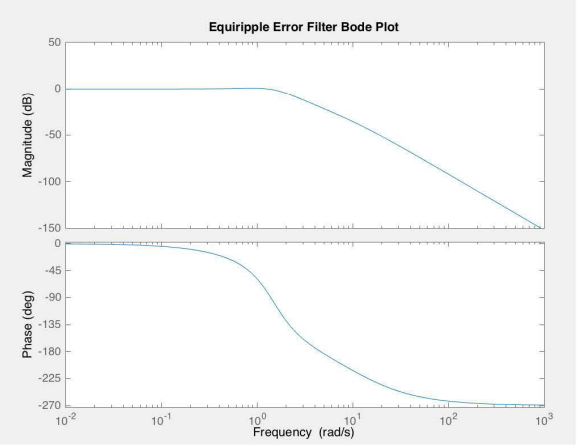

Figure 23: Bode plot for genetic algorithm's result for a third order Equiripple Error filter with cutoff frequency of 100 Hz

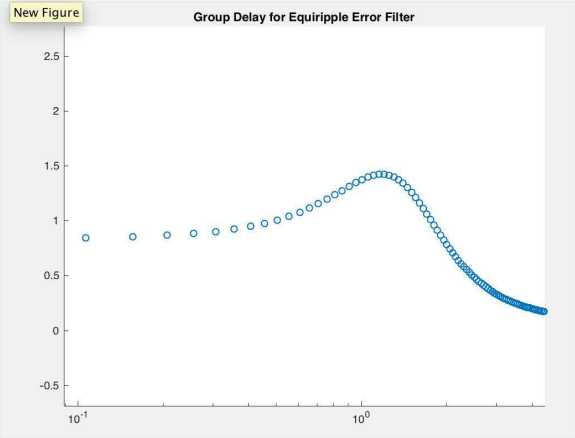

Figure 24: Group delay for genetic algorithm's result for a third order Equiripple Error filter with cutoff frequency of 100 Hz

6 Discussion

The filter coefficients that resulted from the genetic algorithm generally worked well. The Butterworth filter was very successful. The filter coefficients converged very closely on to the derived values, and adjusting the genetic algorithm's values gives coefficients of approximately four percent difference. The constrained values at 0 rad/s and the cutoff frequency are less than the provided tolerance of 0.1.

Designing the second order Chebyshev filter also was successful but was more difficult to achieve. The two initial conditions, thirty percent from the derived values and the Butterworth values, both worked very well. The initial population close to the derived values gave transfer function coefficients were approximately fifteen percent from the correct values, but the bode plot showed even better agreement with the desired behavior. The gain at the cutoff frequency was well within the 0.1 tolerance of the desired value, but was not nearly as close at 0 rad/s which is likely the result of the error in the transfer function coefficients. For the initial population with Butterworth filter coefficients, the results were even more mixed. The two constrained values at 0 rad/s and the cutoff frequency were very close to the desired filters, much closer than for the closer initial population, but some of the transfer function coefficients were almost double the desired values. The bode plot shows similar behavior, but the numerical results show that the final transfer function is not as good a result as it appears. The result of the genetic algorithm with no initial population was similar to the result with the Butterworth values. Similar behavior between the derived filter and the genetic algorithm's filter were again seen, but it is clear that the numbers for the two are not exactly the same. This kind of result is to be expected as the exact designed filter coefficients are not necessary to satisfy the fitness function's coefficients, and so several solutions may be recognized as optimal by the genetic algorithm.

For the third order Chebyshev filter, only one initial population was attempted before a successful result without one was generated. For an initial population thirty percent less than the correct values, similar behavior for the magnitude plot was observed. From Figures 16 and 17, the genetic algorithm's result showed improvement from the initial values toward the designed values in both the passband and the stopband. The numerical results at 0 rad/s and the cutoff frequency are less than one percent from the desired values so there is excellent agreement. The genetic algorithm's

values for the transfer function are significantly less than the designed values, and this difference likely explains the large disagreement between the phase plots.

The result for Bessel filter was extremely close to the designed results when the initial population was ten percent from these values. The coefficients for the transfer function were again less than one percent different from the designed values. The two constrained values, the gain at 0 rad/s and the angle at the cutoff frequency, are less than two percent and one percent, respectively. The bode plot also shows excellent agreement and the group delay shows the desired maximally flat behavior. A result for the Bessel filter with no initial population was extremely difficult to obtain as the code for this filter often took the longest to run, often lasting over eight hours.

The Equiripple Error filter did not work as expected, as can be seen by comparing the group delay of the genetic algorithm's result and the plot from Zverev. This filter had the least easily recognizable constraints of any in this project, which likely meant there was not enough information for a better result.

To find results without an initial population throughout this project, it would often be necessary to increase the population size to find a global rather than local minimum. For an unspecified initial population, MATLAB fills in the array with random values. A larger population size then increases the chances of the random variables being closer to the global minimum, or the designed filter values, and thus that the final result will be closer to these desired values. If the population size is not big enough, the code will often find values that it believes are a minimum, but are really just a local minimum whose behavior is nothing close to what is desired.

7 Conclusions

Using genetic algorithms for filter design seems to work well under certain conditions, but is not always reliable. It is necessary to specify the correct stall conditions as well as total population number, and often fill part of the initial population matrix with values close to the derived values. However, for well understood filters, MATLAB's genetic algorithm function can find coefficients of the transfer functions of these filters that display the desired behavior for suitable fitness and constraint functions. Second or third order Butterworth, Type I Chebyshev, and Bessel filters were all designed successfully during this process. A less understood filter such as the Equiripple Error filter was much harder to design using genetic algorithms as doing so took a deep understanding of the behavior of the filter. While designing filters this way can be difficult, it is a rich and interesting subject.

8 Future Work

Given more time on this project, it would be interesting to do more research on the Equiripple Error filter and get a result with behavior closer to that desired. The current version of the code used in this project is very rudimentary, and could easily be made more automated and more robust. For example, now the two Chebyshev filters of different orders are totally separate codes, but a single code for Chebyshev filters of all orders could be conceived. The run time could also be optimized to help find solutions faster, which would likely aid in finding a result for the Bessel filter with no initial population.

9 References

1. Affenzeller, Michael. Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications. Boca Raton: CRC Press, 2009.

2. Bridge, Derek. "Genetic Algorithms." Techniques in Artificial Intelligence. Accessed May 1, 2016. http://www.cs.ucc.ie/ dgb/courses/tai/notes/handout12.pdf.

3. "Bessel Filter Design." SciPy Reference Manual. http://docs.sc ipy.org/doc/scipy-0.14.0/reference/generated/scipy.

4. Stensby, John. "Chebyshev Filters." August 31, 2011. http://www.ece.uah.edu/courses/ee426/Chebyshev.pdf.

5. Zverev, Anatol I. Handbook of Filter Synthesis. New York: Wiley, 1967.

10 Appendix A: Butterworth Filter Code

```
clear
fc0=100/(2*pi);
%Put in initial guess, ten percent lower than the correct values
X0=[fc0^2 1 sqrt(2)*fc0 fc0^2]
options.InitialPopulation = X0
%Set lower bounds, denominator coefficients must be positive
LB=[-Inf 0 0 0];
FitnessFunction = @(x) butter_filter_fitness(x(1), x(2), x(3), x(4),fc0);
nvars = 4;    % Number of variables
%Add constraint
ConstraintFunction = @(x) butter_constraint(x,fc0);
%Set the minimum
%options.FitnessLimit=0;
%Set tolerances for constraint and function
options.TolCon = 0.1;
options.TolFun = 0.1;
options.PopulationSize=300;
options.StallGenLimit=100;
%Set total time
options.TimeLimit=60*60;

[x,fval,exitflag,output,population] = ga(FitnessFunction,nvars,[],[],[],[],LB, [],
    ...
 ConstraintFunction, options);

%Final transfer function, values at zero and cutoff frequencies
Hfinal=tf([x(1)], [x(2) x(3) x(4)])
zero=20*log10(abs(evalfr(Hfinal,1*j)))
cutoff=20*log10(abs(evalfr(Hfinal,fc0*j)))

tf1=tf([X0(1)],[X0(2) X0(3) X0(4)])

bode(Hfinal, tf1)
legend('GA Filter', 'Designed Filter')

load chirp
sound(y,Fs)


function f = butter_filter_fitness(a,b,c,d,fc0)
% error of the fitness function where fc0 is the desired cutoff frequency

H=tf([a],[b c d])
mag=bode(H, {2*pi*0.001,2*pi*fc0*10});
```

```
f=0;
for i=2:length(mag)
    if mag(i-1)< mag(i)
        f=f+1;
    end
end

end

 function [c, ceq] = butter_constraint(x,fc0)
   %c contains all the constraints
   H=tf([x(1)], [x(2) x(3) x(4)]);
   c = [];
   ceq = [20*log10(abs(evalfr(H,1*j))),20*log10(abs(evalfr(H,fc0*j)))+3];
 end
```

11 Appendix B: Second Order Chebyshev Filter Code

```
clear
%Set cutoff frequency, passband ripple
fc0=100/(2*pi);
Rp=3;

%Set order for theoretical design
%Filter order
N=2;
%Design
[b,a]=cheby1(N,Rp,fc0,'s');

%Give it a start value 10% of the correct value.
Y0=[b a];
X0=Y0(find(Y0))
options.InitialPopulation=X0*0.7

%Try butterworth coefficients as a start value
%X0=[fc0^2 sqrt(2)*fc0 fc0^2]
%options.InitialPopulation = repmat(X0,20,1)

tf1=tf(b,a)
%tf1=tf([fc0^2] ,[1 sqrt(2)*fc0 fc0^2])

%Set lower bounds, denominator coefficients must be positive
LB=[-Inf 0 0 0];
FitnessFunction = @ (x) cheby_filter_fitness2(x(1), x(2), x(3),fc0,Rp);
nvars = 3;    % Number of variables
%Add constraint
ConstraintFunction = @(x) cheby_constraint2(x,fc0,Rp);
%Set tolerances for constraint and function
%options.TolCon = 0.01;
%options.TolFun = 0.01;
%options.PopulationSize=500;
%options.StallGenLimit=100;
%Set total time
%options.TimeLimit=180*60;
options=gaoptimset('PlotFcn',@gaplotbestf, 'TolCon', 0.01, 'TolFun',...
    0.01,'PopulationSize', 500, 'StallGenLimit', 100, 'TimeLimit', 180*60)

[x,fval,exitflag,output,population] = ga(FitnessFunction,nvars,[],[],...
    [],[],LB, [],ConstraintFunction, options);
```

```
Hfinal=tf([x(1)], [1 x(2) x(3)])
zero=20*log10(abs(evalfr(Hfinal,1*j)))
cutoff=20*log10(abs(evalfr(Hfinal,fc0*j)))
figure
bode(Hfinal, tf1)
legend('Cheby GA Filter', 'Designed Filter')

load chirp
sound(y,Fs)

function f = cheby_filter_fitness2(a,c,d,fc0,Rp)


% error of the fitness function where fc0 is the desired cutoff frequency
H=tf([a],[1 c d])
fr=linspace(2*pi*0.001,2*pi*fc0*100,10000);

%Compute the bode plot
bodemag=zeros(1,length(fr));

for i=1:length(fr)
bodemag(i)=20*log10(abs(evalfr(H,fr(i)*j)));
end
%Find index closest to the cutoff frequency
[m,k] = min(abs(fr-fc0))
%Start the fitness function at zero
f=0;

% Aiming for certain dB ripple, don't let it get higher or lower
for i=1:k-1
    if abs(bodemag(i))>Rp
     f=f+1;
    end
end

%Make sure it actually ripples between -3 dB and 0 dB
%if not penalize heavily

minibodemag=abs(bodemag(1:k-1));

y = find(minibodemag>-Rp & minibodemag<-Rp+0.1);
z = find(abs(minibodemag)<0.1);

if isempty(y)
     f=f+100;
end

if isempty(z)
     f=f+100;
end

%Needs to monotonically decrease in the stopband
for i=k:length(bodemag)
    if bodemag(i-1)< bodemag(i)
         f=f+1;
    end
end

%Penalize if the drop off isn't good enough
m2=(bodemag(end)-bodemag(k))/(fr(end)-fr(k));
f=f+1/abs(m2)*100;


end
```

```
function [c, ceq] = cheby_constraint2(x,fc0,Rp)
   %c contains all the constraints
   H=tf([x(1)], [1 x(2) x(3)]);
   c = [];
   ceq = [20*log10(abs(evalfr(H,1*j)))+Rp, 20*log10(abs(evalfr(H,fc0*j)))+Rp];
end
```

12 Appendix C: Third Order Chebyshev Filter Code

```
clear
%Set cutoff frequency, passband ripple
fc0=100/(2*pi);
Rp=3;

%Set order for theoretical design
%Filter order
N=3;
%Design
[b,a]=cheby1(N,Rp,fc0,'s');

%Give it a start value 30% of the correct value.
Y0=[b a];
X0=Y0(find(Y0));
options.InitialPopulation=X0*0.7

tf1=tf([X0(1)],[X0(2) X0(3) X0(4) X0(5)])


%Set lower bounds, denominator coefficients must be positive
LB=[-Inf 0 0 0 0];
FitnessFunction = @ (x) cheby_order3_fitness(x(1), x(2), x(3), x(4),...
    x(5),fc0,Rp);
nvars = 5;    % Number of variables
%Add constraint
ConstraintFunction = @(x) cheby_order3_constraint(x,fc0,Rp);
%Set tolerances for constraint and function
options.TolCon = 0.1;
options.TolFun = 0.1;
options.PopulationSize=300;
options.StallGenLimit=50;
%Set total time
options.TimeLimit=500*60;

[x,fval,exitflag,output,population] = ga(FitnessFunction,nvars,[],[],[],[],LB, [],
    ...
 ConstraintFunction, options);
Hfinal=tf([x(1)], [x(2) x(3) x(4) x(5)])
zero=20*log10(abs(evalfr(Hfinal,1*j)))
cutoff=20*log10(abs(evalfr(Hfinal,fc0*j)))

figure
bode(Hfinal, tf1)
legend('GA Filter', 'Derived filter')

load chirp
sound(y,Fs)

function f = cheby_order3_fitness(a,b,c,d,f,fc0,Rp)
```

```
50
% error of the fitness function where fc0 is the desired cutoff frequency
52 H=tf([a],[b c d f])
fr=linspace(2*pi*0.001,2*pi*fc0*100,10000);
54
%Compute the bode plot
56 bodemag=zeros(1,length(fr));

58 for i=1:length(fr)
bodemag(i)=20*log10(abs(evalfr(H,fr(i)*j)));
60 end
%Find index closest to the cutoff frequency
62 [m,k] = min(abs(fr-fc0))
%Start the fitness function at zero
64 f=0;

66 % Aiming for certain dB ripple, don't let it get higher or lower
for i=1:k-1
68     if abs(bodemag(i))>Rp
        f=f+1;
70     end
end
72
%Make sure it actually ripples between -3 dB and 0 dB
74 %if not penalize heavily

76 minibodemag=abs(bodemag(1:k-1));

78 y = find(minibodemag>-Rp & minibodemag<-Rp+0.1);
z = find(abs(minibodemag)<0.1);
80
if isempty(y)
82      f=f+100;
end
84
if length(z)<2
86      f=f+100;
end
88
%Needs to monotonically decrease in the stopband
90 for i=k:length(bodemag)
    if bodemag(i-1)< bodemag(i)
92          f=f+1;
    end
94 end

96 %Penalize if the drop off isn't good enough
m2=(bodemag(end)-bodemag(k))/(fr(end)-fr(k));
98 f=f+1/abs(m2)*100;

100 end

102 function [c, ceq] = cheby_order3_constraint2(x,fc0,Rp)
   %c contains all the constraints
104    H=tf([x(1)], [x(2) x(3) x(4) x(5)]);
    c = [];
106    ceq = [20*log10(abs(evalfr(H,1*j))),20*log10(abs(evalfr(H,fc0*j)))+Rp];
end
```

13 Appendix D: Bessel Filter Code

```
clear
%Set cutoff frequency, passband ripple
fc0=100/(2*pi);
Rp=3;

%Set order for theoretical design
%Filter order
N=3;
%Design
[b,a]=besself(N,fc0);

%Give it a start value 10% of the correct value.
Y0=[b a];
X0=Y0(find(Y0));
options.InitialPopulation=X0*0.9

tf1=tf([X0(1)],[X0(2) X0(3) X0(4) X0(5)])


%Set lower bounds, denominator coefficients must be positive
LB=[-Inf 0 0 0 0];
FitnessFunction = @(x) bessel_order3_fitness(x(1), x(2), x(3), x(4),...
    x(5),fc0);
nvars = 5;    % Number of variables
%Add constraint
ConstraintFunction = @(x) bessel_order3_constraint(x,fc0);
%Set tolerances for constraint and function
options.TolFun=0.1;
options.TolCon = 0.1;
options.PopulationSize=300;
options.StallGenLimit=25;
%Set total time
options.TimeLimit=500*60;
options.EliteCount=300*0.1;
options.PlotFcns=@gaplotbestf

[x,fval,exitflag,output,population] = ga(FitnessFunction,nvars,[],[],...
    [],[],LB, [],ConstraintFunction, options);

Hfinal=tf([x(1)], [x(2) x(3) x(4) x(5)])
zero=20*log10(abs(evalfr(Hfinal,1*j)))
angle2=angle(evalfr(Hfinal,fc0*j))*180/pi

figure
bode(Hfinal, tf1)
legend('GA Filter', 'Derived filter')

%Compute the final group delay
fr=linspace(2*pi*0.001,2*pi*fc0*100,1000);

%Compute the bode plot phase
bodeangle=zeros(1,length(fr));

for i=1:length(fr)
bodeangle(i)=angle(evalfr(Hfinal,fr(i)*j));
end

h=(2*pi*fc0*100-2*pi*0.001)/1000;
gd = -diff(bodeangle)/h;

```

```
figure
scatter(fr(1:999),gd)

load chirp
sound(y,Fs)


function f = bessel_order3_fitness(a,b,c,d,g,fc0,Rp)

% error of the fitness function where fc0 is the desired cutoff frequency
H=tf([a],[b c d g]);
fr=linspace(2*pi*0.001,2*pi*fc0*100,1000);

%Compute the bode plot phase
bodeangle=zeros(1,length(fr));

for i=1:length(fr)
bodeangle(i)=angle(evalfr(H,fr(i)*j));
end

%Need to calculate group delay, which is negative derivative of phase
%derivative
h=(2*pi*fc0*100-2*pi*0.001)/1000;
gd = -diff(bodeangle)/h;

f=0;
%Needs to monotonically decrease
for i=2:length(gd)
    if gd(i-1)< gd(i)
        f=f+1;
    end
end
f
end

function [c, ceq] = cheby_order3_constraint2(x,fc0,Rp)
   %c contains all the constraints
   H=tf([x(1)], [x(2) x(3) x(4) x(5)]);
    c = [];
    ceq = [20*log10(abs(evalfr(H,1*j))),20*log10(abs(evalfr(H,fc0*j)))+Rp];
end
```

14 Appendix E: Equiripple Error Filter Code

```
clear
%Set cutoff frequency
fc0=100/(2*pi);

%Filter order
N=3;
%Design
[b,a]=besself(N,fc0);

%Give it Bessel filter values as initial values
Y0=[b a];
X0=Y0(find(Y0));
options.InitialPopulation=repmat([X0(1) X0(3) X0(4) X0(5)],10,1)

%Set lower bounds, denominator coefficients must be positive
```

```
LB=[-Inf 0 0 0 0];
FitnessFunction = @ (x) EP_order3_fitness(x(1), x(2), x(3), x(4),fc0);
nvars = 4;     % Number of variables
%Add constraint
ConstraintFunction = @(x) EP_order3_constraint(x,fc0);
%Set tolerances for constraint and function
options.TolFun = 0.1;
options.TolCon = 0.1;
options.PopulationSize=400;
options.StallGenLimit=50;
%Set total time
options.TimeLimit=500*60;
options.PlotFcns=@gaplotbestf

[x,fval,exitflag,output,population] = ga(FitnessFunction,nvars,[],[],...
    [],[],LB, [],ConstraintFunction, options);

Hfinal=tf([x(1)], [1 x(2) x(3) x(4)])
figure
bode(Hfinal)
title('Equiripple Error Filter Bode Plot')

%Compute the final group delay
fr=linspace(2*pi*0.001,2*pi*fc0*5,10000);

%Compute the bode plot phase
bodeangle=zeros(1,length(fr));

for i=1:length(fr)
bodeangle(i)=angle(evalfr(Hfinal,fr(i)*j));
end

h=(2*pi*fc0*5-2*pi*0.001)/10000;
gd = -diff(bodeangle)/h;


figure
scatter(fr(1:9999),gd)
set(gca,'XScale','log')
title('Group Delay for Equiripple Error Filter')

load chirp
sound(y,Fs)

function f = EP_order3_fitness(a,b,c,d,fc0)
%Designing for 0.5 phase error
% error of the fitness function where fc0 is the desired cutoff frequency
H=tf([a],[1 b c d]);
fr=linspace(2*pi*0.001,2*pi*fc0*100,10000);

%Compute the bode plot phase
bodeangle=zeros(1,length(fr));

for i=1:length(fr)
bodeangle(i)=angle(evalfr(H,fr(i)*j));
end


%Need to calculate group delay, which is negative derivative of phase
%derivative
h=(2*pi*fc0*100-2*pi*0.001)/10000;
gd = -diff(bodeangle)/h;

%Find index closest to the cutoff frequency
```

```
[m,k] = min(abs(fr-fc0));
%Start the fitness function at zero
f=0;

% Aiming for certain dB ripple, don't let it get higher or lower
for i=1:k-1
    if 1.85<abs(gd(i))<2.05
     f=f+1;
    end
end

%Make sure it actually ripples above and below 2 s
%if not penalize heavily

minigd=abs(gd(1:k-1));

y = find(minigd>2 & minigd<0.05+2);
z = find(minigd>1.85 & minigd<2);

if isempty(y)<2
     f=f+100;
end

if length(z)
     f=f+100;
end

%Needs to monotonically decrease in the stopband
for i=k:length(gd)
    if gd(i-1)< gd(i)
         f=f+1;
    end
end

%Penalize if the drop off isn't good enough
%m2=(gd(end)-gd(k))/(fr(end)-fr(k));
%f=f+1/abs(m2)*100;

f
end

function [c, ceq] = EP_order3_constraint(x,fc0)
   %c contains all the constraints
   %Make sure magnitude at 1 rad/s is at 1
   %Make sure at fc0, phase is -135
   H=tf([x(1)], [1 x(2) x(3) x(4)]);
    c = [];
    ceq = [20*log10(abs(evalfr(H,0)))];
end
```