Synthesis of Component Values for Equiripple Group Delay Filters using MATLAB

A Project Report

submitted by

SHITIN SAHU

in partial fulfilment of the requirements
for the award of the dual degree of

BACHELOR OF TECHNOLOGY
AND
MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

June 2021

THESIS CERTIFICATE

This is to certify that the thesis titled **Synthesis of component values for Equiripple Group Delay filter using MATLAB**, submitted by **Shitin Sahu**, to the Indian Institute of technology, Madras, for the award of the degree of **Dual Degree in Electrical Engineering**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other institute or university for the award of any degree or diploma.

Prof. Nagendra Krishnapura
Research Guide
Professor
Dept. of Electrical Engineering
IIT Madras, 600 036

Place: Chennai

Date: 30th June 2021

ACKNOWLEDGEMENTS

I would like to thank my guide **Prof. Nagendra Krishnapura** of the Department of Electrical Engineering for his support throughout the project, whose expertise and excellent teaching attracted me to the field of Analog Design. I would like to thank all my professors at IIT Madras who deliver their best to the students. I would like to thank my parents for their support and all my friends who were by my side in all the ups and downs of life.

ABSTRACT

True-time delays are often implemented using all-pass architectures. These have been limited to first and second orders. The component values required to realize a desired group delay response from these systems can generally be estimated using conventional design methods. However, a new architecture described in [1], allows us to design all-pass filter based true-time-delay elements that can be scaled to any order. The presence of several component values that need to be determined to get a desired response poses a design problem.

Based on a MATLAB algorithm that can be used to numerically evaluate these component values to design equiripple group delays, this project aims to extend the capability of this algorithm to synthesize higher orders and larger delays than originally possible. We then look at synthesizing bandpass shaped group delays, instead of the original lowpass group delays, using this architecture to synthesize large equiripple group delays, albeit in reduced bandwidths. The added MATLAB routines help to avoid issues of numerical convergence while synthesizing large delays or for higher orders. Finally, we look at results of zero addition to the all-pass system in order to improve the magnitude droop in a lowpass system that is intrinsically present in the above all-pass architecture.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

APF	All-Pass Filter
DBW	Delay Bandwidth Product
EGD	Equiripple Group Delay
EM	Electromagnetic
SNR	Signal to Noise Ratio

CHAPTER 1

True-Time-Delay Elements - Uses and Architectures

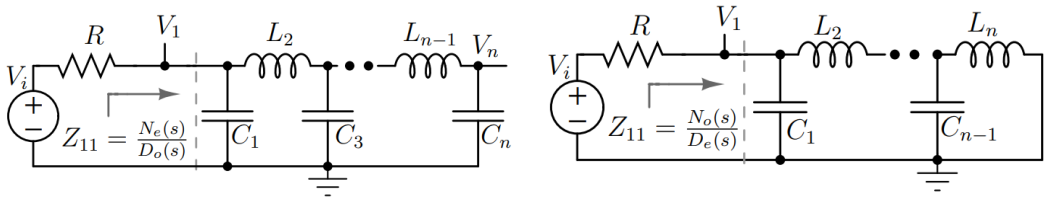
1.1 Broadband beamforming Systems

Delay elements are essential to most modern systems. They are fundamental to radars, beamforming systems and continuous time equalizers where multiple antennas can be used to offer spatial selectivity. For beamforming systems, delay elements allow us to focus and steer the beam in the direction of choice. Higher number of antennas offer higher spatial selectivity, but also increase the delay requirements of the systems. For continuous time equalizers, a longer impulse response requires a longer delay span from the equalizer. The signals processed by these systems can be narrowband or wideband. While for narrowband signals, delays are often approximated as phase shifts at the center frequency, wideband band signals require time-delay elements. Phase shifts for narrowband delays are very common, as they can be easily implemented in CMOS. The ideal true-time-delay element is a transmission line, with a transfer function e^{-sT_d}. This has linear phase and unity magnitude response across all frequencies. For applications targeted at lower GHz frequencies, the length of transmission lines are impractically large for a CMOS implementation. Hence, realizing filters with long delays over wide bandwidths, i.e. large delay-bandwidth products, is necessary. All-pass filters with linear phase over the signal bandwidth are a popular choice for implementing true-time-delays. In the next section we describe some of the architectures.

1.2 All-pass filter architectures

Active implementations of all-pass filters are often used to implement true-time-delays. However, such architectures have been restricted to the first or second orders. Higher order APFs are realized by cascading individual delay cells. However, each connecting node adds a parasitic pole to the system, ultimately limiting the achievable bandwidth

of the system. This limits the number of cells which can be cascaded and thus the maximum achievable delay [3].



(a) Odd order (b) Even order

Figure 1.1: Singly Terminated Ladder Filters 'capacitor first'.

However, if a filter topology has just as many nodes as its order, all parasitic capacitors can be absorbed into the integrating capacitors of the filter. Such an architecture is introduced in [1]. The filter is an all-pass filter architecture based on a singly terminated ladder filter.

Fig. 1.1 shows 'capacitor-first' singly terminated ladder filters of odd and even orders. The working can be understood in terms of transmission lines. We know that a pulse launched into a transmission line terminated by an open or a short circuit returns to the input port with a round trip delay. It can be shown that the final node of the ladder, V_n, is a lowpass function of the form $\frac{1}{D(s)}$. Also, from the properties of the ladder filter response, $2V_1 - V_i$ is an all-pass function of the form $\frac{D(-s)}{D(s)}$ [3]. This is shown in Fig. 1.2.

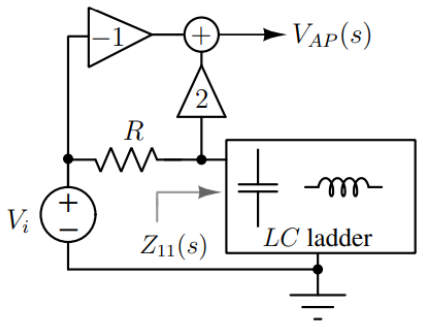


Figure 1.2: All-pass filter using singly terminated ladder.

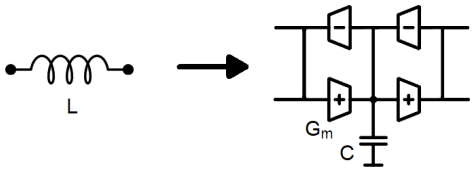


Figure 1.3: Active implementation of an inductor.

Since active filters are more compact and easily tunable, the architecture implements the ladder using active versions of the inductors as shown in Fig. 1.3. Each inductor is replaced by a capacitor C and a pair of transconductances G_m. The resulting active

inductance is of value $L = C/G_m^2$. For $G_m = 1\ S$, which we will be using in our synthesis, $L = C$. This substitution for G_m represents a termination resistance of 1Ω.

This results in a Gm–C filter architecture as shown in Fig. 1.4. The order of the filter equals the number of integrating capacitors. The architecture has only one parasitic pole, present at the output node. The order of the filter can be changed by turning off the transconductances from right to left.

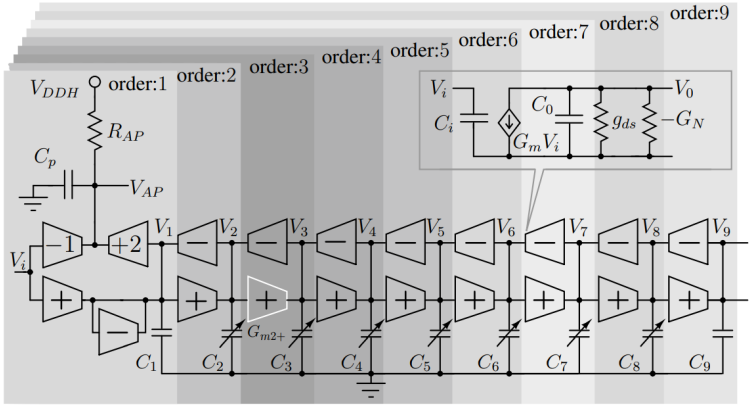
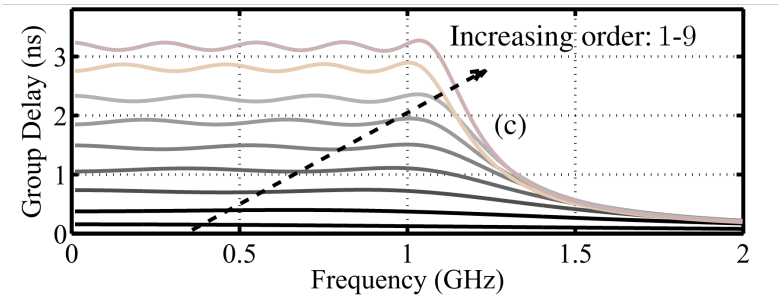
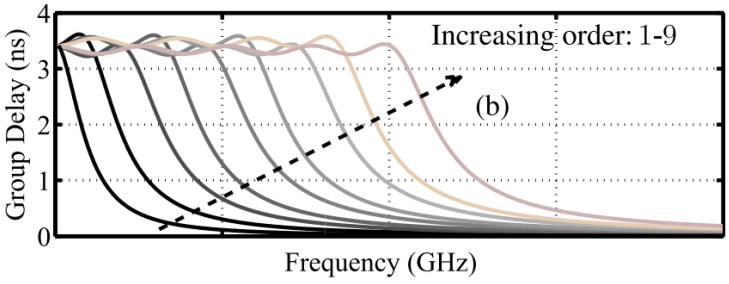


Figure 1.4: Gm–C representation of the APF architecture.

With suitable component values, the filter can be made to have an equiripple group delay as shown in Fig. 1.5. This is what we will call a 'lowpass' type EGD response.



(a)



(b)

Figure 1.5: Equiripple Group Delay responses of the APF with (a) constant group delay and (b) constant bandwidth.

The problem of interest is to find the component values needed so that the filter gives us a desired group delay response. Component values for synthesizing equiripple group delays have been previously tabulated by Zverev [4]. These are, however, only for a few specific ripple values and no general solutions are present. Component values for Blinchikoff filters [5] are known, but only for specific cases of flat bandpass delays. Finding an algorithm to find these component values for filters of any general order and ripple is of interest. An iterative MATLAB routine using the Newton-Raphson method is described. The algorithm is based upon a simplified state-space model of the architecture given in Fig. 1.4. EGD filters of a given order and ripple can be synthesized. We look at the details of the algorithm in Chapter 2 of the thesis.

1.3 Organization of the thesis

The project aims at using the algortihm to synthesize 'bandpass' versions of this group delay i.e. having linear phase in a certain range of frequencies. This allows us to utilize the high delay-bandwidth product of the architecture discussed to give large delays, albeit in narrower bandwidths. We then improve this algorithm to accommodate for a larger range of orders and peak delays both in the lowpass and the bandpass case.

The rest of the thesis is organized as follows:

Chapter 2 discusses in detail the algorithm used to synthesize lowpass shaped group delays of desired peak group delay and ripple. Synthesized filters and data on delay-bandwidth products is shown.

In **Chapter 3** we do a bandpass transformation of the Gm–C architecture to create a bandpass ladder structure. Properties of its group delay response are observed to synthesize a bandpass equiripple group delay filter.

In light of the convergence issues in the bandpass transformed version of the APF, **Chapter 4** presents an alternative method to synthesize bandpass shaped group delays using the lowpass ladder structure. Then we discuss the limitations of the original algorithm and provide new MATLAB routines that extend the range of orders and ripples that can be synthesized.

The intrinsic lowpass nature of the filter contributes to the errors in the time domain

inputs and outputs. This can be improved by trying to flattening the magnitude response of the lowpass filter at the node V_n as shown in Figure 1.1. **Chapter 5** shows preliminary results on how this can be done.

Chapter 6 concludes the thesis.

CHAPTER 2

Synthesizing the Lowpass Equiripple Group Delay

2.1 Target group delay response

The group delay response desired from the circuit is flat but with a constant ripple in a certain bandwidth. This translates to linear phase with some ripples. The magnitude response is exactly flat. The desired characteristics for a 4th order filter are shown in Fig. 2.1.

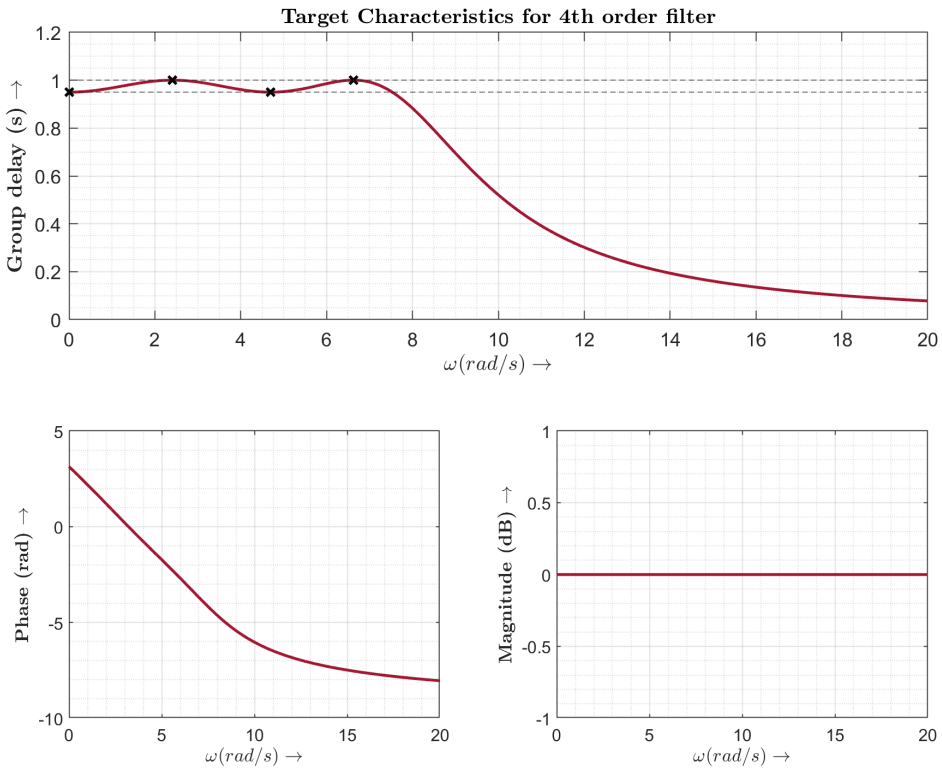


Figure 2.1: Target Characteristics for a 4^{th} order filter.

The group delay response shown above has ripples of constant magnitude. In this case, the group delay oscillates between 1 and 0.95 (a ripple of 0.05 seconds) before rolling off to zero at high frequencies. The number of extremums (4) is equal to the order of the filter synthesized. Since the capacitors can be scaled to have any group

delay while keeping the delay-bandwidth product constant, we normalize by keeping the peak group delay of the system to be equal to unity.

We observe that the number of extremums in the equiripple group delay should be equal to N, the order of the filter. For an odd order filter, the extremums of the group delay should be 1 at dc, then alternate between $(1 - r)$ and 1, where r is the amount of ripple. For an even order filter, the extremums have a value of $(1 - r)$ at dc and alternates between $(1 - r)$ and 1 till a final value of 1. There is always an extremum at dc and the total number of peaks is equal to N. The behavior of the desired target is illustrated in Fig. 2.2.

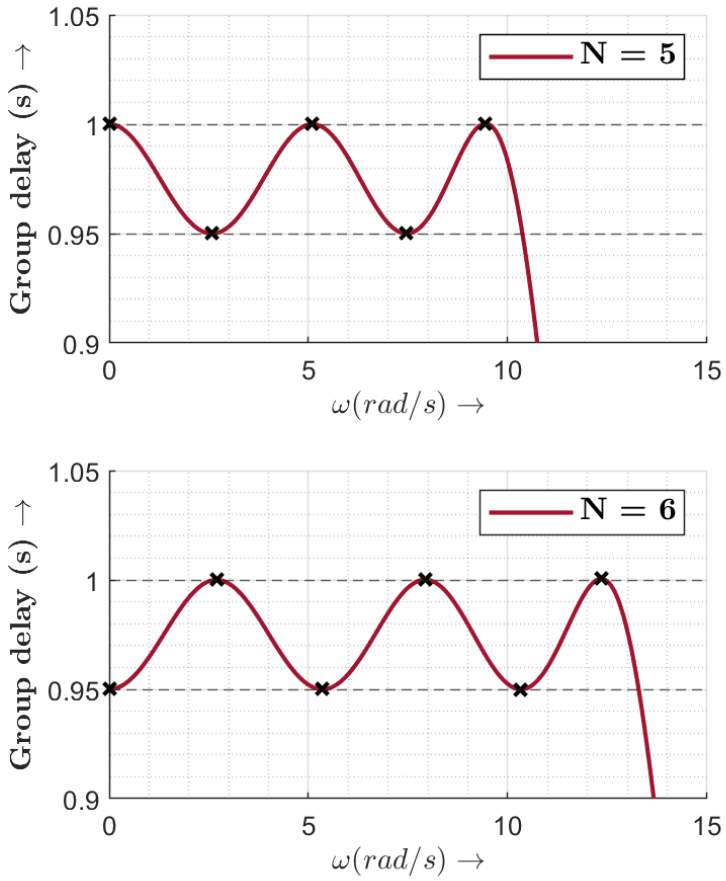


Figure 2.2: Typical group delay desired, shown for order 5 and 6 with a ripple of 0.05 seconds.

This defines the objective functions of our optimization as the values of the extremums of the group delay. An important requirement is also set: all the N extremums should be present in the initialization of the routine and at all steps during the optimization.

2.2 State-space modelling

In order to have the desired group delay response, we have to know the required values of the capacitors. The algorithm we present is a MATLAB routine based on a simplified state space model of the circuit in Fig. 1.4. A part of the Gm–C architecture is reproduced for reference in Fig. 2.3.

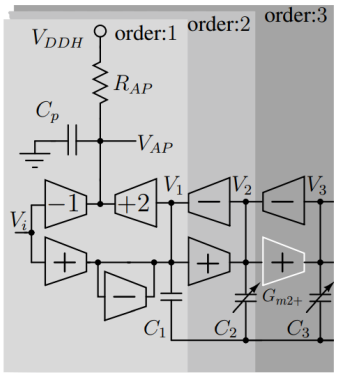


Figure 2.3: A part of the circuit from Fig. 1.4.

A general state space model can be represented as:

$$\begin{aligned} \dot{\mathbf{X}} &= \mathbf{AX} + \mathbf{BU} \\ \mathbf{Y} &= \mathbf{CX} + \mathbf{DU} \end{aligned} \tag{2.1}$$

We can write differential equations relating the node voltages and create a state space model of the system. We can see that:

$$\begin{aligned} C\dot{V_1} &= -G_m V_1 - G_m V_2 + G_m V_i \\ C\dot{V_2} &= -G_m V_1 - G_m V_3 \end{aligned} \tag{2.2}$$

and similar equations govern later nodes. The output V_o can be written as:

$$V_o = 2V_1 - V_i \tag{2.3}$$

The transfer function from V_o to V_i is an all-pass transfer function of the form:

$$\frac{V_o(s)}{V_i(s)} = \frac{D(-s)}{D(s)} \tag{2.4}$$

where $D(s)$ is an N^{th} order polynomial for a filter with N capacitors. The all-pass

system $D(s)/D(-s)$ has twice as much the group delay as the lowpass counterpart $1/D(s)$.

The dependence of the derivatives of the node voltages on other node voltages and the input can be described to the code in the function *GmMatrix.m*. Using this information, the state space matrices for the circuit can be generated. This is done by the function *ladderFilFormGmat*. From equations 2.1 and 2.2 we can see that the node voltages are the states. However, note that we also need the capacitor values to write the state-space matrices. We discuss what the capacitor values are initialized to in the next section. Details on all the MATLAB functions can be found in Appendix A.

For the purpose of our optimization, only the ratio G_m/C matters and not the individual values. Hence we choose to set G_m to be 1 and run the optimization only using the capacitor values.

2.3 Initialization

The group delay target must be 1 at dc. Since, we can arbitrarily scale the capacitor values while keeping the delay bandwidth product constant, this functions as our normalization condition. It is then natural to expect that the initialization also has this property. Another requirement is that it has N extremums, as described towards the end of section 2.1. Fortunately, a simple initialization using equal capacitor values at all nodes satisfies the above requirements. For a given order N and ripple r, we choose the following initialization:

N is Odd	N is Even
$\frac{1}{(N+1)}$	$\frac{1}{N}(1-r)$

Table 2.1: Capacitor initialization for the algorithm.

This scaling ensures that the dc group delay is 1 for odd orders and $(1-r)$ for even orders, which matches our target. The characteristics of the APF at the initialization are shown in Fig. 2.4. With the initial capacitor values, the state space model of the system and its group delay response can be evaluated. This is done by the function *ladderFiltCalc*.

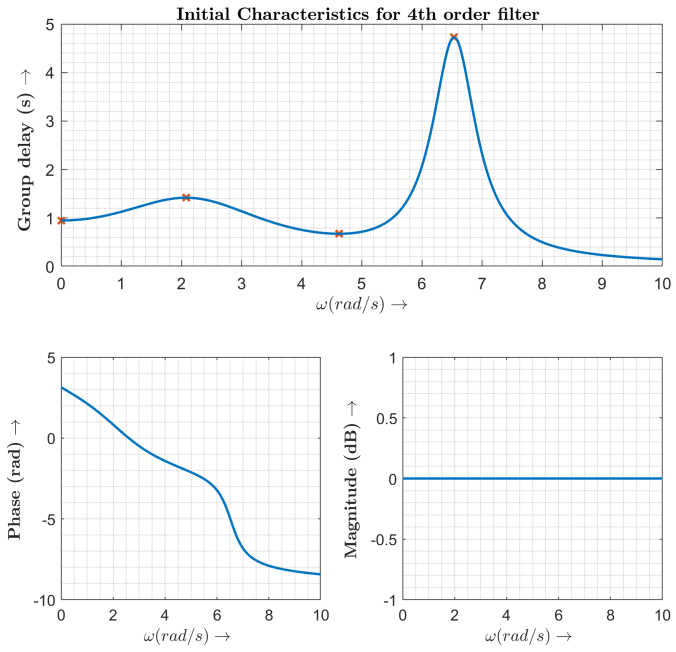
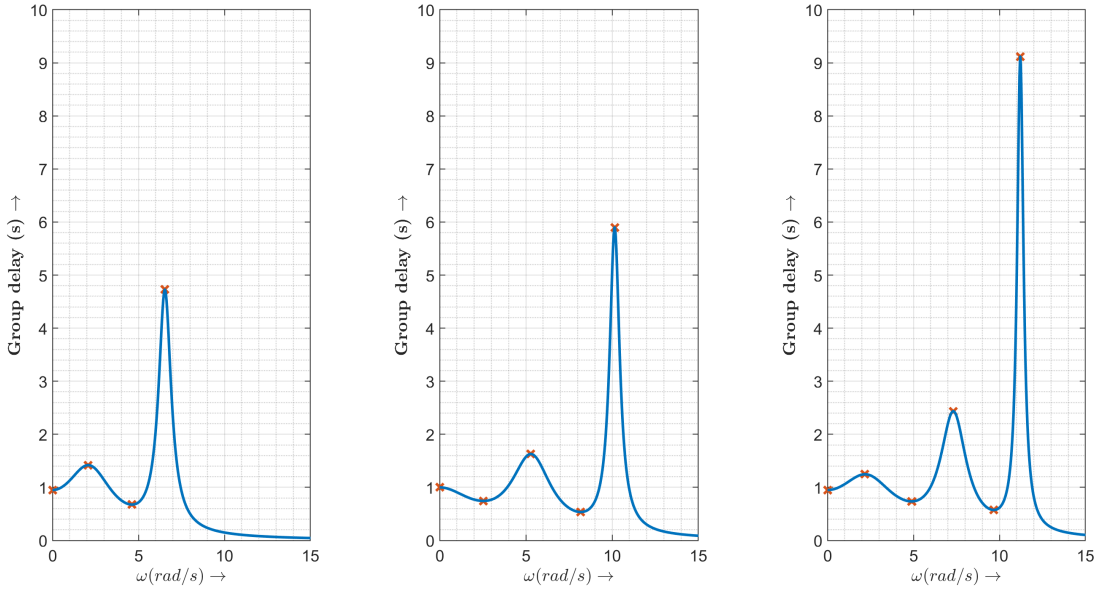


Figure 2.4: Characteristics of an order 4 filter on initialization.

We can see that the number of extremums in the initial group delay is equal to the order, although they are of varying sizes. Group delays at the initialization for orders 4, 5 and 6 are shown below.



(a) $N = 4$ (b) $N = 5$ (c) $N = 6$

Figure 2.5: Initial group delay responses for orders 4, 5, and 6.

Note that the height of the highest peak increases with order. This points to increased difficulties for convergence as order increases.

2.4 Process

Once the target is set and the model is initialized, we can start iterating towards the target. The problem is set-up as a multi-variable optimization using the Newton-Raphson method. The variables are the capacitor values, given as an array. The objective function is the array of desired extremums. For this, the Jacobian is evaluated numerically by perturbing the capacitor values on both sides. This gives us the Jacobian using central difference as:

$$\left[Capval(k) = Capval(k) \pm \Delta Capval_k\right]$$

$$\mathbf{J}_{N\times N} = \frac{\Delta GD}{2\Delta Capval_{k=1\ldots N}} \tag{2.5}$$

where ΔGD denotes the difference produced in the N extremums.

The the update equation for the capacitor array can be written as:

$$Capval = Capval - \gamma\, \mathbf{J}^{-1}\left(Current\, peaks - Target\right) \tag{2.6}$$

where, γ is the step-size modifier and 'peaks' refers to all the extremums alike.

These steps viz.:

1. Evaluate group delay with current capacitor values
2. Numerically evaluate Jacobian
3. Update capacitor array

are repeated for a fixed number of steps OR till a desired tolerance w.r.t the target is reached. The convergence from the initial to the final group delay for order $N = 6$ is shown in Fig. 2.6

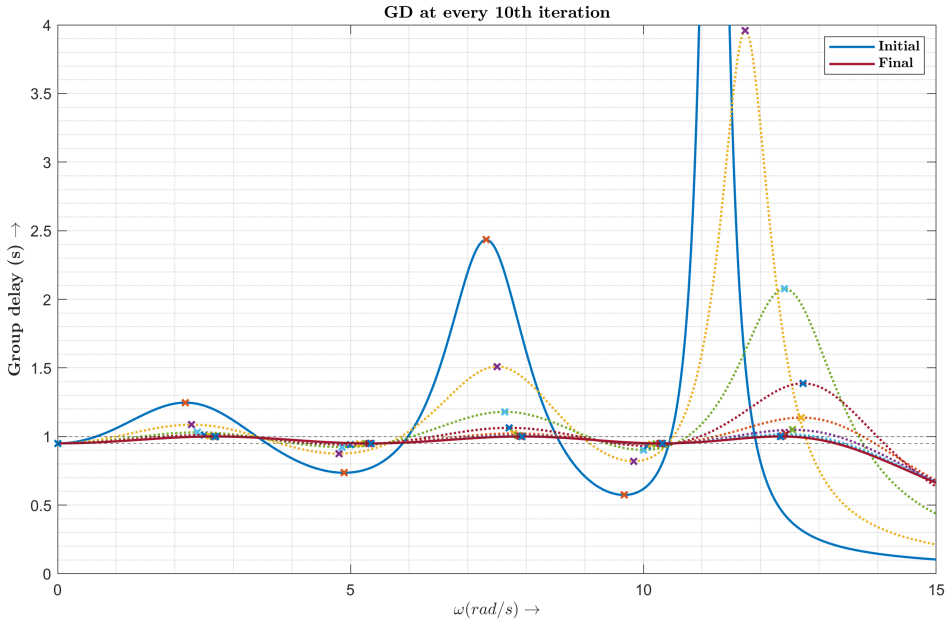


Figure 2.6: Every 10th iteration in the optimization for a 6th order filter with target ripple = 0.05 seconds.

2.5 Results

We can synthesize filters of several orders using the above algorithm. These can be synthesized as a constant group delay or a constant bandwidth systems. Synthesized group delays for orders 4 to 12 are shown in Fig. 2.7.

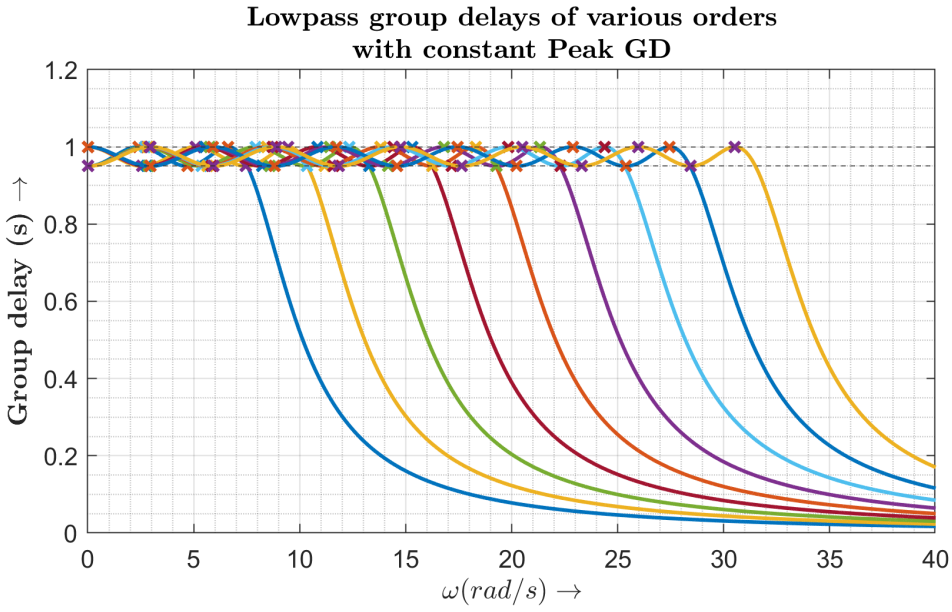


Figure 2.7: Equiripple group delays from orders 4 to 12 with a ripple of 0.05 .

The filters can also be synthesized as constant bandwidth. Fig. 2.8 shows filters normalized to a bandwidth of 10. Here, the bandwidth is defined as the frequency at which the group delay crosses $(1 - r)$ for the last time. We later also refer to this as the 'upper cut-off' frequency.

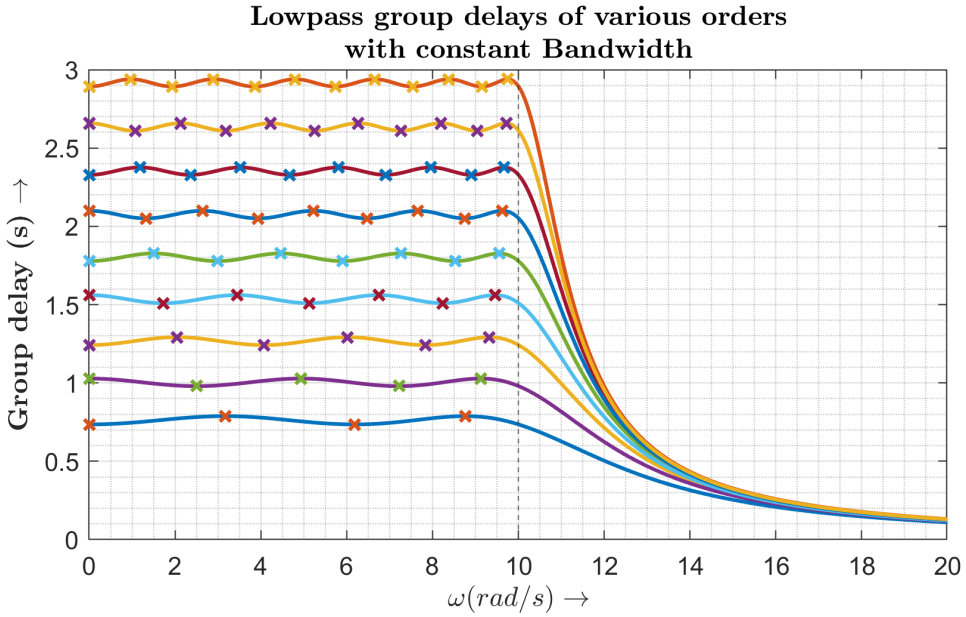


Figure 2.8: Equiripple group delay filters with ripple of 0.5 for a constant bandwidth.

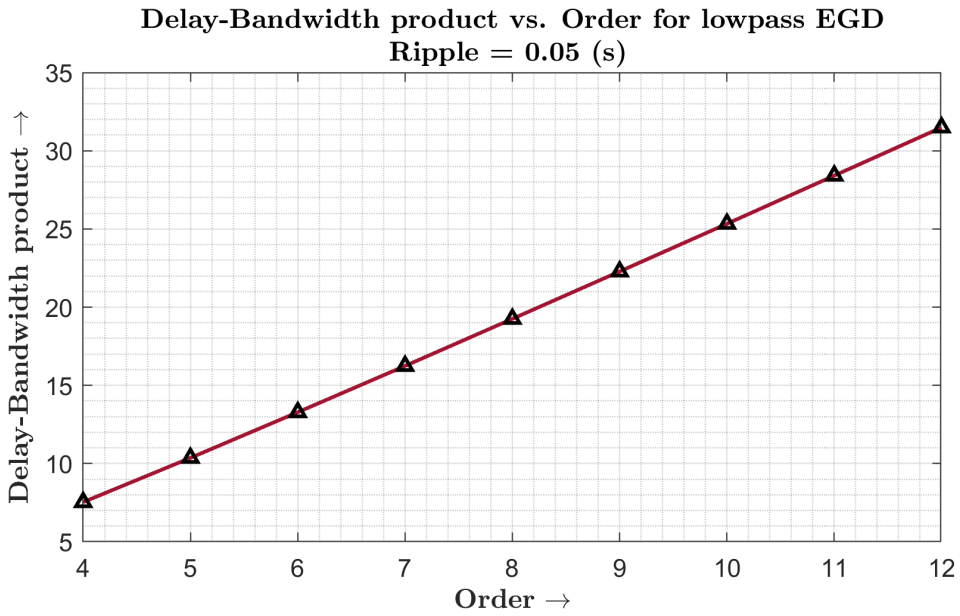


Figure 2.9: Delay bandwidth products for lowpass equiripple group delay filters of order 4 to 12.

CHAPTER 3

Bandpass Transformation of the All-Pass Filter

3.1 Implementation

Bandpass systems often offer higher group delays, but in narrower bandwidths than lowpass systems. With this architecture having large delay-bandwidth products, In this chapter we transform the Gm–C architecture to its bandpass version, and try to create a 'bandpass' equiripple group delay. This is done by transforming the LC ladder form Fig. 1.1 to its bandpass version and then converting the resulting circuit to its active Gm–C equivalent. The steps for transforming the architecture from lowpass to bandpass are shown from Fig. 3.1 to Fig. 3.4.

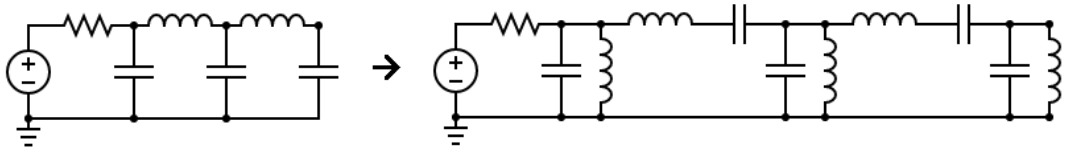
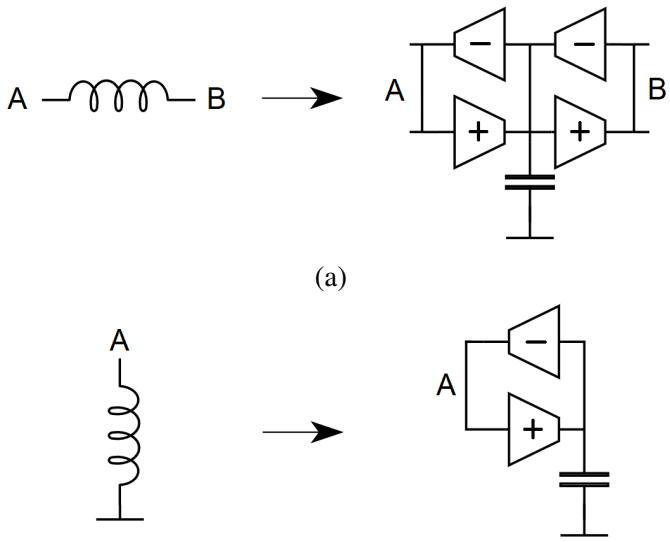


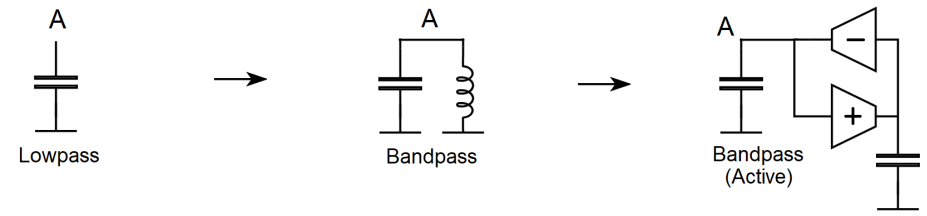
Figure 3.1: Lowpass to bandpass transformation of the LC ladder.



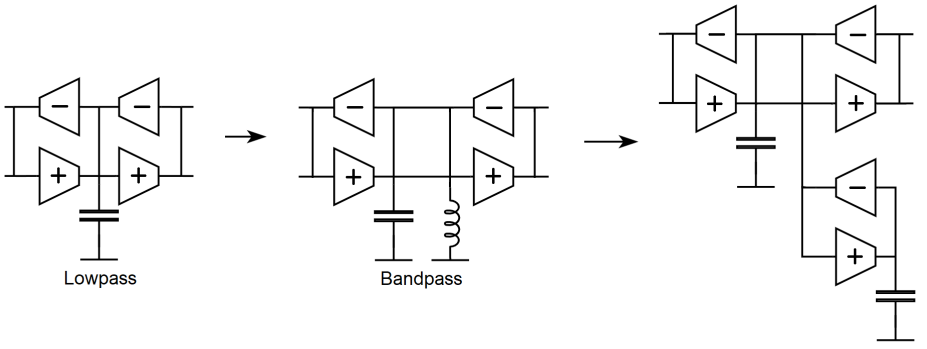
(a)

(b)

Figure 3.2: Active implementation of an inductor (a) between two nodes and (b) between a node and ground



(a)



(b)

Figure 3.3: Transformation to active bandpass versions for (a) Capacitor (b) Inductor

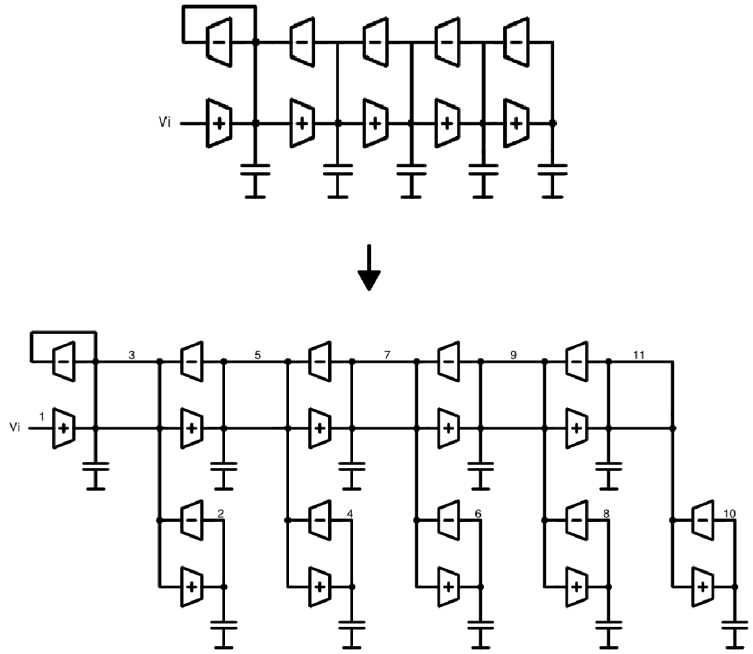


Figure 3.4: Bandpass transformation of the Gm–C architecture.

This circuit can be described in a similar fashion as done in Section 2.2. With some modifications to the rest of the MATLAB routine, it can be adapted to work for a bandpass shaped group delay. We can again start with equal capacitor values, and the group delay response with this initialization is shown in Fig. 3.5.

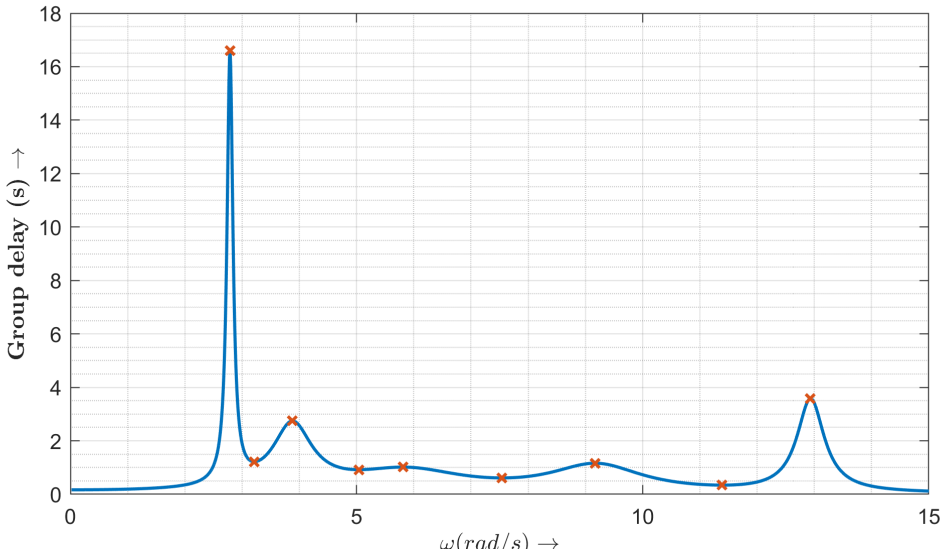
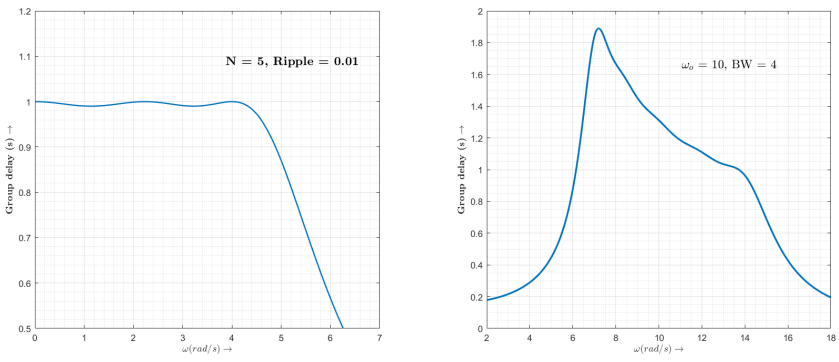


Figure 3.5: Group delay of the bandpass system transformed from lowpass filter of order 5 with equal capacitor values.

It is observed that this initialization is not very efficient in generating stable iterations to reach the equiripple group delay. Another method is to transform the lowpass equiripple group delay system and transform it to create the initialization. However, this does not directly create an equiripple bandpass system as the group delay gets multiplied by a frequency dependent term. It can be shown that the group delays are related as:

$$gd(H_{BP}(j\omega)) = gd(H_{LP}(\frac{j}{\omega_b}\frac{\omega^2-\omega_0^2}{\omega})) \cdot \frac{BW_{LP}}{BW_{BP}} \times \left(1+\frac{\omega_0^2}{\omega^2}\right)$$

Transformation of a 5th order delay with a ripple of 0.01 seconds is shown in Fig. 3.6.



(a) (b)

Figure 3.6: Lowpass (a) to bandpass (b) transformation of 5th order lowpass filter.

Note that the extremums required for our algorithm to work are absent in the bandpass case. They appear if we increase the ripple in the lowpass filter. With some modifications to the code, the routine can be made to synthesize a bandpass equiripple group delay as shown in Fig. 3.7.

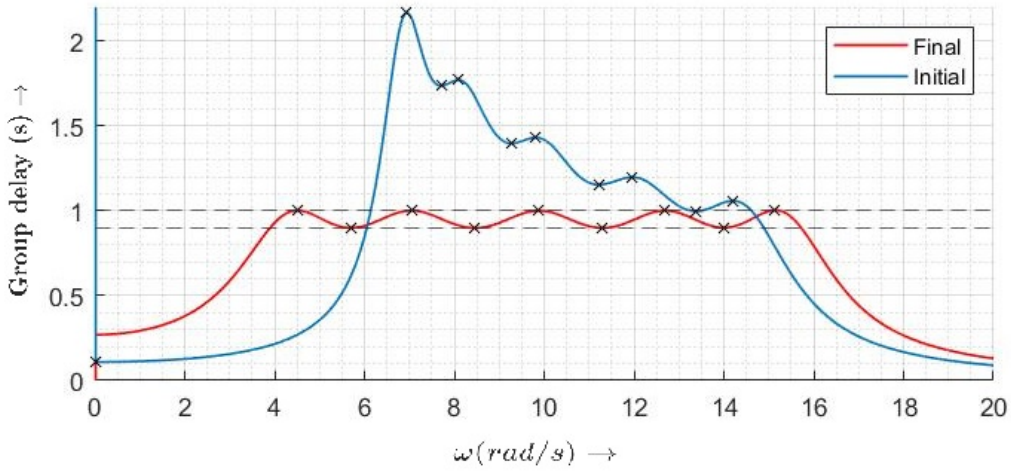


Figure 3.7: Initialization and final output for transformed version of 5th order lowpass.

3.2 Issues

It is observed that this method of generating bandpass shaped group delays is not very stable. Beyond transformations of a 5th order filter, the Newton-Raphson method quickly starts becoming unpredictable and often unbounded. tiny changes in capacitor values often cause substantial changes to the group delay, resulting in unpredictable outcomes. In such cases, one or more of the following situations occur:

1. The Newton-Raphson iterations requires impractically small step-sizes to be able to converge to the final EGD response.
2. One or more peaks are absent from the group delay response, which are required by the algorithm.
3. One or more peaks start increasing in value with each iteration to become unbounded.

In light of these issues, an alternative method to synthesize bandpass shaped group delays has been provided in the next chapter.

CHAPTER 4

Alternate Method to Synthesize Bandpass Shaped Equiripple Group Delays

4.1 Using the lowpass ladder structure

Given the convergence issues with synthesizing using the bandpass transformed ladder structure, we look for another method to synthesize bandpass shaped group delays. Using the lowpass system $1/D(s)$, we can generate a bandpass system $s^n/D(s)$ with the same group delay. In this chapter, we generate bandpass shaped group delays using the lowpass ladder itself and then present another MATLAB routine that extends the range of equiripple group delays that can be synthesized.

4.2 Implementation

The idea is to use the lowpass ladder structure with a modified group delay target. The group delay target for a lowpass equiripple group delay has all the values to be either 1 or $(1 - r)$. However, we can choose to have the target at dc to be lower than the rest of the target. For instance, we choose the target to be : $\left[\ \mathbf{0.95}, 1.5, 1.45, 1.5, 1.45, 1.5\ \right]$ for an order 6 filter. The output then converges accordingly, as shown in Fig. 4.1. The figure also shows the 6th order lowpass equiripple group delay with ripple of 0.05.

The iterations proceed towards convergence with relative ease. So it is possible to synthesize bandpass shaped group delays using the lowpass ladder structure.

However, it is difficult to gauge the amount of advantage this gives us in terms of peak group delay. In order to do so, we define the 'upper cut-off' frequency of the system as the highest frequency where the group delay profile crosses $(1 - r)$, r being the desired ripple in the flat top. The upper cut-off frequencies of the two group delays have been marked in Fig. 4.1. Now, we normalize the new group delay to the same

upper cut-off as the lowpass group delay. This is done by scaling the capacitor values for this group delay by the inverse of the ratio of the two cut-off frequencies, giving us the scaled group delay response as shown in Fig. 4.2(a).

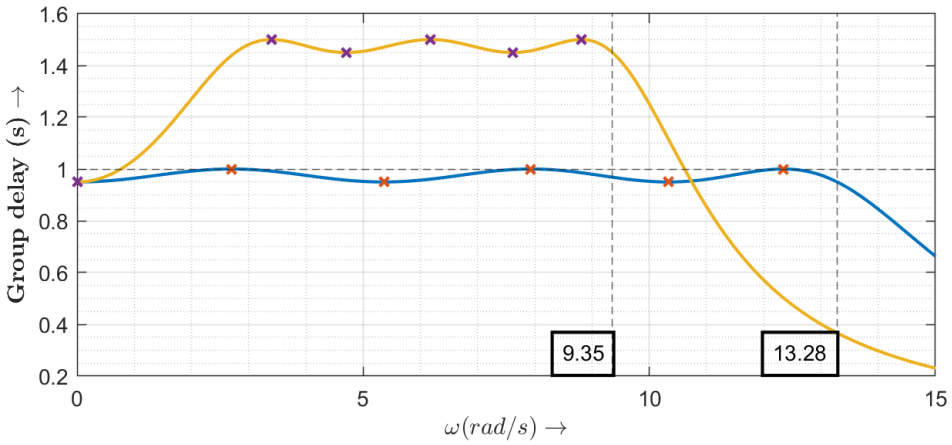
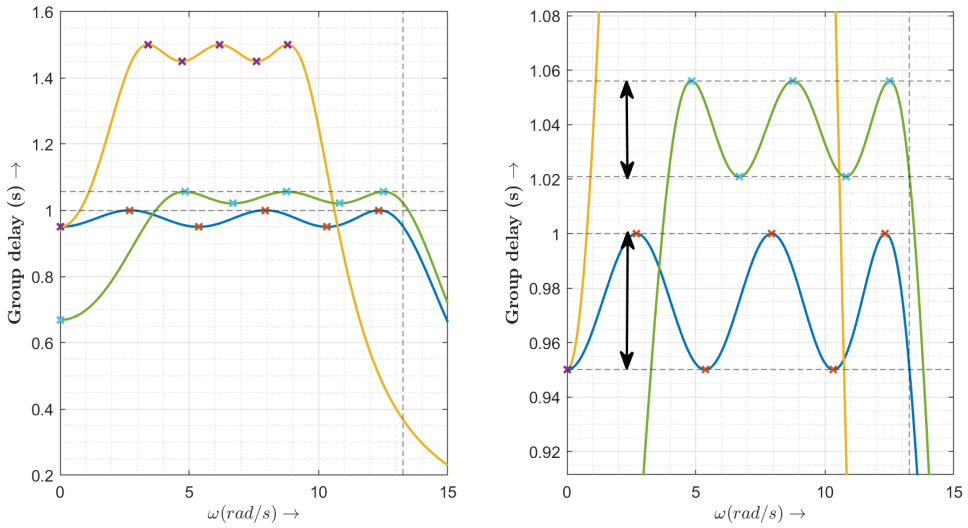


Figure 4.1: Using the lowpass ladder structure with a partly equiripple group delay target.



(a) (b)

Figure 4.2: (a) Normalizing the bandpass shaped group delay using the upper cut-off frequency (b) Ripple reduction as a result of normalization.

We also observe that the ripple of 0.05 in the un-normalized group delay reduces when the group delay is scaled to a higher cut-off frequency, as shown in Fig. 4.2(b). It can be verified that the ripple also reduces by the same factor as the group delay, which in turn is the ratio of the cut-off frequencies. This poses a problem for the problem we would like to solve next.

It is desired that we can get the required peak group delay and ripple in the normal-

ized filter i.e. with the same upper cut-off frequency as the lowpass filter with the same ripple. This is shown in Fig. 4.3

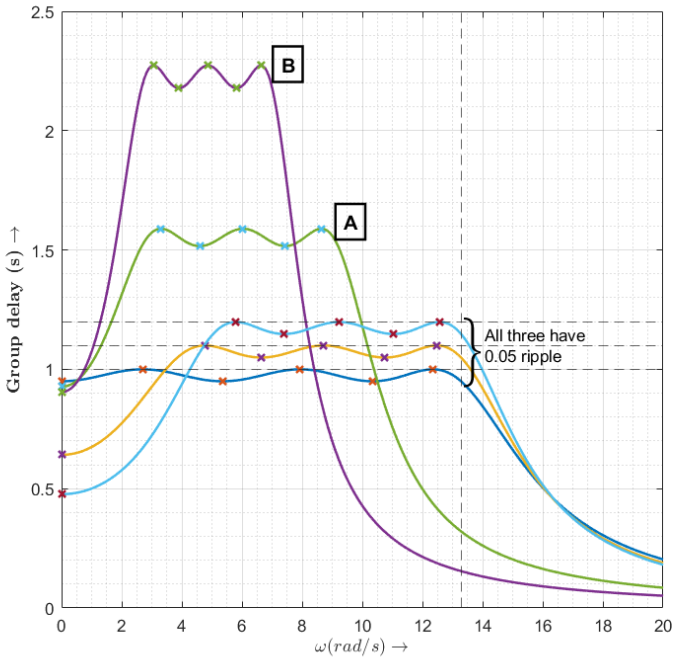


Figure 4.3: We should be able to specify the ripple in the normalized filter.

Note that we can only synthesize the un-normalized filters shown in Fig. 4.3 as A and B, since we cannot specify the upper cut-off frequency to be synthesized. N capacitor values are translated to the N target values that are given for the group delay. We do not know the functional dependence of the upper cut-off frequency of this un-normalized filter and so it is unknown a priori. This translates to unknown scaling factor for the normalization and thus unknown values for the peak group delay and the ripple to be synthesized. This means that we will need an iterative algorithm that can search for the right parameters.

4.3 Basic algorithm

We need to synthesize a group delay normalized to a given upper cut-off frequency with a given order and ripple.The basic algorithm uses the fact that as we increase the peak group delay synthesized in the un-normalized case, the normalized group delay increases accordingly. The core algorithm for the synthesis of normalized group delays as described above is:

1. Synthesize the lowpass equiripple group delay with the target ripple
2. Increase the peak group delay by a fixed amount
3. Normalize to check the normalized filter parameters
4. Repeat 2 and 3 till the normalized group delay exceeds the target
5. The right parameters then lie between the last and the current peak group delays
6. Perform a 'binary search' within these two peak group delays
7. Stop when the desired tolerance from the target is reached

The following subsections describe the algorithm stepwise. Synthesis of an equiripple group delay with peak group delay of 1.25 seconds and ripple of 0.05 seconds is taken as an example i.e. the final target to converge on is: $\big[$0.95, 1.25, 1.2, 1.25, 1.2, 1.25 $\big]$

4.3.1 Searching bounds for peak group delay

The algorithm starts by synthesizing the lowpass group delay filter with ripple equal to the target ripple. The upper cutoff frequency is calculated and stored. Next, the group delay target is modified for a higher peak group delay Say the peak group delay is increased by a fixed amount p. The group delay target is now $[(1-r)$,$(1+p)$, $(1+p-r), (1+p)...]$. Once the group delay converges to this target, it is normalized to the upper cut-off frequency of the lowpass filter saved in the first step. This process of incrementing the peak group delay by this amount p continues till the peak group delay of the normalized filter becomes greater than the desired peak group delay. This is shown in Fig. 4.4. The orange group delay normalizes to a peak group delay greater than 1.25, while the green group delay normalized to a peak group delay less than 1.25.

Note the following: It can be seen from Fig. 4.3 that the required ripple in the un-normalized filter to synthesize a given ripple in the normalized filter increases as we increase the desired peak group delay. In order to not stray away from the target, we increase the ripple of the un-normalized filter as we increase its peak group delay. While increasing the peak group delay, the ripple is set to be a constant fraction m of the peak group delay of the un-normalized case. This increases the ripple as we increase the peak group delay as we look for the bounds for peak group delay. So r in the process described above is equal to $(1+p) \times m$.

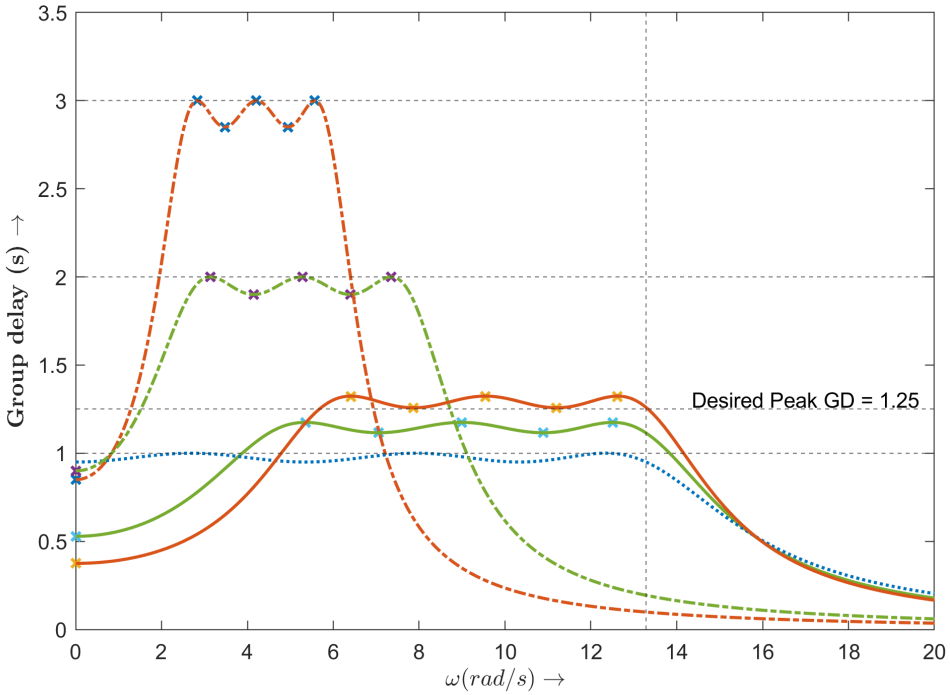


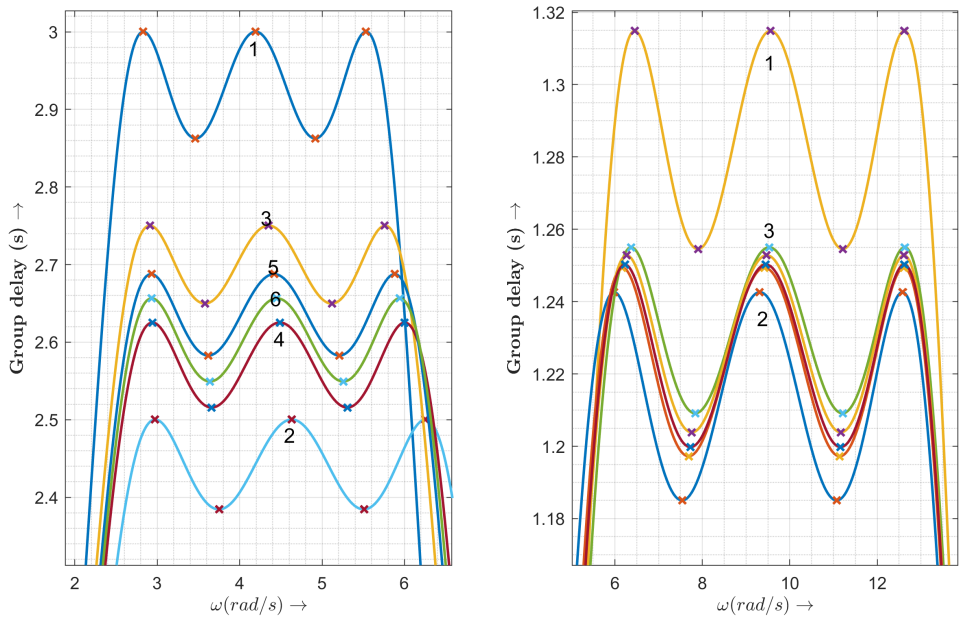
Figure 4.4: Increase peak group delay of un-normalized filter till normalized peak crosses the desired value.

As discussed before, each increment in the peak group delay also increases the ripple. This is done because we do not know the scaling factor for the normalization. Once we have set the bounds for the peak group delay, we are close to the target peak group delay, which means that we are close to the correct value of the scaling factor for normalization. We use this to correct the value of the ripple. If we assume the current state to have the correct scaling factor, we can set the value of the ripple in the un-normalized filter to be the target normalized ripple times the ratio of the cut-off frequencies of the un-normalized and the normalized delays.

4.3.2 Searching between the bounds

Once the bounds for the peak group delay are set, we can look for the target group delay within these limits. The next peak group delay given to the algorithm is the average of these bounds. The ripple value given is the desired ripple after normalization scaled by the ratio of the cut-off frequencies of the un-normalized to the normalized filter. This ripple is updated with each iteration. As we get closer to the target, this approximation of the required ripple approaches the correct value. Updating of the bounds progresses

in a binary search fashion. If the normalized peak group delay is greater than the target, the next search occurs between the average and the lower bound. On the other hand, if the normalized peak group delay is smaller than the target, the next search occurs between the average and the upper bound. This search continues till the target is reached within pre-specified tolerance values. For our current example, the tolerances set are 2×10^{-3} for the peak group delay and 5×10^{-4} for the ripple. The algorithm reaches the target peak of 1.25 and ripple of 0.05 in 5 iterations of 'binary search'. The actual values reached are: 1.250254 for peak delay and 0.050482 for ripple. The convergence is shown in Fig. 4.5.



(a) Un-normalized group delays (b) Normalized group delays

Figure 4.5: Convergence of binary search to target within given tolerances.

The capacitor values for this filter are:

$$\left[\, 0.0655,\ 0.0885,\ 0.1800,\ 0.0786,\ 0.4209,\ 0.0429 \,\right]$$

The total capacitance is 0.8765 and the ratio of the largest to the smallest capacitor is 9.8042 .

4.4 Other features to improve chances of convergence

4.4.1 Flexible parameters in the code

The parameters described previously namely, the increment in group delay (p), multiplier for increasing ripple size (m) can be varied according to the results required. This changes the speed and reliability of the code. For example, increasing the increment p makes the code more liberal, since it has to make larger jumps of group delay. While decreasing the ripple size before entering binary search, another parameter is used to decrease it in multiple steps instead of a single step. This is useful when synthesizing large delays. With large delays, the convergence becomes less stable, and reducing ripple size must be down slowly in small steps to avoid instability. The peak group delay and ripple tolerances can also be modified.

4.4.2 Changing parameters on failure of convergence

If the algorithm fails to converge at any intermediate step, it tries again starting from the last step with a smaller stepsize for the Newton-Raphson iterations and a smaller value for the jump p. In particular, both these values are halved in this case. This continues till a threshold is reached. If the algorithm still fails to converge, it stops and returns the last converged state to the output.

4.4.3 Effect of ripple on cut-off

For a given peak group delay, the ripple also affects the upper cut-off frequency. In general, if the ripple is reduced for a constant peak group delay, the upper cut-off frequency reduces. This results in a slightly higher scaling factor for normalization, which translates to a lower normalized peak group delay. One of the critical decision points for the algorithm is when the normalized group delay exceeds the target value. This is when the code stops incrementing the peak group delay further. If the peak group delay drops below the target due to this reduction, the bounds are no longer correct, and must be updated. This is shown in Fig. 4.6. The peak drops below 1.3 when this is the target. Without this correction, the algorithm will get stuck within the wrong bounds, never

reaching convergence.

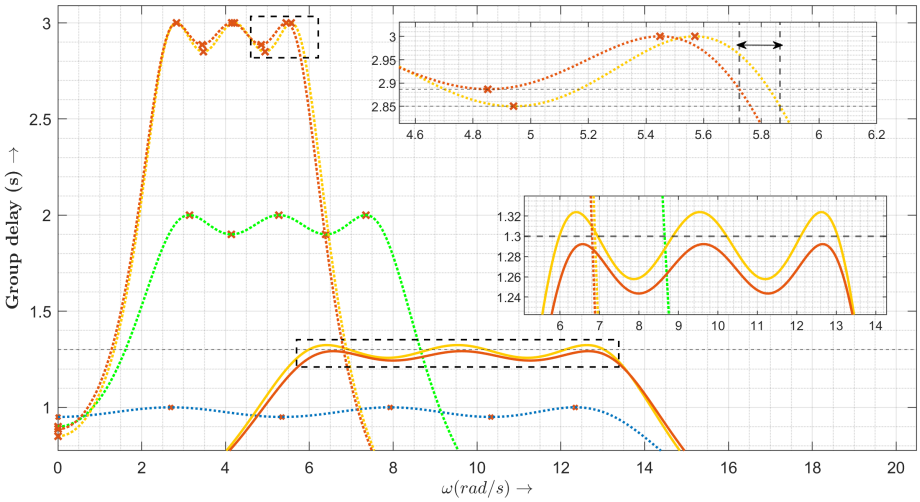


Figure 4.6: Effect of ripple on cut-off.

4.5 Results

Filters of many orders and ripple sizes can be synthesized using the above algorithm. Some of the results and the respective capacitor values are shown in the following figures.

Results for order 6:

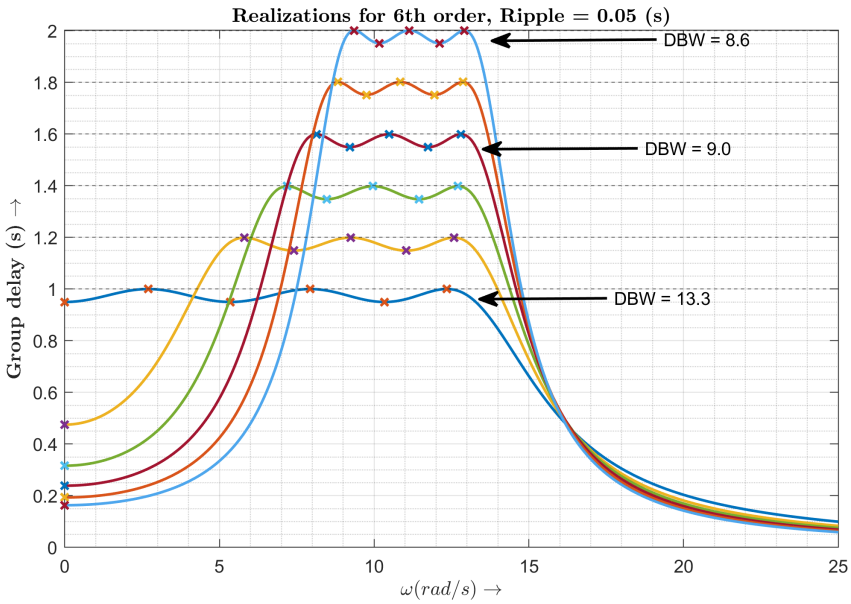


Figure 4.7: Filters with increasing peak group delays for order N = 6.

We can also look at the capacitor values for each of these cases:

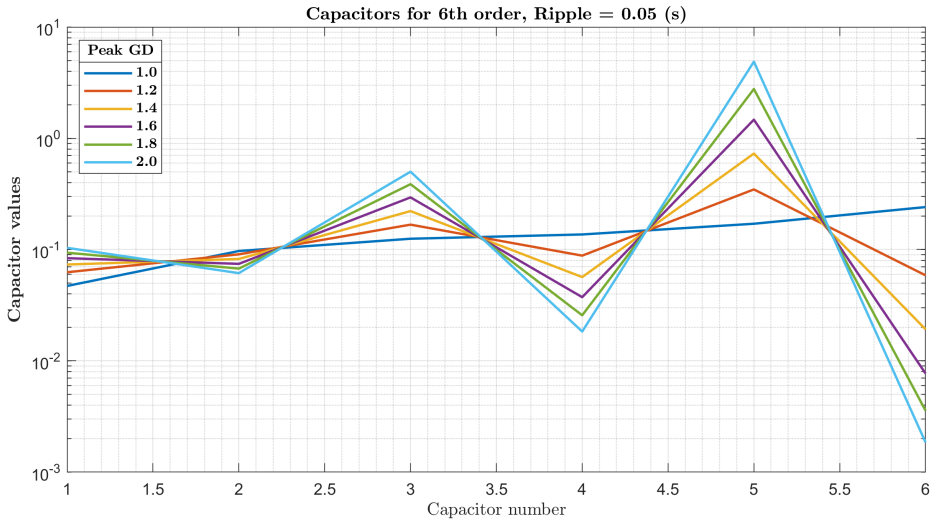
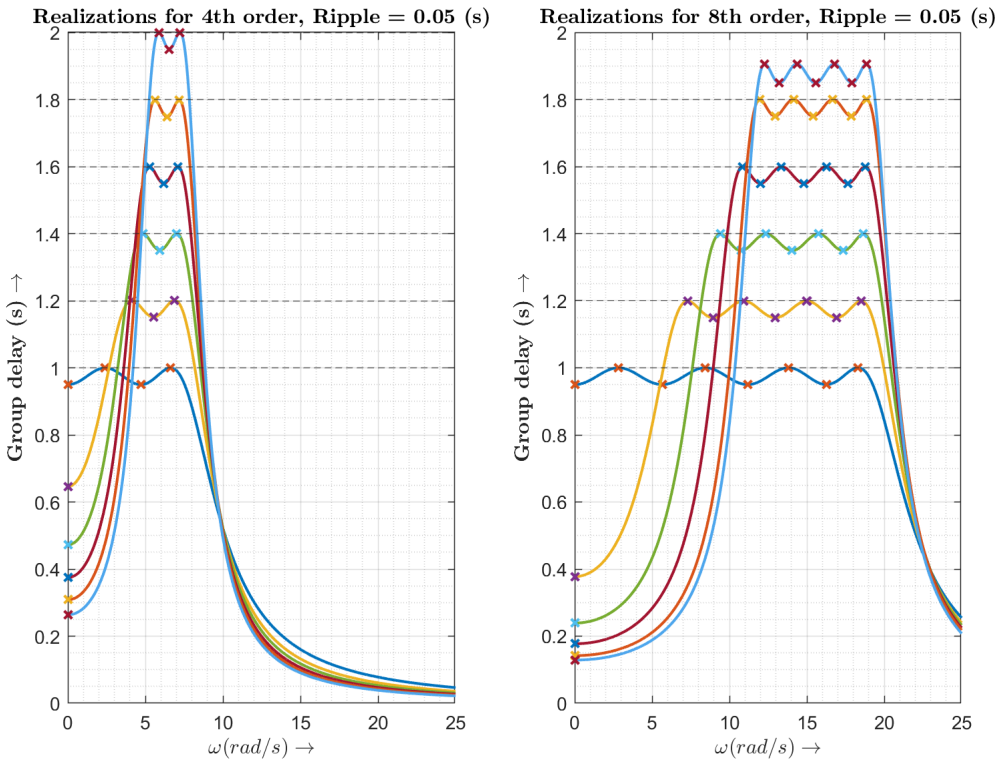


Figure 4.8: Capacitor values for different peak group delays for order N = 6.

Some filters synthesized for orders 4 and 8:



(a) (b)

Figure 4.9: Group delays with increasing values for order N = (a) 4 and (b) 8.

We can compare the capacitor values required for different orders. Fig. 4.10 (a) plots capacitor values for lowpass equiripples of N = 4, 6, and 8 together. We can

see that the capacitors required are smaller for higher orders, even with a larger delay-bandwidth product. However, the total capacitance increases with order. Fig. 4.10 (b) plots capacitor values for equiripple group delay for a peak of 1.8. the amount and the ratio of the maximum to the minimum value of capacitor increases with the order. This is a trade-off for higher delay-bandwidth products.

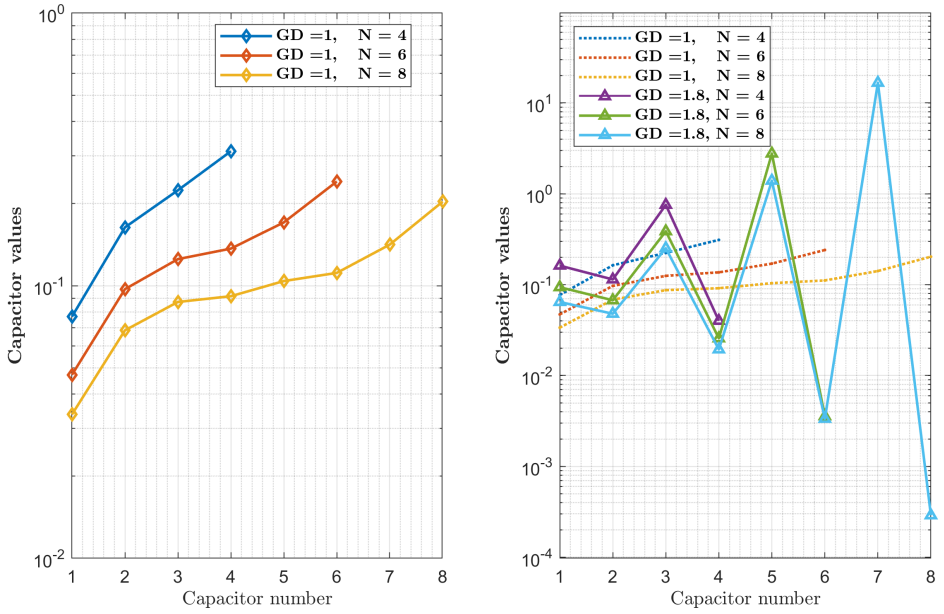


Figure 4.10: Capacitor values for lowpass and bandpass with peak of 1.8 for orders N = 4, 6, 8 and ripple of 0.05.

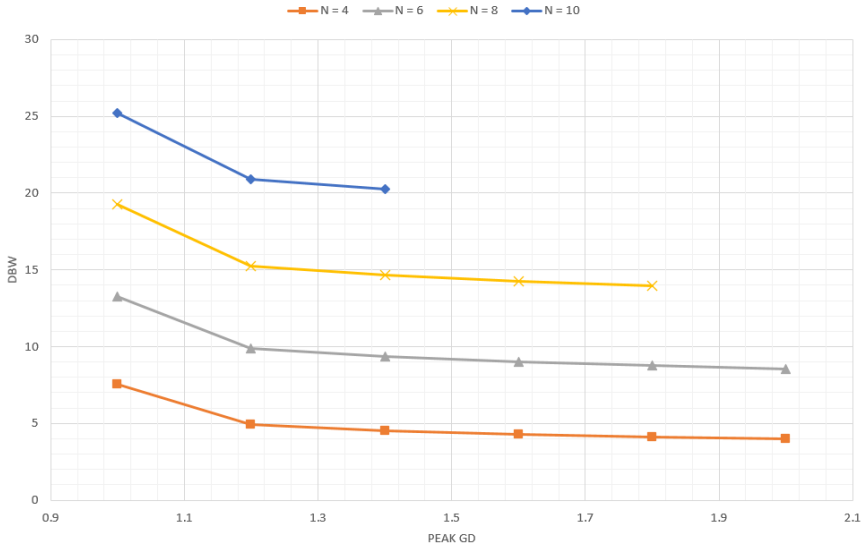


Figure 4.11: Plot of delay-bandwidth product achieved versus peak delay for different orders.

CHAPTER 5

Improving the Magnitude Response of the Lowpass Filter

We know that the all-pass filter architecture has an intrinsic low-pass filter. This can be realized at the final node V_n of the LC ladder. The lowpass transfer function is of the form $1/D(s)$. For an N^{th} order filter, $D(s)$ is of order N. The dc term is 1 for both numerator and denominator. The group delay and magnitude response for the lowpass transfer function for an order 6 filter is shown in Fig. 5.1.

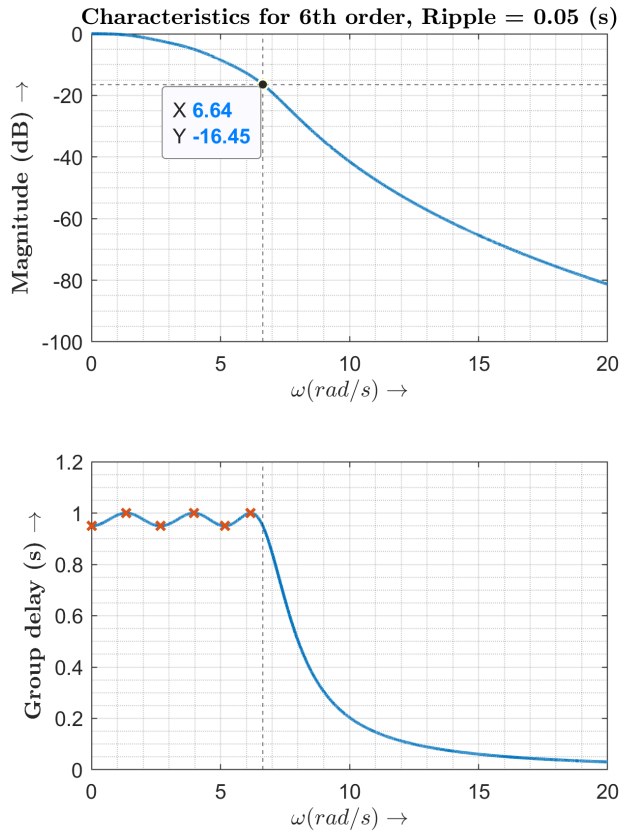


Figure 5.1: Magnitude and group delay response for a 6th order lowpass filter in the same architecture.

At the cut-off frequency of the filter, the group delay has dropped to -16.45 dB. Improving this magnitude response can help reduce error between the input and the output. In order to do this, we can add zeros to the system by adding higher powers of

‘s’ to the numerator. However, we can only add even powers of ‘s’ so that the group delay is not affected.

The results of some improvements with zero addition are given in the following figures. We can add up to power (N-2) in the numerator for an N order filter.

Adding a second order numerator to the 6th order filter, we can improve the magnitude to -8.27 dB from -16.45 dB without the magnitude peaking above the 0 dB level. This is shown in Fig. 5.2. a1, a2, and a3 in the figures represent the coefficients of the second, fourth, and sixth orders of ‘s’ respectively.

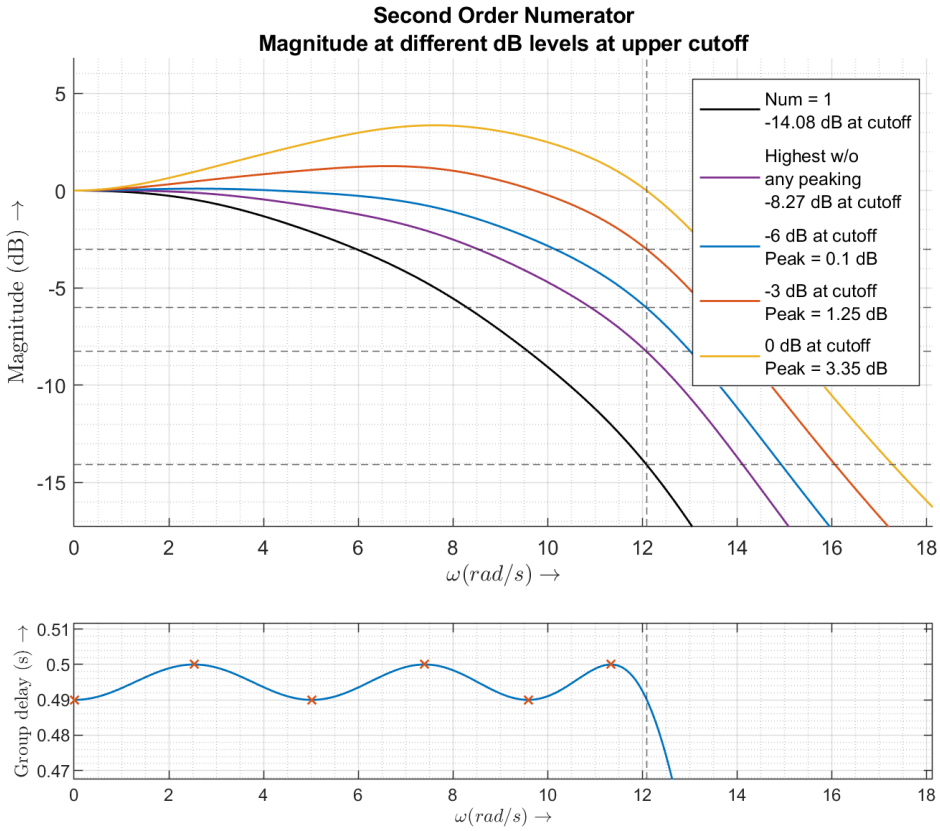


Figure 5.2: Improvement in magnitude response using a second order numerator for a 6th order filter.

If we allow for a 4th order numerator, the magnitude response can be improved further. This is shown in Fig. 5.3. For an order 8 filter, we can add up to a numerator of order 6. This is shown in Fig. 5.4.

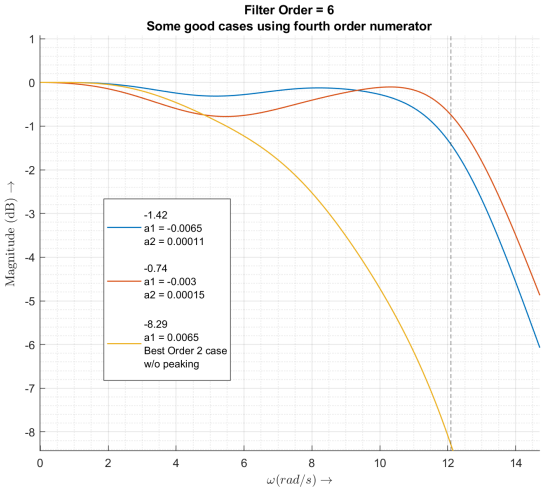


Figure 5.3: Improvement in magnitude response using a fourth order numerator for a 6th order filter.

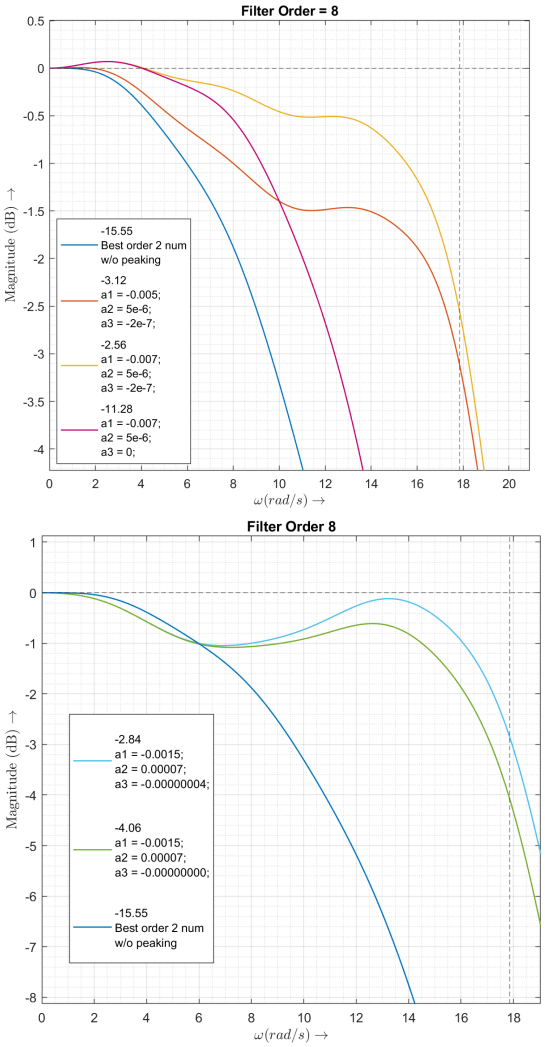


Figure 5.4: Improvements in magnitude response using 4th and 6th order numerators for an 8th order filter.

CHAPTER 6

Conclusion

An algorithm to synthesize a wide range of equiripple group delay filters using a wide-band tunable all-pass delay architecture is described. The algorithm has several features that allow it to avert the issues in the convergence of synthesis for large group delays or high order filters. Several filters with large delay-bandwidth products that have been synthesized are presented. Finally, results of improvement in the magnitude response of a lowpass transfer function present in the architecture using zero addition is described.

APPENDIX A

MATLAB Codes: Working and Program Flow

A.1 Inputs to the algorithm

The outermost function performing I/O from the user is *AnyNormGD.m*. The inputs to the function are:

type	Equiripple group delay to be synthesized. Only 'delay' accepted.
ripple	Desired ripple in normalized filter
order	Order of the filter to synthesized
target_maxgd_norm	Peak group delay of normalized filter
termination	'single' or 'double' terminated LC ladder
frequency	Range of frequencies for evaluation of group delay

A.2 Functions and execution flow

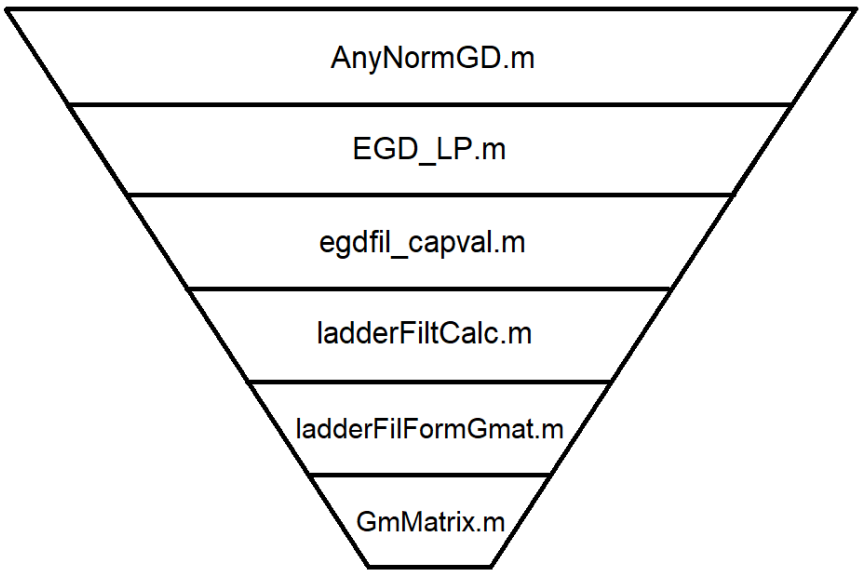


Figure A.1: Heirarchy of functions.

AnyNormGD.m: Makes all decisions regarding what peak GD and ripple to synthesize next. This function is what constitutes the 'algorithm' in general.

EGD_LP: Synthesizes the target given by AnyNormGD starting from equal capacitors or from any given state (capacitor array). This is the master to all the functions described lower in this list. However, it is AnyNormGD's job to make sure the given target is synthesizable by EGD_LP.

egdfil_capval: Performs the Newton-Raphson iterations using inputs from EGD_LP.

ladderFiltCalc: Evaluates all the data pertaining to a given capacitor array including group delay, peak values and locations to give to egdfil_capval.

ladderFilFormGmat: Forms the transconductance matrix given the circuit model.

GmMatrix: Constructs the matrix Gmmat to be given to ladderFilFormGmat using the filter order. If the circuit needs to be changed to another Gm–C architecture, code for GmMatrix should be changed.

REFERENCES

1. I. Mondal and N. Krishnapura, "A 2-GHz Bandwidth, 0.25–1.7 ns True-Time-Delay Element Using a Variable-Order All-Pass Filter Architecture in 0.13 μm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 8, pp. 2180-2193, Aug. 2017.

2. Imon Mondal, "Wideband Tunable True-Time-Delay Architecture Using a Variable Order All-Pass Filter and its Applications to Continuous - Time Pulse Processing," Ph.D dissertation, Dept. Elect. Eng., Indian Institute of Technology Madras, December 2017.

3. I. Mondal and N. Krishnapura, "Effects of AC Response Imperfections in True-Time-Delay Lines," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 4, pp. 1173-1177, April 2021.

4. A. I. Zverev, *Handbook of Filter Synthesis*, Wiley-Interscience, 1st edition, 1967.

5. Blinchikoff, Herman J., Zverev, Anatol I., *'Filtering in the Time and Frequency Domains' (Electromagnetic Waves, 2001)* DOI: IET Digital Library, https://digital-library.theiet.org/content/books/ew/sbew008e