ELEC9721 Assignment

Technical Report

Zid:Z5511123

Name:Danni Jin

Email:z5511123@ad.unsw.edu.au

Submission date: 23 June 2024

Table of Content

Introduction

IIR (Infinite Impulse Response) filters are very important in digital signal processing today. Compared with FIR (Finite Impulse Response) filters, IIR filters can usually achieve the same frequency response with lower orders, so generally speaking, IIR requires fewer computing resources. At the same time, lower orders mean less coefficient storage and fewer state variables. Although, if stability and linear phase are considered, FIR filters are better. To design an IIR filter, we need to consider handling quantization errors and managing computational complexity. When we use digital methods to design filters, q coefficient quantization is a critical step. To enhance filter performance, various methods can be used to optimize filter coefficients.

This experiment aims to design an IIR filter meeting specific specifications and use the RAG-n algorithm to optimize the coefficients of the IIR filter and compare the coefficient complexity before and after optimization.

The study is based on the RAG-n (Reduced Adder Graph-n) algorithm proposed by Andrew G. Dempster and Malcolm D. Macled for IIR filter design[1]. The RAG-n algorithm is primarily used to reduce the coefficient complexity of the FIR filter-by reducing the number of multipliers.

Through this experiment, it can be verified that the optimization effect of the RAG-n algorithm on the coefficients when applied to the IIR algorithm can provide a reference for the design and implementation of IIR filters.

Background

1. **Experimental requirements**：

Design an IIR filter with the following specifications:

Sampling rate: 16kHz.

Filter Type：Lowpass.

Passband ripple ： 1dB.

Stopband attenuation ： 30dB.

Passband cutoff frequency： 3.4kHz.

Stopband cutoff frequency – a random number you generate between 3.8 and 4.2kHz；

The experiment requires us to quantize the filter coefficients to 12 bits and and verify whether the quantized filter still meets the specifications. Configure the filter as direct form 1 and 2, cascade and parallel. Use the RAG-N algorithm to reduce the complexity of the coefficients in each case. Compare complexity with using separate binary coefficients.

2. Background theory

IIR Filter：IIR filter is a recursive filter, which is characterized by being able to meet frequency response requirements with a lower order. Its transfer function H(z) can be expressed as:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}} \tag{1}$$

Where B(z) is the numerator polynomial and A(z) is the denominator polynomial. The feedback characteristics of the IIR filter make the required order much smaller than that of the FIR filter to achieve the same frequency response.

The general form of the difference equation for an IIR filter is as follows:

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k] \tag{2}$$

Among them, y[n] is the output of the filter, x[n] is the input of the filter, a and b are the feedback coefficient and feedforward coefficient respectively. The output of the IIR filter at time n depends not only on the input before time n, but also on the output before time n. This is the recursiveness of IIR and the reason why it has a complex phase.

Coefficient quantization: In IIR filter design, coefficient quantization is an indispensable part. The filter coefficients obtained from theoretical design are usually floating-point numbers, which need to be converted into fixed-point numbers to adapt to hardware implementation. Using fewer bits can improve computational efficiency and reduce power consumption. It is important to note that quantization may introduce quantization errors, so it is necessary to verify the performance of the filter after quantizing the coefficients to ensure that the IIR filter is still stable. The experiment requires us to use 12-bit quantization. Compared with quantization with lower bits, it can significantly reduce nonlinear distortion and strike a good balance between the accuracy and complexity of filter design.

RAG-n Algorithm: The RAG-n algorithm[1] proposed in the literature is mainly used to optimize the coefficients of FIR filters by reducing the number of adders. This algorithm is divided into two parts: the first is optimal. Using graph theory methods, a minimum adder graph is constructed to reduce the number of adders. The second part is heuristic. It uses two lookup tables generated by the MAG algorithm of [2], which, at present, cover the range 1 to 4096. For each coefficient value in this range, the minimum single coefficient cost that includes this multiplication can be found by looking up the table.

The steps of the algorithm are as follows:

1. Normalize coefficients: Convert all even coefficients to odd fundamentals by dividing by 2 and store the results in the 'incomplete set'-the set of fundamentals not yet synthesized.

2. Evaluate all single-coefficient costs by using the cost lookup table.

3. Remove 0 cost coefficients: Remove from the incomplete set all cost-0 fundamentals or repeated fundamentals, as these incur no cost.

4. Create the 'graph set' for the storage of selected fundamentals: Create a partial sum set and remove all basic coefficients with a cost of 1 from the incomplete set and transfer them to a new set.

5. Generate new coefficients by combining them: Check the sum of the combinations of fundamental coefficients in the graph set (e.g., 3+1=4) and the power of 2 of each fundamental coefficient (e.g., for 3, check 3*2^1, 3*2^2).

6. Remove generated coefficients: Compare the new values generated above with the values in the incomplete set. If there are duplicates, remove the corresponding number from the incomplete set and add it to the graph set. Repeat until no more fundamentals are added to the graph set.

If there are still coefficients in the incomplete set after the above steps, you need to enter the second part of the algorithm: the heuristic part. The steps are as follows:

1. Calculate the additive distance between the coefficients in the incomplete set and the coefficients in the existing graph set, add the new vertices with an additive distance of 2 to the graph set and remove them from the incomplete set.

2. For those coefficients that are still in the incomplete set, find the minimum additive distance between the existing part and them to find the minimum cost path.

3. Generate a new cardinality based on the minimum cost path (generated by shifting the existing part a little).

4. Add the newly generated value to the graph set and remove it from the original set.

The literature shows that compared with the traditional CSD (Canonical Signed Digit) representation method and binary representation method, the RAG-n algorithm has a significant effect in reducing the number of adders.

Four types of filters：

1. Direct Form I: The most basic filter implementation method. The difference equation of direct form I is as follows：

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k] \tag{3}$$

2. Direct Form II: Direct Form II reduces the storage unit by combining the feedforward part and the feedback part of Direct Form I. The disadvantage is that it may cause numerical stability problems in high-order filters in actual use. Its differential equation is as follows：

$$\begin{gathered} w[n] = x[n] - \sum_{k=1}^{N} a_k w[n-k] \\ y[n] = \sum_{k=0}^{M} b_k w[n-k] \end{gathered} \tag{4}$$

3. Cascade Form: Cascade Form is the result of cascading multiple low-order filters, usually these low-order filters are second order. The transfer function of each second-order filter is as follows：

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{5}$$

Cascade Form is suitable for high-order filters, greatly improving filter stability and numerical accuracy.

4. Parallel Form: Parallel Form decomposes the filter into multiple parallel branches, each representing a portion of the filter's frequency response. The filter's transfer function is the sum of the transfer functions of each branch. The form is as follows：

$$H(z) = \sum_{i=1}^{L} H_i(z) \tag{6}$$

The disadvantage of Parallel Form is that it is difficult to implement, but it is also more flexible when designing the filter frequency response.

Main Body

Problem Statement

In short, the requirement of this experiment is to design an IIR filter that meets a specific frequency, quantize its coefficients and configure them into four different filter forms, use an algorithm to optimize the coefficients, and compare the coefficient complexity before and after optimization. Specifically, the experiment needs to solve the following problems:

1. **Filter design:** How to design an IIR filter? According to the question, we need to design a low-pass IIR filter. The filter order and normalized cutoff frequency are not provided and need

to be calculated. It is very important to select the appropriate function to design the filter. Common IIR filters include Butterworth filter, Chebyshev Type I Filter, Chebyshev Type II Filter, Elliptic Filter and Bessel Filter. When choosing a suitable filter design method, factors to consider include passband fluctuations of frequency response, stopband fluctuations, and transition band width. Whether the phase response needs to remain linear. If the order is too high, the filter must remain stable. Implementation complexity is also an important part to consider. Compared with other filters, the Butterworth filter can retain the integrity of the signal to the greatest extent when passing through the filter because it has the flattest passband and no ripples. Although its transition band is not steep, it is sufficient in most designs. Considering the stability requirements, the zero-pole diagram of the Butterworth filter shows that its zeros and poles are within the unit element, which is usually more stable than other filters. And the implementation method is also very simple, and you can use the 'buttord' function provided by MATLAB for direct design.

2. **Coefficient quantization:** The experiment requires us to quantize the filter coefficients to 12 bits and check whether the quantized filter still meets the design requirements of the question.
3. **Filter configuration:** The filter needs to be configured as: Direct Form I; Direct Form II; Cascade Form; Parallel Form. Use matlab's drawing function to analyze the frequency response and phase response of each form.
4. **Coefficient optimization:** The RAG-n algorithm proposed in the literature is used to optimize the coefficients. The complexity of the filter implementation is reduced by reducing the number of adders and multipliers. Finally, the results are compared intuitively with the binary coefficient representation method through MATLAB.

Software design

We use MATLAB for filter simulation and design. The specific ideas are as follows:

1. Design the specific parameters of the filter according to the requirements of the question. The sampling frequency fs, passband cutoff frequency fp, passband attenuation and stopband attenuation are all provided in the question. The stopband cutoff frequency fsb requires us to generate randomly. Here we can use the rand () function command of MATLAB to generate a random number between 3800 and 4200 in Hz.
2. Use functions to design filters. Based on the analysis in the previous problem statement, we choose Butterworth filter for design. MATLAB provides the buttord function for us to directly design. The required input parameters include the boundary frequencies of the passband and stopband, as well as the ripples of the passband and stopband (in dB). The function returns a specification of the minimum filter order n and the corresponding cutoff frequency Wn as output.
3. Normalized frequency: Before using the buttord function, you need to normalize the frequency of fp and fs. Normalization is to facilitate calculations. The essence is to convert different frequencies into a standard range (0 to 1). The formula is as follows:

$$W = \frac{f}{f_s/2} \tag{7}$$

Normalizing the frequency is not only an input requirement for the buttord function in MATLAB, but also ensures the stability of the filter. Normalizing the frequency ensures that the characteristics of the filter are relatively consistent at any sampling rate. It also simplifies the mathematical processing and reduces errors.

After determining the filter order n and the normalized cutoff frequency Wp through the butter function, n and Wp are used as the input of the butter function to obtain the filter coefficients b and a. b is the coefficient vector composed of the numerator polynomial, and a is the coefficient vector composed of the denominator polynomial. The nature of the butter function determines that the first parameter of a is 1.

4. Coefficient quantization: The coefficients can be quantized using the round() command of MATLAB. For a given filter coefficient ai and the number of quantization bits B, the quantized coefficient ai,quant can be expressed as:

$$a_{i,\text{ quant}} = \frac{\text{round}\left(a_i \times 2^{(B-1)}\right)}{2^{(B-1)}} \tag{8}$$

After obtaining the quantized b_quant and a_quant, the experiment requires us to compare whether the quantized filter still meets the parameter requirements provided by the question, so we use the freqz() function to input b_quant, a_quant, number of sampling points, and sampling rate to obtain the filter's frequency response value H_quant(z) and frequency point f_quant (range 0 to fs/2).

5. Check if the filter after quantization still meets the specifications. Convert H_quant(z) obtained by freqz function into logarithmic units, and then use find command to find the position of -3dB cutoff frequency. The corresponding f_quant value is the cutoff frequency. The same method can be used to calculate the passband ripple and stopband ripple. Print the value on the console and compare it with the value of the filter before quantization.
6. Configure the filter to Direct Form I, Direct Form II, Cascade Form and Parallel Form. Use the MATLAB functions dfilt.df1 () and dfilt.df2() to configure the filter to Direct Form I and Direct Form II. These two functions accept the quantized filter numerator coefficient b_quant and denominator coefficient a_quant as input and output a Direct Form I or Direct Form II filter object, which contains the filter coefficient information. You can use MATLAB's fvtool command to view the frequency response and phase response. Regarding how to create a cascade filter form, you can first try the tf2sos () function to convert the filter into the Second-Order Sections (SOS) form, and then use the output of this function as the input of the dfilt.df2sos () function to get a cascade filter object. For the creation of a parallel filter, you can first try the tf2zp () function to convert the filter into the Zero-Pole-Gain (ZPK) form. Then use the output of this function z (zero point), p (pole), k (gain) as the input of the zp2sos () function to create a cascaded second-order section object, and then use the dfilt.df2sos () function to get multiple second-order section filter objects, and finally use the dfilt.parallel() function to get the parallel form of the filter.
7. Use the RAG-n algorithm. Because the RAG-n algorithm can only process positive integers, we must first perform some processing on the quantized coefficients. Here, we choose to make the coefficients positive and then multiply them by a specific quantization scale factor (2^(B-1)). The obtained coefficients can be used as the coefficient input of the RAG-n

algorithm. This algorithm has a total of 3 outputs: addercost (adder cost), vertices (radix array sorted by addition order in the figure), and optimalflag (indicates whether the adder cost is the optimal cost of the input coefficient, 1 is optimal, 0 is not). The optimized coefficients are used in step 6 to form four forms of new filters.

8. Calculate the complexity and compare them. The complexity can be compared by comparing the number of adders and the number of multipliers. Direct Form I and Direct Form II can be calculated using the same function. The number of adders can be obtained by calculating the sum of the number of non-zero elements in the numerator and denominator coefficients and then subtracting 2 (the first element does not require an adder). The number of multipliers is equal to the sum of the non-zero elements in the numerator and denominator coefficients. Regarding the calculation of Cascade Form complexity, use the for function to traverse each row of the sos coefficient matrix. Each row represents a second-order filter. The first three coefficients represent the numerator coefficients, and the last three represent the denominator coefficients. Calculate the sum of their non-zero elements and then subtract 2 to get the number of adders required for this second-order filter. Finally, sum the number of adders in each row to get the number of adders required for the entire cascade filter. Similarly, calculate the number of non-zero elements in each row to get the number of multipliers for this second-order filter. The complexity calculation method of Parallel Form is very similar to that of Cascade Form. It is necessary to calculate the number of adders and multipliers in each branch and finally sum them up.

Software Design Flowchart

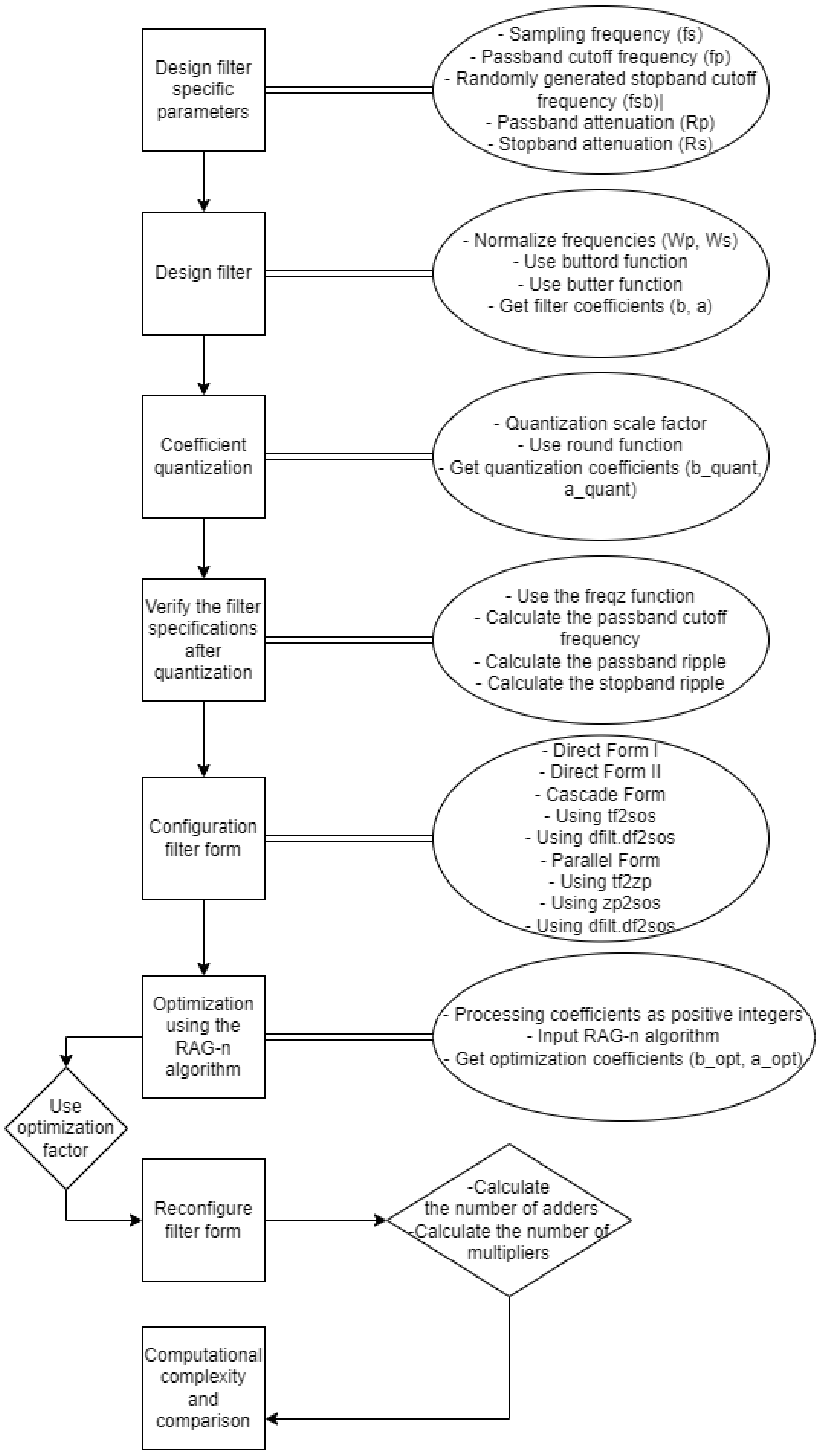
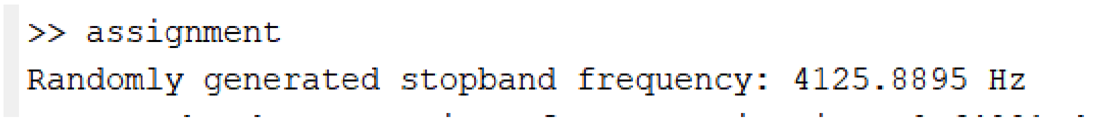

Design filter specific parameters
- Sampling frequency (fs)
- Passband cutoff frequency (fp)
- Randomly generated stopband cutoff frequency (fsb)|
- Passband attenuation (Rp)
- Stopband attenuation (Rs)
Design filter
- Normalize frequencies (Wp, Ws)
- Use buttord function
- Use butter function
- Get filter coefficients (b, a)
Coefficient quantization
- Quantization scale factor
- Use round function
- Get quantization coefficients (b_quant, a_quant)
Verify the filter specifications after quantization
- Use the freqz function
- Calculate the passband cutoff frequency
- Calculate the passband ripple
- Calculate the stopband ripple
Configuration filter form
- Direct Form I
- Direct Form II
- Cascade Form
- Using tf2sos
- Using dfilt.df2sos
- Parallel Form
- Using tf2zp
- Using zp2sos
- Using dfilt.df2sos
Optimization using the RAG-n algorithm
- Processing coefficients as positive integers
- Input RAG-n algorithm
- Get optimization coefficients (b_opt, a_opt)
Use optimization factor
Reconfigure filter form
-Calculate the number of adders
-Calculate the number of multipliers
Computational complexity and comparison

Fig.1. Software code flow chart

Fig.1 shows the main steps of the experiment, which is mainly a visual graphic description of the text description of the software design chapter. The square box represents the main steps of the experiment, and each corresponding circular box is a detailed description of the experimental step in the box. The triangle box represents the necessary process of two experimental steps.

Experimental results

1. Display: Randomly generated cutoff frequencies within the experimental requirements; filter frequency response and phase response under this condition.



```
>> assignment
Randomly generated stopband frequency: 4125.8895 Hz
```

Fig.2. Console print result of cutoff frequency fsb

Fig.2 Shows the filter cutoff frequency fsb result generated by the rand() function in the console output.

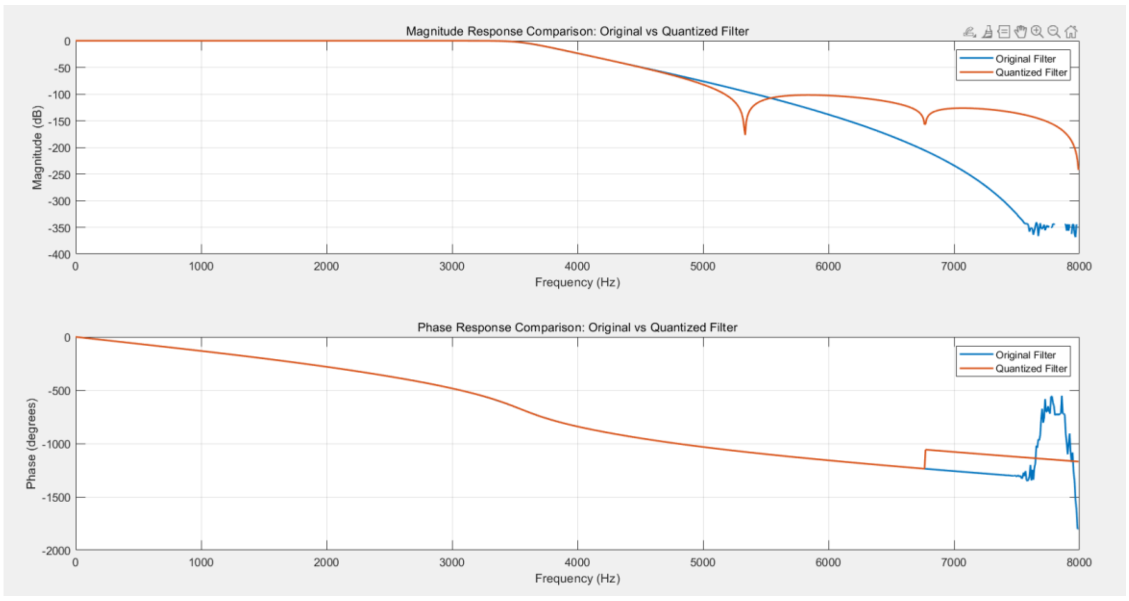


Fig.3. Magnitude and phase response comparison for original and quantized filters

From Fig.3, we can see that the upper part of the figure is a comparison of the frequency responses of the two filters, and the lower part is a comparison of the phase responses of the two filters. The blue line is the filter result without quantization of the coefficients, and the red line is the result of the filter after 12-bit quantization. Comparing the frequency responses of the filters, there is not much difference in the low-frequency part. In the high-frequency part, the stopband of the quantized filter has certain fluctuations, which may be caused by the quantization error. Comparing the phase response, we can see that the phase response of the quantized filter in the high-frequency part is more stable and there is no sudden change. The original filter has a higher phase sensitivity in the high-frequency part and changes more dramatically.

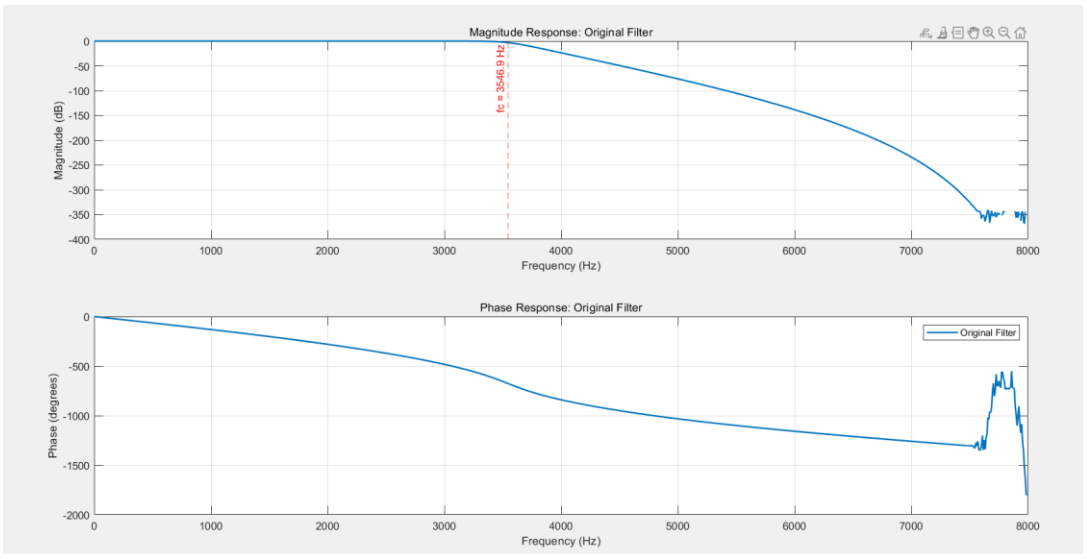


Fig.4. Original filter frequency response and phase response, with the -3dB cutoff frequency marked by the red line

Fig.4 shows the frequency response and phase response of the original filter, and the marked cutoff frequency fc=3546.9Hz. It is different from the filter cutoff frequency generated by the rand () function before. The reason is that Wn calculated when designing the filter using the butter function is the normalized frequency, and there may be certain numerical accuracy issues when looking up the -3dB cutoff frequency through freqz. The -3dB cutoff frequency is obtained by using freqz to find the frequency response H and frequency axis f of the filter, and then by converting H to a logarithm (20*log10(abs(H))), finding the index of the point where the frequency is less than -3dB, and then finding the corresponding f value, the -3dB cutoff frequency can be calculated. Similarly, the cutoff frequency fc of the filter after coefficient quantization can be calculated to be 3554.7Hz.

2. Display: The passband fluctuation and stopband fluctuation of the filter after coefficient quantization, and judge whether it meets the requirements of the question.

```
Max passband attenuation after quantization: 0.64901 dB
Min stopband attenuation after quantization: 30.2612 dB
```

Fig.5. Passband ripple and stopband ripple of the filter after quantization of coefficients

Fig.5 shows the passband ripple and stopband ripple of the quantized filter printed by the console. It meets the requirements of the question: passband ripple 1dB and Stopband attenuation 30dB. The passband ripple calculation method is still to find the frequency response range less than the passband cutoff frequency and find the minimum value and negate it. The stopband ripple calculation method is to find the frequency response range greater than the stopband cutoff frequency and find the maximum value and negate it.

3. Display: Frequency and phase response after quantizing the coefficients Configure the filter as direct form 1 and 2, cascade and parallel.

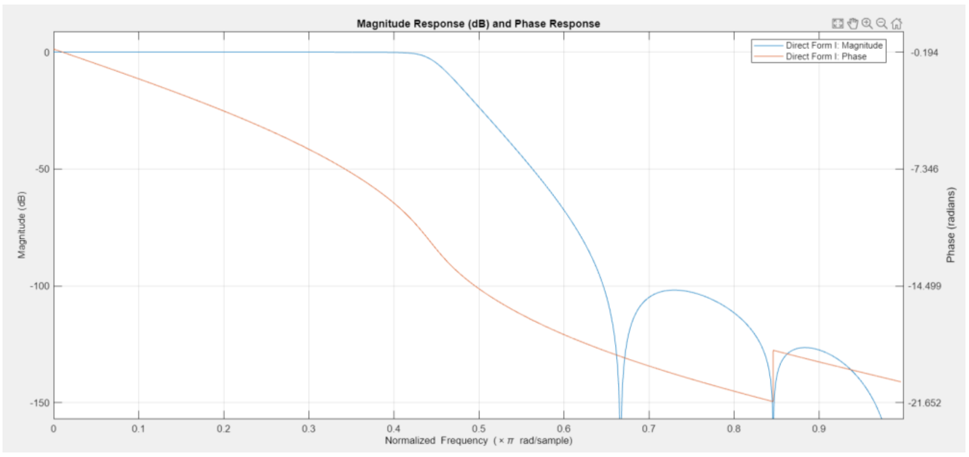
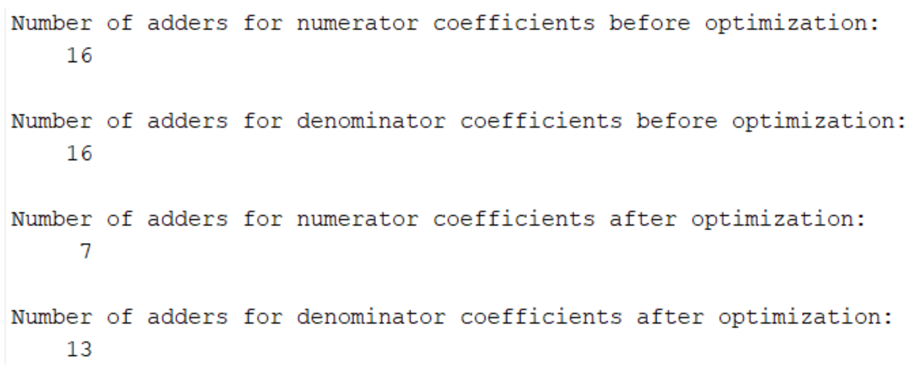


Fig.6. Frequency response and phase response of Direct form I after quantization coefficients

Fig.6 shows the frequency response and phase response of Direct Form I. Because the order of the filter is relatively high (n=15), some ripples will be generated in the transition band, and the low-pass filter has a strong suppression of high-frequency signals. The phase of the filter is linear. Direct Form II, Cascade Form and Parallel Form all have the same frequency response and phase response. I will not go into details here. The biggest difference between them is the structure, which also determines the different optimization effects of the coefficients using the RAG-n algorithm.

4. Display: Comparison of the number of adders required for Direct Form I and Direct Form II before and after using the RAG-n algorithm - Complexity calculation.

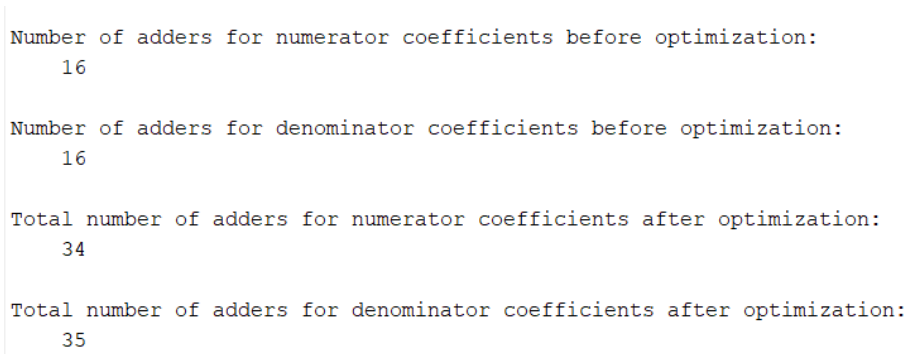


```
Number of adders for numerator coefficients before optimization:
    16

Number of adders for denominator coefficients before optimization:
    16

Number of adders for numerator coefficients after optimization:
     7

Number of adders for denominator coefficients after optimization:
    13
```

Fig.7. Comparison of the number of adders before and after the optimization coefficients of Direct Form I and Direct Form II

Fig.7 shows the comparison of the number of adders before and after optimization of Direct Form I and Direct Form II using the RAGN algorithm. Analyzing the filter structure of Direct Form I and Direct Form II, I found that if I want to use the RAG-n algorithm to optimize their coefficients, I must first transpose the IIR filter into small multiplier blocks and optimize the number of adders in each multiplier block. The coefficients of the numerator and denominator represent the feedforward and feedback, which need to be optimized separately, and the number of adders required is calculated. The final sum is the number of adders required for the entire system. The running results printed in the figure show that the RAG-n algorithm has a good effect of reducing adders for both Direct Form I and Direct Form II filters.

5.Display: Comparison of the number of adders required for Cascade Form before and after using the RAG-n algorithm - Complexity calculation.

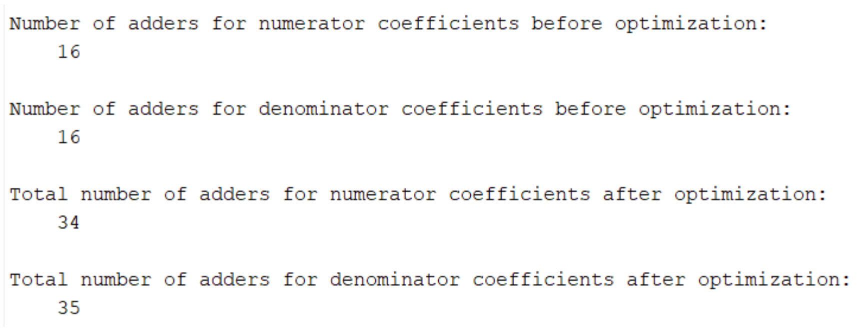


```
Number of adders for numerator coefficients before optimization:
    16

Number of adders for denominator coefficients before optimization:
    16

Total number of adders for numerator coefficients after optimization:
    34

Total number of adders for denominator coefficients after optimization:
    35
```

Fig.8. Comparison of the number of adders before and after the optimization coefficients of Cascade Form

Fig.8 shows the result: after using the RAG-n algorithm, the number of filter adders increased, and the coefficient optimization effect was not achieved. Analyzing the structure of the Cascade Form, it is composed of multiple second-order sections in cascade, which is the cascade result of multiple second-order filters. For each filter, the numerator and denominator need to be added 4 times in total, which is fixed, so the space for optimizing its adders is relatively limited. When the coefficient matrix is processed by the RAG-n algorithm, the floating point number needs to be quantized into integers before optimization. The quantized integer set is relatively large, so when using the RAG-n algorithm for calculation, the algorithm needs to generate more basic coefficients and possible sums, which increases the total number of adders. It is written in the literature that when the set is large, if the RAG-n algorithm cannot find enough coefficient parts with a cost of 1, it cannot get the optimal result. The use of suboptimal parts leads to an increase in the number of adders.

6.Display: Comparison of the number of adders required for Parallel Form before and after using the RAG-n algorithm - Complexity calculation.



```
Number of adders for numerator coefficients before optimization:
    16

Number of adders for denominator coefficients before optimization:
    16

Total number of adders for numerator coefficients after optimization:
    34

Total number of adders for denominator coefficients after optimization:
    35
```

Fig.9. Comparison of the number of adders before and after the optimization coefficients of Parallel Form

Fig.9 shows the result: after using the RAG-n algorithm, the number of filter adders also increased, and there was no effect of optimizing the coefficients. Analyzing the structure of Parallel Form, it is composed of multiple second-order section filters added together. These added second-order sections cannot be optimized because the RAG-n algorithm has a better optimization effect on the structure of one input and multiple outputs, a typical example is Direct Form II. Therefore, the coefficients can only be optimized for each second-order matrix structure separately. For each filter, the numerator and denominator need to be added 4 times in total.

Conclusions

Through this experiment, we successfully designed an IIR filter that meets specific specifications, with a cutoff frequency of 4125.8895Hz. The Butterworth design method was used. After 12-bit coefficient quantization, it was verified that the performance of the filter still meets the requirements of the topic. The performance of the quantized filter and the original filter was compared and analyzed. The filter was converted into four forms: direct form 1 and 2, cascade and Parallel, and analyzed. IIR filters have feedback paths. For Direct Form I and Direct Form II, the optimization effect of RAG-n algorithm is obvious, and the number of adders is significantly reduced. However, for Cascade Form and Parallel Form, the optimization effect of RAG-n algorithm is very general. This is due to the different structures of each filter type. The left side of Fig.10 below is a simple second-order matrix, and the right side is its transposition. We found that after transposing this structure, it can be transformed into a structure with an input flowing through multiple delays. In this structure, the RAG-n algorithm works best because it will calculate the best results for these coefficients.

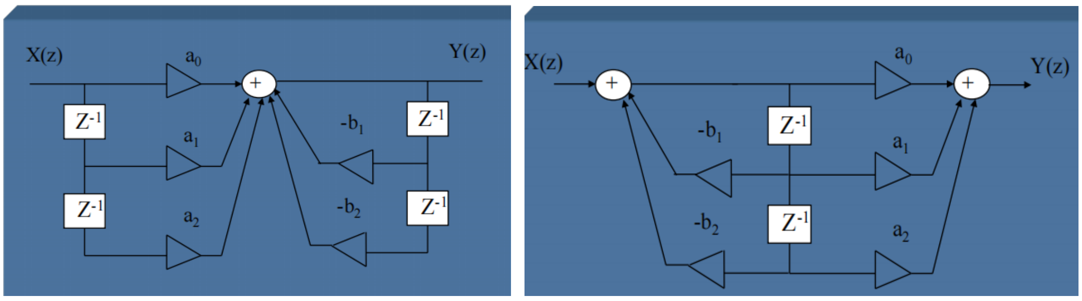


Fig.10. left: simple second-order Direct Form I structure; right: simple second-order Direct Form II structure

The left side of Fig.11 below shows the Cascade Form with three second-order sections in cascade connection, and the right side shows the Parallel Form with three second-order sections in parallel connection. This structure is optimized by the RAG-n algorithm to optimize the second-order sections in each branch, and the optimization of the overall filter structure is not high.

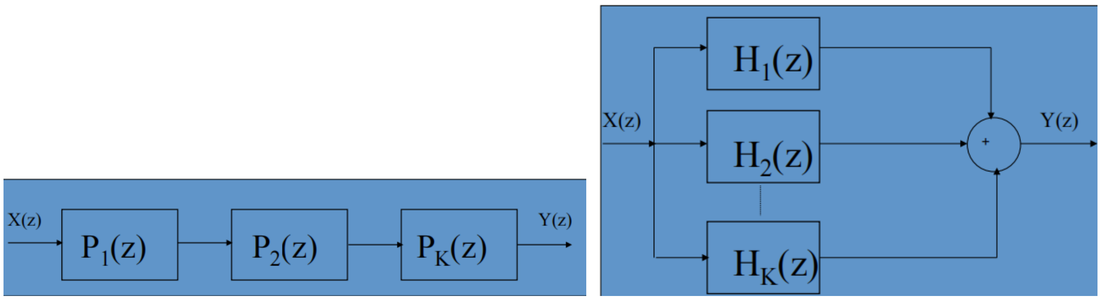


Fig.11. left: Cascade Form of three second-order filters in cascade; right: Parallel Form of three second-order filters in parallel

The IIR filter in the literature [2] also uses the Minimum Adder Multiplier Blocks method. A second-order transpose type filter can use one multiplier block to replace five multiplication coefficients. The optimization results of Direct Form II introduced in the paper are slightly better than Direct Form I, but their complexity is still very close.

References

[1] Dempster, A.G. and M.D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters", IEEE Trans Circuits & Systems II - Digital & Analog Signal

Processing, vol. 42(8), no. 9, pp. 569-577, September 1995.
[2] Dempster, A G and M D Macleod, "IIR Digital Filter Design Using Minimum-Adder Multiplier Blocks", IEEE Trans Circuits & Systems II - Digital & Analog Signal Processing, vol. 45, no. 6, pp. 761-763, June 1998.