Advanced Systems & Signals Engineering (E101)

Final Project

Project Team

Isaiah Jeter

George Davis

Emmett Stralka

Matthew Molinar

December 2024

2. Background

This project aims to analyze the effect of analog and digital filters, which have discrete and continuous time signal processing implications. The fundamental signal of the system is a PWM square wave generated by a pulse train oscillator chip. This signal is fed into an analog filter, which we designed as a 4th-order Butterworth using the Sallen Key circuit configuration (see Section 1.1). This analog filter is fed into an ADC pin on an Arduino, which applies a digital 10th-order comb filter that we designed (see Section 1.2). Then, MATLAB was used to conduct a time and frequency domain analysis of the Arduino data. The project aims to reinforce our understanding of time and frequency domain analysis while practicing signal processing techniques.

2.1 Butterworth Design Calculations

The Butterworth filter is a low-pass filter that attenuates all frequencies above its corner frequency. It has a transfer function that characterizes the system's frequency response. The general form of this function is seen in Equation 1. Since we are designing a fourth-order filter, we plugged in four for N and evaluated the poles. We only kept the poles on the left side of the y-axis because it would ensure a stable system. Then, we expressed our new transfer function using these poles (see Eq 2).

Finally, we could plug the transfer function for the 4^{th} order Butterworth filter into MATLAB to make the bode plot. Since the passband frequency was given as 10 Hz, we knew that our corner frequency (w_c) would be 20π.

$$H(s)H(-s) = \frac{1}{1 + (\frac{1}{jw_c})^{2N}} \qquad \text{(Eq 1)}$$

where; *N = Filter Order* & w_c *= Corner Frequency [Hz]*

$$H(s) = \frac{w_c^4}{(s^2+A\cdot s\cdot w_c - w_c^2)(s^2+B\cdot s\cdot w_c - w_c^2)} \qquad \text{(Eq 2)}$$

where; w_c = *Corner Frequency [Hz]* & $A = \sqrt{2-\sqrt{2}}$ & $B = \sqrt{2+\sqrt{2}}$

$$H(s) = \frac{1}{s^2 R_1 R_2 C_1 C_2 + s(R_1+R_2)C_2 + 1} \qquad \text{(Eq 3)}$$

We were able to use the "Sallen Key Circuit" transfer function given in Eq 3 alongside the capacitor values given to find our necessary resistor values. The Sallen Key transfer function describes one stage of our 4th order Butterworth filter, which has two stages with one op-amp each (see Figure 1).

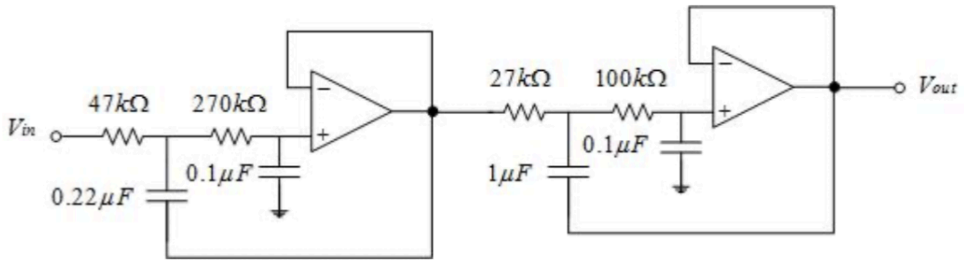


Figure 1: Two-Stage Sallen-Key Filter Schematic

Once we solved for the necessary resistor values (see Tables 1 and 2), we had all the components necessary to build the filter on a breadboard with V_{in} being driven by the pulse train oscillator chip (see Figure 2).

First Stage ζ = Higher Damped	R_1^I [kΩ]	R_2^I [kΩ]	C_1^I [μF]	C_2^I [μF]
Calculated	46.5	247.6	N/A	N/A
Actual	47	270	0.22	0.1

Table 1: First Stage Calculated and Actual Resistor/Capacitor Values

Second Stage ζ = Less Damped	R_1^{II} [kΩ]	R_2^{II} [kΩ]	C_1^{II} [μF]	C_2^I [μF]
Calculated	26.6	95.2	N/A	N/A
Actual	27	100	1	0.1

Table 2: Second Stage Calculated and Actual Resistor/Capacitor Values

Using the transfer function (Eq. 5) and plugging in the resistor and capacitor values from Tables 1 and 2, we can estimate the ratios of the Fourier series coefficient on the analog filter.

$$H(s) = \frac{1}{s^2 R^I_1 R^I_2 C^I_1 C^I_2 + s(R^I_1 + R^I_2)C^I_2 + 1} * \frac{1}{s^2 R^{II}_1 R^{II}_2 C^{II}_1 C^{II}_2 + s(R^{II}_1 + R^{II}_2)C^{II}_2 + 1} \quad \text{(Eq. 4)}$$

2.2 Modified Digital Comb Filter Design Calculations

The Digital Comb filter isolates specific frequencies and attenuates surrounding noise and other signals much like a band pass filter. We designed a low-pass modified 10th-order comb filter to apply to the output from the analog filter in Arduino. The impulse response function on which we modeled our comb filter is shown in Equation 5. Instead of impulses, we used the previous values of the input to multiply by the coefficients. The previous values would be updated on each cycle the Arduino ran. To get the output from the analog filter, we had an Arduino sketch that read the values of the analog pin which we then logged and provided as an input to the digital comb filter. The code for the comb filter can be found in Appendix A.

$$h[n] = 0.0382 + 0.1\delta[n-1] + 0.1618\delta[n-2] + 0.2\delta[n-3] + 0.2\delta[n-4] + 0.1618\delta[n-5] + 0.1\delta[n-6] + 0.0382\delta[n-7]$$

(Eq 5)

Figure 2 shows the combination of the analog and digital filters to collect experiment results.

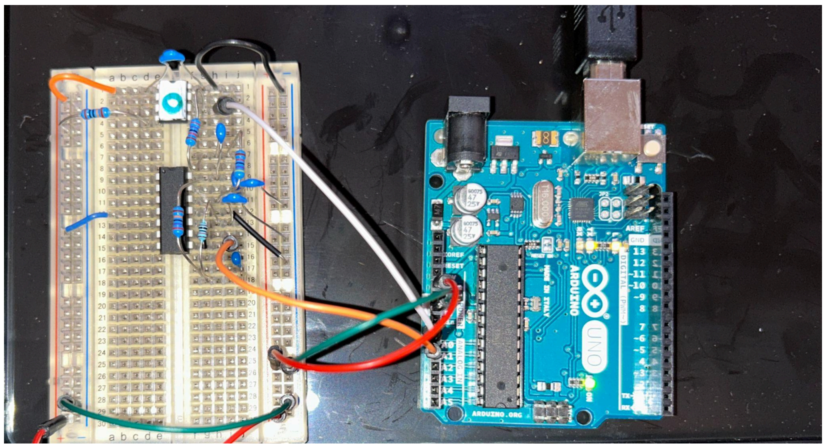


Figure 2: Pulse Train Generator (Top-Left), Fourth Order Butterworth Filter (Bottom-Left), and Modified Digital Comb Filter (Right)

3. Experimental Results

Pictured below are the time and frequency-domain plots for the pulse train, pulse train with a Hanning window applied, Butterworth filter, and comb filter using the Matlab code provided in Appendix B, demonstrating the filters' effectiveness.

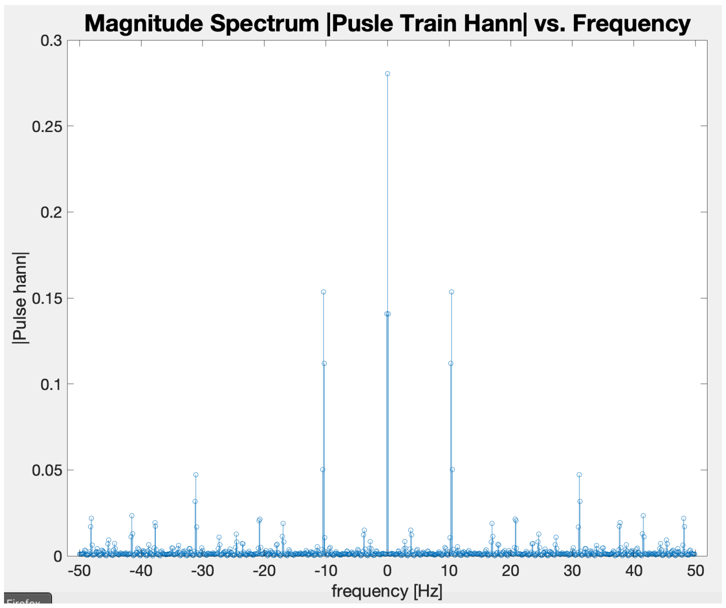
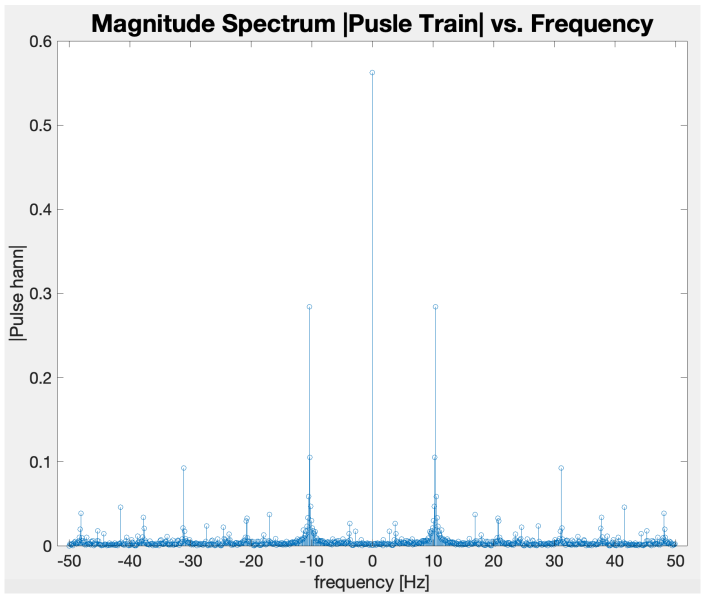
	

Figure 3: Frequency Domain of Windowed Pulse Train	Figure 4: Frequency Domain of Pulse Train

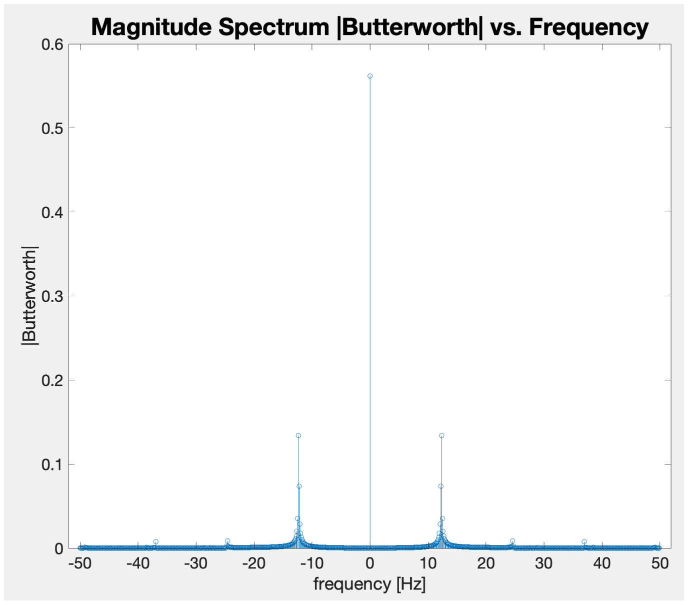
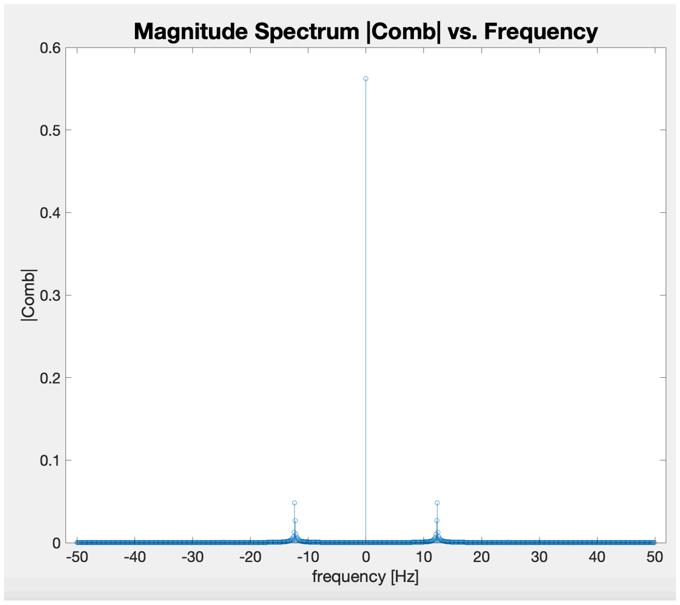
	

Figure 5: Frequency Domain of Analog (Butterworth) Filter	Figure 6: Frequency Domain of Digital (Comb) Filter

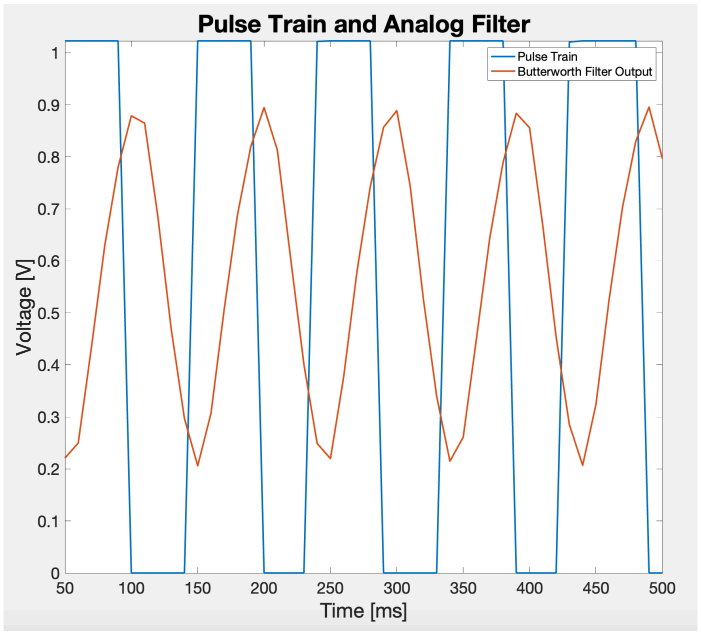
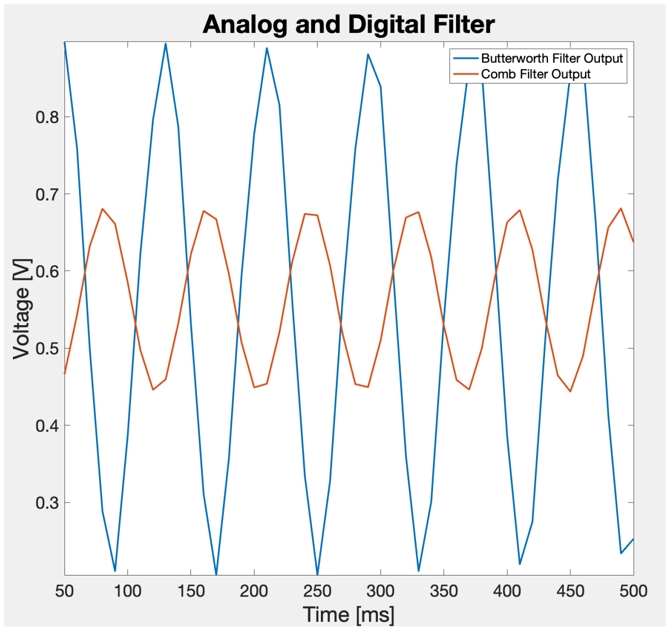

	

Figure 7: Time domain of Analog (output) filter and Pulse Train (input)	Figure 8: Time domain of Digital filter (output) and Analog input (input)

3.1 Fourier Series Coefficients of Sampled Periodic Pulse-Train Using Windowing

Pulse Train				
Measured @ Frequency [Hz]	**10.4**	**20.8**	**31.1**	**41.5**
Estimated Coefficients	0.3274	0.0507	0.1142	0.0599
Measured Coefficients	0.284134	0.0327985	0.0926331	0.0457848
Measured Ratio (Hanning)	0.153553	0.0213828	0.0472237	0.023407

Table 3: Estimated and Measured Coefficients of the Pulse Train

3.2 Fourier Series Coefficients of the Input and Output Signals of the Analog Filter

Butterworth Filter				
Measured @ Frequency [Hz]	**12.349**	**24.576**	**36.943**	**49.153**
Estimated Coefficients	0.08	0.00184548	0.00099354	0.00017371
Measured Coefficients	0.1340000	0.00863393	0.0078170	0.0011282
Estimated Ratio	0.2514	0.0364	0.0087	0.0029

Table 4: Estimated and Measured Coefficients of the Butterworth Filter (Analog)

3.3 Fourier Series Coefficients of the Input and Output Signals of the Digital Filter

Digital Filter				
Measured @ Frequency [Hz]	**12.3487**	**24.5763**	**36.9249**	**49.1525**
Estimated Coefficients	0.04327773569	1.22E-19	1.87E-09	6.76E-21
Measured Coefficients	0.048127	0.000466	0.000182	0.000107
Estimated Ratio	0.5258	6.58E-17	1.89E-06	3.89E-17

Table 5: Estimated and Measured Coefficients of the Modified Comb Filter (Digital)

(See section 4 to see how we determined the measured coefficients and ratio)

4. Analysis of Results

4.1 Details of Methods Used and Evaluation of Performance of Analog and Digital Filters

The first step in processing the signal from the pulse train and the filter was to log the output data using the Arduino. We then used the provided *fdomain,* Appendix B, function to get the Fourier transform in the time domain of the different signals. Since the pulse train was at a frequency of 10hz and periodic, we measured the magnitude of the Fourier transforms of each filter at frequency intervals of 10hz. The magnitudes gave us our measured coefficients for each filter's Fourier Transform. To get the measured ratio of the coefficients, we divided the measured coefficients of the output filter by the coefficients of the input filter. The measured ratio for the analog filter was the measured coefficients of the analog filter divided by the measured coefficients of the Pulse Train. The measured ratio for the digital filter was the measured coefficients of the digital filter divided by the measured coefficients of the analog filter.

First, we applied a Hanning window to the time domain representation of the pulse train. This has the effect of pinching the edges of the data vector so that when the FFT command *forces* the signal to repeat it does not have large high-frequency components that show up in the frequency domain. Then, we used equation 6 to estimate the pulse train coefficients.

$$|a_k| = \left|\frac{A\sin(k\pi d)}{N\sin(k\pi/N)}\right| \qquad \text{(Eq 6)}$$

where; A = Amplitude, k = Coefficient Number, N = Period, d = Duty Cycle

To estimate the Butterworth filter and comb filter coefficients, we followed the process detailed above. The frequencies at which we evaluated the Analog filter (Eq 4) were based on an ideal fundamental frequency of the pulse train of 10 Hz. For the digital filter (Eq 7), we evaluated the transfer functions at $e^{j\hat{w}}$ frequencies of 10 Hz, 20 Hz, 30 Hz, and 40 Hz for the same reason.

$$H_{modified\_comb}(z) = 0.0382(1+2.618z^{-1}+2.618z^{-2}+2.618z^{-3}+2.618z^{-4}+2.618z^{-5}+2.618z^{-6}+2.618z^{-7}) \qquad \text{(Eq 7)}$$

where; b_0 = *Scaling Constant of* & *M = Filter Order of 10*

$$\hat{w} = \frac{2\pi f}{f_s} \qquad \text{(Eq 8)}$$

where; $f_s = 100\,Hz$

4.2 Choosing Op-amp Stages, Aliasing, Frequency Resolution, and Windowing Functions

The Sallen-Key op-amp stages were chosen to make the analog Butterworth low-pass filter attenuate the correct frequencies while maintaining stability. The first stage of the filter was designed with higher damping to ensure stability and minimize overshoot in the transient response. The second stage had lower damping to achieve a sharper frequency selectivity and maintain an effective cutoff, balancing stability, and performance.

We ensured the sampling rate was high enough to prevent aliasing, following the Nyquist sampling theorem. This was important for the entire system because it would ensure signal integrity prior to frequency domain analysis. This was particularly important when designing the digital comb filter because digital filters rely on precise delay, which would become distorted by aliasing if not properly implemented.

A high-frequency resolution comes with many data points that allow the FFT function to have accurate and consistent data to convert. Tangentially, if there are not enough data points the Hanning Window would *pinch* too many periods of the signal and not be accurate when passed through the FFT.

Finally, we applied windowing functions like Hanning Window to reduce the effect of FFT creating new high-frequency components to force the signal to become periodic. This results in an improvement in the accuracy of our frequency domain signal analysis.

4.3 Sources of Error

We identified three potential sources of error while collecting and analyzing this project's digital and analog filters. First, all passive resistors and capacitors have tolerances of about $\pm$ 5%. In our calculations, we assumed that the resistors and capacitors had their “actual” values in the lab (see Tables 1 and 2). However, we could have measured these resistors and capacitors using a multimeter and used these values for our calculations instead. This would have led us to a more accurate transfer function for the Butterworth filter, which would have led to results that more closely matched our expectations. Second, the 10-bit ADC of the Arduino Uno provides a limited resolution of analog data. This effect shows up in rounding errors in the sampled data, especially for small signal amplitudes, because they are much closer to the lower limit of what the binary encoding can handle. Thirdly, parasitic shunt capacitance and series inductance in breadboard circuits or wires can alter the filter’s behavior by increasing the system's impedance.

5. Conclusions

The results of our project confirm the successful design and testing of both the analog Sallen-Key Butterworth filter and the modified digital comb filter. The measured output closely matched the expected performance, confirming our design parameters' appropriateness. The performance matched closely (section 3) for the pulse train, where all coefficients were within 0.05 or less of the estimated values. The Butterworth analog filter exhibited a deviation of 0.05 or less, while the digital modified comb filter had a deviation of less than 0.005, highlighting that the digital filter demonstrates less tolerance for errors than the dependent analog pulse generator and filter as demonstrated in section 4.2. The Butterworth filter attenuated all signals above our input frequency of 10Hz demonstrating that it is an effective low-pass filter. The digital comb filter effectively isolated the signal it received from the analog filter as we expected. Ultimately, this project highlights the importance of filter design in signal processing and the feasibility of implementing these filters in practical applications.

Appendix A: Arduino Code

Analog Filter:

Reads analog values from Arduino from two input pins on an Arduino board and sends the results to the serial monitor for display

```
/*
Reads 2 analog input pins (range 0-1023) and
prints the results to the serial monitor.
The circuit:
* analog signal connected to analog pin A0.
* analog signal connected to analog pin A1.
*/
// These constants won't change. They're used to give names
// to the pins used:
const int inputPin0 = A4; // Analog input pin 0
const int inputPin1 = A2; // Analog input pin 1
void setup() {
Serial.begin(9600);
}
void loop() {
// analogRead values from 0 to 1023 (10 bit ADC)
int sensorVal0 = analogRead(inputPin0);
int sensorVal1 = analogRead(inputPin1);
// print the results to the serial monitor:
Serial.print(sensorVal0);
Serial.print(" ");
Serial.println(sensorVal1);
delay(10); // sample time ms
}
```

Digital FIR Filter:

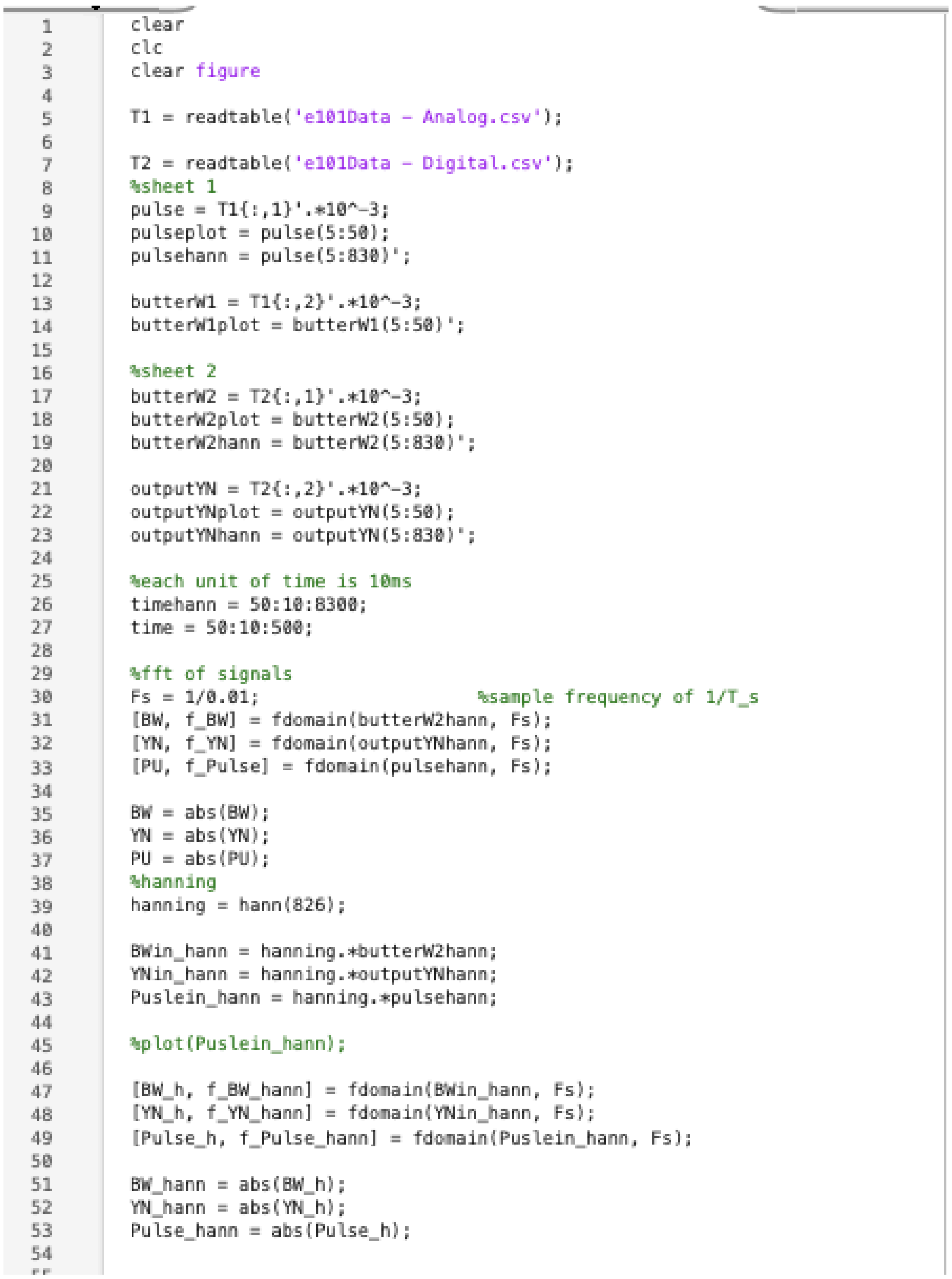
Reads analog input pin (range 0-1023) and E101 2024 Final Project Analog and Digital Filters

prints the filtered output to the serial monitor

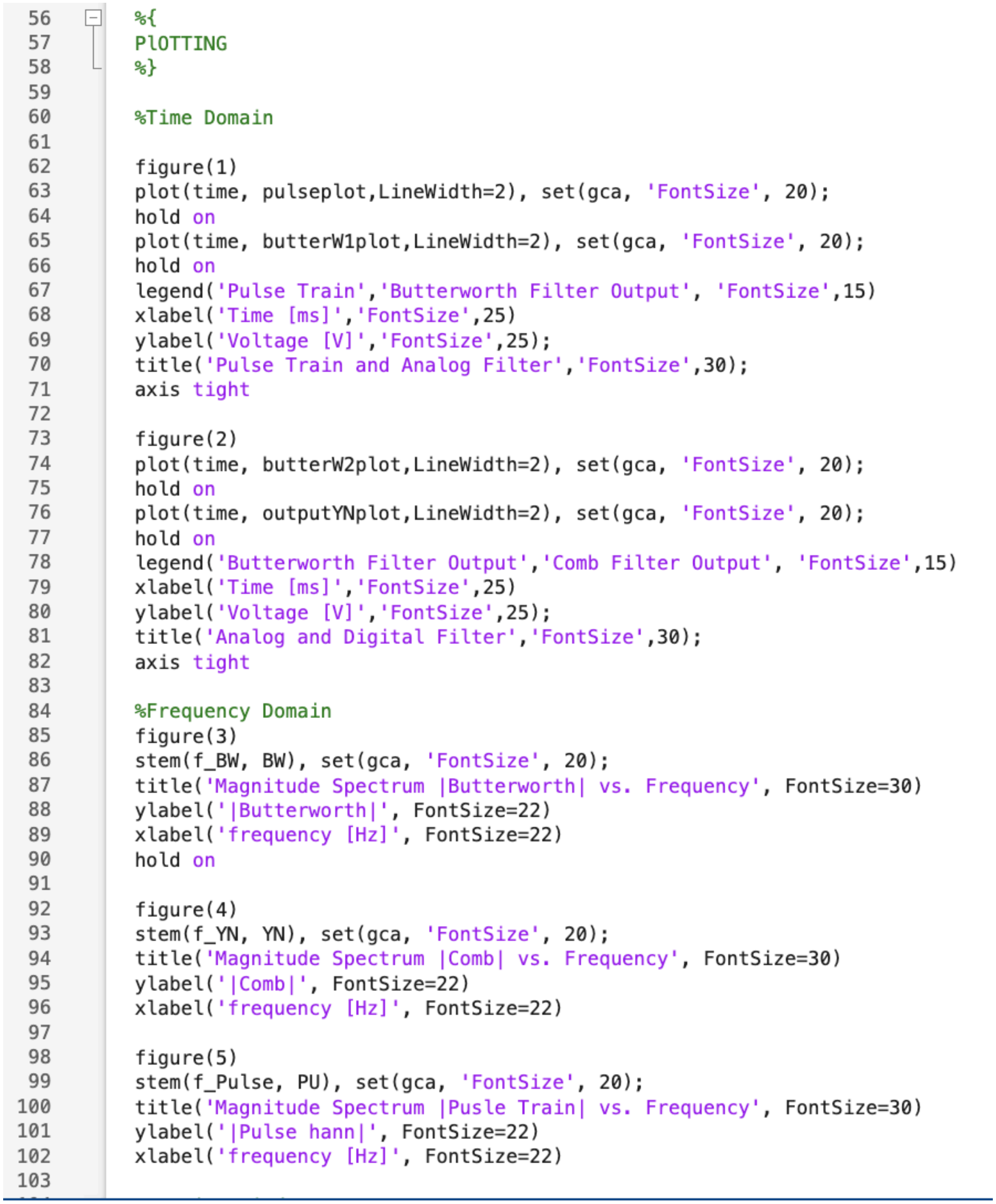
```
/*
Digital FIR Filter
Reads analog input pin (range 0-1023) and
E101 2024 Final Project Analog and Digital Filters
prints the filtered output to the serial monitor
The circuit:
* analog signal connected to analog pin A1
The filter  used  in  this  template  is  a three point averaging filter.
You will  need  to  adapt the program for a modified  comb  filter
*/

const int inputPin = A2; // Analog input pin 1 butterworth
// filter coefficients
const float b0=0.0382;
const float b1=0.1;
const float b2=0.1618;
const float b3=0.2;
const float b4=0.2;
const float b5=0.1618;
const float b6=0.1;
const float b7=0.0382;
// initial values for internal signals
int n = 0;
float yn = 0;
float x1 =0;
float x2 =0;
float x3 =0;

float x4 =0;

float x5 =0;

float x6 =0;

float x7 =0;


void setup() {

Serial.begin(9600);

}


void loop() {


// analogRead values from 0 to 1023 (10 bit ADC)

int xn = analogRead(inputPin); //output of butterworth


// difference equation for filter

yn = xn*b0 + x1*b1 + x2*b2+ x3*b3 + x4*b4 + x5*b5 + x6*b6 + x7*b7;



//update stored values


// print to output

//butterworth

Serial.print(xn);

Serial.print(" ");

//output

Serial.println(yn);



// update stored values

x7=x6;
x6=x5;

x5=x4;

x4=x3;

x3=x2;

x2=x1;

x1=xn;


delay(10); // sample time ms

}
```

Appendix B: MATLAB Code

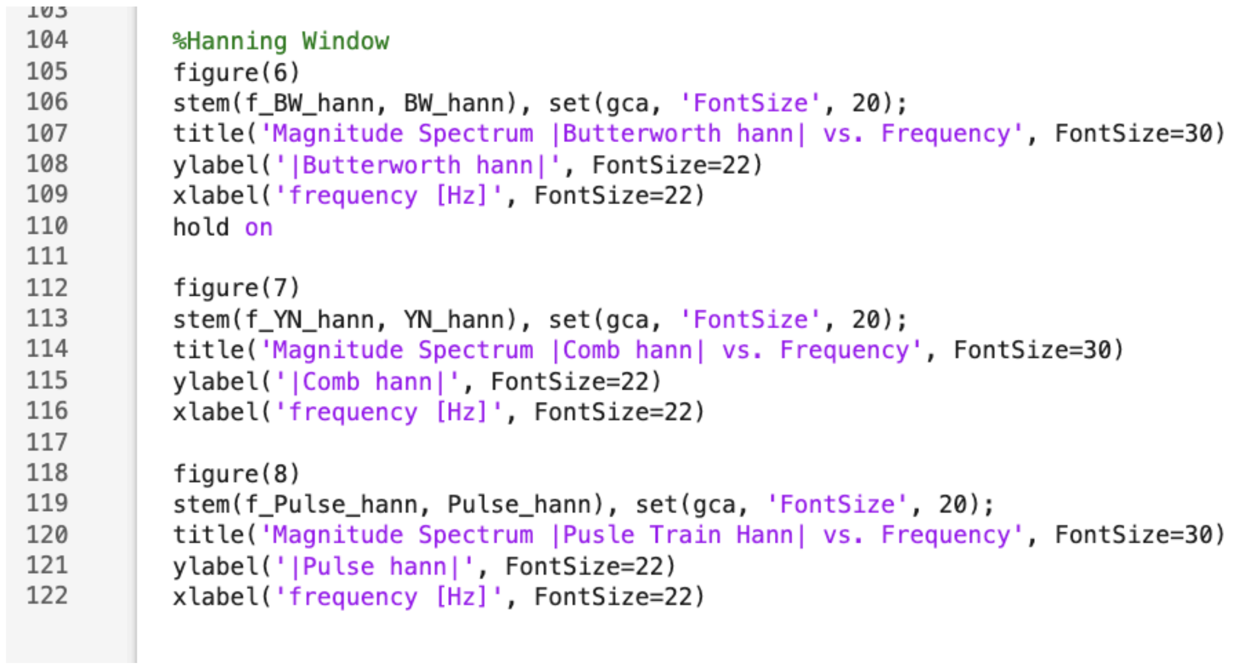
Reads the Arduino data into a MATLAB table, performs fft, and plots the time and frequency representation of each stage of the system.



```matlab
clear
clc
clear figure

T1 = readtable('e101Data - Analog.csv');

T2 = readtable('e101Data - Digital.csv');
%sheet 1
pulse = T1{:,1}'.*10^-3;
pulseplot = pulse(5:50);
pulsehann = pulse(5:830)';

butterW1 = T1{:,2}'.*10^-3;
butterW1plot = butterW1(5:50)';

%sheet 2
butterW2 = T2{:,1}'.*10^-3;
butterW2plot = butterW2(5:50);
butterW2hann = butterW2(5:830)';

outputYN = T2{:,2}'.*10^-3;
outputYNplot = outputYN(5:50);
outputYNhann = outputYN(5:830)';

%each unit of time is 10ms
timehann = 50:10:8300;
time = 50:10:500;

%fft of signals
Fs = 1/0.01;                    %sample frequency of 1/T_s
[BW, f_BW] = fdomain(butterW2hann, Fs);
[YN, f_YN] = fdomain(outputYNhann, Fs);
[PU, f_Pulse] = fdomain(pulsehann, Fs);

BW = abs(BW);
YN = abs(YN);
PU = abs(PU);
%hanning
hanning = hann(826);

BWin_hann = hanning.*butterW2hann;
YNin_hann = hanning.*outputYNhann;
Puslein_hann = hanning.*pulsehann;

%plot(Puslein_hann);

[BW_h, f_BW_hann] = fdomain(BWin_hann, Fs);
[YN_h, f_YN_hann] = fdomain(YNin_hann, Fs);
[Pulse_h, f_Pulse_hann] = fdomain(Puslein_hann, Fs);

BW_hann = abs(BW_h);
YN_hann = abs(YN_h);
Pulse_hann = abs(Pulse_h);
```



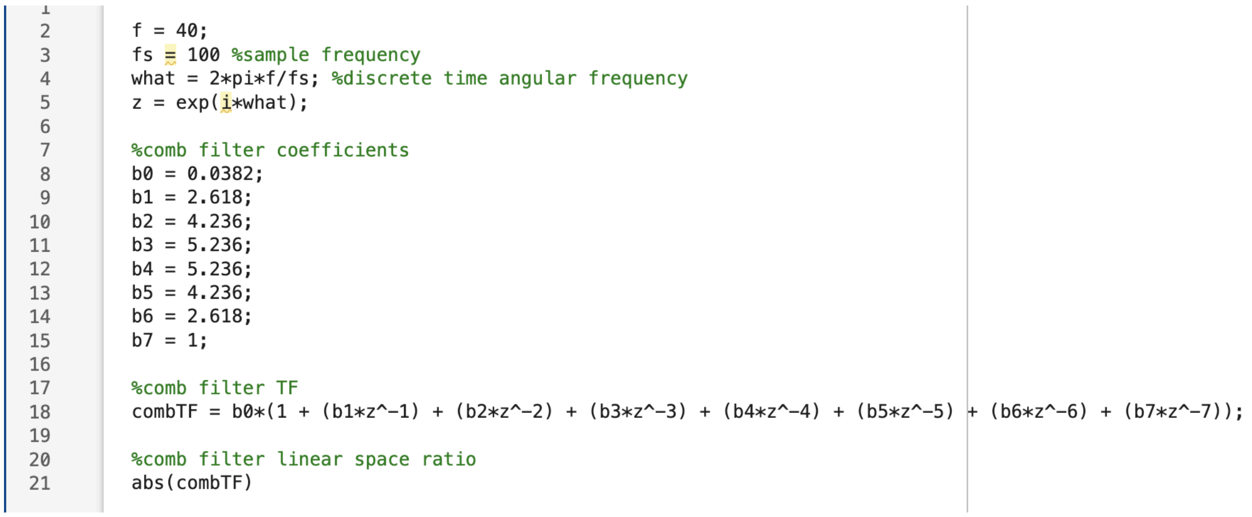
```
%{
PlOTTING
%}

%Time Domain

figure(1)
plot(time, pulseplot,LineWidth=2), set(gca, 'FontSize', 20);
hold on
plot(time, butterW1plot,LineWidth=2), set(gca, 'FontSize', 20);
hold on
legend('Pulse Train','Butterworth Filter Output', 'FontSize',15)
xlabel('Time [ms]','FontSize',25)
ylabel('Voltage [V]','FontSize',25);
title('Pulse Train and Analog Filter','FontSize',30);
axis tight

figure(2)
plot(time, butterW2plot,LineWidth=2), set(gca, 'FontSize', 20);
hold on
plot(time, outputYNplot,LineWidth=2), set(gca, 'FontSize', 20);
hold on
legend('Butterworth Filter Output','Comb Filter Output', 'FontSize',15)
xlabel('Time [ms]','FontSize',25)
ylabel('Voltage [V]','FontSize',25);
title('Analog and Digital Filter','FontSize',30);
axis tight

%Frequency Domain
figure(3)
stem(f_BW, BW), set(gca, 'FontSize', 20);
title('Magnitude Spectrum |Butterworth| vs. Frequency', FontSize=30)
ylabel('|Butterworth|', FontSize=22)
xlabel('frequency [Hz]', FontSize=22)
hold on

figure(4)
stem(f_YN, YN), set(gca, 'FontSize', 20);
title('Magnitude Spectrum |Comb| vs. Frequency', FontSize=30)
ylabel('|Comb|', FontSize=22)
xlabel('frequency [Hz]', FontSize=22)

figure(5)
stem(f_Pulse, PU), set(gca, 'FontSize', 20);
title('Magnitude Spectrum |Pusle Train| vs. Frequency', FontSize=30)
ylabel('|Pulse hann|', FontSize=22)
xlabel('frequency [Hz]', FontSize=22)

```



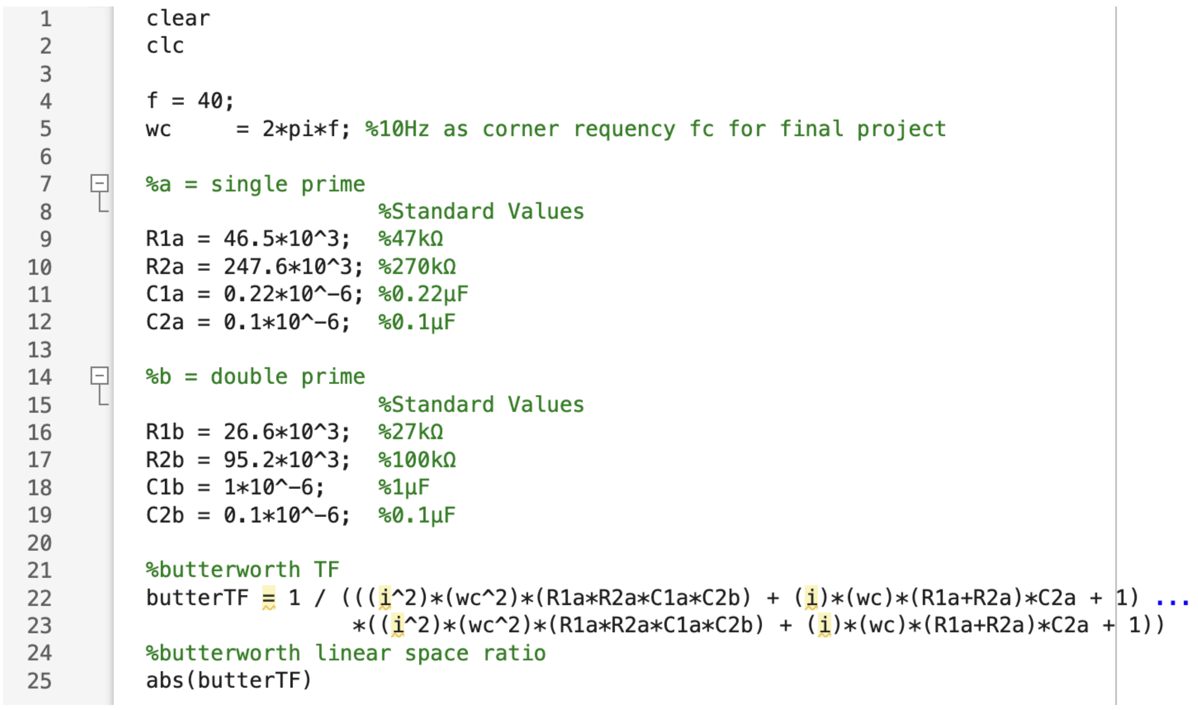
```
%Hanning Window
figure(6)
stem(f_BW_hann, BW_hann), set(gca, 'FontSize', 20);
title('Magnitude Spectrum |Butterworth hann| vs. Frequency', FontSize=30)
ylabel('|Butterworth hann|', FontSize=22)
xlabel('frequency [Hz]', FontSize=22)
hold on

figure(7)
stem(f_YN_hann, YN_hann), set(gca, 'FontSize', 20);
title('Magnitude Spectrum |Comb hann| vs. Frequency', FontSize=30)
ylabel('|Comb hann|', FontSize=22)
xlabel('frequency [Hz]', FontSize=22)

figure(8)
stem(f_Pulse_hann, Pulse_hann), set(gca, 'FontSize', 20);
title('Magnitude Spectrum |Pusle Train Hann| vs. Frequency', FontSize=30)
ylabel('|Pulse hann|', FontSize=22)
xlabel('frequency [Hz]', FontSize=22)
```

```
function [X,f]=fdomain(x,Fs)

% FDOMAIN computes the Fourier coefficients from vector x

% and the corresponding frequencies (two-sided)

% usage: [X,f]=fdomain(x,Fs)

N=length(x);

if mod(N,2)==0

   k=-N/2:N/2-1; % N even

else

   k=-(N-1)/2:(N-1)/2; % N odd

end

T=N/Fs;

f=k/T; %create a vector for two sided frequencies (for the given resolution)

X=fft(x)/N;

X=fftshift(X);
```

These MATLAB files compute the estimated value of the transfer functions of the Butterworth and Comb filters at the various frequencies viewed in the frequency domain representation of the signals.

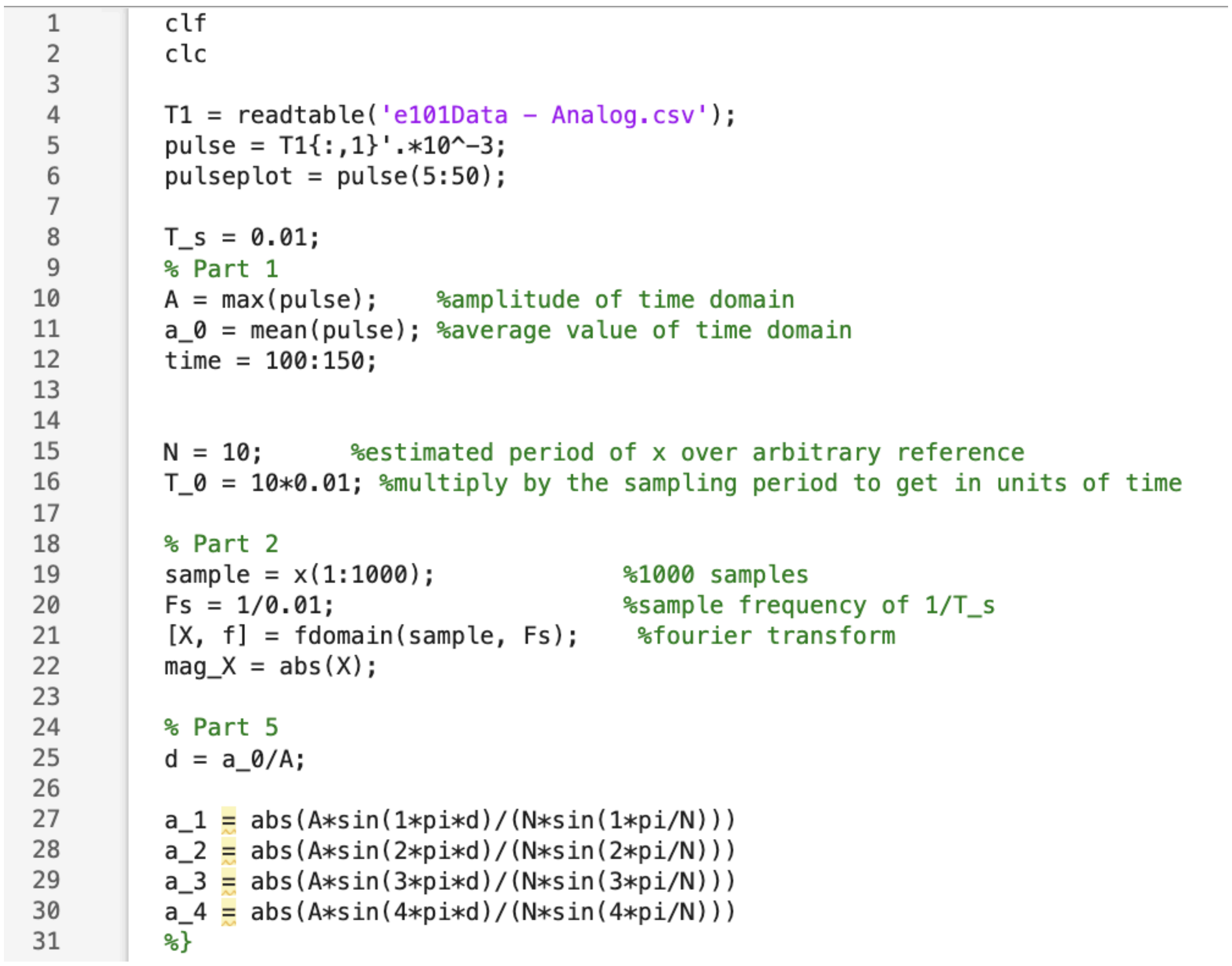


```
f = 40;
fs = 100 %sample frequency
what = 2*pi*f/fs; %discrete time angular frequency
z = exp(i*what);

%comb filter coefficients
b0 = 0.0382;
b1 = 2.618;
b2 = 4.236;
b3 = 5.236;
b4 = 5.236;
b5 = 4.236;
b6 = 2.618;
b7 = 1;

%comb filter TF
combTF = b0*(1 + (b1*z^-1) + (b2*z^-2) + (b3*z^-3) + (b4*z^-4) + (b5*z^-5) + (b6*z^-6) + (b7*z^-7));

%comb filter linear space ratio
abs(combTF)
```



```
clear
clc

f = 40;
wc     = 2*pi*f; %10Hz as corner requency fc for final project

%a = single prime
                  %Standard Values
R1a = 46.5*10^3;  %47kΩ
R2a = 247.6*10^3; %270kΩ
C1a = 0.22*10^-6; %0.22μF
C2a = 0.1*10^-6;  %0.1μF

%b = double prime
                  %Standard Values
R1b = 26.6*10^3;  %27kΩ
R2b = 95.2*10^3;  %100kΩ
C1b = 1*10^-6;    %1μF
C2b = 0.1*10^-6;  %0.1μF

%butterworth TF
butterTF = 1 / (((i^2)*(wc^2)*(R1a*R2a*C1a*C2b) + (i)*(wc)*(R1a+R2a)*C2a + 1) ...
                *((i^2)*(wc^2)*(R1a*R2a*C1a*C2b) + (i)*(wc)*(R1a+R2a)*C2a + 1))
%butterworth linear space ratio
abs(butterTF)
```

This MATLAB code computes the Fourier coefficients of the pulse train taken from the arduino.



```matlab
clf
clc

T1 = readtable('e101Data - Analog.csv');
pulse = T1{:,1}'.*10^-3;
pulseplot = pulse(5:50);

T_s = 0.01;
% Part 1
A = max(pulse);     %amplitude of time domain
a_0 = mean(pulse); %average value of time domain
time = 100:150;


N = 10;       %estimated period of x over arbitrary reference
T_0 = 10*0.01; %multiply by the sampling period to get in units of time

% Part 2
sample = x(1:1000);              %1000 samples
Fs = 1/0.01;                     %sample frequency of 1/T_s
[X, f] = fdomain(sample, Fs);     %fourier transform
mag_X = abs(X);

% Part 5
d = a_0/A;

a_1 = abs(A*sin(1*pi*d)/(N*sin(1*pi/N)))
a_2 = abs(A*sin(2*pi*d)/(N*sin(2*pi/N)))
a_3 = abs(A*sin(3*pi*d)/(N*sin(3*pi/N)))
a_4 = abs(A*sin(4*pi*d)/(N*sin(4*pi/N)))
%}
```