Signal Processing In Theory And In Practice

By Joseph Ashton, SID 27047440

UNIVERSITY OF
LINCOLN

A report for the "Signal Processing and System Identification" module "IIR Filter Design Assignment and Image processing" coursework.

Table of Contents

Table of Figures

Introduction

This report is comprised of 2 major sections the first discusses the design and implementation of a Butterworth low pass filter and the second briefly discusses digital filter techniques such as convolutions for image processing and psudo-random binary sequences for system identification.

Symbols

Below is a list of the symbols used throughout this report accompanied by their common names and definitions where relevant.

Symbol	Meaning
j	Imaginary unit ($j^2 = -1$)
s	Complex frequency variable in Laplace domain
s_k	k-th pole of the filter in the s-domain
z	Complex frequency in Z-domain
ω	Angular frequency (rad/s)
Ω	Digital frequency (rad/sample)
ω_b	Butterworth cutoff frequency (rad/s)
ω_s	Sampling frequency (rad/s)
$H(s)$	Transfer function in the Laplace (analogue) domain
$H(z)$	Digital transfer function (Z domain)
$H(j\omega)$	Frequency response evaluated on the $j\omega$ axis
$\lvert H(j\omega)\rvert^2$	Power gain
$X(s), Y(s)$	Input and output signals in the s-domain
n	Filter order (number of poles)
F_s	Sampling frequency in Hz ($F_s = \omega_s / 2\pi$)
T	Sampling period (s), $T = 1/F_s$

Butterworth Low Pass Filter

Theoretical Background

The s and z variables and their respective domains allow engineers to analyse both linear and oscillatory system dynamics intuitively. The s-domain is well suited for the analysis of analogue systems operating in continuous time where the z-domain suits digital systems in discrete time steps.

$$s = \sigma + j\omega \qquad z = re^{j\Omega}$$

The s variable exists in a Cartesian space with it's real part σ representing exponential growth, or decay if negative, and it's imaginary component $j\omega$ representing oscillations and their frequency. A systems natural modes sum to determine how it will react to any kind of input and can be understood by where it's poles lay in the s-domain.

A system with a positive real component grows exponentially out of control but the more negative it is the faster it will settle down. Likewise large imaginary poles indicate a tendency for the system to oscillate violently whereas a system with small imaginary components will have slow manageable oscillations. In filter design, the precise placement of poles allows for deliberate shaping of the system's frequency response. For example, placing poles closer to the imaginary axis allows narrowband filters with sharp frequency selectivity, while placing them farther left results in smoother, faster-decaying filters.

In contrast the z variable is a polar coordinate where the exponential growth/decay is captured by the radius r and the frequency is represented by the angle Ω. Similarly to the s-domain Poles with low Ω near the real axis correspond to lower frequency behaviour and those further away correspond to higher frequency oscillations. In contrast the rate of decay decreases with distance from the origin with stability is indicated by whether the pole exists inside the unit circle.

This can be seen in Fig 1, Fig 2, Fig 3 and Fig4 below showing various s and z poles with their implied impulse response and magnitude vs frequency response. The code for which can be found in the annex.

Figure 1: s-domain Impulse Response

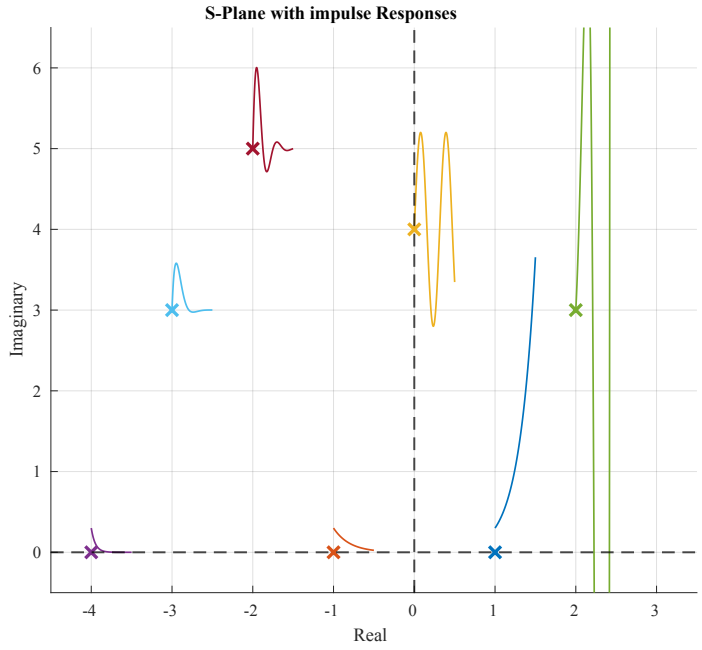


Figure 2: s-domain Frequency Response

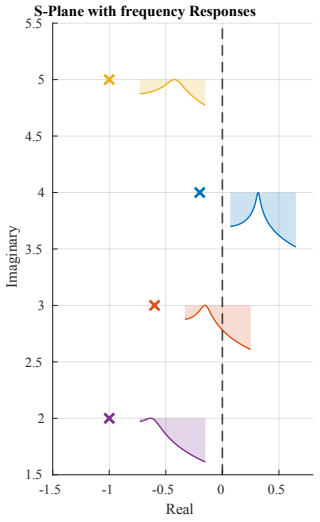


Figure 3: z-domain Impulse Response

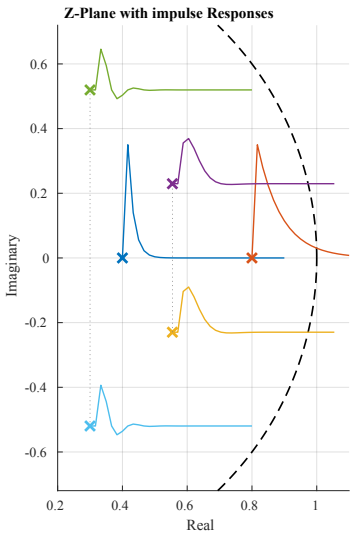
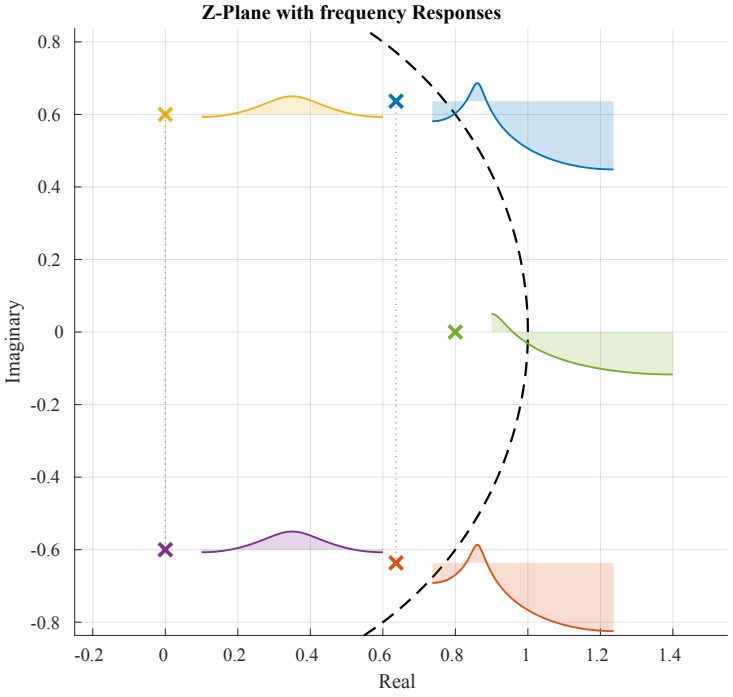


Figure 4: z-domain Frequency Response



Model

Deriving the s-domain Transfer Function

The ideal behaviour of a low pass filter is to reject frequencies (ω) above the cut off (ω_b) while admitting frequencies below it without distortion. The power gain ($|H(j\omega)|^2$), defined as the magnitude of the frequency response ($H(j\omega)$ squared, should ideally resemble a sharp edged brick wall as given by Equation 1.

$$|H(j\omega)|^2 = \begin{cases} \omega \leq \omega_b : 1 \\ \omega > \omega_b : 0 \end{cases}$$

This desirable brick wall response was approximated for simple staged filter amplifiers by Butterworth in 1930, with Equation 2 (left) where n is the number of filter stages. Given the case of a 2 stage filter ($n = 2$) and a cut off frequency of $\omega_b = 5\frac{\text{rad}}{\text{sec}}$ beyond which point signals should be reduced by at least -3 dB, the power gain function becomes Equation 3 (right).

$$|H(j\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_b}\right)^{2n}}$$

$$|H(j\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{5}\right)^{4}}$$

The s-domain transfer function equivalent for Equation 3 can be designed by pole placement as the poles (s_k) of of a Butterworth filter lie evenly spaced across the left hand plane of a circle of radius $r = \omega_b$ according to Equation 4 (left). In the case of a 2 stage filter the poles will be those shown in Equation 5 (right).

$$\text{for } k \text{ in } [0, n-1] : \quad s_k = \omega_b \cdot e^{j\pi\left(\frac{2k+1}{2n}\right)}$$

$$\text{for } n = 2 : \quad \begin{aligned} s_0 &= \omega_b e^{j3\pi/4} \\ s_1 &= \omega_b e^{j5\pi/4} \end{aligned}$$

The s-domain transfer function ($H(s)$) of any filter with Gain A can be represented by Equation 6 (left) by using the product of the poles as the denominator. Thus the prototype filter is given by Equation 7 (right).

$$H(s) = \frac{A{\omega_b}^n}{\prod_{k=0}^{n-1}(s - s_k)}$$

$$H(s) = \frac{{\omega_b}^2}{s^2 + \sqrt{2}\omega_b s + {\omega_b}^2}$$

Converting to a z-domain Transfer Function

An s-domain transfer function can be mapped to the z-domain via the following substitution (left) that maps the infinite analogue frequency axis $j\omega$ of the s-domain onto the unit circle $z = e^{j\Omega}$ of digital frequency in the z-domain. However, having to fit the entire -ve half of the s-domain into the unit circle warps the frequency scale especially effecting higher frequency information. This warping can be reduced by pre-warping the frequencies with the following substitution (right).

$s = \frac{2}{T} \cdot 1 - \frac{z^{-1}}{1 + z^{-1}}$	$\omega' = \frac{2}{T} \cdot \tan\left(\frac{\omega T}{2}\right)$

Performing these substitutions to on Equation 7 found in the the previous section and simplifying gives the following z-domain transfer function. For the full method with a detailed walk through see appendix for a full step by step walk through of the method.

pre-warp

$$H(s) = \frac{{\omega_b}^2}{s^2 + \sqrt{2}\omega_b s + {\omega_b}^2} \rightarrow \frac{{\omega'_b}^2}{s^2 + \sqrt{2}\omega'_b s + {\omega'_b}^2}$$

bilinear transform

$$H(z) = \frac{{\omega'_b}^2}{\left(\frac{2}{T} \cdot \frac{1-z^{-1}}{1+z^{-1}}\right)^2 + \sqrt{2}\omega'_b \cdot \left(\frac{2}{T} \cdot \frac{1-z^{-1}}{1+z^{-1}}\right) + {\omega'_b}^2}$$

cancel out complex fractions

$$H(z) = \frac{{\omega'_b}^2}{\left(\frac{2}{T}\right)^2 \frac{(1-z^{-1})^2}{(1+z^{-1})^2} + \sqrt{2}\omega'_b \cdot \left(\frac{2}{T}\right) \cdot \frac{(1-z^{-1})}{(1+z^{-1})} + \omega_{b'^2}} \times \frac{(1+z^{-1})^2}{(1+z^{-1})^2}$$

expand polynomials

$$H(z) = \frac{{\omega'_b}^2(1 + 2z^{-1} + z^{-2})}{\left(\frac{2}{T}\right)^2 + \sqrt{2}\omega'_b \cdot \frac{2}{T} + {\omega'_b}^2 + \left[-2\left(\frac{2}{T}\right)^2 + 2{\omega'_b}^2\right]z^{-1} + \left[\left(\frac{2}{T}\right)^2 - \sqrt{2}\omega'_b \cdot \frac{2}{T} + {\omega'_b}^2\right]z^{-2}}$$

substitute numerical values

$$H(z) \approx \frac{0.0675 + 0.1350z^{-1} + 0.0675z^{-2}}{1 - 1.2765z^{-1} + 0.4129z^{-2}}$$

The digital systems represented by z-domain transfer functions operate in discrete time samples unlike a continuous s-domain representation. Given that the z variable is $z = r \cdot e^{i\omega}$ then z^{-n} can be expressed as:

$$z^{-n} = (r \cdot e^{i\omega})^{-n} = e^{-i\omega n} \times r^{-n}$$

with $e^{-i\omega n}$ shifting the phase and r^n scaling the amplitude so that the coefficients of each z^{-n} term represent the influence of a sample from n sampling periods ago.
The difference equation uses this property to give a recursive relationship that predicts the system output $y[n]$ for a given set of input samples $x[n]$. This can be found by collecting the z-domain transfer function into z^{-n} terms.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} \dots}{1 - b_1 z^{-1} + b_2 z^{-2} \dots}$$
$$\downarrow \qquad \times X(z)$$
$$Y(z)(1 - b_1 z^{-1} + b_2 z^{-2}) = X(z)(a_0 + a_1 z^{-1} + a_2 z^{-2})$$

These coefficients can be used directly in difference equation based on the mapping:

$$\begin{aligned} z^{-1}Y(z) &\to y[n-1] \qquad & z^{-1}X(z) &\to x[n-1] \\ z^{-2}Y(z) &\to y[n-2] \qquad & z^{-2}X(z) &\to x[n-2] \\ &\dots \qquad & &\dots \end{aligned}$$

$$y[n] = (a_0\, x[n] + a_1\, x[n-1] + a_2\, x[n-2] \dots) - (b_1\, y[n-1] + b_2\, y[n-2] \dots)$$

For the case of the butterworth z-domain filter found previously this is:

$$H(z) = \frac{0.0675 + 0.1350 z^{-1} + 0.0675 z^{-2}}{1 - 1.2765 z^{-1} + 0.4129 z^{-2}}$$

$$y[n] = 0.0675x[n] + 0.1350x[n-1] + 0.0675x[n-2] + 1.2765y[n-1] - 0.4129y[n-2]$$

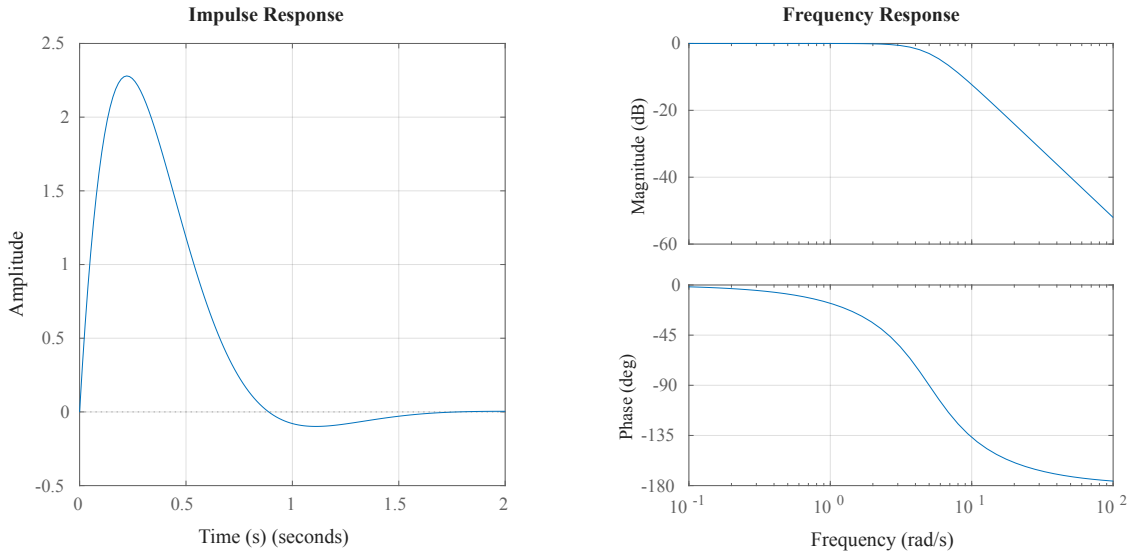
Simulation

Analogue Filter Response

The analogue Butterworth filter is simulated with MATLABs built in `impulse` function and it's frequency response is projected in terms of magnitude and phase using the `bode` function. This script can be downloaded along with the rest of the code used in this report from the github repo at [git@github.com:jasht1/Uni-Projects/Signal Processing/Assesments/CourseWork/]

```
%% Params
omega_b = 5;  % Cutoff frequency (rad/s)

%% Define transfer function H(s)
num_s = [omega_b^2];
den_s = [1, sqrt(2)*omega_b, omega_b^2];

H_s = tf(num_s, den_s);  % Analogue transfer function in Laplace domain

%% Plot results
figure('NumberTitle', 'off');
tiledlayout(1, 2);  % 1 row, 2 columns
sgtitle('Analogue Butterworth Low Pass Filter Response');

%%% Impulse response
%nexttile;
impulse(H_s);
title('Impulse Response');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

%%% Frequency response
%nexttile;
bode(H_s);
title('Frequency Response');
grid on;
```

Figure 5: Analogue Butterworth filter Impulse & Frequency Response

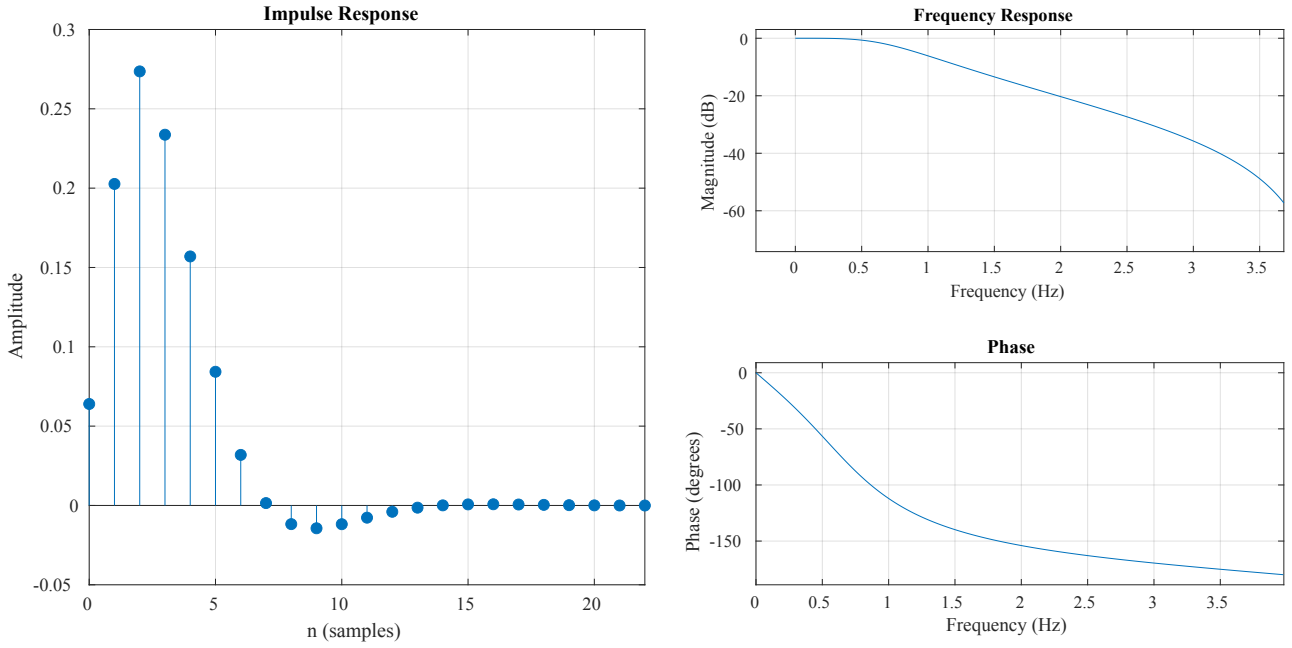


Digital Filter Response

The `bilinear` function is used to convert a the symbolic s-domain transfer function object into it's z-domain form. The `impz` function is used to simulate an impulse response and it's frequency response is projected in terms of magnitude and phase using the `freqz` function. Download this script from the git repo.

```
%% params
omega_b = 5;  % Cutoff frequency (rad/s)
omega_s = 10 * omega_b;  % Sampling frequency (rad/s)
Fs = omega_s / (2*pi);  % Sampling frequency (Hz)

%% Define transfer function H(s)
num_s = omega_b^2;
den_s = [1, sqrt(2)*omega_b, omega_b^2];

H_s = tf(num_s, den_s);  % Analogue transfer function in Laplace domain

%% Bilinear transform to get H(z)
[num_z, den_z] = bilinear(num_s, den_s, Fs);

%% Plot results
figure('NumberTitle', 'off');
tiledlayout(1, 2);  % 1 row, 2 columns
sgtitle('Digital Butterworth Low Pass Filter');

%%% Impulse response
%nexttile;
impz(num_z, den_z);
title('Impulse Response');
xlabel('n (samples)');
ylabel('Amplitude');
grid on;

%%% Frequency response
%nexttile;
figure()
freqz(num_z, den_z, 1024, Fs);
title('Frequency Response');
```

Figure 6: Digital Butterworth filter Impulse & Frequency Response

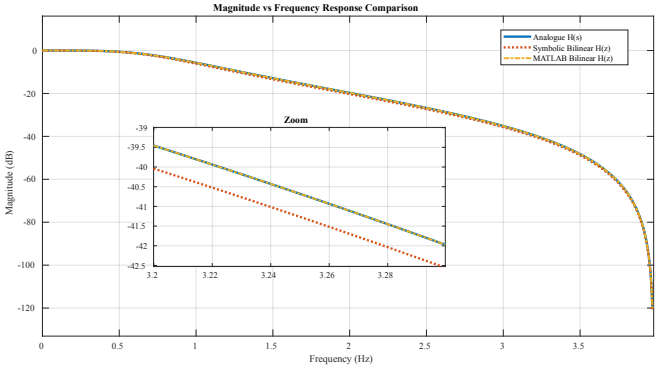


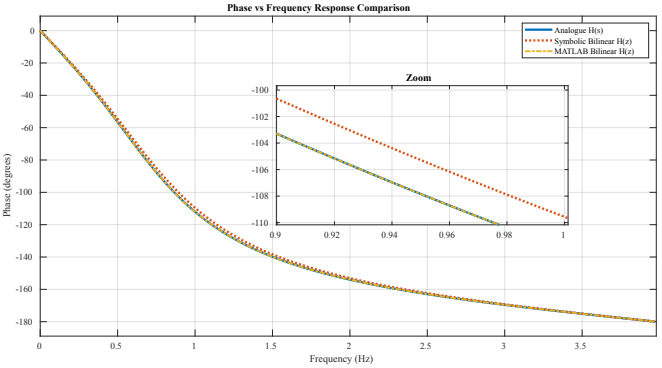
Analogue vs Digital Frequency Response

Note that the previous two figures despite being laid out in a similar format are not directly comparable given the default x units of `bode` are rad/s compared with `freqz` in Hz. Similarly while samples and seconds are analogous between analogue and digital systems it is important to avoid their conflation. The below script, available here, plots the projected frequency response of the analogue and digital models onto the same axis. This shows an almost perfect match with the minor difference in the "symbolic" method likely due to truncation in the `sym2poly` method when the higher resolution symbolic variables are converted into vector floats.

```
%% params
omega_b = 5;  % Cutoff frequency (rad/s)
omega_s = 10 * omega_b;  % Sampling frequency (rad/s)
Fs = omega_s / (2*pi);  % Sampling frequency (Hz)
T = 1/Fs;  % sampling period (s)

%% Define transfer function H(s)
num_s = omega_b^2;
den_s = [1, sqrt(2)*omega_b, omega_b^2];

H_s = tf(num_s, den_s);  % Analogue transfer function in Laplace domain

%% Bilinear transform to get H(z)

%%% Auto find
[num_z, den_z] = bilinear(num_s, den_s, Fs);

%%% Manual~ish find
syms s z % symbolic for subs and simplifications
omega_b_pw = (2/T) * tan(omega_b * T / 2); % frequency pre warp
bilinear_identity = 2 / T * (1 - z^-1) / (1 + z^-1);
Hs = omega_b_pw^2 / (s^2 + sqrt(2)*omega_b_pw*s + omega_b_pw^2); % Analogue TF but symbolic
Hz = simplify(subs(Hs, s, bilinear_identity));
Hz = collect(Hz, z^-1);
[N_sym, D_sym] = numden(Hz); % symbolic to numeric
N_sym = expand(N_sym);
D_sym = expand(D_sym);
num_z_sym = sym2poly(N_sym);
den_z_sym = sym2poly(D_sym);
num_z_sym = num_z_sym / den_z_sym(1); % see if it reduces
den_z_sym = den_z_sym / den_z_sym(1);

%% values from my hand derivation
num_z_hand_deriv = [0.0675, 0.1350, 0.0675]; % Y(z)
den_z_hand_deriv = [1, -1.2765, 0.4129]; % X(z)

%% Plot and compare

[H_s_mag, w] = bode(H_s, 2*pi*linspace(0, Fs/2, 1024));
H_s_mag = squeeze(H_s_mag);  % Remove singleton dim
f = linspace(0, Fs/2, numel(H_s_mag));
hold on;
freqz(num_z_sym, den_z_sym, 512, 1/T);
freqz(num_z, den_z, 512, 1/T);
freqz(num_z_hand_deriv, den_z_hand_deriv, 512, 1/T);  % Normalized symbolic

legend('Analog H(s)', 'MATLAB Bilinear', 'Symbolic (full)', 'Manual derivation');
title('Frequency Response Comparison');
grid on;
```

Figure 7: Analogue vs Digital Frequency Response


Magnitude vs Frequency Response Comparison
Analogue H(s)
Symbolic Bilinear H(z)
MATLAB Bilinear H(z)
Zoom
Magnitude (dB)
Frequency (Hz)


Phase vs Frequency Response Comparison
Analogue H(s)
Symbolic Bilinear H(z)
MATLAB Bilinear H(z)
Zoom
Phase (degrees)
Frequency (Hz)

Implementation

To test the difference equation derived in the previous section based on the Butterworth filter it was used onboard an Arduino Nano to filter an incoming analogue signal with high noise content.

The firmware uses two synchronous functions. The first generated the noisy signal using a MCP4725 digital to analogue converter (DAC). The signal being a 2 Hz sinusoid ranging from $0.25 - 0.75\text{V}$ disturbed by random noise of $\pm 0.2\text{V}$.

```
void generateSignal() {
  static float t = 0.0;

  // 2Hz sine wave
  // float sineWave = 0.5 * (sin(2 * PI * signalFreq * t) + 2.5); // 1.5 to 3.5 as can't do -ve V
  float sineWave = 0.25 * (sin(2 * PI * signalFreq * t) + 1.0); // 0.25 to 0.75 as clips bad

  // Add high-frequency noise
  float noise = noiseAmplitude * ((float)random(-1000, 1000) / 1000.0);

  float signal = sineWave + noise;
  signal = constrain(signal, 0.0, 5.0); // Clamp between 0 and 5

  // Write to DAC (12-bit resolution: 0-4095)
  uint16_t dacValue = (uint16_t)(signal * 4095);
  dac.setVoltage(dacValue, false);

  t += dt;
}
```

This noisy signal is returned to the Nano at pin `A0` where the second function sampled it at $\approx 8\text{Hz}$ and used the difference equation to filter the incoming data. The sampling rate was based on a the cut off frequency $\omega_b = 5\frac{\text{rad}}{\text{sec}}$ times 10.

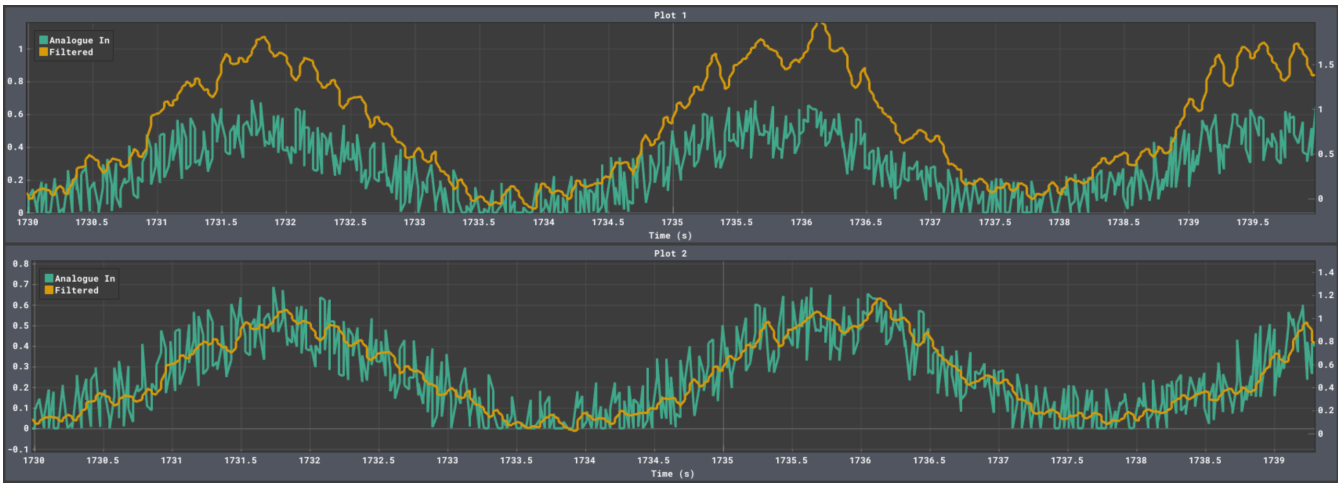
```
void filterSignal() {
  int rawInput = analogRead(analogInputPin); // Read input signal from A0
  float vin = (float)rawInput / 1023.0; // Normalize to 0-1

  // Shift old samples
  x[2] = x[1];
  x[1] = x[0];
  x[0] = vin;

  y[2] = y[1];
  y[1] = y[0];

  // Apply difference equation
  y[0] = 0.0675 * x[0] + 0.1350 * x[1] + 0.0675 * x[2]
         + 1.2765 * y[1] - 0.4129 * y[2];

  // Print for monitoring
  Serial.print(vin, 4);
  // Serial.print(", "); // the Arduion IDE serial plotter likes commas
  Serial.print(" "); // However, my Serial plotter dosen't like commas
  Serial.print(y[0], 4);
  Serial.println();
}
```

The main body of the script can be found here on the git repo allong with it's platformIO `.ini` build config and can be read in the snippet below.

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>

Adafruit_MCP4725 dac;

const int analogInputPin = A0;

// Filter state
float x[3] = {0.0, 0.0, 0.0}; // Input samples x[n], x[n-1], x[n-2]
float y[3] = {0.0, 0.0, 0.0}; // Output samples y[n], y[n-1], y[n-2]

// Sampling parameters
const float fs = 500.0; // sample rate (Hz)
const float dt = 1.0 / fs;

// Signal generation parameters
const float signalFreq = 2.0; // 2Hz sine wave
const float noiseAmplitude = 0.2; // Noise amplitude (relative)

unsigned long lastSampleTime = 0;
const unsigned long sampleIntervalMicros = 7957747.1637 / fs; // Microseconds between samples

void generateSignal() {
    // See Def "Noisy Signal Function" Above
}

void filterSignal() {
    // See Def "Signal Filtering Function" Above
}

void setup() {
  Serial.begin(9600);
  Wire.begin();
  dac.begin(0x60); // MCP4725 I2C address

  analogReference(DEFAULT);
}

void loop() {
  unsigned long now = micros();
  if (now - lastSampleTime >= sampleIntervalMicros) {
    lastSampleTime = now;

    generateSignal();
    filterSignal();
  }
}
```

The traces from which can be seen in the following 2 plots. The first has both traces on the main axis (left) for consistency, the second has the "Filtered" trace on axis 2 (right) scaled and translated to match "Analogue In" to more easily compare noise.

Figure 8: Arduino Butterworth Filter Traces



Serial Plots where generated using nathandunk's BetterSerialPlotter.

Digital Signal Processing Principles

The use of Convolutions

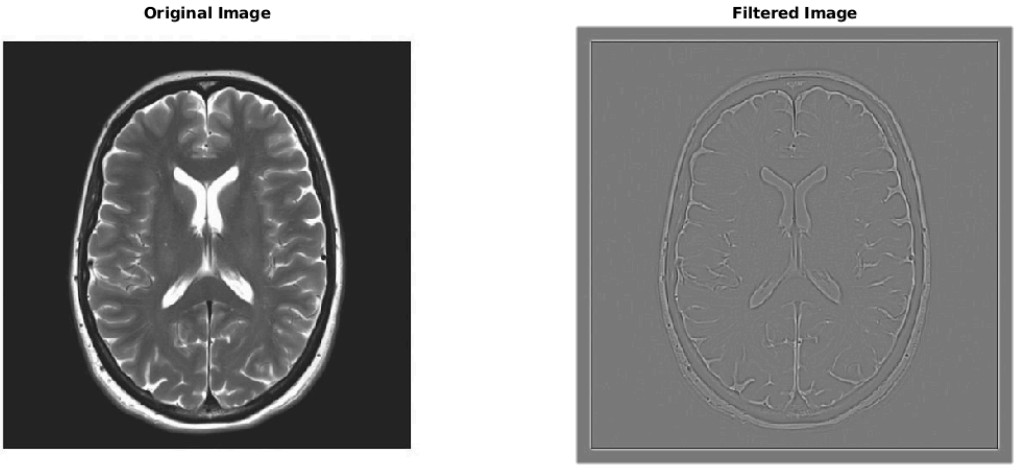
A convolution is a fundamental mathematical operation utilised extensively in digital systems especially in image processing. Abstractly it can refer to any operation that combines 2 functions or sets to produce a third, but in digital signal processing applications the first function/set is typically some kind of dataset while the second is often some kind of filter to be applied, in these cases the second "filter" function/set is referred to as a **kernel** often referred to as f & g respectively. The operation flips one of the inputs $g^{\circlearrowleft}$ then from the start of f and $g^{\circlearrowleft}$ the output is the sum of the element-wise product of all opposing members of f and $g^{\circlearrowleft}$, incrementing the overlap by one each time until the final element of each array meet.

Sharpening

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 24 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

This kernel highlights pixels by how they differ from pixels in their immediate vicinity, when applied to the brain scan this makes the edges of bright features show up as light lines with darker outlines or the inverse for darker features.

Figure 9: Brain Filter Side By Side



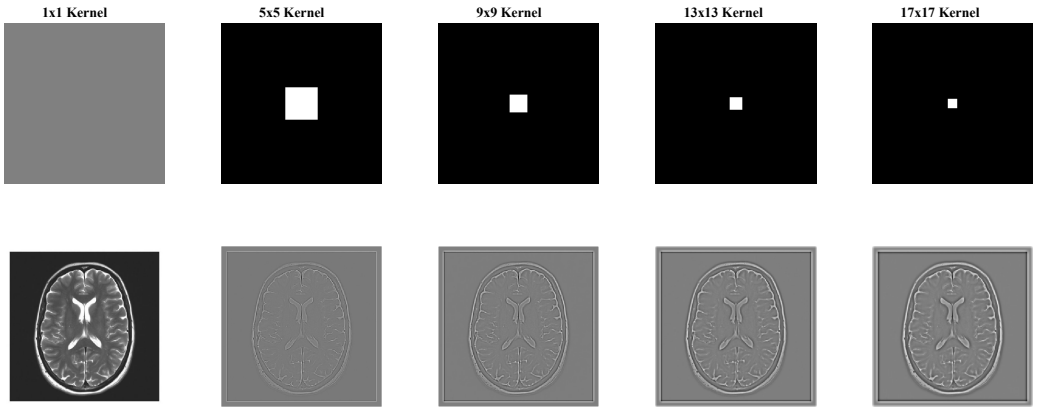
This is done by considering a region of up to 2 pixels in every direction and subtracting the value of each neighbouring pixel from the one at the centre of this region, to compensate the the pixel at the centre is multiplied in magnitude by the number of neighbouring pixels compared against.
This can be achived by defining the kernel shown above in MATLAB and using teh `conv2` function to perfom a 2D convolution.

```
original_image = imread('Brain_BW512x512.png');
```

```
kernel = [
  -1 -1 -1 -1 -1;
  -1 -1 -1 -1 -1;
  -1 -1 24 -1 -1;
  -1 -1 -1 -1 -1;
  -1 -1 -1 -1 -1
  ];

filtered_image = conv2(double(original_image), kernel);
filtered_image = uint8(mat2gray(filtered_image) * 255);
```

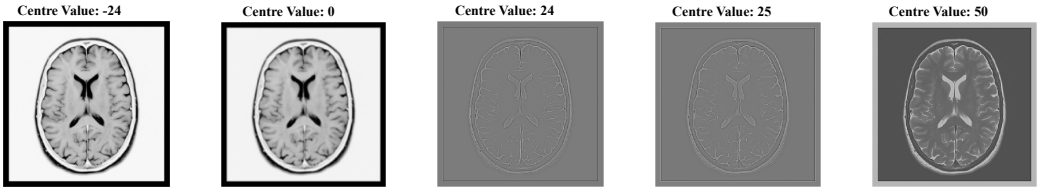
The filter could be adjusted to only pick up larger features by increasing the size of both the centre and the surroundings considered, or discriminate against a larger region by increasing just the surroundings like shown below.

Figure 10: Kernal Size Comparison



By changing the centre value we can essentially add this on top of the original image (or an inverted version) with the effect of increasing it's sharpness.

Figure 11: Kernal Centre Value Comparison



The details of the scripts to generate the two figures above can be found in the annex.

Edge Detection

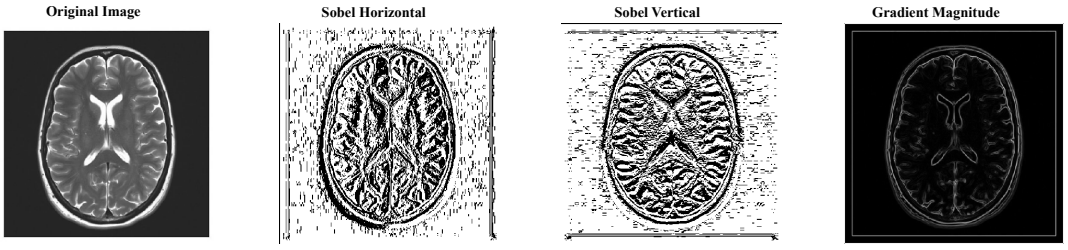
In applications such as computer vision systems it is often necessary to extract details like edges, the convolution of images with specialised kernels is a computationally effective way of extracting information like this.

Adding the effective brightness of pixels to one side and subtracting the brightness of those on the other will show the effective gradient along that axis like taking a local derivative. This is the principle behind the Sobel filter, where the magnitude of two axial gradient detecting kernels are summed to the effect of a rudimentary edge detection filter.

Horisontal Sobel Filter	Vertical Sobel Filter	Edge Detect
$G_x = \begin{bmatrix} -1, & 0, & 1 \\ -2, & 0, & 2 \\ -1, & 0, & 1 \end{bmatrix}$	$G_y = \begin{bmatrix} -1, & -2, & -1 \\ 0, & 0, & 0 \\ 1, & 2, & 1 \end{bmatrix}$	$E = \sqrt{{G_x}^2 + {G_y}^2}$

Applied to the black and white brain image considered in the previous section this method can clearly outline the features of the brain.

Figure 12: 3x3 Sobel Edge Detection



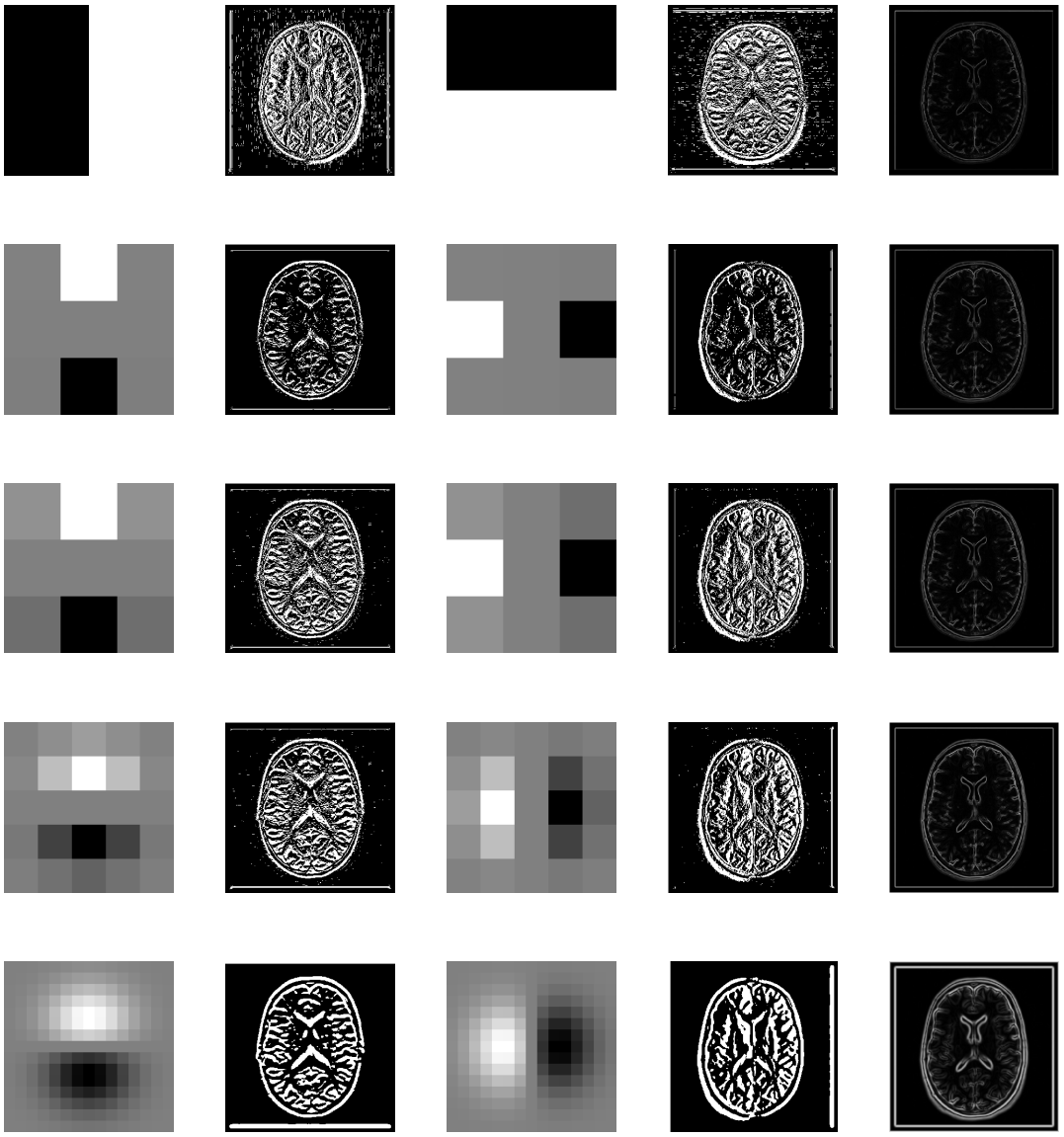
This was achieved in MATLAB utilising the `conv2` function.

```
original_image = double(imread('Brain_BW512x512.png'));

% Sobel kernels
Gx = [
  -1 0 1;
  -2 0 2;
  -1 0 1
  ];

Gy = [
  -1 -2 -1;
  0  0  0;
  1  2  1
  ];

% Apply convolution
filtered_x = conv2(original_image, Gx);
filtered_y = conv2(original_image, Gy);

% Compute gradient magnitude
sobel_magnitude = sqrt(filtered_x.^2 + filtered_y.^2);
sobel_magnitude = mat2gray(sobel_magnitude);

% Plot results
```

```
figure('Name', 'Sobel Filter Results');
tiledlayout(1,4);
nexttile;
imshow(uint8(original_image));
title('Original Image');
nexttile;
imshow(uint8(mat2gray(filtered_x)) * 255);
title('Sobel Horizontal');
nexttile;
imshow(uint8(mat2gray(filtered_y)) * 255);
title('Sobel Vertical');
nexttile;
imshow(uint8(sobel_magnitude * 255));
title('Gradient Magnitude');
```

It's worth noting the impact of the size of the sobel filter, the reader will notice the filters picking up noise in the background in figure ! . As far as the filter is concerned this noise is just a very small feature so by increasing the size of the kernel, effectivley blurring it's input, the filter can be tuned to only detect larger features.

Figure 13: Sobel Kernel Size Comparison



Details on the script used to generate the figure above can be found in the annex.

Pseudo Random Binary Sequences

It is often the case that a digital system must be determined to match an existing black box system where an engineer can only test inputs and observe outputs. A predictable system can be expected to fit a standard form such as:

$$G(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}}{1 + b_1 z^{-1} + b_2 z^{-2} - \cdots + b_n z^{-n}}$$

And thus the problem is reduced to tuning the values coefficients to match the observed behaviour of the system. The engineer must be able to determine and distinguish the systems tendency to react to the input vs itself, measured by its cross-correlation and its auto-correlation respectively. Cross-correlation correlates the system output with a time shift of the input data a measure of it's reactivity, whereas auto-correlation measures the system outputs similarity to a time shifted version of it's self, in other words, the systems own resonance. It should then be considered what input data is the best test to get meaningful output data to compare, contrast and thus improve the modelling. The input should at once be random and chaotic to be able to observe the system responding to the greatest verity of stimuli but also be periodic so that the system response to the input can be seen multiple times in different contexts / starting points. These are the properties of pseudo random binary sequences (PRBS)s along with being computationally trivial to produce and tune to fit a given need. This makes PRBS optimal as input data to fit/train a model of an unknown system, as it is has near 0 auto-correlation of it's own apart from at 0 time shift with it being periodic making it easy to account for and contrast with genuine auto-correlation of the system.

Annex

Pole Dynamics Visuals

The pole dynamic visuals seen in the Theoretical Background section where generated using the following function available here on the Git repo along with the `.svg` figures produced here.

Full z-domain transfer function derivation walk through

Beginning with ../Answers/Question 1/Q1i - s domain TF > ^Equation7-PTFilterTF for the 2 stage Butterworth low pass filter s-domain transfer function:

$$H(s) = \frac{\omega_b^2}{s^2 + \sqrt{2}\omega_b s + \omega_b^2}$$

The frequency pre-warping substitution may be applied for a minor benefit in accuracy. The sampling period T in seconds is based on the Butterworth cut-off frequency ω_b such that $\omega_s = 10 \times \omega_b$. The cutoff frequency $\omega_b = 5\ \frac{\text{rad}}{\text{s}}$ gives the sampling frequency $\omega_s = 50\ \text{rad/s}$ or $F_s = \frac{\omega_s}{2\pi} = 25/\pi\ \text{Hz}$. The sampling period being the reciprocal $T = \frac{1}{F_s} = \pi/25\ \text{s}$.

$$\omega_b' = \frac{2}{T} \cdot \tan\left(\frac{\omega_b T}{2}\right) \quad \text{Where} \quad \begin{matrix} \omega_b = 5 \\ T = \pi/25 \end{matrix} \quad \text{such that} \quad \omega_b' = \frac{50}{\pi} \cdot \tan\left(\frac{\pi}{10}\right) = 5.17125757634\,.$$

If deemed necessary this instead makes the starting point:

$$H(s) = \frac{{\omega_b'}^2}{s^2 + \sqrt{2}\omega_b' s + {\omega_b'}^2}$$

In either case the s variable must be mapped to it's corresponding z variable using the bilinear transform:

$$s = \frac{2}{T} \cdot \frac{1 - z^{-1}}{1 + z^{-1}}$$

Doing so gives a messy intermediary with many complex fractions but by considering the the powers factor-wise these can be eliminated by multiplying the numerator and denominator by $(1 + z^{-1})^2$:

$$H(z) = \frac{{\omega_b'}^2}{\left(\frac{2}{T} \cdot \frac{1-z^{-1}}{1+z^{-1}}\right)^2 + \sqrt{2}\omega_b' \cdot \left(\frac{2}{T} \cdot \frac{1-z^{-1}}{1+z^{-1}}\right) + {\omega_b'}^2}$$

$$= \frac{{\omega_b'}^2}{\left(\frac{2}{T}\right)^2 \frac{(1-z^{-1})^2}{(1+z^{-1})^2} + \sqrt{2}\omega_b' \cdot \left(\frac{2}{T}\right) \cdot \frac{(1-z^{-1})}{(1+z^{-1})} + \omega_{b'^2}} \times \frac{(1+z^{-1})^2}{(1+z^{-1})^2}$$

$$= \frac{{\omega_b'}^2 (1+z^{-1})^2}{\left(\frac{2}{T}\right)^2 (1-z^{-1})^2 + \sqrt{2}\omega_b' \frac{2}{T}(1-z^{-1})(1+z^{-1}) + {\omega_b'}^2 (1+z^{-1})^2}$$

Expanding the remaining polynomials:

$$(1+z^{-1})^2 = 1 + 2z^{-1} + z^{-2}$$

$$\frac{2}{T}^2 (1-z^{-1})^2 = \frac{2}{T}^2 (1 - 2z^{-1} + z^{-2})$$

$$\sqrt{2}\omega_b' \frac{2}{T}(1-z^{-1})(1+z^{-1}) = \sqrt{2}\omega_b' \frac{2}{T}(1-z^{-2})$$

$$\omega_b'^2 (1+z^{-1})^2 = \omega_b'^2 (1 + 2z^{-1} + z^{-2})$$

Substituting back in at this point gives the most complete and accurate form:

$$H(z) = \frac{\omega_b'^2(1 + 2z^{-1} + z^{-2})}{\left(\frac{2}{T}\right)^2 + \sqrt{2}\omega_b' \cdot \frac{2}{T} + \omega_b'^2 + \left[-2\left(\frac{2}{T}\right)^2 + 2\omega_b'^2\right]z^{-1} + \left[\left(\frac{2}{T}\right)^2 - \sqrt{2}\omega_b' \cdot \frac{2}{T} + \omega_b'^2\right]z^{-2}}$$

Considering the numerical values of variables:

$T = \pi/25$	≈ 0.125663706
$\omega_b' = 2/T \cdot \tan(5T/2)$	≈ 5.171257576
$2/T = 50/\pi$	≈ 15.91549431
$2/T \cdot \omega_b'$	≈ 82.30312053
$\omega_b'^2$	≈ 26.74190492

The numerator and denominator can be approximated by sections as:

numerator

$\omega_b'^2(1 + 2z^{-1} + z^{-2}) \qquad \approx 26.74789 + 53.49578z^{-1} + 26.74789z^{-2}$

denominator

$$\left(\frac{2}{T}\right)^2 + \sqrt{2}\omega_b' \cdot \frac{2}{T} + \omega_b'^2 \approx 253.30296 + 1.4142 \cdot 82.34737 + 26.74789 \qquad \approx 395.9572$$

$$-2\left(\frac{2}{T}\right)^2 + 2\omega_b'^2 \approx -2 \cdot 253.30296 + 2 \cdot 26.74789 \qquad \approx -505.5581$$

$$\left(\frac{2}{T}\right)^2 - \sqrt{2}\omega_b' \cdot \frac{2}{T} + \omega_b'^2 \approx 253.30296 - 116.5343 + 26.74789 \qquad \approx 163.5165$$

Substituting these values back into the z-domain function, it can be slightly visually simplified by dividing by $\omega_b'^2$ and then put into the standard from by cancelling out the remaining constant coefficient in the denominator.

$$H(z) = \frac{26.74789 + 53.49578z^{-1} + 26.74789z^{-2}}{395.9572 - 505.5581z^{-1} + 163.5165z^{-2}} \qquad \div \frac{26.74190492}{26.74190492}$$

$$H(z) = \frac{1 + 2z^{-1} + z^{-2}}{14.8065 - 18.9021z^{-1} + 6.1138z^{-2}} \qquad \div \frac{14.8065}{14.8065}$$

$$H(z) = \frac{0.0675 + 0.1350z^{-1} + 0.0675z^{-2}}{1 - 1.2765z^{-1} + 0.4129z^{-2}}$$

This is within rounding error of the answer given by MATLAB using the symbolic math Toolbox.

$$\frac{211871198900911234191 5\,(z+1)^2}{3140913774204324806390 9\,z^2 - 3590003204074556614268 9\,z + 1296574225473876744644 1}$$

Sharpening Kernel Visuals

The kernel size and centre value visuals seen in the Sharpening section where generated using the functions available here and here on the Git repo along with the `.svg` figures produced here.

Edge Detect Visuals

The kernel size comparison visuals seen in the Edge Detection section where generated using the function available here on the Git repo along with the `.svg` figures produced here.