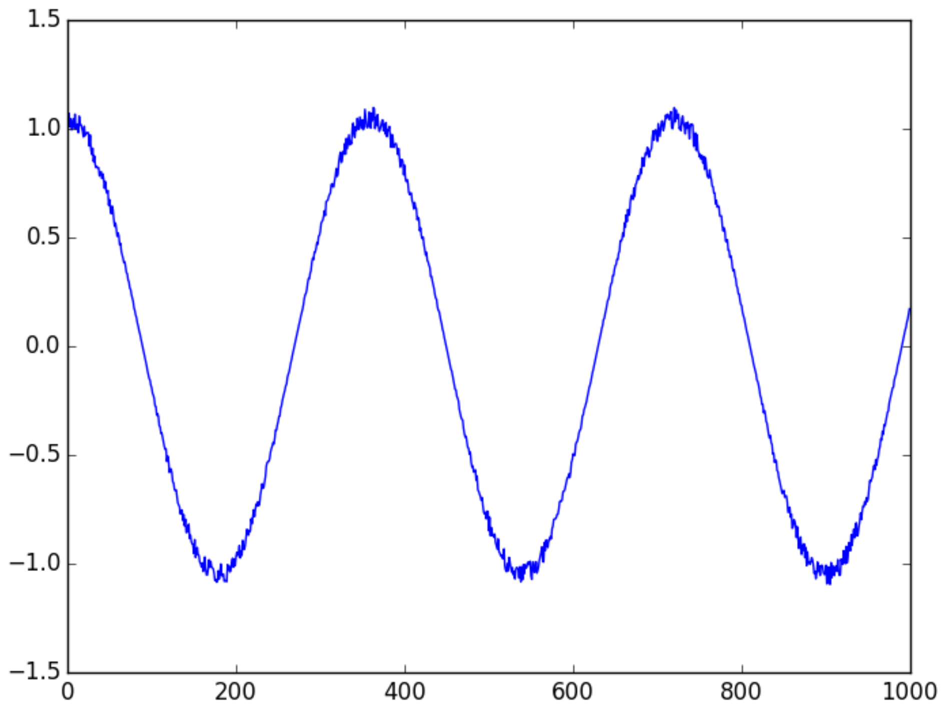
Final
H. Altuğ Yıldırım
090100252

The project that this report covers is a butterworth filter in time domain applied to a noisy signal. First of all we need specifications for our filter and this will be our bandpass, bandstop frequencies and gains. From there we pre-warp our frequencies and then write our gains in decibel form. There we can obtain the order of the butterworth filter. After the order is known we can construct the filter. First we should find the poles and zeros of the transfer function. After that we should use the bilinear transform to transform from s to z, projecting it to complex plane. After that now we know the poles, zeros and gain in form of z, then we can construct the polynomial transfer function and than can obtain the a and b of the difference equation(This part I used a ready-to-use function from scipy library).

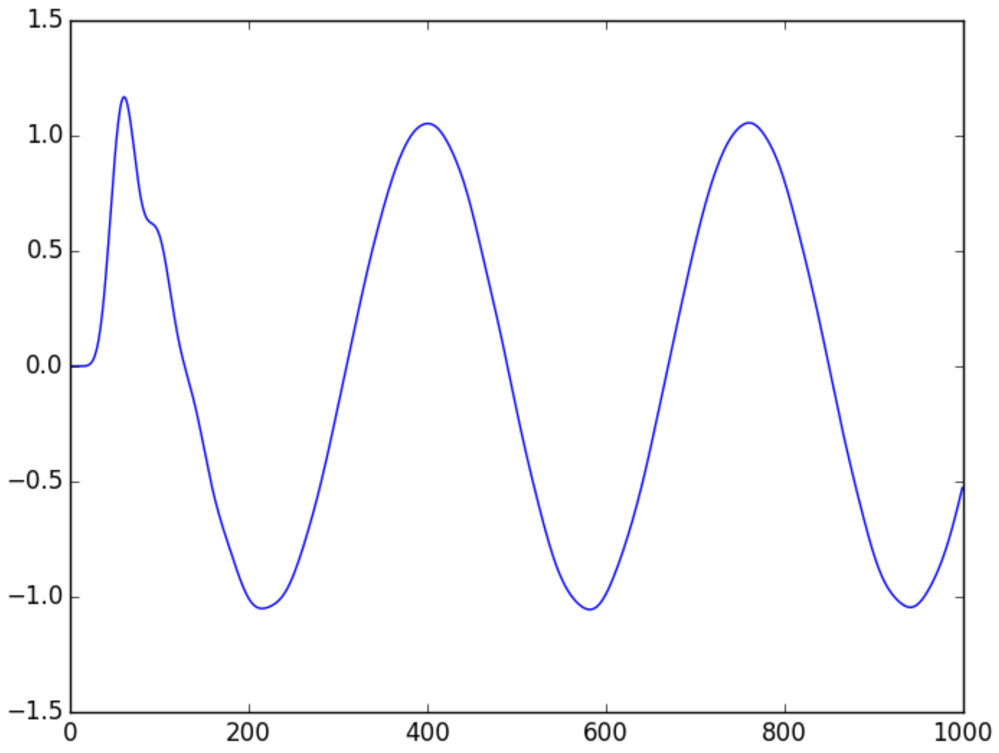
After that I generate a cosine signal with a random noise. And now I know the a and b and signal now I can construct my difference equation which resembles my filter.

$$\begin{aligned} y(n) &= b_0\,x(n) + b_1\,x(n-1) + \cdots + b_M\,x(n-M) \\ &\qquad - a_1\,y(n-1) - \cdots - a_N\,y(n-N) \\ &= \sum_{i=0}^{M} b_i\,x(n-i) - \sum_{j=1}^{N} a_j\,y(n-j) \end{aligned}$$

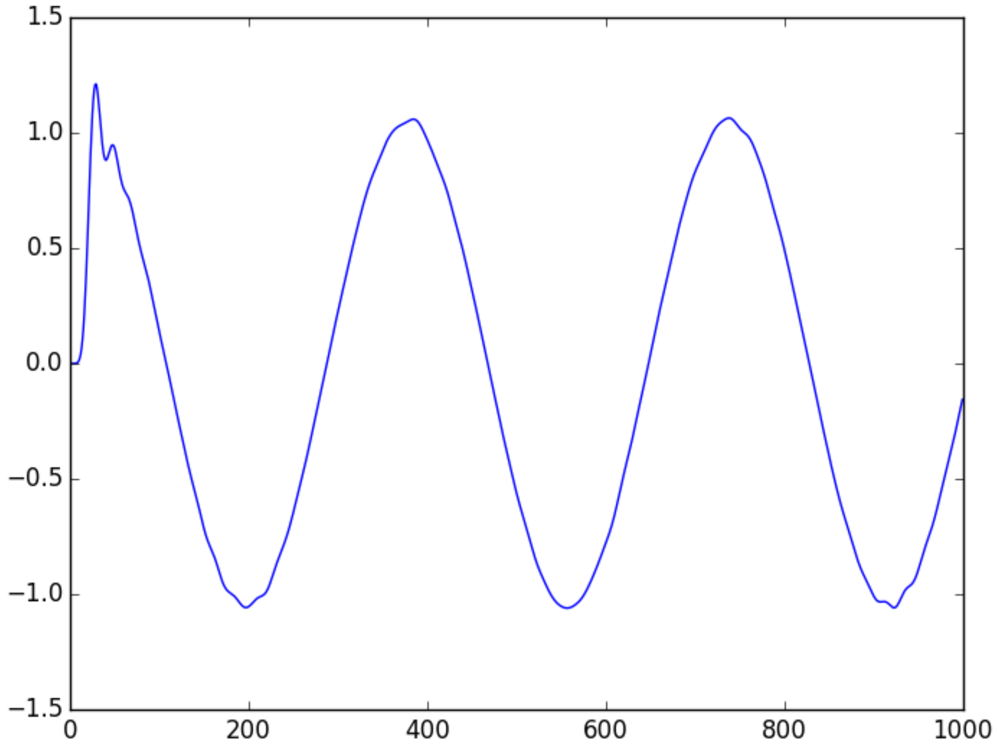
Cosine signal that generated with noise;



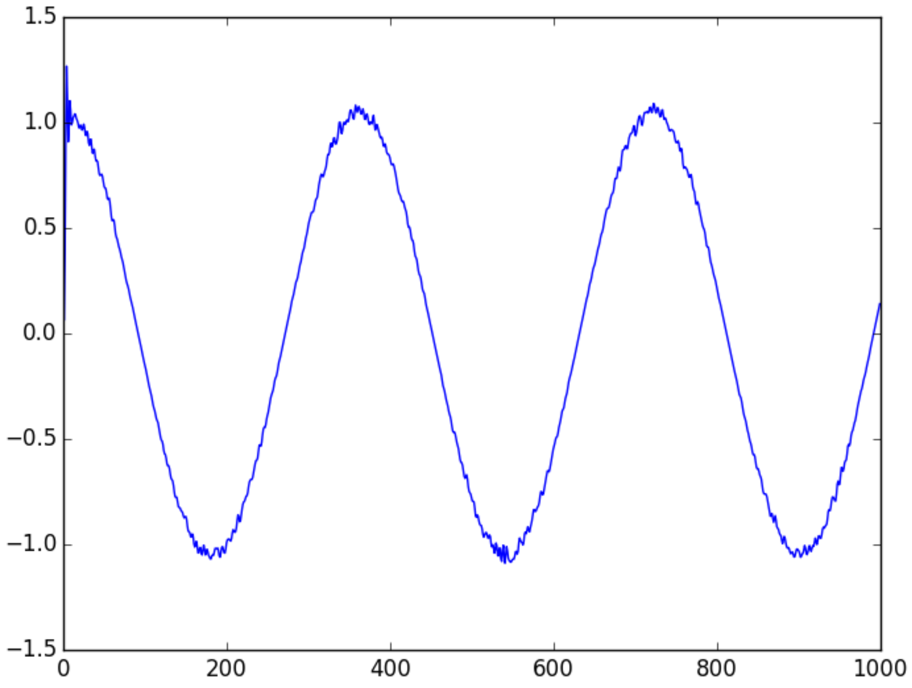
The signal that is filtered by a cutoff frequency=0.05 and gain at passband=2, gain at stop=30 with a sample rate of 2;



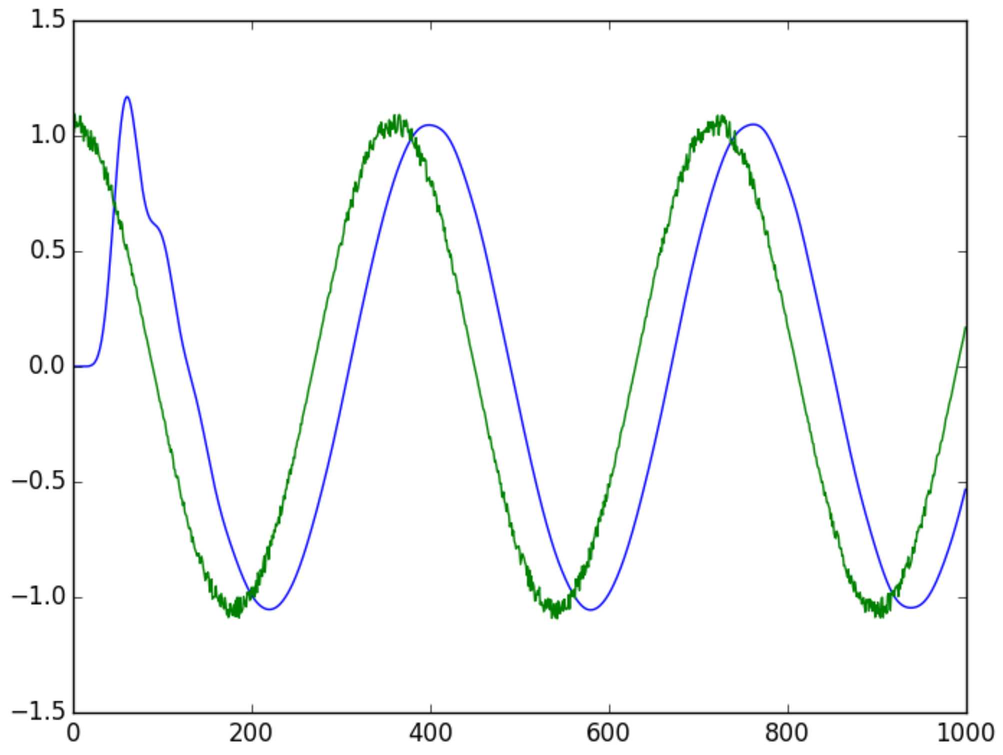
The signal that is filtered by a cutoff frequency=0.1 and gain at passband=2, gain at stop=30 with a sample rate of 2;



The signal that is filtered by a cutoff frequency=0.5 and gain at passband=2, gain at stop=30 with a sample rate of 2;



The comparison of the input signal and the signal that is filtered by a cutoff frequency=0.05 and gain at passband=2, gain at stop=30 with a sample rate of 2;



Code:

```
from sympy import *
from numpy import *
from pylab import *
import random as ran
from scipy.signal import butter, lfilter, buttord

#sampling rate
fs=2
cuttoff_freq=0.05
wp = cuttoff_freq/(fs/2)
ws = 1.5*wp

gpass=2
gstop=30
#Butterworth filter order selection
# Pre-warp frequencies
passb = tan(pi * wp / 2.0)
```

```
stopb = tan(pi * ws / 2.0)
lw = stopb / passb
GSTOP = 10 ** (0.1 * abs(gstop))
GPASS = 10 ** (0.1 * abs(gpass))
n = int(ceil(log10((GSTOP - 1.0) / (GPASS - 1.0)) / (2 * log10(lw))))
W0 = (GPASS - 1.0) ** (-1.0 / (2.0 * n))
WN = W0 * passb
wn = (2.0 / pi) * arctan(WN)

#zeros values
z=array([])
#to ensure that middle value is 0
m=arange(-n+1,n,2)
#constructing poles
p=-exp(1j*pi*m/(2*n))
#defining gain
k=1

#warping the frequency
w0=2*fs*tan(pi*wn/fs)
w0=float(w0)

#relative degree of transfer function
degree=len(p)-len(z)

k=k*w0**degree
z=w0*z
p=w0*p

#calculating nyquist ???
fs2=2*fs
#compansate for the gain change???(no change)
k=k*real(prod(fs2-z)/prod(fs2-p))
#bilinear transform the pads and zeros
z=(fs2+z)/(fs2-z)
p=(fs2+p)/(fs2-p)
#zeros at infinity moved to nyquist
z=append(z,-ones(degree))

#Return polynomial transfer function representation from zeros and
poles
#This part is taken from the scipy library
if len(z.shape) > 1:
    temp = poly(z[0])
    b = zeros((z.shape[0], z.shape[1] + 1), temp.dtype.char)
    if len(k) == 1:
        k = [k[0]] * z.shape[0]
    for i in range(z.shape[0]):
        b[i] = k[i] * poly(z[i])
else:
    b = k * poly(z)
a = atleast_1d(poly(p))


#generating time domain
t=linspace(1, 1000,1000,endpoint=False)

#generating a clean signal
clean_signal = cos(t*pi/180)

#adding noise to clean signal
noise_signal=clean_signal
for i in range(len(clean_signal)):
    noise_signal[i]=noise_signal[i]*ran.uniform(1,1.1)

y=lfilter(b, a, noise_signal, axis=-1, zi=None)
plot(t,y)
#plot(noise_signal)
show()
```