Research Methods

Robert E Simpson

Singapore University of Technology & Design

robert_simpson@sutd.edu.sg

November 9, 2018

Useful Noes

`Rob[72:144]=0` Sets the index range 72 to 144 of list Rob to 0.

`numpy.fft.fft` computes the fast Fourier transform

`numpy.fft.ifft` computes the inverse fast Fourier transform

Case Problem 1: Fourier Series

1. Write a function to plot the amplitude spectrum for the signal described by:

$$f(t) = \sum_{k}^{n} (n-k)\sin(2\pi kt) \tag{1}$$

2. Plot the waveform for this signal for $n = 20$ and $k = 10$ between $t = 0$ and $t = 4$ ensuring that the signal is not undersampled.
3. Redo the plot, only this time under and over sample the signal at $f_s/2$ and $2f_s$ respectively. What do you notice?
4. Use numpy's built in FFT function to compute and then plot the magnitude of the frequency spectrum for the signal. Make sure that the signal is not under sampled. What do you notice?

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Case Problem 1 Solution

```
def w9cp1(t, n, k):
    out=0
    for kk in range(k,n):
        out=out+ (n-kk)*np.sin(2*np.pi*kk*t)
    return out

Ns=600 #number of samples
t=np.linspace(0,4, Ns)
Maxtime=4.0
time_step=Maxtime/Ns
FreqStep=1./(Maxtime)
print "max freq is", 1/time_step
freq=[]
for i in range(Ns):
    freq.append(i*FreqStep)

ft=w9cp1(t,20,10)
```

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Case Problem 1 Solution

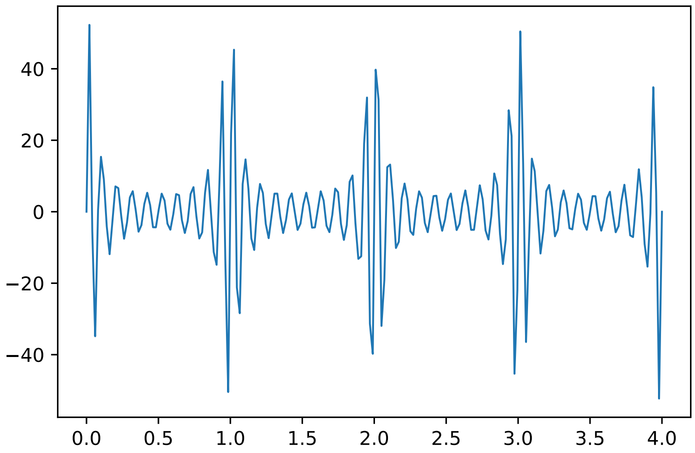
```
w=np.fft.fft(ft)  #compute forward Fourier transform

plt.figure()
plt.plot(t, ft, lw=1)

mag=[]
for line in w:
    mag.append(np.linalg.norm(line))  #compute the magnitude

plt.figure()
plt.plot(freq,mag)
plt.xlabel('Freq (Hz)')
plt.ylabel('FT Magnitude')
plt.xlim(0,50)
```

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Case Problem 1 solution

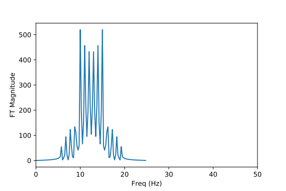
2.
The maximum frequency component in the signal is $k = 20$.
Therefore we need at least 40 points (f_s =>40 Hz) along one wavelength to reconstruct the signal.
We are asked to plot the wave from 0 to 4. When k is 1 this would be four wavelengths, but k goes up to 20. Therefore, we need to multiply this by 40 (because f_s =>40 Hz). So the minimum number of data points is: $4 \times 40 = 160$. But let's add more points to ensure that we capture all the features of the signal, so we make a wave with 200 points and scale the x-axis from 0 to 4.

Case Problem 1 solution

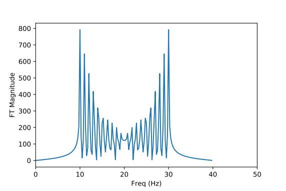


Case Problem 1 solution

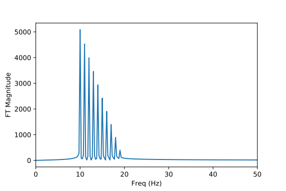
3. When we over sample the signal shape doesn't change too much, but when we under sample there is big change in the signal because we have removed the higher frequency components.



(a) 100 points



(b) 160 points



(c) 1000 points

Case Problem 1 solution

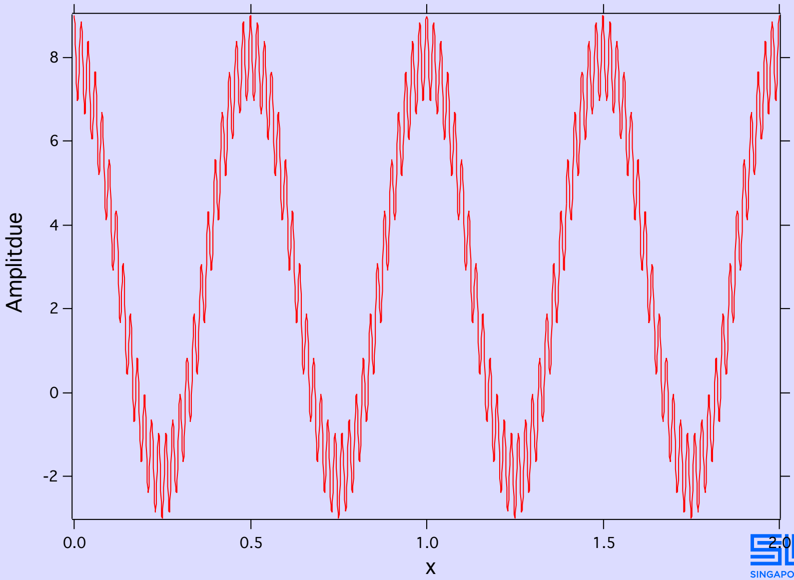
4. As we would expect, the signal is built from sinusoids at frequencies ranging from 10 Hz, 11 Hz, . . . , 20 Hz.

Case Problem 2: Distinguishing Signals

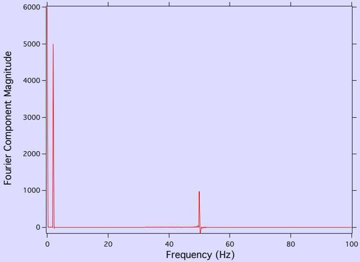
- Plot the wave: $5\cos((2\pi)2x) + \cos((2\pi)50x) + 3$ between x=0 and x=2.
- Plot the amplitude of the Fourier components that are present in the signal
- Now create a new wave of the complex FFT signal.
- Filter out the 50 Hz signal in frequency space by seting the amplitudes of the frequency spectrum to zero for frequencies close to 50 Hz (I created a 'window' function to do this).
- Now synthesise the signal from the modified/filtered Fourier component spectrum.
- Plot and compare the 'filtered' and original signals.
- Repeat the process and filter out the low frequency signals.

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

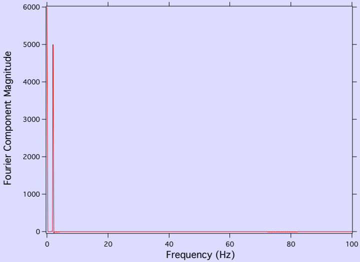
Case Problem 2 Solution



Case Problem 2 Solution



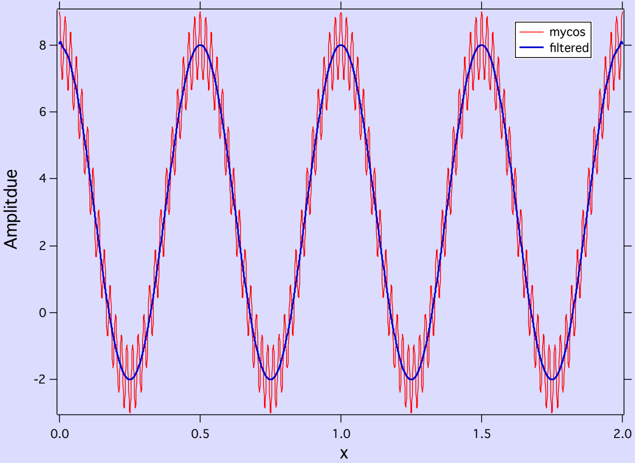
(d) Original FFT Magnitude



(e) Filtered FFT Magnitude

Case Problem 2 Solution

The filtered signal now does not have the high frequency noise.



Case Problem 2: Distinguishing Signals

```
Ns=1001  #number of samples

x=np.linspace(0,2,Ns)
y=5*np.cos((2*np.pi)*2*x) + np.cos((2*np.pi)*50*x) +

Maxtime=2.
time_step=Maxtime/Ns
FreqStep=1./(Maxtime)
print "freq step", FreqStep
print "max freq is", 1./time_step

freq=[]
for i in range(Ns):
    freq.append(i*FreqStep)
```

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Case Problem 2: Distinguishing Signals

```
w=np.fft.fft(y)

plt.figure()
plt.plot(x,y)

plt.figure()
plt.plot(freq,w)
plt.xlim(0,60)
```

Case Problem 2: Distinguishing Signals

```
win=[]
def window(freq,spec, f0,Delta_f):    #Create a widow function
    indx0=freq.index(f0)
    indx02=len(freq)-indx0
    print indx0, indx02
    freqstep=freq[1]-freq[0]
    print freqstep
    Deltaindex=Delta_f/freqstep
    print "Delta index", Deltaindex
    IndexStartDelete0=indx0-(Deltaindex/2)
    IndexEndDelete0=indx0+(Deltaindex/2)
    IndexStartDelete02=indx02-(Deltaindex/2)
    IndexEndDelete02=indx02+(Deltaindex/2)
    F_spec=spec
    F_spec[int(IndexStartDelete0):int(IndexEndDelete0)]=0
    F_spec[int(IndexStartDelete02):int(IndexEndDelete02)]=0
    return F_spec
```

Case Problem 2: Distinguishing Signals

```
filtered50=window(freq,w, 50,10)

plt.figure()
plt.plot(x,y)
plt.figure()
plt.plot(freq,filtered50)
plt.xlim(0,60)

F_spec=np.fft.ifft(filtered50)
plt.figure()
plt.plot(X,F_spec)
```

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Often a signal can be distinguished from noise by analysing the frequency components in the signal. E.g. The signal may contain low frequencies whereas the noise may be at a high frequency.

Case Problem 3

- Set the scale to go from 0 to 10 seconds
- Set a sinusoidal wave with an amplitude of 10 and a frequency of 2 Hz
- Add random white noise to the signal (numpy.random.normal)
- Now you have a noisy signal, perform Fourier filtering by setting all Fourier components to zero except for the component with the largest contribution to the signal.
- Compare the synthesised signal with the original signal.