PHYS :501 Mathematical Physics

Homework #6

Prakash Gautam

December 7, 2017

1. The function $f(t_k)$ adn $F(\omega_n)$ are discrete Fourier transforms of one anotherr, where $t_k = k\Delta, \omega_n = 3\pi n/N\Delta$, for $k, n = 0, \cdots, N-1$. Show that.
   (a) if f is real, then $F(\omega_n) = F^*(4\pi f_c - \omega_n)$,
   (b) if f is pure imaginary, then $F(\omega_n) = -F^*(4\pi f_c - \omega_n)$,
   where $f_c = 1/2\Delta$ is the Nyquist frequency.
   Solution:
   The fourier discrete fourier transform fo $f(t_k)$ by definition is

$$F(\omega_n) = \sum_{k=0}^{N-1} f(t_k) e^{i\omega_n t_k} = \sum_{k=0}^{N-1} f(k\Delta) e^{i\frac{2\pi n}{N\Delta}k\Delta} = \sum_{k=0}^{N-1} f(k\Delta) e^{2\pi i n k/N}$$

   Taking conjugate of the above expression with ω_n replaced by $4\pi f_c - \omega_n$ we get.

$$F^*(4\pi f_c - \omega_n) = \sum_{k=0}^{N-1} f^*(k\Delta) \left(e^{i\left(4\pi f_c - \frac{2\pi n}{N\Delta}\right)k\Delta}\right)^* = \sum_{k=0}^{N-1} f^*(k\Delta) \left(\overbrace{e^{-2i\pi}}^{1} + 2i\pi n/N\right)^k = \sum_{k=0}^{N-1} f^*(k\Delta)(e^{2\pi i n k/N})$$

   if f is real then $f^*(k\Delta) = f(k\Delta)$ if f is pure imaginary then $f^*(k\Delta) = -f(k\Delta)$

$$F^*(4\pi f_c - \omega_n) = \sum_{k=0}^{N-1} f(k\Delta)(e^{2\pi i n k/N}) = F(\omega_n) \qquad F^*(4\pi f_c - \omega_n) = \sum_{k=0}^{N-1} -f(k\Delta)(e^{2\pi i n k/N}) = -F(\omega_n)$$

   Which completes the proof. ■

2(a) Let $\mathcal{R}_j$ be a random sequence of real numbers, with $\mathcal{R}_j$ distributed uniformly between -1 and 1. For N-piont discrete Fourier transform of $\mathcal{R}_j$: $r_k = \sum_{j=0}^{N-1} \mathcal{R}_j e^{2\pi i j k/N}$, calculate the expectation value and variance of the "periodogram estimate" of the power spectrum, $P_k = |r_k|^2 + |r_{N-k}|^2$, for $k = 1, \cdots, N/2$.
   Solution:
   Given $P_k = |r_k|^2 + |r_{N-k}|^2$ we can calculate the expectation value of the function P_k as

$$\langle P_k \rangle = \frac{1}{N}\left[\sum_{1}^{N/2} |r_k|^2 + \sum_{1}^{N/2} |r_{N-k}|^2\right] = \frac{1}{N}\sum_{0}^{N-1} |r_k|^2 \overset{\text{Parseval's Theorem}}{=} \sum_{0}^{N-1} |\mathcal{R}_k|^2$$

   This the expectation value of the function P_k Now for the variance

$$Var(P_k) = \left\langle P_k^2 \right\rangle - \langle P_k \rangle^2 = \frac{1}{N}\sum_{1}^{N/2} \left[|r_k|^2 + |r_{N-k}|^2\right]^2 - \sum_{0}^{N-1} |\mathcal{R}_k|^2$$

   The simplification should give the variance. ■

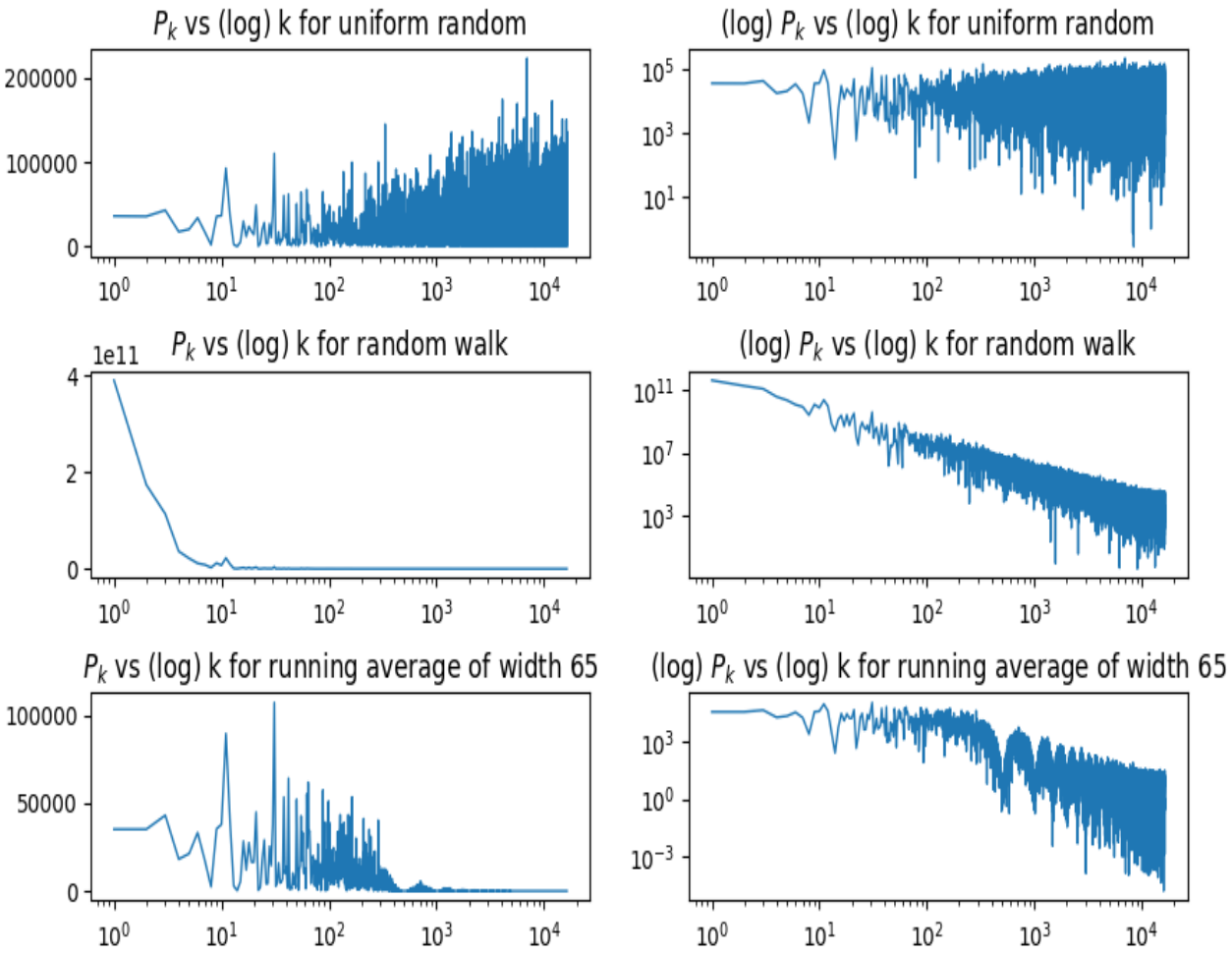
(b) Generate a sequence of random numbers with properties as in part (a), and compute P_k numerically using a fast Fourier transform with $N = 32768$. Plot first P_k, then $\log_{10} P_k$ against $\log_{10} k$, for $k = 1, \cdots, N/2$. How does this graph compater with the analytic expectations from part (a)?

Repeat the calcluations, averaging the data over an interval of width 65 centered on each frequency data point, and plot the results.

(c) Repeat the computation in part (b) for *random walk*w_j definde by $w_0 = 0$; $w_{j+1} = w_j + \mathcal{R}_j$. Can you account for the differences in appearence between this graph and one you obtained in part (b)?

Solution:

For the graph of uniform random the power is flat curve for up to a high frequency range but for random



walk the power at higher frequency is significantly lower than the power at lower frequencies. ■

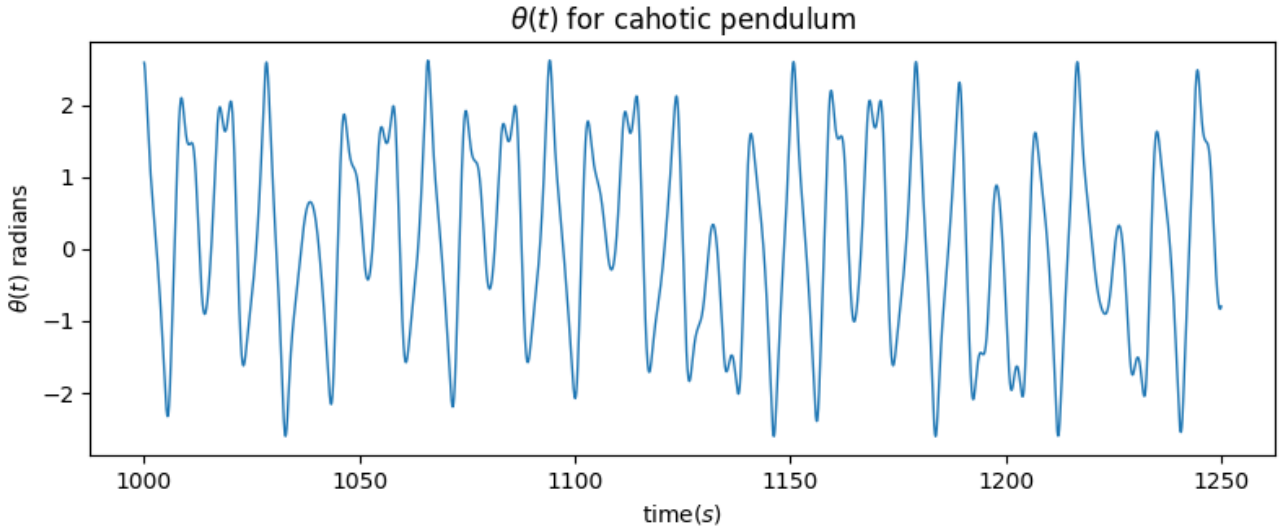
3. The file `pendulum2.dat` contains a chaotic data set generated as a solution to the equations of motion of the damped, driven, nonlinear pendulum:

$$\frac{d^2\theta}{dt^2} + \frac{1}{q}\frac{d\theta}{dt} + sin(\theta) = g\cos(\omega_d t).$$

Contains four column t, θ, ω and $\phi = \omega_d t$

(a) Plot the time sequence (t) for 1000 t 1250.

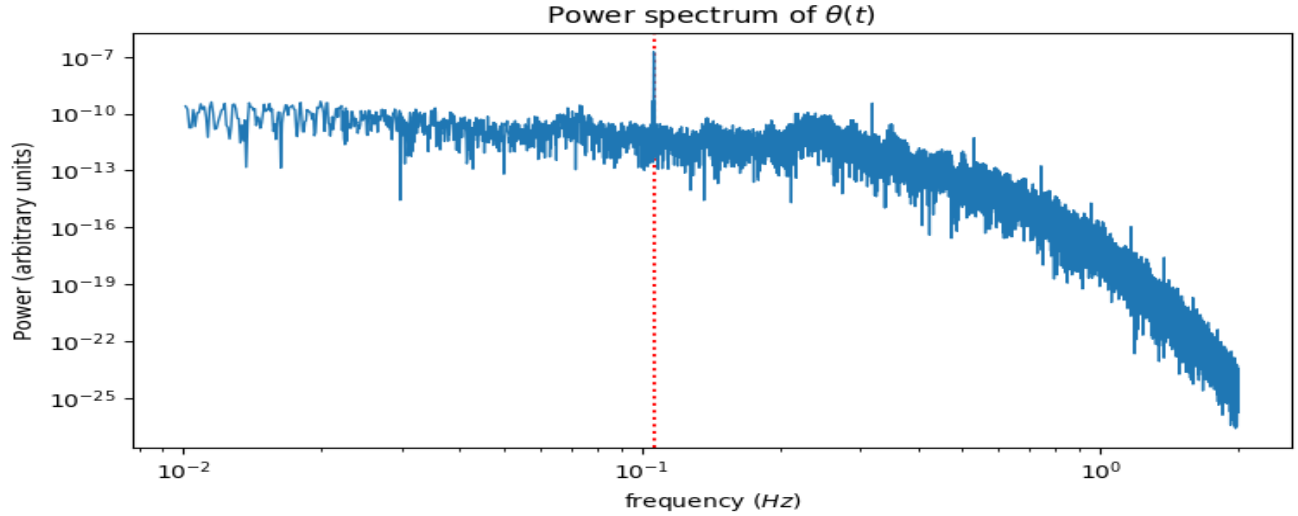
Solution:



This graph shows the plot of $\theta(t)$ vs t for the time range $t = 1000$ to $t = 1250$ ■

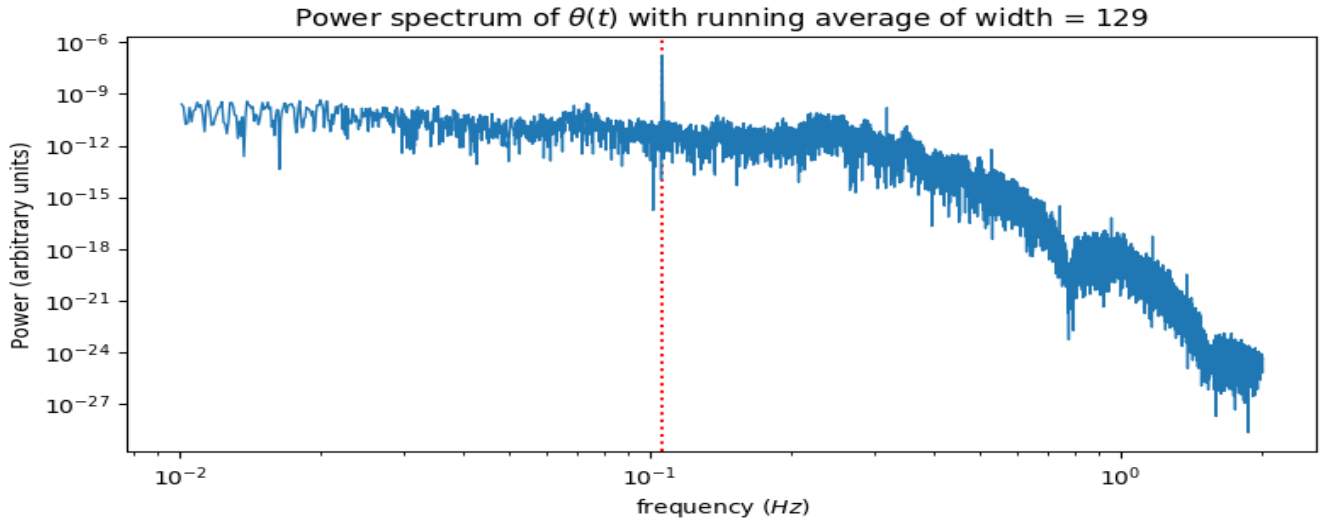
(b) Use an FFT to compute the power spectrum $P(f)$ of $\theta(t)$, where f is frequency. Use the entire dataset, with a Bartlett data window, and plot $P(f)$ with loglog axes for $0.01 \leq f \leq 2$. Can you identify any features in the plot?

Solution:

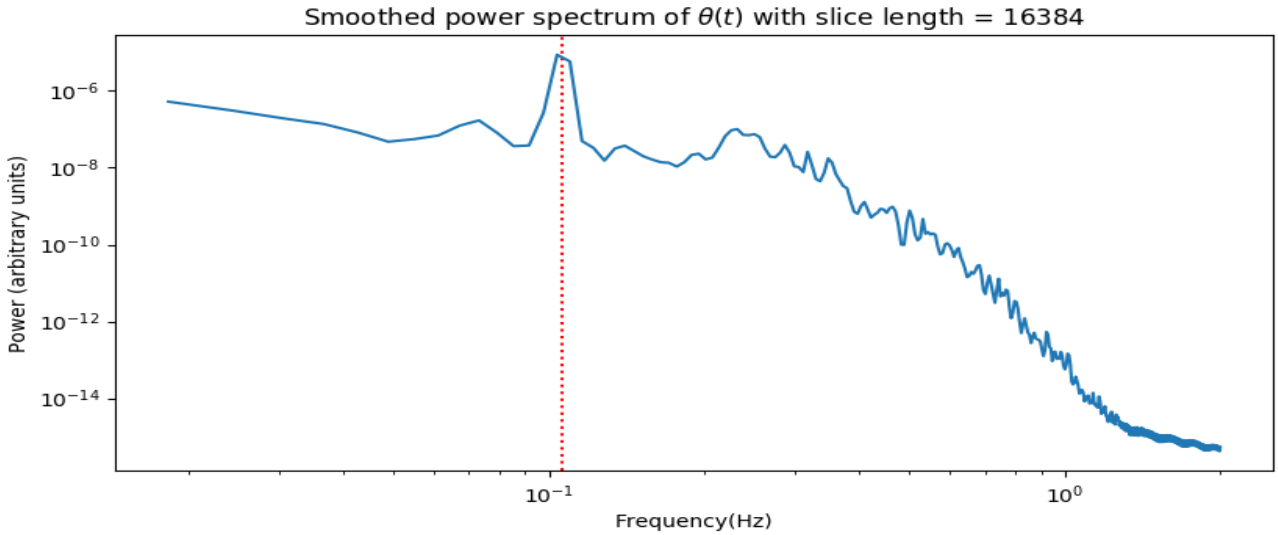


There is a sharp spike at frequency $f = 0.106$ which equals the frequency of the driving force. $w_d = 2/3$

(c) As in Problem 2, smooth the data by averaging over an interval of width 129 centered on each frequency data point, and plot the results as in part (b).



(d) Implement the alternative smoothing strategy of dividing the input dataset into a series of slices each of length 16384, computing the power spectra of each, and then averaging all the individual power spectra. Again plot the results as in part (b). Dont forget the Bartlett windows!



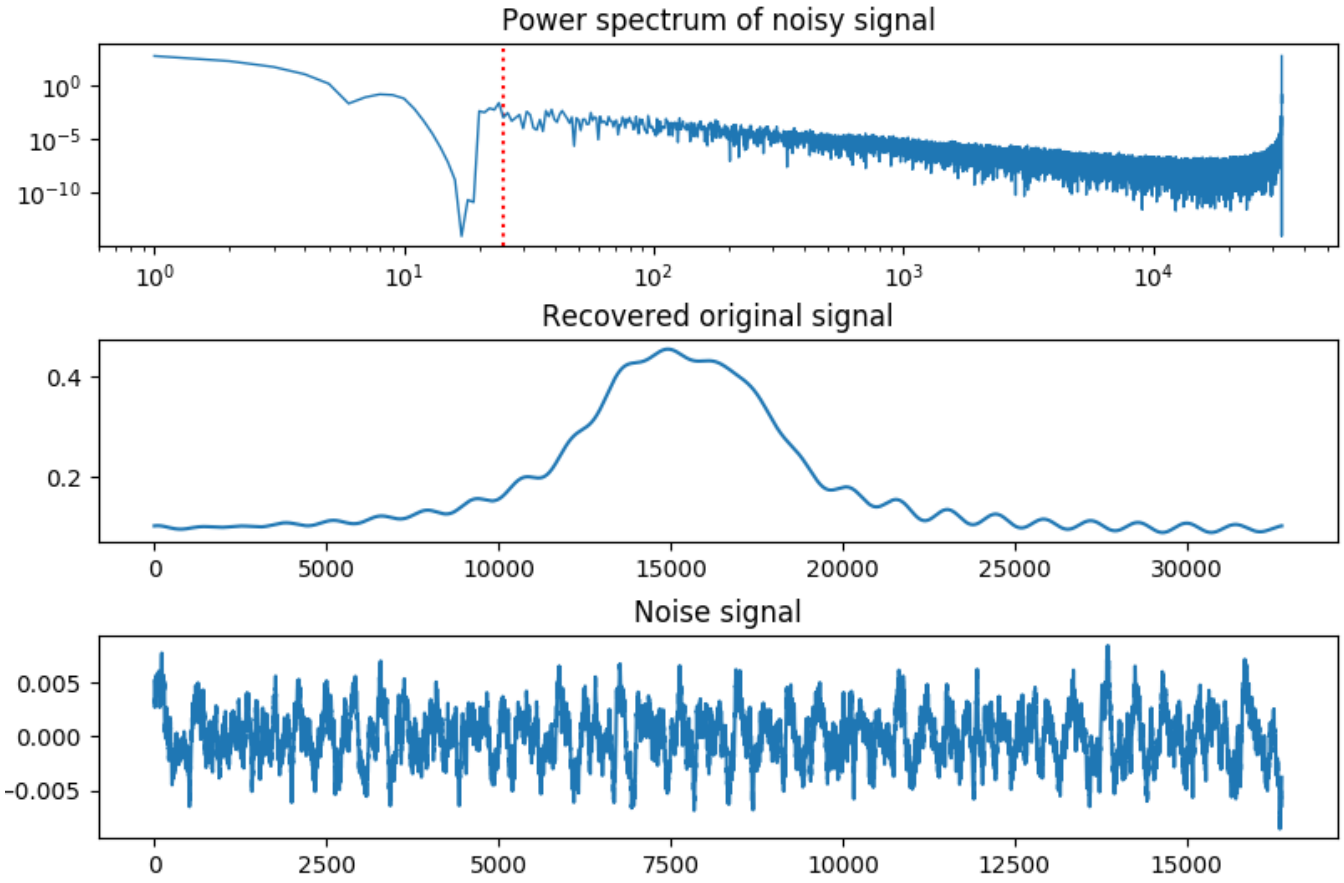
On each of these power spectrum there the spectrum is flat for lower frequency and it drops sharply after some frequency. A common feature in all of these graphs is the presence of a spike in power spectrum at $f = 0.106$. This corresponds to the frequency of the driving force. ■

4. A corrupted real-valued dataset may be found in the file `corrupt.dat`. It is a time sequence consisting of two columns of data, j and c j , for $j = 0, \cdots N1$. The original data have been convolved with a Gaussian transfer function of the form $g \approx exp(j^2/a^2)$ (normalized so that $\sum_j g_j = 1$), with a = 2048, and are subject to random noiseof some sort at some level.

   Find a filter to apply to the data, and plot your best-guess reconstruction of the original uncorrupted dataset. Can you characterize the type of noise in the data?

   Solution:

The power spectrum of the corrupted signal has high value for lowe frequencies and it is significantly small



for higher frquency. At around 25^{th} frequency bin the noise is completely dominates. So I chose 25^{th} frequency bin as the cutoff point.
The recovered signal looks like a gaussian function. The recovered noise looks like a white noise. ■

QuestionTwo

```
#!/usr/bin/env python3

import itertools
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt

class PowerSpectrum():
    def __init__(self,N):
        self.N = N
        self.pcnt = 1
        self.spl = 220
        self.wid = 65

    def get_rnd_sequence(self):
        rnd = np.random.uniform(-1,1,self.N)
        return rnd

    def get_rnd_walk(self,sig):
        wlk = list(itertools.accumulate(sig))
        return wlk

    def get_power(self,sig):
        r = np.fft.fft(sig); N = len(sig)
        Pk = [np.abs(r[k])**2 + np.abs(r[N-k])**2 for k in
            range(1,int(N/2))]
        return Pk

    def plot_spectrum(self,sig,title=''):
        Pk = self.get_power(sig)
        k = range(len(Pk))
        plt.subplot(self.spl+self.pcnt); plt.xscale('log');
            self.pcnt+=1
        plt.plot(k,Pk,lw=1); plt.title(r'$P_k$ vs (log) k for
             '+title)
        plt.subplot(self.spl+self.pcnt); plt.xscale('log');
            plt.yscale('log');self.pcnt+=1
        plt.plot(k,Pk,lw=1); plt.title(r'(log) $P_k$ vs (log)
             k for '+title)

    def get_run_average(self,sig,wid):
        hwid = int((wid-1)/2)
        ravg = [np.average(sig[k-hwid:k+hwid+1]) for k in
            range(hwid,len(sig)-hwid)]
        return ravg

    def machinge_periodogram(self,sig):
        plt.subplot(self.spl+self.pcnt); self.pcnt +=1
        plt.xscale('log')
        prd,Pxx_den = signal.periodogram(rsig)
        plt.plot(prd,Pxx_den,lw=1)
        plt.subplot(self.spl+self.pcnt); self.pcnt +=1
        plt.xscale('log'); plt.yscale('log')
        plt.plot(prd,Pxx_den,lw=1)

    def new_experiment(self):
        rsig = self.get_rnd_sequence()
        wsig = self.get_rnd_walk(rsig)
        ravg = self.get_run_average(rsig,PS.wid)

        plt.xscale('log'); plt.yscale('log')
        rprd,rpxxden = signal.periodogram(rsig)
        aprd,apxxden = signal.periodogram(ravg)
        plt.plot(rprd,rpxxden)
        plt.plot(aprd,apxxden)
        plt.show()


    def experiment(self):
        rsig = PS.get_rnd_sequence()
        wsig = PS.get_rnd_walk(rsig)
        ravg = PS.get_run_average(rsig,PS.wid)

        plt.xscale('log'); plt.yscale('log')
        plt.plot(self.get_power(rsig))
        plt.plot(self.get_power(ravg))
        plt.show()


if __name__ == '__main__':
    PS = PowerSpectrum(32768)
    PS.spl = 320; PS.wid = 65

    rsig = PS.get_rnd_sequence()
    wsig = PS.get_rnd_walk(rsig)
    ravg = PS.get_run_average(rsig,PS.wid)

    PS.plot_spectrum(rsig, 'uniform random ')
    PS.plot_spectrum(wsig, 'random walk')
    PS.plot_spectrum(ravg, 'running average of width {}'.
        format(PS.wid))
    #PS.machinge_periodogram(rsig)
    #PS.new_experiment()

    plt.show()
```

Question Three

```
#!/usr/bin/env python3

import itertools
import numpy as np
import matplotlib.pyplot as plt

from  matplotlib2tikz import save as tikz_save

class ForcedPendulum():
    def __init__(self,flname):
        self.filename = flname
        self.slc = 20
        self.spl = 210; self.pcnt = 1
        self.read_file()

    def read_file(self):
        content = np.genfromtxt(self.filename)
        self.t     = content[:,0];
        self.theta = content[:,1]
        self.omega = content[:,2]
        self.phi   = content[:,3]
        return content

    def bratlett_window(self,sig):
        N = len(sig); hlen = int(N/2)
        sig2 = np.copy(sig)
        sig2[0:hlen] = [k*sig[k] for k in range(hlen)]
        sig2[hlen:N] = [(N-k)*sig[k] for k in range(hlen,N)]
        wss = N*np.sum([1-np.abs(2*j-N)/N for j in range(N)])
        return sig2/(wss)

    def get_run_average(self,sig,wid):
        hwid = int((wid-1)/2)
        ravg = [np.average(sig[k-hwid:k+hwid+1]) for k in
            range(hwid,len(sig)-hwid)]
        return ravg

    def plot_theta(self,min_time=1000,max_time=1250):
        tmin = min_time; tmax = max_time
        minidx = np.searchsorted(self.t,tmin)+1
        maxidx = np.searchsorted(self.t,tmax)+1
        plt.subplot(self.spl+self.pcnt); self.pcnt += 1; plt.
            tight_layout()
        tdata  = self.t[minidx:maxidx]
        thdata = self.theta[minidx:maxidx]
        plt.plot(tdata,thdata,lw=1)
        plt.title(r'$\theta(t)$ for cahotic pendulum');
        plt.xlabel(r'time$(s)$'); plt.ylabel(r'$\theta(t)$
            radians')

    def get_power(self,sig):
        sft = np.fft.fft(sig)
        N = len(sig)
        power = 1/N*(np.abs(sft))**2
        dt = (max(self.t) - min(self.t))/len(self.t)
        dw = 2*np.pi/(N*dt)
        omega = np.arange(N)*dw;

        return power, omega

    def plot_power_spectrum(self,sig,minfrq=None,maxfrq=None)
        :
        power,omega = self.get_power(self.bratlett_window(sig
            )); freq = omega/(2*np.pi)
        minfloc = np.searchsorted(freq,minfrq)+1 if minfrq !=
             None else 0
        maxfloc = np.searchsorted(freq,maxfrq)+1 if maxfrq !=
             None else len(omega)
        maxpowloc = power.argmax(); maxpowfreq= freq[
            maxpowloc]

        plt.subplot(self.spl+self.pcnt); self.pcnt += 1;plt.
            tight_layout()
        plt.yscale('log'); plt.xscale('log');
        plt.axvline(x=maxpowfreq,color='r',ls=':')
        plt.plot(freq[minfloc:maxfloc],power[minfloc:maxfloc
            ],lw=1)
        plt.title(r'Power spectrum of $\theta(t)$')
        plt.xlabel(r'frequency ($Hz$)'); plt.ylabel('Power (
            arbitrary units)')

    def plot_run_power_spectrum(self,sig,wid=129,minfrq=None,
        maxfrq=None):
        runavgsig = self.get_run_average(sig,wid)
        power,omega = self.get_power(self.bratlett_window(
            runavgsig)); freq = omega/(2*np.pi)
        minfloc = np.searchsorted(freq,minfrq)+1 if minfrq !=
             None else 0
        maxfloc = np.searchsorted(freq,maxfrq)+1 if maxfrq !=
             None else len(omega)
        maxpowloc = power.argmax(); maxpowfreq= freq[
            maxpowloc]

        plt.subplot(self.spl+self.pcnt); self.pcnt += 1; plt.
            tight_layout()
        plt.yscale('log'); plt.xscale('log');
```

```
        plt.axvline(x=maxpowfreq,color='r',ls=':')
        plt.plot(freq[minfloc:maxfloc],power[minfloc:maxfloc
            ],lw=1)
        plt.title(r'Max power at $\omega$= {:2}, f={:2}'.
            format(maxpowfreq*2*np.pi,maxpowfreq))
        plt.title(r'Power spectrum of $\theta(t)$ with
            running average of width = {}'.format(wid))
        plt.xlabel(r'frequency ($Hz$)'); plt.ylabel('Power (
            arbitrary units)')

    def plot_sliced_power_spectrum(self,sig,slwid=16384,
        minfrq=None,maxfrq=None):
        tmin = self.t[0]; tmax = self.t[slwid]; dt = (tmax-
            tmin)/slwid;
        dw = 2*np.pi/(slwid*dt); omega = np.arange(slwid)*dw;
             freq=omega/(2*np.pi)

        N = len(sig); padlength = slwid - N % slwid
        longsig = np.concatenate((sig,np.zeros(padlength)),
            axis=0);

        minfloc = np.searchsorted(freq,minfrq)+1 if minfrq !=
             None else 0
        maxfloc = np.searchsorted(freq,maxfrq)+1 if maxfrq !=
             None else len(omega)

        cnt = 0; avpwr  = 0
        for pos in range(0,len(longsig),slwid):
            piece = longsig[pos:pos+slwid]; cnt+=1
            power,omg = self.get_power(self.bratlett_window(
                piece));
            avpwr += power

        avpwr /= cnt

        maxpowloc = avpwr.argmax(); maxpowfreq= freq[
            maxpowloc]
        print('max power freq',maxpowfreq)

        plt.subplot(self.spl+self.pcnt); self.pcnt += 1; plt.
            tight_layout()
        plt.yscale('log'); plt.xscale('log')
        plt.axvline(x=0.1061,color='r',ls=':')
        plt.plot(freq[minfloc:maxfloc],avpwr[minfloc:maxfloc
            ])
        plt.title(r'Smoothed power spectrum of $\theta(t)$
            with slice length = {}'.format(slwid))
        plt.xlabel(r'Frequency(Hz)'); plt.ylabel('Power (
            arbitrary units)');

if __name__ == '__main__':
    FP = ForcedPendulum('./files/pendulum2.dat')
    FP.spl = 110
    #FP.plot_theta(); #tikz_save('images/Pendulum.tex')
    #FP.plot_power_spectrum(FP.theta,minfrq=0.01,maxfrq=2)
    #FP.plot_run_power_spectrum(FP.theta,wid=129,minfrq=0.01,
        maxfrq=2)
    FP.plot_sliced_power_spectrum(FP.theta,minfrq=0.01,maxfrq
        =2)

    plt.show()
```

Question Four

```
#!/usr/bin/env python3

import itertools
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal


class NoiseReduction():
    def __init__(self,flname):
        self.filename = flname
        self.slc = 20
        self.spl = 210; self.pcnt = 1
        self.read_file()
        self.a = 2048
        self.trc = 10

    def read_file(self):
        content = np.genfromtxt(self.filename)
        self.j     = content[:,0];
        self.c     = content[:,1]
        self.N     = len(self.c)

    def get_threshold_loc(self,sig,thr):
        return self.trc

    def get_power(self,sig):
        sft = np.fft.fft(sig)
        power = 1/len(sft)*(np.abs(sft))**2
        return power
```

```
    def sys_function(self):
        hn = int(self.N/2)
        j = np.arange(hn+1)
        sysf = np.zeros(self.N)
        sysf[:hn+1] = np.exp(-j**2/self.a**2)
        sysf[-hn:] = sysf[hn:0:-1]
        sysf /= np.sum(sysf)
        return sysf

    def truncate_noise(self,sig):
        thr = 1e-4
        scp = np.fft.fft(sig)
        thl = self.get_threshold_loc(scp,thr)
        scp[thl:-thl] = 0
        return scp

    def divide_complex(self,Nr,Dr):
        N = len(Nr)
        nsig = np.zeros(N,dtype=complex)
        nsig = np.where(abs(Nr)>0,Nr/Dr,nsig)
        return nsig

    def get_back_signal(self,sig):
        csig = sig
        truncated = self.truncate_noise(csig)
        ftgaussian = np.fft.fft(self.sys_function())
        ftsig = self.divide_complex(truncated,ftgaussian)
        orgsig = np.fft.ifft(ftsig)
        return orgsig


    def get_back_noise(self,corrupt_sig):
        osig = self.get_back_signal(corrupt_sig)
        fftosig = np.fft.fft(osig)
        fftgaussian = np.fft.fft(self.sys_function())
        fftcorrupt_sig = np.fft.fft(corrupt_sig)

        fftnoise = fftcorrupt_sig/fftgaussian - fftosig

        noise = np.fft.ifft(fftnoise)
        #noise = np.fft.ifft(fftcorrupt_sig)
        plt.plot(noise)
        #return noise


    def plot_org_sig(self,sig):
        osig = self.get_back_signal(sig)
        plt.plot(osig)


    def plot_power_spectrum(self,sig):
        power = self.get_power(sig); lng = len(power)
        omega = 1/len(sig) *2*np.pi* np.arange(lng)
        plt.subplot(self.spl+self.pcnt); self.pcnt += 1
        #plt.yscale('log'); plt.xscale('log');
        plt.plot(omega,power)
        plt.subplot(self.spl+self.pcnt); self.pcnt += 1
        plt.yscale('log'); plt.xscale('log');
        plt.plot(power,lw=1)

if __name__ == '__main__':
    NR = NoiseReduction('./files/corrupt.dat')
    NR.spl = 210
    #NR.plot_power_spectrum(NR.c)
    NR.plot_org_sig(NR.c)
    #NR.get_back_noise(NR.c)
    #plt.plot(NR.c)

    plt.show()
```