Problem 1_a:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy


matdata = scipy.io.loadmat('vostok.mat')
g = np.array(matdata['OxygenIsotopesd18O'][0])
x = np.arange(0, 414)

var_set = np.var(g)
std_set = np.std(g)
mean_set = np.mean(g)
num_samples = len(g)

#------------- white noise -------------

w = np.random.normal(mean_set, std_set, size=num_samples)
mean_w = np.mean(w)
var_w = np.var(w)

#------------- red noise -------------

r = [0]*num_samples
dt = [-1, 1]

for i in range(1, num_samples):
   r[i] = r[i-1] + np.random.choice(dt)

r = np.array(r)
var_red = np.var(r)
std_red = np.std(r)
mean_red = np.mean(r)

factor = var_set/var_red
r = r*np.sqrt(factor)

diff = mean_set - np.mean(r)
r = r + diff

mean_r = np.mean(r)
var_r = np.var(r)
```

```
#------------- plotting -------------

plt.plot(x, g, label = 'Oxygen Isotope delta-18O Dataset (Mean = %s, Variance = %s)' %
(round(mean_set,2), round(var_set, 2)), color = 'k')
plt.plot(x, w, label = 'White Noise (Mean = %s, Variance = %s)' % (round(mean_w, 2),
round(var_w, 2)), color = 'b')
plt.plot(x, r, label = 'Red Noise (Mean = %s, Variance = %s)' % (round(mean_r, 2),
round(var_r, 2)), color = 'r')
plt.legend()
plt.title('Oxygen Isotope delta-18O Data vs Noise')
plt.xlabel('Time')
plt.ylabel('delta-18O')
```

Problem 1_b:

```
import numpy as np
import scipy
import matplotlib.pyplot as plt

matdata = scipy.io.loadmat('vostok.mat')
g = matdata['OxygenIsotopesd18O'][0]
data = np.array(g)


var_set = np.var(data)
std_set = np.std(data)
mean_set = np.mean(data)
num_samples = len(data)

data = data[:] - mean_set
transform = np.fft.rfft(data)
transform = np.abs(transform)**2

f = np.linspace(0, .5, np.size(transform))

def w_noise_floor():
    array = np.zeros((208, 100))

    for i in range(99):
        x = np.random.normal(mean_set, std_set, size=num_samples)
        mean = np.mean(x)
```

```
        x = x - mean
        transx = np.fft.rfft(x)
        transx = np.abs(transx)**2

        for j in range(len(transx)-1):
            array[j][i] = transx[j]

    return array

def r_noise_floor():
    array = np.zeros((208, 100))

    for i in range(99):
        r = [0]*num_samples
        dt = [-1, 1]

        for k in range(1, num_samples):
            r[k] = r[k-1] + np.random.choice(dt)

        r = np.array(r)
        var_red = np.var(r)

        factor = var_set/var_red
        r = r*np.sqrt(factor)

        r = r - np.mean(r)

        transx = np.fft.rfft(r)
        transx = np.abs(transx)**2

        for j in range(len(transx) -1):
            array[j][i] = transx[j]

    return array


def sort_floor(array):
    x = np.array(array, copy = True)

    for i in range(len(x)):
        l = x[i][:]
        l.sort()
```

```
        x[i] = l

    return x

def get_floor(array):
    x = array[:]
    floor = []

    for i in range(len(x)):
        floor.append(x[i][94])

    return floor

red_floor = r_noise_floor()
red_floor = sort_floor(red_floor)
red_floor = get_floor(red_floor)

white_floor = w_noise_floor()
white_floor = sort_floor(white_floor)
white_floor = get_floor(white_floor)

#plt.plot(f, transform, color = 'k', label = 'Oxygen Isotope d18O Data')
#plt.plot(f, white_floor, color = 'b', label = 'White Noise FLoor')
plt.plot(f, red_floor, color ='r', label = 'Red Noise Floor')

'''
plt.axvline(x = .01, label = 'Combined Eccentricity Frequency', color= '#ef1010', linestyle='--')
plt.axvline(x = .0388, label = 'Axial Precession Frequency', color='#ff6600', linestyle='--')
plt.axvline(x = .008928, label = 'Apsidal Precession Frequency', color='#6708db',
linestyle='--')
plt.axvline(x = .014285, label = 'Orbital Inclination Frequency', color='#b700b7', linestyle='--')
'''
plt.title('Period Power of Oxygen Isotope d18O Data Compared to Colored Noise Floors')
plt.xlabel('Frequency 1/1000yrs')
plt.ylabel('Power')
plt.legend()
plt.xscale('log')
plt.yscale('log')

plt.xlim(0, .45)
plt.ylim(1, 7000)
```

Problem 2_a:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

data = []

with open('mass_coastline', 'r') as f:

    for line in f:
        n = line.split()
        n[0], n[1] = float(n[0]), float(n[1])

        data.append(n)

data = np.array(data)

x_coords = data[:, 0]
y_coords = data[:, 1]

min_x = min(x_coords)
max_x = max(x_coords)
min_y = min(y_coords)
max_y = max(y_coords)
print max_y

def matrix(r):

    x_length = np.ceil(max_x / float(r))
    #matrix[:, 0]
    y_length = np.ceil(max_y / float(r))
    #matrix[0, :]

    matrix = np.zeros((x_length, y_length))
    return matrix


def loop_it(r):

    grid = matrix(r)

    for i in data:
```

```
        x = i[0]
        y = i[1]

        x_grid = np.ceil(x / float(r)) - 1
        y_grid = np.ceil(y / float(r)) -1

        if grid[x_grid, y_grid] != 1:

            grid[x_grid, y_grid] = 1

        else:
            pass

    return grid


def all_together_now():
    r = np.array(range(250, 300, 5))
    n_r = np.zeros(len(r))

    for i in range(len(r)-1):

        n_r[i] = np.sum(loop_it(r[i]))

    return r, n_r

'''
r, n_r = all_together_now()

def func(x, a, b):
    return a*x**b

popt, pcov = curve_fit(func, r, n_r)

a, b = popt[0], popt[1]
'''

plt.plot(r, n_r, label = 'N(r)')
plt.plot(r, func(r, *popt), label = "Fit %s * r ** %s" % (round(a, 2), round(b, 2)))
plt.xlabel('r (meters)')
plt.ylabel('N (number of boxes)')
plt.legend()
```

```
plt.title('Number of Boxes to Cover the Massachusetts Coastline as a Function of box
Sidelength (Middling r)')
plt.xscale('log')
plt.yscale('log')
```

Problem 3_a:

```
import numpy as np
import matplotlib.pyplot as plt
import invperc
from scipy.optimize import curve_fit

x_range = range(5, 500, 5)

def get_thing():

    r = []

    for i in x_range:

        n = []

        for j in range(5):
            n.append(invperc.invperc(i))

        value = np.mean(n)
        r.append(value)

    return r

r = get_thing()

def func(x, a, b):
    return a*x**b

popt, pcov = curve_fit(func, x_range, r)

a, b = popt[0], popt[1]

plt.plot(x_range, r, label = "Simulated Oil Volume")
plt.plot(x_range, func(x_range, *popt), label = "Fit %s * L ** %s" % (round(a, 2), round(b, 2)))
plt.title("Displaced Volume of Oil as a Function of Side Length")
```

```
plt.xlabel("Side Length of Grid")
plt.ylabel("Volume of Oil Displaced")
plt.xscale('log')
plt.yscale('log')
plt.legend()
```

Problem 3_b:

```
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import numpy as np
import random

data = []

with open('invperc_example', 'r') as f:

    for line in f:
        n = line.split()

        for i in range(len(n)):
            n[i] = int(n[i])

        data.append(n)

data = np.array(data)


def start():
    start = random.randrange(0, 999)
    return (start, 0)

def get_direction():
    return random.choice([[-1, 0], [1, 0], [0, 1], [0, -1]])

def is_adj(point, direction):

    x = (point[0] + direction[0]) % len(data[:, 0])
    y = point[1] + direction[1]

    if y < 0:
        return point
```

```
    elif y == len(data[0, :]):
        return point
    elif data[x, y] == 1:
        return [x, y]
    else:
        return point

def get_dist(point, start):
    x_point = point[0]
    y_point = point[1]
    x_start = start[0]
    y_start = start[1]

    dist = np.sqrt((x_point - x_start)**2 + (y_point - y_start)**2)

    return dist

def move_n_times(n):

    dist = []
    start_pos = start()
    old = start_pos
    new = ''

    for i in range(n):
        new = is_adj(old, get_direction())
        dist.append(get_dist(new, start_pos))
        old = new

    return np.array(dist)

def loopdidoo(n, simnum):

    global_dist = np.zeros(n)

    for i in range(1, simnum):
        distance = move_n_times(n)
        total = (global_dist + distance)

        global_dist = total

    return global_dist/float(simnum)
```

```
x_list = range(1000)
y_list = loopdidoo(1000, 15000)

def func(x, a, b):
    return a*x**b

popt, pcov = curve_fit(func, x_list, y_list)

a, b = popt[0], popt[1]

plt.plot(x_list, y_list, label = 'Simulated Particle Distance')
plt.plot(x_list, func(x_list, *popt), label = "Fit %s * t ** %s" % (round(a, 2), round(b, 2)))
plt.xlabel('Time')
plt.ylabel('Distance')
plt.title('Distance of a Particle from its Origin as a Function of Time')
plt.legend()
plt.xscale('log')
plt.yscale('log')
```