Problem 1

```
import matplotlib.pyplot as plt
import random

# takes in the position of a walker and moves it up or down with a 50/50 chance.
def move_walker(pos):
    position = pos
    if random.uniform(0, 1) <= 0.5:
    position -= 1
    else:
    position += 1

    return position


# deletes walkers based on probability, or if their position is positive (they have left the microbial mat)
def delete_walkers(walkers, p, zero_num):

    for i in walkers[:]:

    if random.uniform(0, 1) <= p:
    walkers.remove(i)

    for i in walkers[:]:
    if i <= 0:
    walkers.remove(i)
    else:
    pass

    return walkers + [0] * zero_num


# takes time step, probability of uptake, and number of walkers at z = 0 and moves walkers
def go(t, p, n):
    walkers = [0] * n

    for i in xrange(t):
    for i in xrange(len(walkers)):
    pos = walkers[i]
    walkers[i] = move_walker(pos)
```

```
	walkers = delete_walkers(walkers, p, n)
	return walkers

def plot_it(t, p, n, b):

	plt.hist(go(t, p, n), bins = b)
	plt.xlabel('Position')
	plt.ylabel('Carbon Dioxide Molecules at Position')
	plt.title(('Uptake of Carbon Dioxide in Microbial Mat p = '+ str(p)))

plot_it(1000, .5, 100, 4)
```

Problem 2

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


def matrix(r):
	mat = np.zeros((r, r))

	for i in xrange(r):
	for j in xrange(r):
	if i == 0:
		mat[i][j] = 0
	elif i == (r - 1) and j == (r - 1):
		mat[i][j] = 1
	elif j == (i + 1) or j == (i - 1):
		if i != (r - 1):
		mat[i][j] = .5
		else:
		pass
	else:
		pass
	return mat

def do_it(r, t):

	t_0 = np.array([1] * r)
```

```
        A = np.linalg.matrix_power(matrix(r), t)
        b = np.dot(A, np.transpose(t_0))

        return b

def build_heatplot(r, t):

        a = do_it(r, t)
        matrix = [a] * r
        matrix = np.array(matrix)

        return matrix.T

def find_flux(r, t):

        a = do_it(r, t)
        flux = []

        for i in range(len(a)):
        if i + 1 >= len(a):
        return flux
        else:
        flux.append(a[i + 1] - a[i])
        return flux

print find_flux(20, 1000)

a = build_heatplot(20, 1000)
plt.imshow(a, cmap='hot', interpolation='nearest')
plt.title('Temperature of Seafloor Slab t = 1000')
plt.xlabel('X Position')
plt.ylabel('Z Position')
plt.colorbar(cmap = 'hot')
plt.show()
```

Problem 3

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

```
matdata = scipy.io.loadmat("Fracture.mat")
data_raw = matdata['Fracture']

def get_const_points(data):

    const = (data == 1)
    return const

def build_usable_data(data):

    new = np.copy(data)

    for i in range(len(data)):
    for j in range(len(data)):
    if i == (len(data) - 1):
        new[i][j] = 1
    elif data[i][j] == 1:
        new[i][j] = 0
    else:
        new[i][j] = 1

    new = new.astype(np.float32)
    return new

def average(data, const):

    dat = np.copy(data)
    for i in xrange(1, data.shape[0] - 1):

    j = 0
    av = ((data[i + 1][j] + data[i - 1][j] + data[i][j + 1])/3.0)
    dat[i][j] = av

    j = len(data) -1

    av = ((data[i + 1][j] + data[i - 1][j] + data[i][j - 1])/3.0)
    dat[i][j] = av

    dat[1:-1, 1:-1] = (data[2:, 1:-1] + data[:-2, 1:-1] + data[1:-1, 2:] + data[1:-1, :-2])/4.0
    dat[const] = 0
    dat[0, :] = 0
    dat[-1, :] = 1
```

```
    return dat


def loop_average(data, t):

    const_points = get_const_points(data)
    data = build_usable_data(data)
    dat = np.copy(data)

    for k in range(t):
    m = average(dat, const_points)
    dat = m

    return dat

def find_flux_profile(data):
    dat = np.copy(data)

    for i in xrange(1, data.shape[0] - 1):
    j = 0
    flux = np.sqrt((data[i + 1][j] - data[i - 1][j])**2 + (data[i][j + 1])**2)
    dat[i, j] = flux

    j = len(data) - 1
    flux = np.sqrt((data[i + 1][j] - data[i - 1][j])**2 + (data[i][j - 1])**2)
    dat[i, j] = flux

    dat[1:-1, 1:-1] = ((data[2:, 1:-1] - data[:-2, 1:-1])**2 + (data[1:-1, 2:] - data[1:-1, :-2])**2)

    return dat

processed_data = loop_average(data_raw, 30000)
flux = find_flux_profile(processed_data)


plt.imshow(flux, cmap='hot', interpolation='nearest')
plt.colorbar(cmap = 'hot')
plt.xlabel('X Position')
plt.ylabel('Z Position')
plt.title('Magnitude of Flux for Fractured Seafloor Slab')
plt.show()
```

Problem 4a

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


def this_time(t, repeat):
	yes = 0.0

	for r in xrange(repeat):
	pos = 0
	for i in xrange(t):
	pos += np.random.choice([-1, 1])
	if pos == 0:
		yes += 1.0
		break
	else:
		pass

	return (1.0- (yes/(float(repeat))))

def all_times(t, repeat):
	probs = []

	for i in range(t):
	probs.append(this_time(i, repeat))

	return probs

y = all_times(100, 500)

plt.plot(y, label = "Simulated Probability Data")
plt.title('Probability of a Random Walker Never Returning to Origin in 1D')
plt.xlabel('Time Steps')
plt.ylabel('Probabilty')
plt.xscale('log')
plt.yscale('log')
plt.legend()
```

Problem 4c

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

def this_time(t, repeat):
	yes = 0.0

	for r in xrange(repeat):
	posx = 0
	posy = 0
	for i in xrange(t):
	if np.random.choice([-1, 1]) == 1:
		posx += np.random.choice([-1, 1])
	else:
		posy += np.random.choice([-1, 1])

	if posx == 0 and posy == 0:
		yes += 1.0
		break
	else:
		pass

	return (1.0- (yes/(float(repeat))))

def all_times(t, repeat):
	probs = []

	for i in range(t):
	probs.append(this_time(i, repeat))

	return probs

y = all_times(200, 500)
x = range(200)


plt.plot(y, label = "Simulated Probability Data")
plt.title('Probability of a Random Walker Never Returning to Origin in 2D')
plt.xlabel('Time Steps')
plt.ylabel('Probabilty')
plt.xscale('log')
plt.yscale('log')
```

```
plt.legend()
```