ISTA 421 / INFO 521 - Homework 1

Charles Kenneth "Ken" Youens-Clark

Graduate

Note: I discussed problems 3-5 with TA Farig Sadeque and students Matthew Miller and Kai Blumberg.

1. [0 points] Python setup.

2. [1 point] **Exercise 1.1** from FCMA p.35

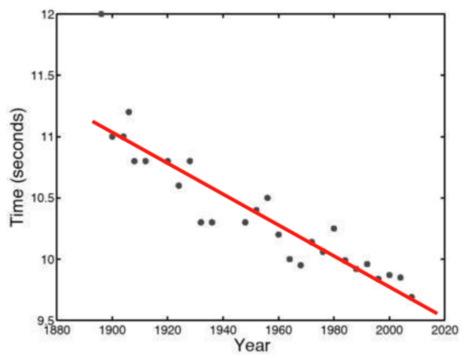

Figure 1: Reproduction of figure 1.1, Olympic men's 100m data

By examining Figure 1.1 [from p. 2 of FCMA, reproduced here], estimate (by hand / in your head) the kind of values we should expect for w_0 (y-intercept) and w_1 (slope) as parameters of a line fit to the data (e.g., High? Low? Positive? Negative?). (No computer or calculator calculation is needed here – just estimate!)

Solution.

I added a red line to the image above to indicate the kind of slope I would expect. Y intercept would be around 11.25, slope drops about 0.5 seconds every 20 years, so -20/0.5 = -40.

3. [2 points] **Exercise 1.3** from FCMA p.35

Show that:

$$\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} = w_0^2 \left(\sum_{n=1}^{N} x_{n1}^2 \right) + 2w_0 w_1 \left(\sum_{n=1}^{N} x_{n1} x_{n2} \right) + w_1^2 \left(\sum_{n=1}^{N} x_{n2}^2 \right),$$

where

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix}.$$

Solution.

$$\mathbf{X}^\top = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{12} & x_{22} & x_{32} & \dots & x_{n2} \end{bmatrix}$$

$$\mathbf{X}^\top\mathbf{X} = \begin{bmatrix} \sum_{n=1}^N x_{n1}^2 & \sum_{n=1}^N x_{n1}x_{n2} \\ \sum_{n=1}^N x_{n2}x_{n1} & \sum_{n=1}^N x_{n2}^2 \end{bmatrix}$$

$$\mathbf{w}^\top = \begin{bmatrix} w_0 & w_1 \end{bmatrix}$$

$$\mathbf{w}^\top\mathbf{X}^\top\mathbf{X} = \begin{bmatrix} w_0\sum_{n=1}^N x_{n1}^2 + w_1\sum_{n=1}^N x_{n2}x_{n1} & w_0\sum_{n=1}^N x_{n1}x_{n2} + w_{\sum_{n=1}^N x_{n2}^2} \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w} = w_0^2\left(\sum_{n=1}^N x_{n1}^2\right) + w_0w_1\left(\sum_{n=1}^N x_{n2}x_{n1}\right) + w_0w_1\left(\sum_{n=1}^N x_{n1}x_{n2}\right) + w_1^2\left(\sum_{n=1}^N x_{n2}^2\right)$$

$$\mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w} = w_0^2\left(\sum_{n=1}^N x_{n1}^2\right) + 2w_0w_1\left(\sum_{n=1}^N x_{n1}x_{n2}\right) + w_1^2\left(\sum_{n=1}^N x_{n2}^2\right)$$

4. [1 point] **Exercise 1.4** from FCMA p.35

   Using $\mathbf{w}$ and $\mathbf{X}$ as defined in the previous exercise, show that $(\mathbf{X}\mathbf{w})^\top = \mathbf{w}^\top\mathbf{X}^\top$ by multiplying out both sides.

   Solution.

   Given:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

   Then:

$$\mathbf{Xw} = \begin{bmatrix} x_{11}w_0 + x_{12}w_1 \\ x_{21}w_0 + x_{22}w_1 \\ x_{31}w_0 + x_{32}w_1 \\ \dots \\ x_{N1}w_0 + x_{N2}w_1 \end{bmatrix}$$
$$(\mathbf{Xw})^\top = \begin{bmatrix} x_{11}w_0 + x_{12}w_1 & x_{21}w_0 + x_{22}w_1 & x_{31}w_0 + x_{32}w_1 & \dots & x_{N1}w_0 + x_{N2}w_1 \end{bmatrix}$$

Given:

$$\mathbf{w}^\top = \begin{bmatrix} w_0 & w_1 \end{bmatrix}, \mathbf{X}^\top = \begin{bmatrix} x_{11} & x_{21} & x_{31} & \dots & x_{N1} \\ x_{12} & x_{22} & x_{32} & \dots & x_{N2} \end{bmatrix}$$

Then:

$$\mathbf{w}^\top \mathbf{X}^\top = \begin{bmatrix} w_0x_{11} + w_1x_{12} & w_0x_{21} + w_1x_{22} & w_0x_{31} + w_1x_{32} & \dots & w_0x_{N1} + w_1x_{N2} & \end{bmatrix}$$

Therefore $(\mathbf{Xw})^\top = \mathbf{w}^\top \mathbf{X}^\top$

5. [2 points] **Exercise 1.5** from FCMA p.35

When multiplying a scalar by a vector (or matrix), we multiply each element of the vector (or matrix) by that scalar. For $\mathbf{x}_n = [x_{n1}, x_{n2}]^\top$, $\mathbf{t} = [t_1, ..., t_N]^\top$, $\mathbf{w} = [w_0, w_1]^\top$, and

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1{}^\top \\ \mathbf{x}_2{}^\top \\ \vdots \\ \mathbf{x}_N{}^\top \end{bmatrix}$$

show that

$$\sum_n \mathbf{x}_n t_n = \mathbf{X}^\top \mathbf{t}$$

and

$$\sum_n \mathbf{x}_n \mathbf{x}_n^\top \mathbf{w} = \mathbf{X}^\top \mathbf{X} \mathbf{w}$$

Solution Part 1

Given:

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}$$
$$\mathbf{X}^\top = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \dots & \mathbf{x}_N \end{bmatrix}$$
$$\mathbf{X}^\top \mathbf{t} = \mathbf{x}_1 t_1 + \mathbf{x}_2 t_2 + \mathbf{x}_3 t_3 \dots \mathbf{x}_N t_N$$
$$= \sum_n \mathbf{x}_n t_n$$

Solution Part 2

$$\sum_n \mathbf{x}_n \mathbf{x}_n^\top \mathbf{w} = \mathbf{X}^\top \mathbf{X} \mathbf{w}$$

Divide both sides by $\mathbf{w}$:

$$\sum_n \mathbf{x}_n \mathbf{x}_n^\top = \mathbf{X}^\top \mathbf{X}$$

Solve the right side given:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1{}^\top \\ \mathbf{x}_2{}^\top \\ \mathbf{x}_3{}^\top \\ \vdots \\ \mathbf{x}_N{}^\top \end{bmatrix}$$
$$\mathbf{X}^\top = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \dots & \mathbf{x}_N \end{bmatrix}$$

Then:

$$\begin{aligned} \mathbf{X}^\top \mathbf{X} &= \begin{bmatrix} \mathbf{x}_1 \mathbf{x}_1{}^\top + \mathbf{x}_2 \mathbf{x}_2{}^\top + \mathbf{x}_3 \mathbf{x}_3{}^\top + \dots \mathbf{x}_N \mathbf{x}_N{}^\top \end{bmatrix} \\ &= \sum_n \mathbf{x}_n \mathbf{x}_n{}^\top \end{aligned}$$

Therefore $\sum_n \mathbf{x}_n \mathbf{x}_n^\top = \mathbf{X}^\top \mathbf{X}$

6. [5 points] Reading and Displaying Numpy Arrays:

Solution.

```
cat -n hw1.1.py
#!/usr/bin/env python3
"""
Assignment: INFO521 HW1
Date:       31 Aug 2018
Author:     Ken Youens-Clark
"""

import numpy as np
import matplotlib.pyplot as plt

dat = np.loadtxt("../data/humu.txt")
print('type = {}'.format(type(dat)))
print('size = {}'.format(dat.size))
print('shape = {}'.format(dat.shape))
```

```
print('max = {}'.format(dat.max()))  # also np.amax(dat)
print('min = {}'.format(dat.min()))  # also np.amin(dat)

scaled = dat / dat.max()
print('scaled min = {} max = {} shape = {}'.format(scaled.min(),
                                                   scaled.max(),
                                                   scaled.shape))

plt.figure()
plt.imshow(dat)
plt.show()

print(plt.cm.cmapname)

plt.imshow(dat, cmap='gray')
plt.show()

outfile = 'random.png'
for _ in range(0, 2):
    ran = np.random.random(dat.shape)
    plt.imshow(ran)
    plt.show()
    np.savetxt(outfile, ran)

ran1 = np.loadtxt(outfile)
plt.imshow(ran1)
plt.savefig('random.png')
plt.show()

print('Done.')
```

```
$ ./hw1.1.py
type = <class 'numpy.ndarray'>
size = 210816
shape = (366, 576)
max = 0.9450980392156862
min = 0.0
scaled min = 0.0 max = 1.0 shape = (366, 576)
tab20c_r
Done.
```

The humuhumunukunukuapua'a is the reef trigger fish and the state fish of Hawai'i.

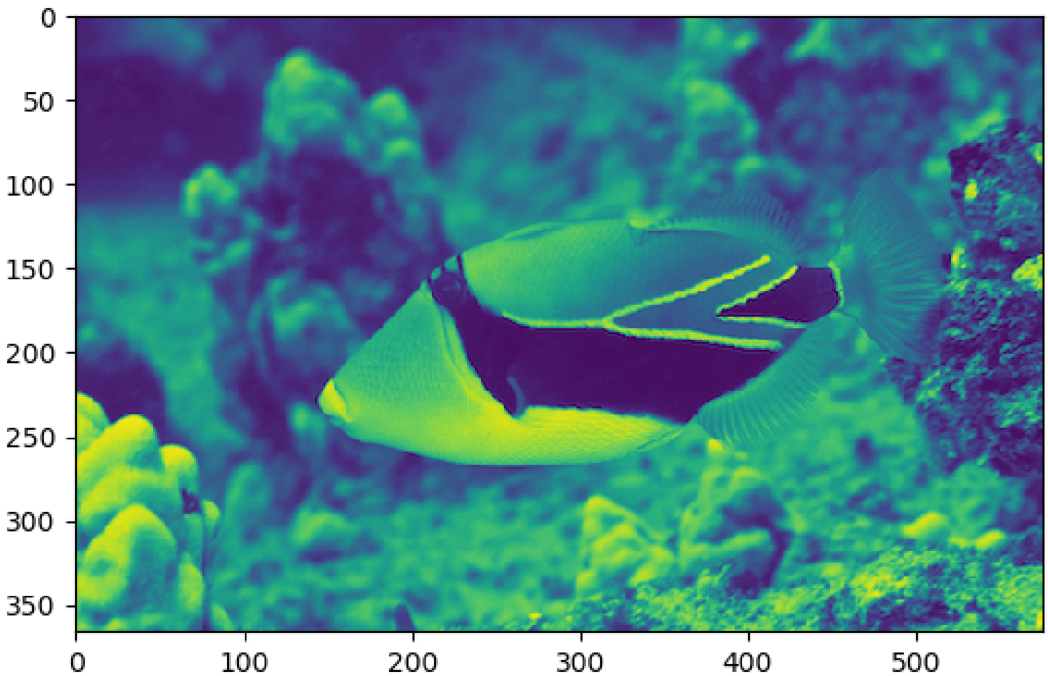

Figure 2: Humu Color

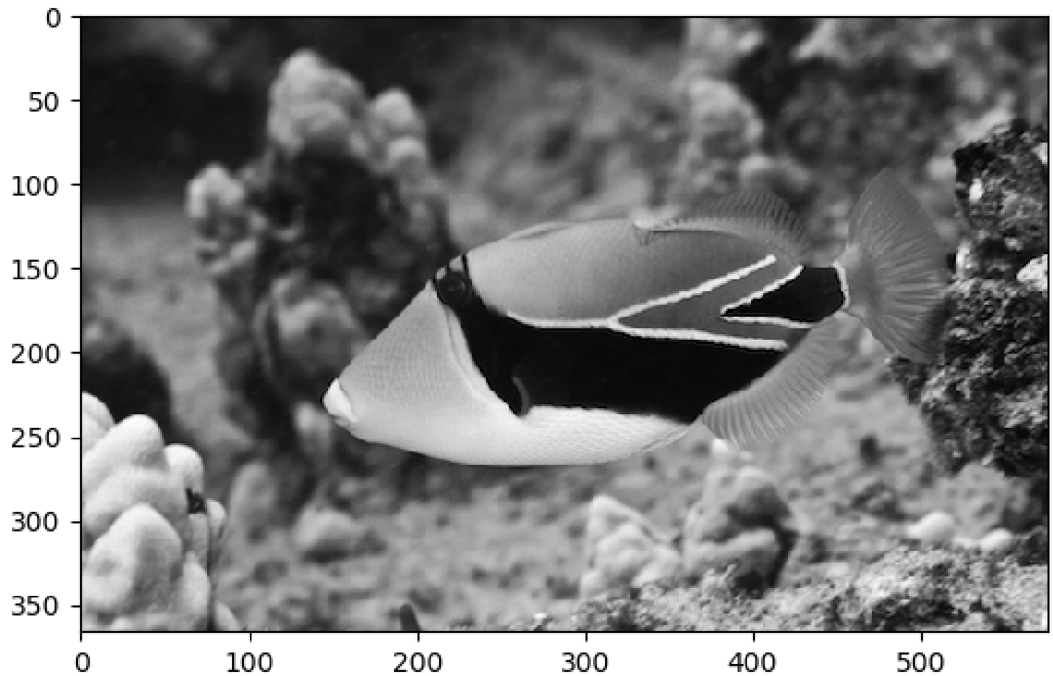

Figure 3: Humu Greyscale

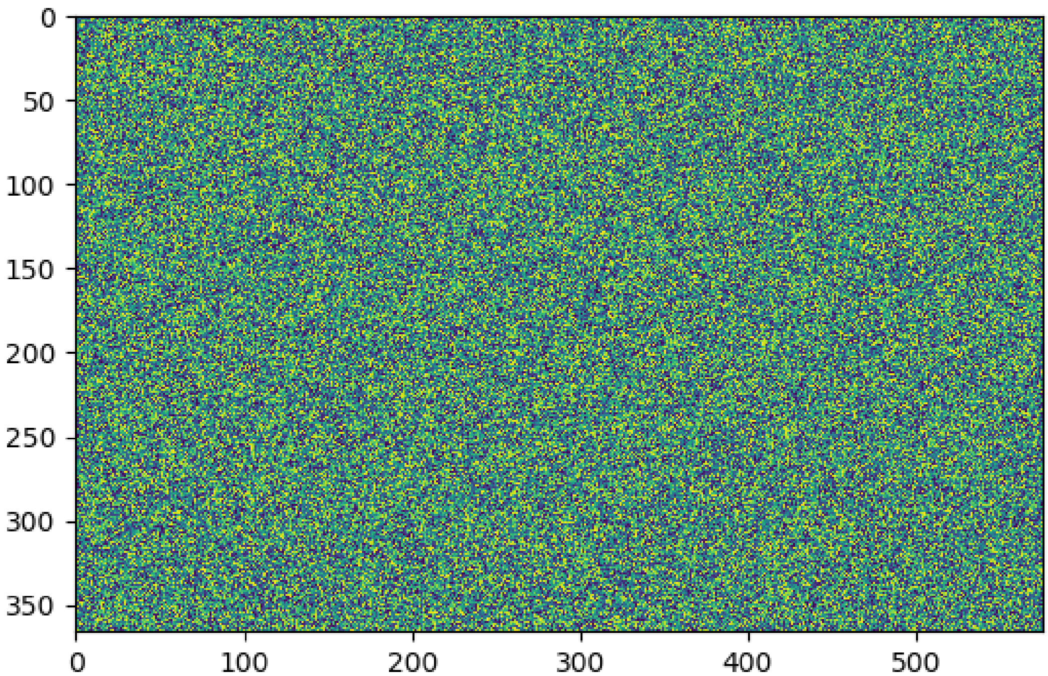

Figure 4: Random Image

PART B

Solution.

```
$ cat -n walk.py
#!/usr/bin/env python3

"""
walk.py - manipulate and plot "walk.txt"
Ken Youens-Clark
27 August 2018
"""

import numpy as np
import matplotlib.pyplot as plt


# --------------------------------------------------
def main():
    """
    main()
    """
    dat = np.loadtxt('../data/walk.txt')
    print('Data  : min "{:5}, max "{:5}", shape "{}"'.format(
        dat.min(), dat.max(), dat.shape))

    abs_min = np.abs(dat.min())
    scaled = (dat + abs_min) / (dat.max() + abs_min)
    print('Scaled: min "{:5}, max "{:5}", shape "{}"'.format(
        scaled.min(), scaled.max(), scaled.shape))

    outfile = '../data/walk_scale01.txt'
    np.savetxt(outfile, scaled)
    print('Scaled data saved to "{}"'.format(outfile))

    plot_1d_array(arr=dat, title='Original', outfile='walk.png')
    plot_1d_array(arr=scaled, title='Scaled', outfile='walk_scaled.png')


# --------------------------------------------------
def plot_1d_array(arr, title=None, outfile=None):
    """
    Plot a 1D array
    :param: arr - a 1D Numpy array
    :param: title - figure title (str)
    :param: outfile - path to write image (str)

    :return: void
    """

    plt.figure()
    if title:
```

```
        plt.title(title)
    plt.plot(arr)

    if outfile:
        plt.savefig(outfile)
        print('Wrote to "{}"'.format(outfile))

    plt.show()


# --------------------------------------------------
if __name__ == '__main__':
    main()
```

```
$ ./walk.py
Data  : min " -1.0, max "  5.0", shape "(200,)"
Scaled: min "  0.0, max "  1.0", shape "(200,)"
Scaled data saved to "../data/walk_scale01.txt"
Wrote to "walk.png"
Wrote to "walk_scaled.png"
```

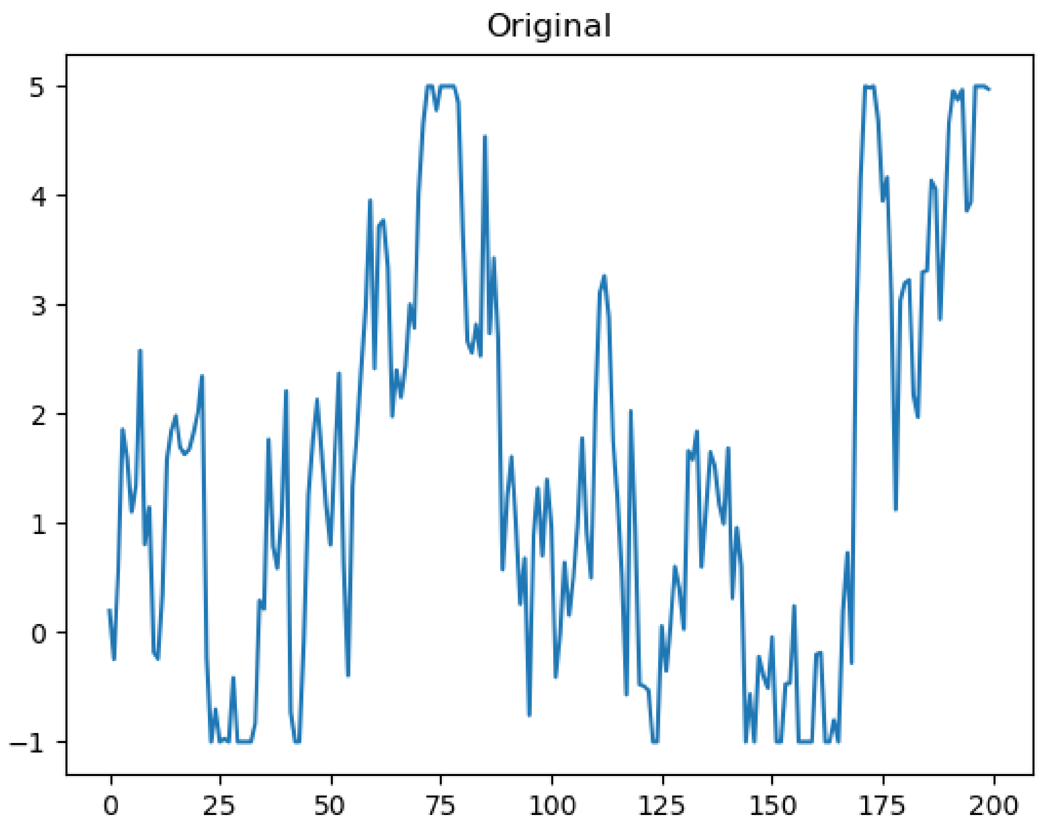

Figure 5: Plot of original walk data

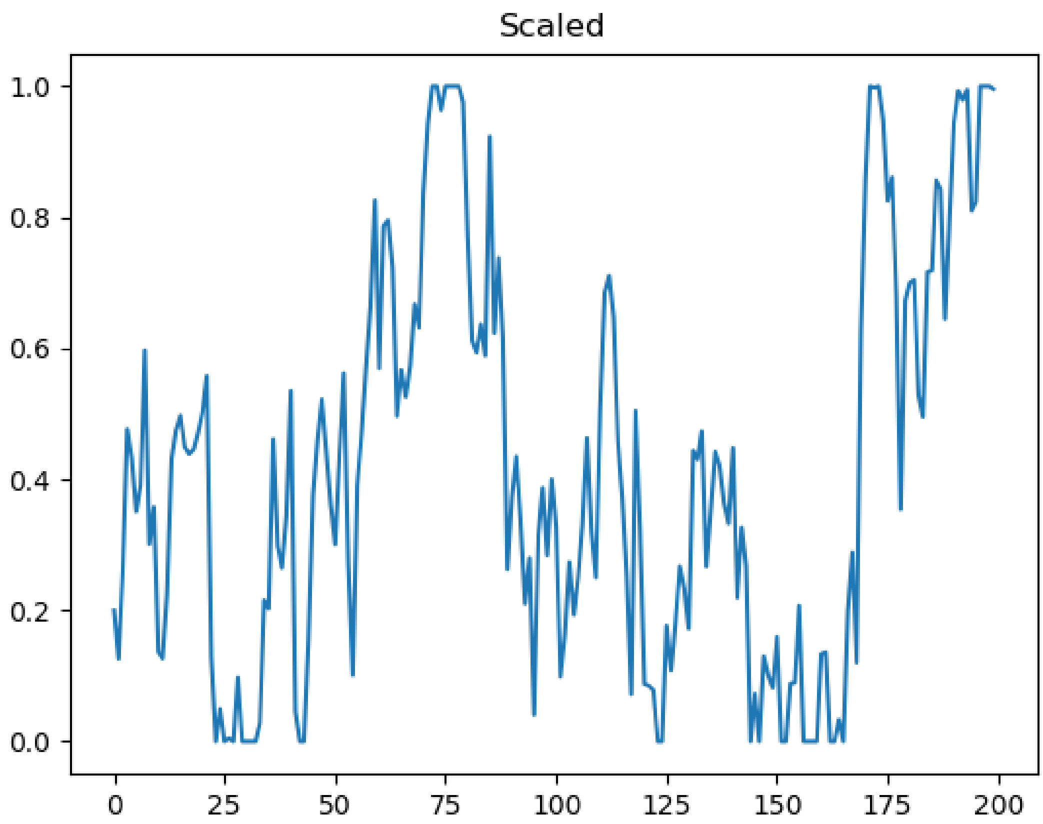

Figure 6: Plot of scaled walk

7. [1 point] Functions:

Solution.

```
def main():
    """
    main
    :param: none
    :return: void
    """
    exercise6("../data/humu.txt", "out.txt")

def scale01(arr):
    """
    Linearly scale the values of an array in the range [0,1]
    :param arr: input ndarray
    :return: scaled ndarray
    """
    return arr / arr.max()

def exercise6(infile, outfile):
    """
    Read a file into a Numpy ndarray
    :param infile: the file to read
    :param outfile: where to write the output
    :return: void
    """
    dat = np.loadtxt(infile)

    scaled = scale01(dat)
    print('scaled min = {} max = {} shape = {}'.format(scaled.min(),
                                                       scaled.max(),
                                                       scaled.shape))

    plt.figure()
    plt.imshow(dat)
    plt.show()

    print(plt.cm.cmapname)

    plt.imshow(dat, cmap='gray')
    plt.show()

    for _ in range(0, 2):
        ran = np.random.random(dat.shape)
        plt.imshow(ran)
        plt.show()
        np.savetxt(outfile, ran)

    ran1 = np.loadtxt(outfile)
    plt.imshow(ran1)
```

```
    plt.show()

    print('Done.')

if __name__ == '__main__':
    main()

$ ./hw1.py
type = <class 'numpy.ndarray'>
size = 210816
shape = (366, 576)
max = 0.9450980392156862
min = 0.0
scaled min = 0.0 max = 1.0 shape = (366, 576)
tab20c_r
Done.
```

8. [2 points] Documenting:

   Functions documented above

9. [2 points] Random Numbers:

   Solution.

   I wrote the following function:

```
def exercise9():
    """
    Estimate the randomness of throwing double-sixes
    :param: none
    :return: void
    """

    np.random.seed(seed=8)

    throws = 1000
    dbl6 = 0
    for _ in range(0, throws):
        (n1, n2) = np.random.randint(low=1, high=7, size=2, dtype=int)
        #print('{} {}'.format(n1, n2))
        if n1 == 6 and n2 == 6:
            dbl6 += 1

    print('Threw double-six {:.2}%'.format((dbl6 / throws) * 100))
```

   Everytime I run it, I get the same result, "2.6%".

```
$ ./hw1.py
Threw double-six 2.6%
$ ./hw1.py
Threw double-six 2.6%
```

   If I comment out `seed=8`, I will get different results:

```
$ for i in $(seq 1 9); do echo -n "$i: " && ./hw1.py; done
1: Threw double-six 2.7%
2: Threw double-six 2.4%
3: Threw double-six 3.4%
4: Threw double-six 2.1%
5: Threw double-six 2.6%
6: Threw double-six 2.6%
7: Threw double-six 2.3%
8: Threw double-six 3.2%
9: Threw double-six 3.2%
```

Putting back in `seed=8`, I go back to the same result:

```
$ for i in $(seq 1 10); do echo -n "$i: " && ./hw1.py; done
1: Threw double-six 2.6%
2: Threw double-six 2.6%
3: Threw double-six 2.6%
4: Threw double-six 2.6%
5: Threw double-six 2.6%
6: Threw double-six 2.6%
7: Threw double-six 2.6%
8: Threw double-six 2.6%
9: Threw double-six 2.6%
10: Threw double-six 2.6%
```

Explain why it is often important to have random number sequences that are not really random, and can be controlled

Being able to count on a "random" number allows one to write tests for such functions.

The generation of random numbers is too important to be left to chance. – Robert R. Coveyou

10. [5 points] Random Numbers, Vectors, Matrices, and Operations

Solution 10a.

```
def exercise10a():
    """
    Print two three-dimensional column vectors
    :param: none
    :return: void
    """

    np.random.seed(seed=5)
    a = np.random.rand(3, 1)
    b = np.random.rand(3, 1)
    print(a)
    print(b)

$ ./hw1.py
[[0.22199317]
 [0.87073231]
 [0.20671916]]
[[0.91861091]
 [0.48841119]
```

```
 [0.61174386]]
```

Solution 10b.

```
def exercise10b():
    """
    Print two three-dimensional column vectors
    :param: none
    :return: void
    """

    np.random.seed(seed=5)
    a = np.random.rand(3, 1)
    b = np.random.rand(3, 1)
    print("a\n {}".format(a))
    print("b\n{}".format(b))
    print("a + b\n{}".format(a + b))
    print("a * b\n{}".format(a * b))
    print("aT . b\n{}".format(a.transpose().dot(b)))
```

```
a
 [[0.22199317]
 [0.87073231]
 [0.20671916]]
b
[[0.91861091]
 [0.48841119]
 [0.61174386]]
a + b
[[1.14060408]
 [1.35914349]
 [0.81846302]]
a * b
[[0.20392535]
 [0.4252754 ]
 [0.12645917]]
aT . b
[[0.75565992]]
```

$$\begin{aligned}
\mathbf{a} &= \begin{bmatrix}0.22199317 & 0.87073231 & 0.20671916\end{bmatrix} \\
\mathbf{b} &= \begin{bmatrix}0.91861091 & 0.48841119 & 0.61174386\end{bmatrix} \\
\mathbf{a}+\mathbf{b} &= \begin{bmatrix}1.14060408 & 1.35914349 & 0.81846302\end{bmatrix} \\
\mathbf{a}\circ\mathbf{b} &= \begin{bmatrix}0.20392535 & 0.4252754 & 0.12645917\end{bmatrix} \\
\mathbf{a}^\top\mathbf{b} &= 0.75565992
\end{aligned} \tag{1}$$

Solution 10c.

```
def exercise10c():
    """
    Vector/matrix manipulation
    :param: none
```

```
    :return: void
    """

    np.random.seed(seed=5)
    a = np.random.rand(3, 1)
    b = np.random.rand(3, 1)

    np.random.seed(seed=2)
    X = np.asmatrix(np.random.rand(3, 3))

    print("a\n {}".format(a))
    print("b\n{}".format(b))
    print("X\n{}".format(X))
    print("aTX\n{}".format(a.transpose() * X))
    print("aTXb\n{}".format(a.transpose() * X * b))
    print("X-1\n{}".format(X.getI()))
a
 [[0.22199317]
 [0.87073231]
 [0.20671916]]
b
[[0.91861091]
 [0.48841119]
 [0.61174386]]
X
[[0.4359949  0.02592623 0.54966248]
 [0.43532239 0.4203678  0.33033482]
 [0.20464863 0.61927097 0.29965467]]
aTX
[[0.51814195 0.49979844 0.47159888]]
aTXb
[[1.00857572]]
X-1
[[-1.20936675  5.11771977 -3.42333228]
 [-0.96691719  0.279414    1.46561347]
 [ 2.82418088 -4.07257903  2.64627411]]
```

$$
\begin{aligned}
\mathbf{a} &= \begin{bmatrix} 0.22199317 & 0.87073231 & 0.20671916 \end{bmatrix} \\
\mathbf{b} &= \begin{bmatrix} 0.91861091 & 0.48841119 & 0.61174386 \end{bmatrix} \\
\mathbf{X} &= \begin{bmatrix} 0.4359949 & 0.02592623 & 0.54966248 \\ 0.43532239 & 0.4203678 & 0.33033482 \\ 0.20464863 & 0.61927097 & 0.29965467 \end{bmatrix} \\
\mathbf{a}^\top \mathbf{X} &= \begin{bmatrix} 0.51814195 & 0.49979844 & 0.47159888 \end{bmatrix} \\
\mathbf{a}^\top \mathbf{X} \mathbf{b} &= 1.00857572 \\
\mathbf{X}^{-1} &= \begin{bmatrix} 1.20936675 & 5.11771977 & -3.42333228 \\ -0.96691719 & 0.279414 & 1.46561347 \\ 2.82418088 & -4.07257903 & 2.64627411 \end{bmatrix}
\end{aligned}
\tag{2}
$$

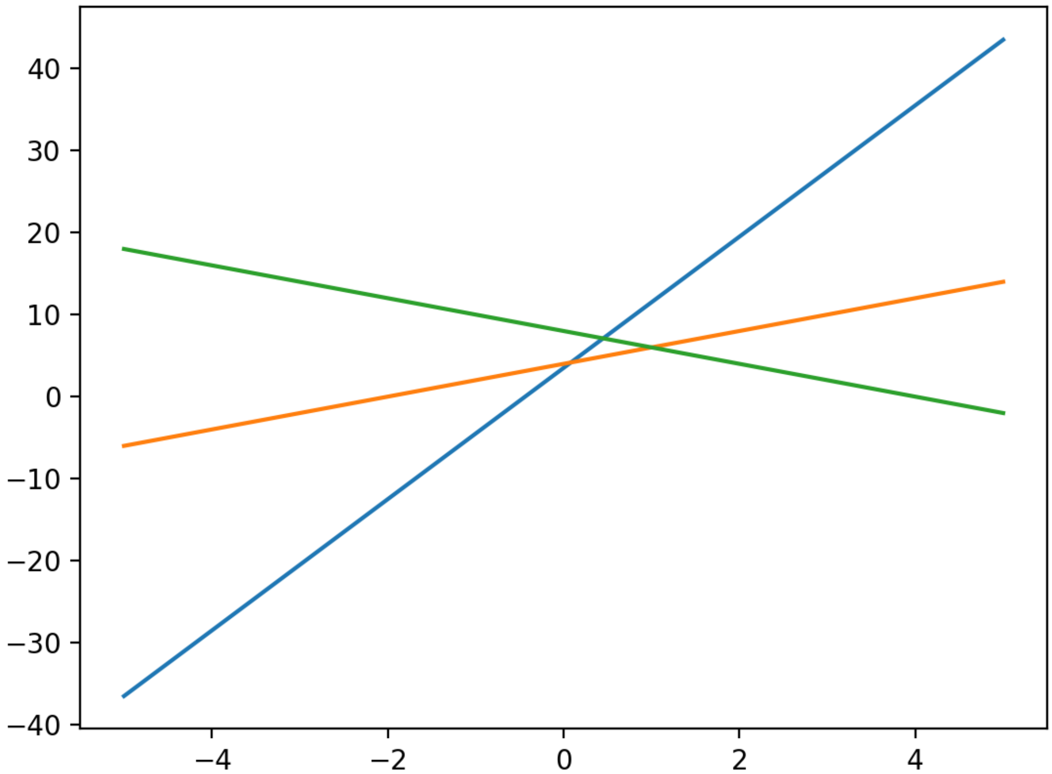

Figure 7: y = 3.5 + 8.0x; y = 4.0 + 2.0x; y = 8.0 + -2.0x

11. [3 points] Simple Plotting

Solution11a.

$$\begin{aligned} y &= 3.5 + 8.0x \\ y &= 4.0 + 2.0x \\ y &= 8.0 + -2.0x \end{aligned} \tag{3}$$

Solution11b.

```
def exercise11():
    """
    Plotting
    :param: none
    :return: void
    """

    x = np.arange(0, 10, .01)
    y = np.sin(2 * np.pi * x)
```

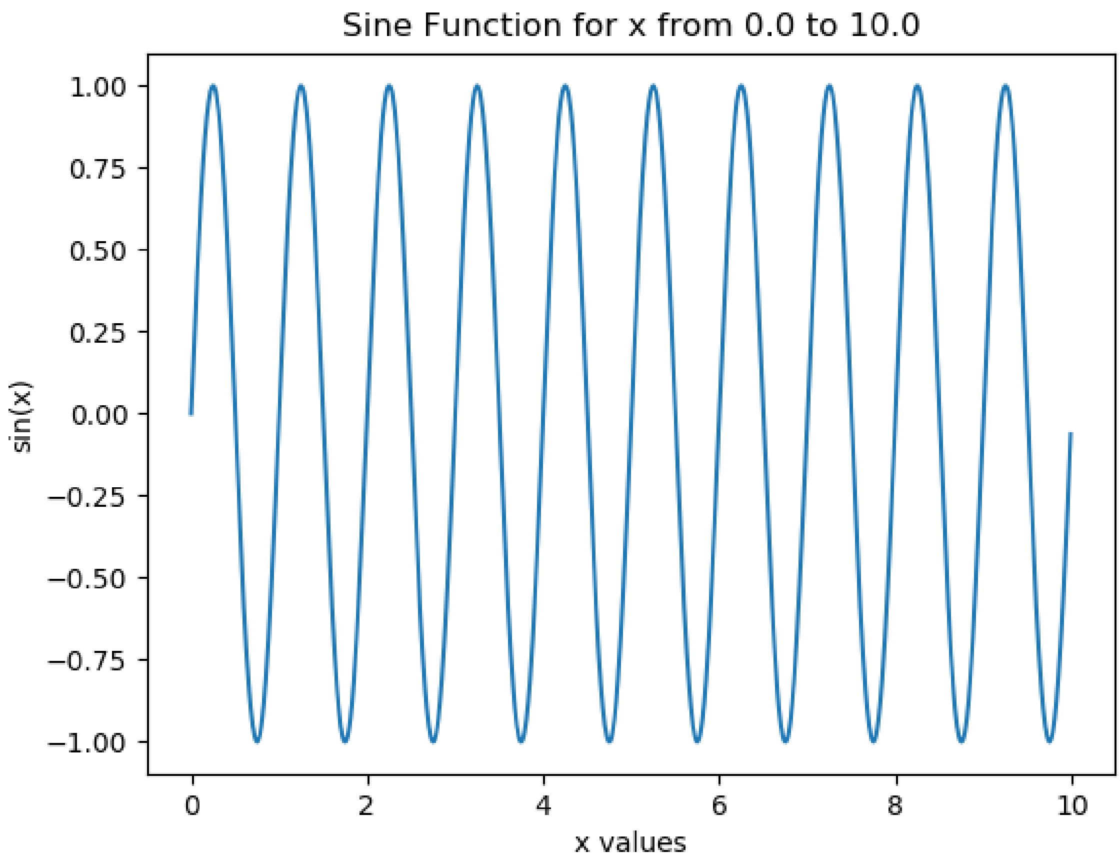

Figure 8: Plot sin(x)

```
plt.plot(x, y)
plt.title('Sine Function for x from 0.0 to 10.0')
plt.xlabel('x values')
plt.ylabel('sin(x)')
plt.show()
plt.savefig('sine.png')
```