Taylor Tanita
A11216947

COGS 118A: Assignment 3

1 Entropy and Mutual Information

1) Compute the entropy of X, subject to P(X), that is, compute H(X)
2) Compute the entropy of Y, subject to P(Y), that is, compute H(Y)
3) Compute the entropy of X, subject to P(X|Y), that is, compute H(X|Y)
4) Compute the entropy of Y, subject to P(Y|X), that is compute H(Y|X)
5) Compute the mutual information of X and Y, subject to P(X,Y), that is compute I(X;Y)

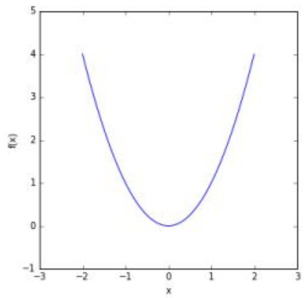
```
Taylors-MacBook-Pro-8:assignment3 taylortanita$ python ./a3_p1.py
1.1 Entropy of X subject to P(X): 1.37622660434
1.2 Entropy of Y subject to P(Y): 1.33508516509
1.3 Conditional Entropy of X subject to P(X|Y): 1.13154897977
1.4 Conditional Entropy of Y subject to P(Y|X): 1.17269041903
1.5 Mutual Entropy of X and Y subject to P(Y,X): 0.411723885998
```

```
from entropy import *

a = np.array([[.15,.03,.05,.07],[.02,.05,.03,.05],[.03,.2,.02,.1],[.05,.02,.1,.03]])
x = np.array([.25,.3,.2,.25])
y = np.array([.3,.15,.35,.2])

print '1.1 Entropy of X subject to P(X): ' + str(entropy(x,4))
print '1.2 Entropy of Y subject to P(Y): ' + str(entropy(y,4))
print '1.3 Conditional Entropy of X subject to P(X|Y): ' + str(conditional_entropy(a,4,4))
print '1.4 Conditional Entropy of Y subject to P(Y|X): ' + str(conditional_entropy(a.T,4,4))
print '1.5 Mutual Entropy of X and Y subject to P(Y,X): ' + str(mutual_entropy(a,4,4))
```

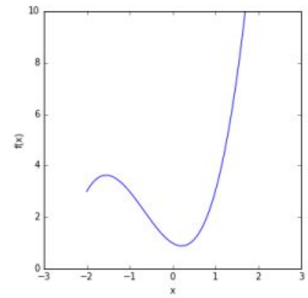
```
import numpy as np
import math

def entropy(values, num):
  rtn_val = 0
  for i in range(num):
    rtn_val = rtn_val + values[i]*math.log(values[i])
  return rtn_val*(-1)

def conditional_entropy(values, num_rows, num_col):
  rtn = 0
  for j in range(num_col):
    denominator = 0

    for i in range(num_rows):
      denominator = denominator + values[i,j]

    ent = 0
    for i in range(num_rows):
      x = values[i,j]/denominator
      ent = ent + x*math.log(x)

    rtn = rtn + denominator*ent
  return rtn*(-1)

def mutual_entropy(values, num_rows, num_col):
  rtn = 0
  x_sum = np.sum(values, axis=0)
```

```
y_sum = np.sum(values, axis=1)

for j in range(num_col):
  for i in range(num_rows):
    rtn = rtn + values[i,j]*math.log(values[i,j]/(x_sum[i]*y_sum[j]))
return rtn
```

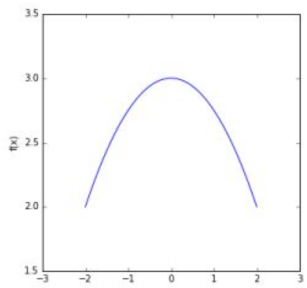
2 Convex



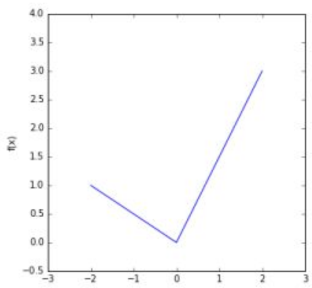
(a)



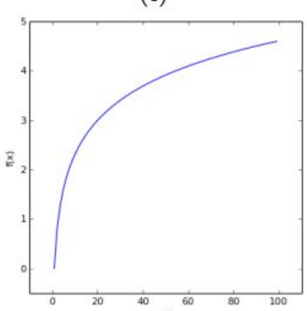
(b)



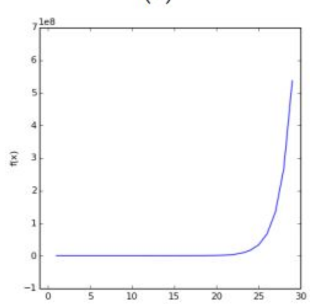
(c)



(d)



(e)

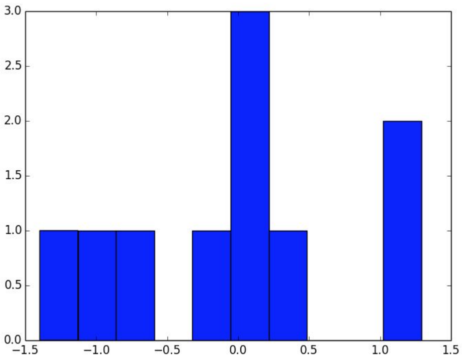


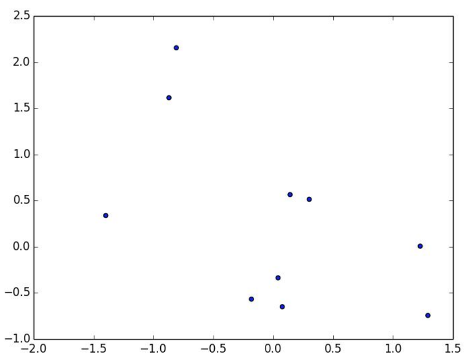
(f)

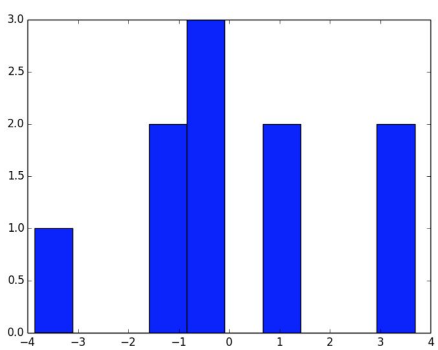
a) Convex
b) Non-convex
c) Non-convex
d) Convex
e) Non-convex
f) Convex

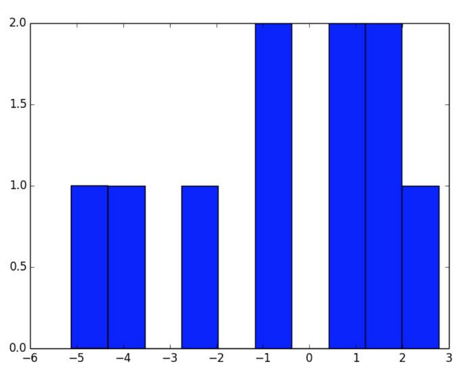
Taylor Tanita
A11216947

3 Normal Distribution









From this experiment, the following results can be interpreted

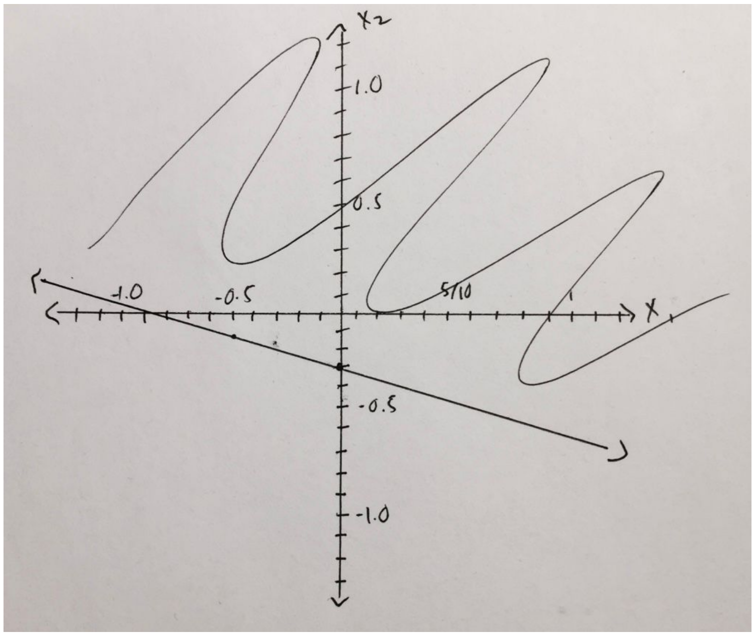
1. X is approximately symmetric and negatively skewed
2. X and Y are negatively correlated
3. X’ is approximately symmetric
4. Y’ is negatively skewed

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

x = np.random.randn(10)
plt.hist(x)
plt.savefig('x_hist.png')
plt.clf()

y = np.random.randn(10)
plt.scatter(x,y)
plt.savefig('scatter.png')
plt.clf()

xp = np.array([])
yp = np.array([])
for i in range(0,10):
  if i == 0:
    xp = np.array([3*x[i]+y[i]])
    yp = np.array([x[i]-2*y[i]])
  else:
    xp = np.append(xp, [3*x[i]+y[i]], axis=0)
    yp = np.append(yp, [x[i]-2*y[i]], axis=0)

plt.hist(xp)
plt.savefig('xp_hist.png')
plt.clf()

plt.hist(yp)
plt.savefig('yp_hist.png')
plt.clf()
```

Taylor Tanita
A11216947

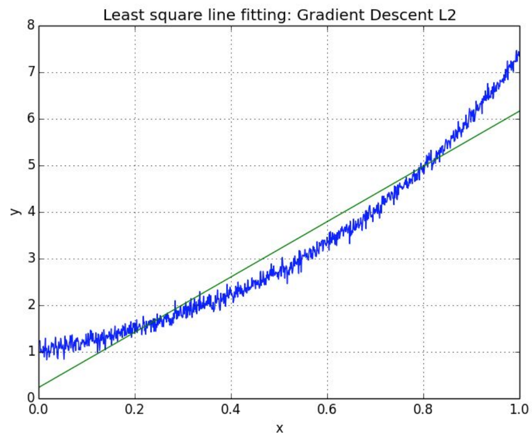
4 Decision Boundary

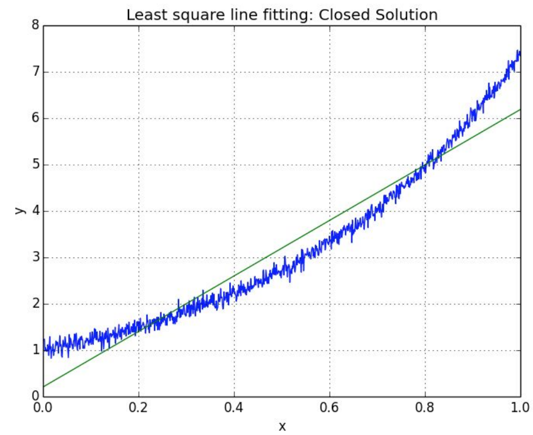
$$4x_1 + 10x_2 + 3 \geq 0$$
$$10x_2 \geq -4x_1 - 3$$
$$x_2 \geq -\frac{2}{5}x_1 - \frac{3}{10}$$



Taylor Tanita
A11216947

5 Least Square Estimation Via Gradient Descent





Dimension of $\frac{dg(W)}{dW}$: $(2,1)$
Dimension of W : $(2,1)$

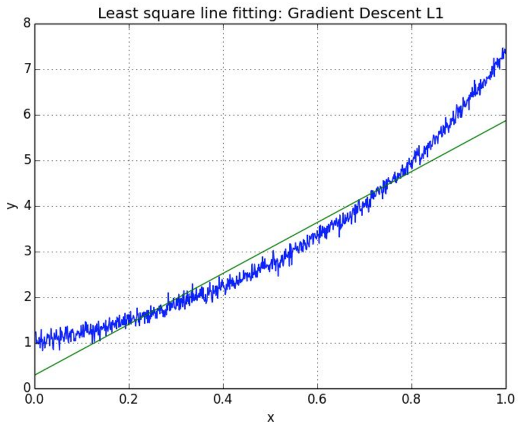
```
Taylors-MacBook-Pro-8:assignment3 taylortanita$ python ./a3_p56.py
dimension of derivative is (2,1)
dimension of W is (2,1)
Gradient Descent: w0: [ 0.20961483], w1: [ 5.97608281]
Closed Form: w0: [ 0.20961483], w1: [ 5.97608281]
```

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

data = sio.loadmat('data.mat')
x = data['x'].reshape([-1,1])
y = data['y'].reshape([-1,1])
X = np.hstack((np.ones((len(x),1)),np.power(x,1)))

w = np.array([[0.0],[0.0]])

## Number 5 ##
lamb = 0.0001
lim = 0.001

dif = 1
while (dif >= lim):
  a = 2*(np.dot(np.dot(X.transpose(),X),w))
  b = 2*np.dot(X.transpose(),y)
  deriv = a - b
  w_old = w
  w = w - lamb*deriv
# unsure as to how to compute dif
```

```
  dif = np.sum(np.absolute(w-w_old))

(n,m) = deriv.shape
(k,l) = w.shape
print('dimension of derivative is (' + str(n) + ',' + str(m) + ')')
print('dimension of W is (' + str(k) + ',' + str(l) + ')')

plt.plot(x,y)
plt.grid()
plt.hold(True)
plt.plot(x,w[0]+w[1]*x)
plt.title('Least square line fitting: Gradient Descent L2')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('prob5.png')
plt.clf()

(w_closed,_,_,_) = np.linalg.lstsq(X,y)
plt.plot(x,y)
plt.grid()
plt.hold(True)
plt.plot(x,w_closed[0]+w_closed[1]*x)
plt.title('Least square line fitting: Closed Solution')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('prob5_closed.png')
plt.clf()

print('Gradient Descent: w0: ' + str(w[0]) + ', w1: ' + str(w[1]))
print('Closed Form: w0: ' + str(w[0]) + ', w1: ' + str(w[1]))
```

Taylor Tanita
A11216947

6 L1 Distance



Using L1 as the error function did not produce identical w values as L2 did; however they are still similar. This also makes sense because L1 is less stable of a solution than L2 per wikipedia- mainly due to its resistance against outliers. However, the graphs produced in all three solutions were virtually identical- which should indicate the model was well represented.

```
Taylors-MacBook-Pro-8:assignment3 taylortanita$ python ./a3_p6.py
Gradient Descent L1: w0: [ 0.2897], w1: [ 5.580318]
```

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

data = sio.loadmat('data.mat')
x = data['x'].reshape([-1,1])
y = data['y'].reshape([-1,1])
X = np.hstack((np.ones((len(x),1)),np.power(x,1)))

size = X.size/2
dif = 1
lamb = 0.0001
lim = 0.001
w = np.array([[0.0],[0.0]])
while (dif >= lim):
  deriv = np.array([[0.0],[0.0]])
  for i in range(0,size):
    deriv = deriv +
np.sign(np.dot(X[i][np.newaxis],w)-y[i])*X[i][np.newaxis].transpose()
  w_old = w
  w = w - lamb*deriv
  dif = np.linalg.norm((w-w_old),1)

plt.plot(x,y)
plt.grid()
```

```
plt.hold(True)
plt.plot(x,w[0]+w[1]*x)
plt.title('Least square line fitting: Gradient Descent L1')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('prob6.png')
plt.clf()
print('Gradient Descent L1: w0: ' + str(w[0]) + ', w1: ' + str(w[1]))
```