DS5 HW1

1) a)

$$p(Y_1, Y_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[-\frac{z^*}{2(1-\rho^2)}\right]$$

$$z^* \equiv \frac{(Y_1-\mu_1)}{\sigma_1^2} - \frac{2\rho(Y_1-\mu_1)(Y_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(Y_2-\mu_2)^2}{\sigma_2^2}$$

$$\rho \equiv cor(Y_1, Y_2) = \frac{cov(Y_1, Y_2)}{\sigma\sigma^2}$$

b) For the continuous case and by Bayes theorem :

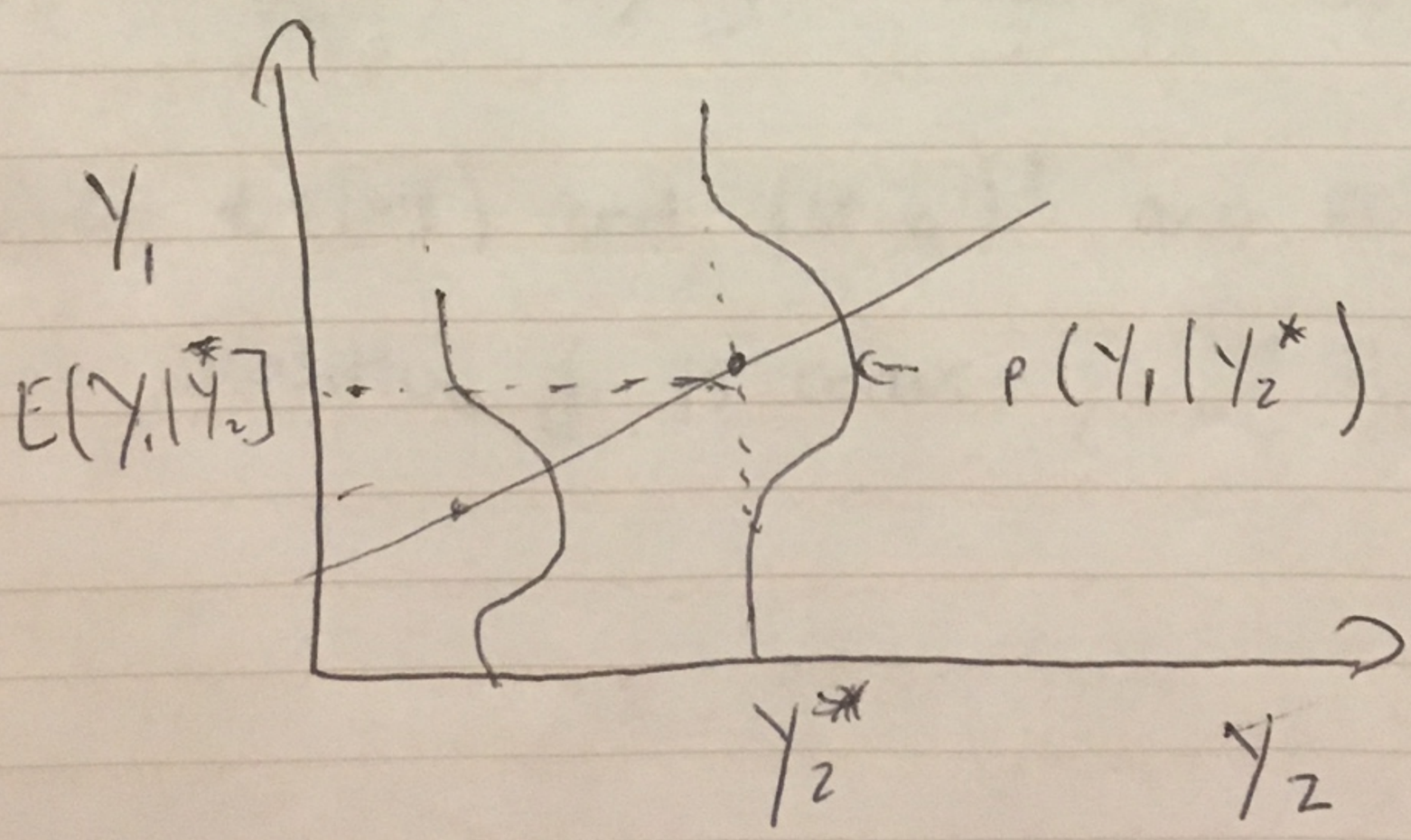
$$f_{y_1}(y_1|y_2) = \frac{f_{y_1,y_2}(y_1,y_2)}{f_{y_2}(y_2)} \leftarrow \text{joint}$$

$$= \frac{\frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[-\frac{z^*}{2(1-\rho^2)}\right]}{\frac{1}{2\pi\sigma_2} \exp\left[-\frac{(y_2-\mu_2)^2}{2\sigma_2}\right]} \leftarrow \text{marginal } \rho$$

$$= f_{y_1|y_2}(y_1|y_2) = \frac{1}{\sqrt{2\pi}\sqrt{1-\rho^2}\sigma_2} \exp\left[\frac{-\left(y_2 - \rho\frac{\sigma_2}{\sigma_1}y_1\right)^2}{(1-\rho^2)\sigma_2^2}\right]$$

1) c) The form of the conditional probability distribution $f_{Y_1|Y_2}(y_1|y_2)$ is relevant in the context of fitting linear regression using a maximum likelihood approach to the data where we wish to maximize $\prod_{i=1}^{N} p(y_1|y_2)$ which is achieved by finding the optimal parameters for the conditional expectation $E[Y_1|Y_2]$ and var $(Y_1|Y_2)$

It turns out that those values which minimize the $(y_2 - (\theta_0 - \theta_1 y_1)^2$ also maximize the total likelihood. In terms of the conditional, we also note that figure 1 is relevant



2) a| Sum of convex functions is convex; therefore we only prove the loss function for each individual term is convex to show sum is convex.

$\mathcal{L}_2 = f(\theta_0, \theta_1) = (y_1 - (\theta_0 + \theta_1 x_1))^2 \qquad u = \theta_0 + \theta_1 x$

proving convexity: $f(tu + (1-t)u') - tf(u) - (1-t)f(u') \leq 0$

$(y_1 - tu - (1-t)u' - tf(u) - (1-t)f(u') \leq 0$

$= (y_1 - tu - (1-t)u')^2 - t(y_1 - u)^2 - (1-t)(y_1 - u')^2 \leq 0$

$= y_1^2 - 2y_1tu - 2y_1(1-t)u' + t^2u^2 + 2t(1-t)uu' + (1-t)^2u'^2$

$- ty_1^2 + 2ty_1u - tu^2 - (1-t)y_1^2 + (1-t)2y_1u' - (1-t)u'^2$

$= tu^2(t-1) + (1-t)u^2(1-t-1) + 2t(1-t)uu'$

$= -t(1-t)\left[u^2 + u'^2 - 2uu'\right]$

$= -t(1-t)\left[(u-u')^2\right] \leq 0$, because

the $t(1-t)$ and $(u-u')^2$ are both necessarily positive

□ f is convex, $\mathcal{L}_2 = \sum f_i$ is convex

2b) Show that $L = |y_1 - \theta_0 - \theta_1 x_1|$ $\quad u = \theta_0 - \theta_1 x_1$

$f(u) = |y_1 - u|$ is convex.

$f(tu + (1-t)u')$

$= |y_1 - (tu + (1-t)u')|$

$= |ty_1 - tu + (1-t)y_1 - (1-t)u'|$

$\leq |ty_1 - tu| + |(1-t)y_1 - (1-t)u_1|$ (triangle inequality!)

$= t|y_1 - u| + (1-t)|y_1 - u'|$ (t and $(1-t)$ are positive)

$= t f(u) + (-t) f(u') \rightarrow$ convex

c) A condition to prove convexity is that the functions local minima are global minima. Given that we used individual term summation, we only prove:

$$L(\varepsilon) = \begin{cases} \frac{1}{2}\varepsilon^2 & \text{if } |\varepsilon| \leq \delta \\ \delta|\varepsilon| - \frac{1}{2}\delta^2 & |\varepsilon| > \delta \end{cases} \qquad \delta \in (0, \infty)$$

The second case gives two lines, whose mins are both $\ell = \frac{1}{2}\delta^2$, when they are continuous with the first case (all points $\leq \frac{1}{2}\delta^2$) then the minimum is $\varepsilon = 0$, which is singular + global

3 a) Analytic Solution:

$$\mathcal{L}(\theta_0, \theta_1) = \sum_i^N (y_i - (\theta_0 + \theta_1 x_i))^2$$

Minimize $\frac{\partial \mathcal{L}}{\partial \theta_0} = \sum_i^N 2(y_i - (\theta_0 + \theta_1 x_i)) = 0 \Rightarrow$

$$\sum_{i=1}^N \theta_0 = \sum_{i=1}^N y_i + \theta_1 x_i$$

$$N \cdot \theta_0 = \sum y_i + \theta_1 x_i$$

$$\boxed{\hat{\theta}_0 = \bar{y} - \theta_1 \bar{x}}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_1} = \sum_{i=1}^N 2(y_i - (\theta_0 + \theta_1 x_i)) \cdot -x_i = 0 \Rightarrow$$

$$* \sum_{i=1}^N x_i [(\bar{y} + \theta_1 \bar{x}) + \theta_1 x_i] = \sum_{i=1}^N y_i x_i$$

$$\theta_1 \sum_{i=1}^N \bar{x} x_i + x^2 = \sum_{i=1}^N y_i x_i - \bar{y} \cdot x_i$$

$$= \frac{1}{N} \sum_{i=1}^N (y_i - (\hat{\theta}_0 + \hat{\theta}_1 x_i))^2$$

$$\boxed{\hat{\theta}_1 = \frac{\sum_{i=1}^N x_i (y_i - \bar{y})}{\sum_{i=1}^N x_i (x_i - \bar{x})}}$$

bc

$$\boxed{\hat{\theta}_1 = \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^N (x_i - \bar{x})^2}}$$

*

Homework 1

3 b)

Batch first-order gradient descent

- Initialize $\hat{\theta}$

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \end{bmatrix} \begin{matrix} x_1 \\ x_2 \end{matrix}$$

- Repeat until Convergence ε

  - For $j = 1, \ldots, p$ {

$$\theta_0 \leftarrow \sigma_0 + \alpha \frac{d}{d\theta_0} \sum_{i=1}^{N} (y_i - x_i^T \theta)^2$$

$1\times 1 \quad 1\times p \quad p\times 1$

  }

}

3 c)

Stochastic gradient descent ✓

- Initialize θ

- Repeat until convergence {

  - shuffle the order of observations $i = 1, \ldots N$

For i, in $1 \ldots, N$ {

For j in $1 \ldots, p$ {

$$\sigma_j = \sigma_j + \alpha \frac{d}{d\theta} (y_i - x_i^T \theta)^2$$

$1\times p \quad p\times 1$

}

}

HW - batch gradient desc GD-Copy1 (1)

September 19, 2018

In [1]:
```
import numpy as np
import plotly
import pandas
```

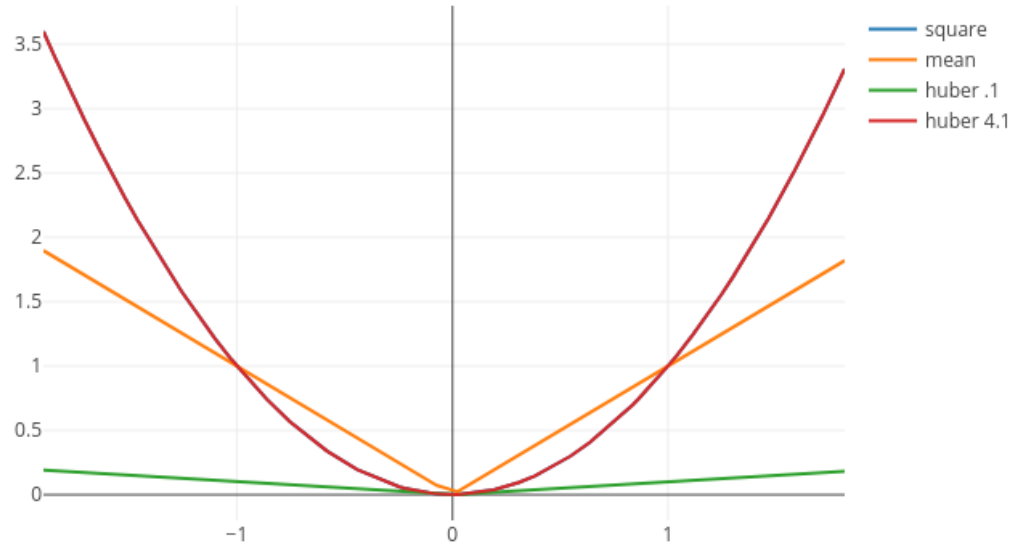
In [2]:
```
#shuffle two arrays in unison
def unison_shuffled_copies(a, b):
    assert len(a) == len(b)
    p = np.random.permutation(len(a))
    return a[p], b[p]
```

In [3]:
```
#error functions
def l1norm(E,l):
    return sum([x**2 for x in E])

#mean absolute error
def mean_absolute_error(E,l):
    return 1/len(E) * sum([np.abs(x) for x in E])

#huber loss
def huber_loss(E, l):
    result = []
    for e in E:
        if e <= l:
            result.append( 1/2 * e**2 )
        elif e > l:
            result.append( l * np.abs(e) - 1/2 * l**2)
    return sum(result)
```

In [4]:
```
x = np.random.uniform(-2,2,50)
%matplotlib inline
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
plotly.tools.set_credentials_file(
    username='zferic85', api_key='TRGTDNcZWyGPnY02dpJV')

# Create random data with numpy
import numpy as np
```

```
    def huberfun(x,h):

        if np.abs(x) <= h:
            return x**2
        if np.abs(x) > h:
            return h*np.abs(x)

    x = np.sort(x)
    # Create traces
    trace0 = go.Scatter(
        x = x,
        y =[ e**2 for e in x],
        mode = 'lines',
        name = 'square'
    )
    trace1 = go.Scatter(
        x = x,
        y = [ np.abs(y) for y in x],
        mode = 'lines',
        name = 'mean'
    )

    trace2 = go.Scatter(
        x = x,
        y = [huberfun(y,.1) for y in x] ,
        mode = 'lines',
        name = 'huber .1'
    )

    trace3 = go.Scatter(
        x = x,
        y = [huberfun(y,4.1) for y in x] ,
        mode = 'lines',
        name = 'huber 4.1'
    )
    data = [trace0, trace1, trace2, trace3]

    fig = go.Figure(data)
    plotly.plotly.image.save_as(fig, filename='line-mode.png')

    from IPython.display import Image
    Image('line-mode.png')
```

Out[4]:




```
In [5]: #line search
        n_max = 1.0
        r = .6
        tolerance = .0001
        max_backtrack_iterations = 50
        def line_search(w_t, c, g,h):
                n = n_max
                w = w_t
                #this also depends on the loss function why we pass the c
                obj = c(np.matmul(data,w)-np.array(target),h)
                backtrack_iterations = 0
                while backtrack_iterations < max_backtrack_iterations:
                    w = np.array(w_t) - n * np.array(g)

                    if c(np.matmul(data,w)-np.array(target),h) < (obj - tolerance):
                        break
                    n = r*n

                    backtrack_iterations += 1

                if backtrack_iterations == max_backtrack_iterations - 1:
                    print('here')
```

```
                return w_t, 0


            return w, n
```

In [6]:
```
#derivative of the mean absolute error
def mean_absolute_err_g(data,w,target):
    g = []
    errors = (np.matmul(data,w)-np.array(target))
    for p in range(0, len(data[0])):
        sums = 0
        for idx, e in enumerate(errors):
            if e > 0:
                sums += data[idx][p]
            if e < 0:
                sums -= data[idx][p]
            if e == 0:
                sums += 0

        g.append(sums)
    return g

#derivative of huber loss
def huber_loss_g(data,w,target,h):
    g = []
    errors = (np.matmul(data,w)-np.array(target))
    for p in range(0, len(data[0])):
        sums = 0
        for idx, e in enumerate(errors):

            if np.abs(e) <= h:

                sums += e * data[idx][p]

            if np.abs(e) > h:

                if e > 0:
                    sums += h*data[idx][p]
                if e < 0:
                    sums -= h*data[idx][p]
                if e == 0:
                    sums += 0

        g.append(sums)
    return g
```

In [7]:
```
list(np.random.randn(1,3)[0])
```

Out[7]:
```
[1.1351593646912232, 0.6920185366346697, -0.857732633728152]
```

In [8]:
```
#batch  gradient descent implementation
def batch_gradient(data,target,errfun,h):
    #initialize w
    w = list(np.random.randn(1,len(data[0]))[0])
    #super large int (inf)
    err = 5000000000000
    tolerance = .0001
    dim = len(data[0])
    max_iterations = 1000
    iteration = 0

    while err > tolerance and iteration < max_iterations:
        err = errfun(np.matmul(data,w)-np.array(target),h)
        #gradient depends on the loss function used
        if errfun == l1norm:
            g =  np.matmul(np.transpose(data) ,(np.matmul(data,w)-np.array(target)))
        if errfun == mean_absolute_error:
            g = mean_absolute_err_g(data,w,target)
        #huber loss is a combination of the l1norm and absolute so the gradient is pie
        if errfun == huber_loss:
            g = huber_loss_g(data,w,target,h)
        #insert line search here for the step size
        #w_ls,n = line_search(w, errfun, g,h)
        n = .001
        #print(n)
        w = np.array(w) - n * np.array(g)
        iteration += 1
    return w
```

In [9]:
```
#Stochastic gradient descent
def stochastic_gradient(data,target,errfun,epochs):

    #initialize w

    w = list(np.random.randn(1,len(data[0]))[0])


    #stochastic gradient descent implementation
    hub_coef = .9
    for i in range(1,epochs):
        data, target = unison_shuffled_copies(np.array(data),np.array(target))
        #number of epochs in
        for j in range(0, len(data)):
            obj = np.matmul(np.transpose(data[j]), w) - target[j]
            if errfun == l1norm:
                g = obj * data[j]
            if errfun == mean_absolute_error:
                if obj > 0:
```

```
                        g = data[j]
                    if obj < 0:
                        g = -data[j]
                    if obj == 0:
                        g = 0
                if errfun == huber_loss:
                    if obj < hub_coef:
                        g = obj * data[j]
                    if obj >= hub_coef:
                        if obj > 0:
                            g = hub_coef*data[j]
                        if obj < 0:
                            g = -hub_coef*data[j]
                        if obj == 0:
                            g = 0
                n = 1/i
                w = np.array(w) - n * np.array(g)

        return w
```

In [10]:
```
def analytic(data,target):

    s1 = np.linalg.inv(np.matmul(np.transpose(data),np.array(data)))
    s2 = np.matmul(s1,np.transpose(data))
    s3 = np.matmul(s2, target)
    coef_pred = s3

    return coef_pred
```

In [11]:
```
#####testing
#Problem 5a

rez5a1 = []
#analyti square loss
print('5a1')
for i in range(0,1000):
    X = np.random.uniform(-2,2,50)
    e = np.random.normal(0, 4, 50)

    Y = 2 + 3*X + e
    intercept = [1]*50

    data = list(zip(intercept,X))
    target = Y
    w = analytic(data,target)
    #w = batch_gradient(data,target,huber_loss,8000)

    d = []
```

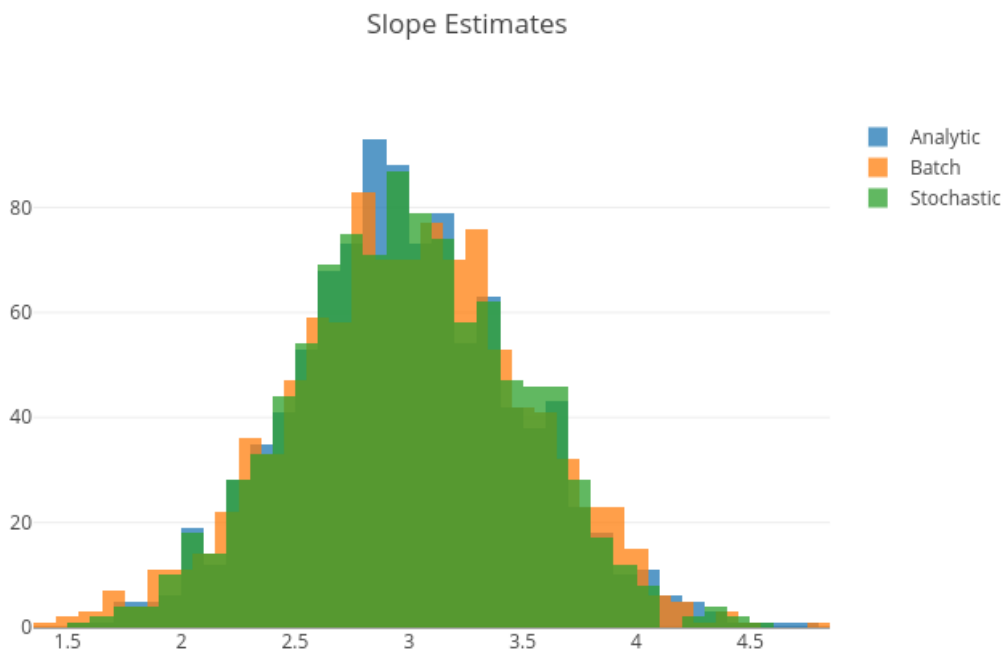
```
        rez5a1.append(w[1])
    #Problem 5b batch square loss
    rez5a2 = []
    print('5a2')
    for i in range(0,1000):
        X = np.random.uniform(-2,2,50)
        e = np.random.normal(0, 4, 50)
        Y = 2 + 3*X + e
        intercept = [1]*50
        data = list(zip(intercept,X))
        target = Y
        w = batch_gradient(data,target,l1norm,8000)
        rez5a2.append(w[1])
    #Problem 5c stochastic_square loss
    rez5a3 = []
    print('5a3')
    for i in range(0,1000):
        X = np.random.uniform(-2,2,50)
        e = np.random.normal(0, 4, 50)

        Y = 2 + 3*X + e
        intercept = [1]*50

        data = list(zip(intercept,X))
        target = Y
        w = stochastic_gradient(data,target,l1norm,100)
        rez5a3.append(w[1])
```

```
5a1
5a2
5a3
```

In [12]:
```
import plotly.plotly as py
import plotly.graph_objs as go
import numpy as np

df1 = pandas.DataFrame({'i': rez5a1, 'ii': rez5a2, 'iii': rez5a3})

def visualize(df):
    trace1 = go.Histogram(
        x=df['i'],
        opacity=0.75,
        name ='Analytic'
    )
    trace2 = go.Histogram(
```

```
            x=df['ii'],
            opacity=0.75,
            name = 'Batch'
        )
        trace3 = go.Histogram(
            x=df['iii'],
            opacity=0.75,
            name = 'Stochastic'
        )
        data = [trace1,trace2,trace3]
        layout = go.Layout(barmode='overlay', title = 'Slope Estimates')
        fig = go.Figure(data=data, layout=layout)

        return data,layout

    data,layout = visualize(df1)
    fig = go.Figure(data=data, layout=layout)

    plotly.plotly.image.save_as(fig, filename='histogram2.png')

    Image('histogram2.png')
```

Out[12]:



(i) analytical solution,
(ii) batch
(iii) stochastic gradient descent implemented in Question 4. Set learning rate to a small value (say, = 0.01).

```
In [13]: df1.describe()
```

```
Out[13]:                  i           ii          iii
         count  1000.000000  1000.000000  1000.000000
         mean      2.996771     3.000782     2.992342
         std       0.499195     0.525452     0.493351
         min       1.579678     1.399667     1.573076
         25%       2.661620     2.650278     2.646529
         50%       2.971682     3.008095     2.981482
         75%       3.338339     3.345889     3.342733
         max       4.724260     4.832236     4.567030
```

5b) It seems that all three methods tend to be normally distributed about the mean 3. However, the gradient descent method as shown by the statistics has the best approximate of the intercept and also has the smallest standard deviation. the analytical solution did a little better than the batch gradient descent but then I did not experiment with the tolerance or the number of iterations was fairly low.

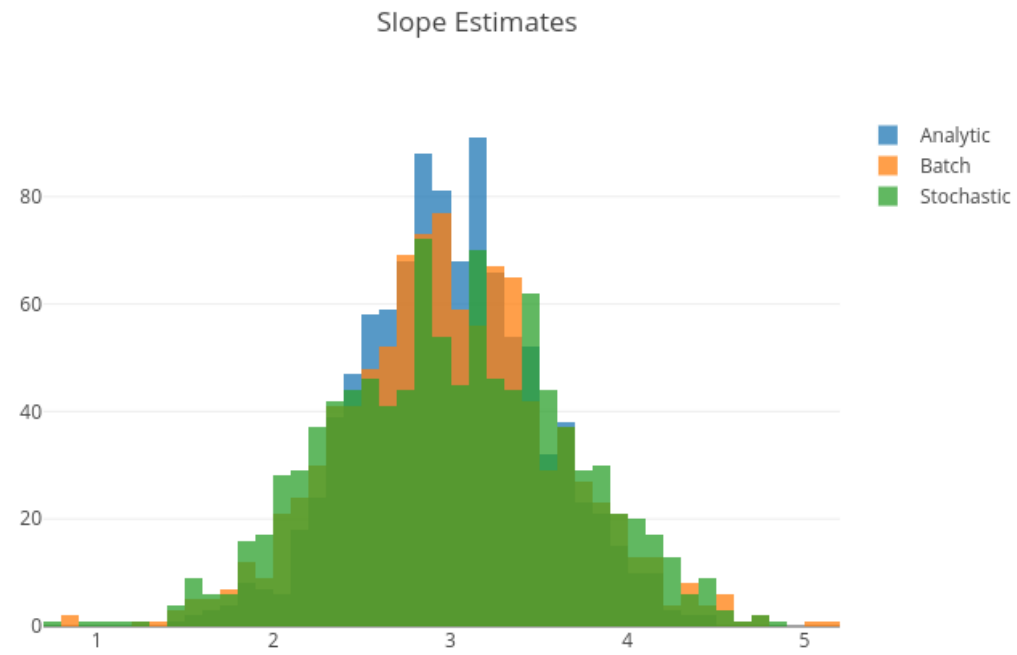
```
In [14]: #####testing
         #Problem 5c

         rez5c1 = []
         #analyti square loss
         print('5c1')
         for i in range(0,1000):
             X = np.random.uniform(-2,2,50)
             e = np.random.normal(0, 4, 50)
             Y = 2 + 3*X + e
             intercept = [1]*50
             data = list(zip(intercept,X))
             target = Y
             w = analytic(data,target)
             #w = batch_gradient(data,target,huber_loss,8000)
             rez5c1.append(w[1])
         #Problem 5b batch square loss
         print('5c2')
         rez5c2 = []
         for i in range(0,1000):
             X = np.random.uniform(-2,2,50)
             e = np.random.normal(0, 4, 50)
             Y = 2 + 3*X + e
             intercept = [1]*50
             data = list(zip(intercept,X))
             target = Y
```

```
        w = batch_gradient(data,target,mean_absolute_error,8000)
        d = []
        rez5c2.append(w[1])
    #Problem 5c stochastic_square loss
    print('5c3')
    rez5c3 = []
    for i in range(0,1000):
        X = np.random.uniform(-2,2,50)
        e = np.random.normal(0, 4, 50)
        Y = 2 + 3*X + e
        intercept = [1]*50
        data = list(zip(intercept,X))
        target = Y
        w = stochastic_gradient(data,target,huber_loss,100)
        d = []
        rez5c3.append(w[1])
```

```
5c1
5c2
5c3
```

In [15]:
```
df2 = pandas.DataFrame({'i': rez5c1, 'ii': rez5c2, 'iii': rez5c3})

data,layout = visualize(df2)
fig = go.Figure(data=data, layout=layout)

plotly.plotly.image.save_as(fig, filename='histogram3.png')

Image('histogram3.png')
```

Out[15]:



(i) squared loss with the analytical solution,
(ii) mean absolute error with batch
(iii) Huber loss with batch gradient descent implemented in

```
In [16]: df2.describe()
```

Out[16]:

	i	ii	iii
count	1000.000000	1000.000000	1000.000000
mean	2.998371	2.994529	3.002683
std	0.502119	0.612918	0.669111
min	1.473251	0.869273	0.718801
25%	2.667545	2.600267	2.509021
50%	2.979453	2.972258	2.999382
75%	3.330112	3.376120	3.471906
max	4.475202	5.190752	4.835009

Question 4. Set learning rate to a small value (say, = 0.01).

1 5d) It seems that in this case, the analytical solution outpreformed the rest of the methods. Again, my number of iterations and tolerances were fixed and I did not experiment enough due to run time.

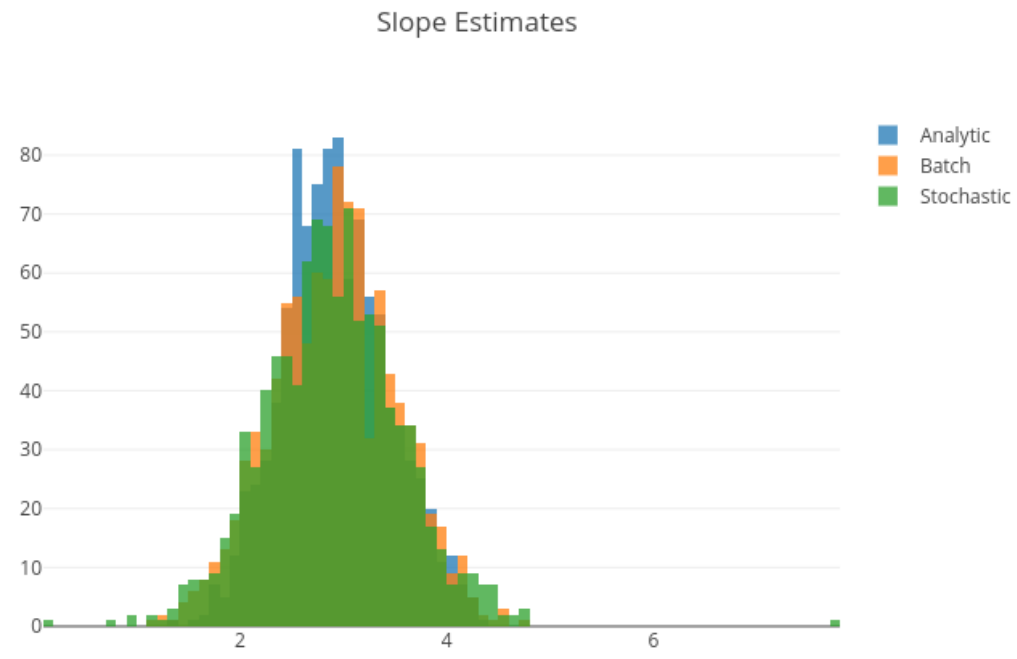
In [17]:
```
import random
#Generate 50 random numbers on interval (0,1]
def generate_data():
    k = [np.random.rand() for x in range(0,50)]

    idx = []
    #store index
    for i in range(0,50):
        if k[i] <= .1: idx.append(i)

    X = np.random.uniform(-2,2,50)
    e = np.random.normal(0, 4, 50)

    Y = 2 + 3*X + e
    intercept = [1]*50
    data = list(zip(intercept,X))
    data = [list(x) for x in data]
    for i in idx:
        r = np.random.rand()
        if r > .5:
            data[i][1] = data[i][1] + 1/2*data[i][1]
        if r<= .5:
            data[i][1] = data[i][1] - 1/2*data[i][1]
    return data,Y
```

In [18]:
```
#####testing
#Problem 5c
rez5e1 = []
print('5e1')
#analyti square loss
for i in range(0,1000):
    data,target = generate_data()
    w = analytic(data,target)
    rez5e1.append(w[1])
#Problem 5b batch square loss
rez5e2 = []
print('5e2')
for i in range(0,1000):
    data,target = generate_data()
    w = batch_gradient(data,target,mean_absolute_error,8000)
    rez5e2.append(w[1])
#Problem 5c stochastic_square loss
rez5e3 = []
print('5e3')
```

```
        for i in range(0,1000):
            data,target = generate_data()
            w = stochastic_gradient(data,target,huber_loss,100)
            rez5e3.append(w[1])

5e1
5e2
5e3
```

In [19]:
```
df3 = pandas.DataFrame({'i': rez5e1, 'ii': rez5e2, 'iii': rez5e3})

data,layout = visualize(df3)
fig = go.Figure(data=data, layout=layout)
plotly.plotly.image.save_as(fig, filename='histogram4.png')

Image('histogram4.png')
```

Out[19]:



In [20]:
```
df3.describe()
```

Out[20]:

	i	ii	iii
count	1000.000000	1000.000000	1000.000000
mean	2.927560	2.915841	2.899921
std	0.511940	0.598849	0.667905
min	1.353416	1.157599	0.186008
25%	2.569836	2.495753	2.464618
50%	2.899689	2.933181	2.890421
75%	3.263618	3.343740	3.311388
max	4.349971	4.703073	7.756445

5f) According to the statistics the closest approximation was achieved by the analytical solution making me believe that my choice of iterations and tolerance needs improvement as well as more experimentation with the huber coeffiecnt. According to the book, the huber loss function should be less sensitive to outliers than the other methods.