Table of Contents

MIDS - w261 Machine Learning At Scale

Course Lead: Dr James G. Shanahan (**email** Jimi via James.Shanahan *AT* gmail.com)

Assignment - HW6

Name: Nilesh Bhoyar
Class: MIDS w261 Summer -2017 Group 2
Email: nilesh.bhoyar@iSchool.Berkeley.edu
StudentId 26302327 **End of StudentId**
Week: 6

NOTE: please replace `1234567` with your student id above
Due Time: HW is due the Tuesday of the following week by 8AM (West coast time).

Instructions

MIDS UC Berkeley, Machine Learning at Scale
DATSCIW261 ASSIGNMENT #6

Version 2017-23-2

IMPORTANT

This homework can be completed locally on your computer

=== INSTRUCTIONS for SUBMISSIONS ===

Follow the instructions for submissions carefully.

Each student has a `HW-<user>` repository for all assignments.

Push the following to your HW github repo into the master branch:

- Your local HW6 directory. Your repo file structure should look like this:

```
HW-<user>
    --HW3
       |__MIDS-W261-HW-03-<Student_id>.ipynb
       |__MIDS-W261-HW-03-<Student_id>.pdf
       |__some other hw3 file
    --HW4
       |__MIDS-W261-HW-04-<Student_id>.ipynb
       |__MIDS-W261-HW-04-<Student_id>.pdf
       |__some other hw4 file
    etc..
```

HW6.0.

- In mathematics, computer science, economics, or management science what is mathematical optimization?
- Give an example of a optimization problem that you have worked with directly or that your organization has worked on.
- Please describe the objective function and the decision variables.
- Was the project successful (deployed in the real world)? Describe.

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import sys
sys.executable
```

Out[2]: '/opt/anaconda/bin/python'

In [3]:
```
from IPython.display import display, Math, Latex
```

START STUDENT SOLUTION HW6.0

Optimization algorithms try to find the optimal solution for a problem, for instance, finding the maximum or the minimum of a function. The function can be linear or non-linear. The solution could also have special constraints.

$$\begin{aligned} \text{minimize}_x \quad & f_0(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \ i = 1, \ldots, m \\ & h_i(x) = 0, \ i = 1, \ldots, p \\ & x \in S \end{aligned}$$

Function $f_0(x)$ is called the problem objective function, first condition is the equality constraints,second stands for inequality constraints and last condition is set constraints.

I have not solved any optmization problem but I work for insurance company where actuarial science deals with such issues on daily basis. In fact their risk models are designed for optimizing gross profit based on total products/total policy holders and loss paid.

http://www.actuaries.asn.au/library/events/GIS/2010/Convex%20Optimization.pdf (http://www.actuaries.asn.au/library/events/GIS/2010/Convex%20Optimization.pdf)

Article describes how to optimize premium rating i.e. loss/total premium loss ratio using convex optimization.

Yes , project is successful and we are generating more premium than paid losses.

END STUDENT SOLUTION HW6.0

HW6.1 Optimization theory:

HW6.1.A

- For unconstrained univariate optimization what are the first order Necessary Conditions for Optimality (FOC)?
- What are the second order optimality conditions (SOC)?
- Give a mathematical defintion.

START STUDENT SOLUTION HW6.1.A

First order Conditons for Optimality

If $f(x_0)$ is a local minimum of f, then $\nabla f(x_0) = 0$.

second order optimality conditions (SOC)

If $f(x_0)$ is a local minimum of f, then $\nabla^2 f(x_0) > 0$.

END STUDENT SOLUTION HW6.1.A

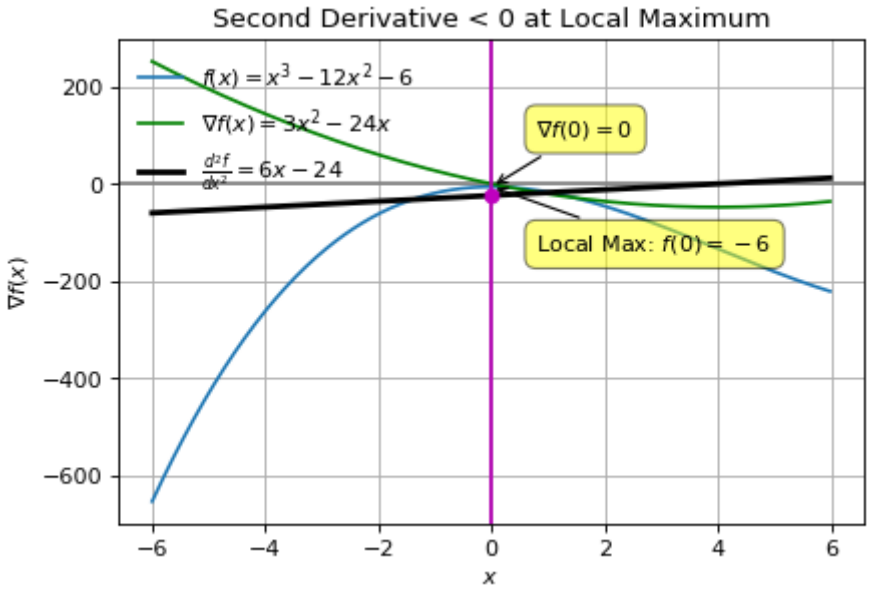
HW6.1.B

In python, plot the univartiate function

$$X^3 - 12x^2 - 6 \; defined \; over \; [-6, 6]$$

Also plot its corresponding first and second derivative functions. Eyeballing these graphs, identify candidate optimal points and then classify them as local minimums or maximums. Highlight and label these points in your graphs. Justify your responses using the FOC and SOC.

In [4]:
```
# START STUDENT PLOTS HW6.1.B
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return x**3 - 12*x**2 - 6
def first_derivative(x):
    return 3*x**2 - 24*x
def second_derivative(x):
    return 6*x -24

X = np.arange(-6.0, 6.0, 0.02)
Y_zero = [0] * ((6 - (-6))*100/2)
Y_neg_24 = [-24] * ((6 - (-6))*100/2)
plt.figure(num=None, figsize=(6, 4), dpi=80, facecolor='w', edge
color='k')
plt.grid(True)
plt.axhline(0, color='grey')
plt.axvline(0, color='grey')
plt.plot(X, f(X),label='$f(x)=x^3-12x^2-6$')
plt.ylabel(r"$f(x)$")
plt.xlabel(r"$x$")
#plt.tittle(r"Objective function: $f(x)=x^3-12x^2-6$")
plt.annotate(r'Local Max: $f(0)=-6$',
            xy = (0, -6), xytext = (20, -20),
            textcoords = 'offset points', ha = 'left', va = 'top
',
            bbox = dict(boxstyle = 'round,pad=0.5', fc = 'yellow
', alpha = 0.5),
           arrowprops = dict(arrowstyle = '->', connectionstyle
= 'arc3,rad=0'))
plt.plot(X, first_derivative(X),label='$\\nabla f(x)=3x^2-24x$',
color='green')
plt.ylabel(r"$\nabla f(x)$")
plt.xlabel(r"$x$")
#plt.title(r"First derivative: $\nabla f(x)=3x^2-24x$")
plt.annotate(r'$\nabla f(0)=0$',
            xy = (0, -6), xytext = (20, 20),
        textcoords = 'offset points', ha = 'left', va = 'bottom'
,
        bbox = dict(boxstyle = 'round,pad=0.5', fc = 'yellow', a
lpha = 0.5),
        arrowprops = dict(arrowstyle = '->', connectionstyle = '
arc3,rad=0'))

plt.plot(X, second_derivative(X), 'k', label = '$\\frac{d^2f}{dx
^2} = 6x - 24$', linewidth = 2.5)

plt.plot(0, -24, 'mo')
plt.axvline(x=0, ymin=-700, ymax=100, color='m')
plt.legend(loc='upper left', frameon=False)
plt.title('Second Derivative < 0 at Local Maximum')

plt.show()
# END STUDENT PLOTS HW6.1.B
```


Second Derivative < 0 at Local Maximum
$f(x) = x^3 - 12x^2 - 6$
$\nabla f(x) = 3x^2 - 24x$
$\frac{d^2f}{dx^2} = 6x - 24$
$\nabla f(0) = 0$
Local Max: $f(0) = -6$
$\nabla f(x)$
x
200
0
−200
−400
−600
−6
−4
−2
0
2
4
6

$$X^3 - 12x^2 - 6 \; defined \; over \; [-6, 6]$$

as per first order condition , first derivative is zero at maxima and minima so

$$\frac{df(x)}{dx} = 0$$

$$3 * x^2 - 24 * x = 0$$

$$x^2 - 8x = 0$$

$$x * (x - 8) = 0$$

$$solutions : x : 0, 8$$

But this does not tell whether its maxiuma or minima . We need to take SOC for this to find it out

$f''(x) > 0$ then_its local minima

$f''(x) < 0$ then its local maxima

second ordder derivative

$$6 * x - 24 = 0$$

when $x = 0$

$$solution = 6 * 0 - 24$$

$$solution = -24$$

So function is maximum at 0

When $x = 8$

$$solution = 6 * 8 - 24$$

$$solution = 24$$

So function is minimum at 8

But constraints for solutioin space is [-6,6] so answer is 0.

HW6.1.C

- For unconstrained multi-variate optimization what are the first order Necessary Conditions for Optimality (FOC)?
- What are the second order optimality conditions (SOC)?
- Give a mathematical defintion.
- What is the Hessian matrix in this context?

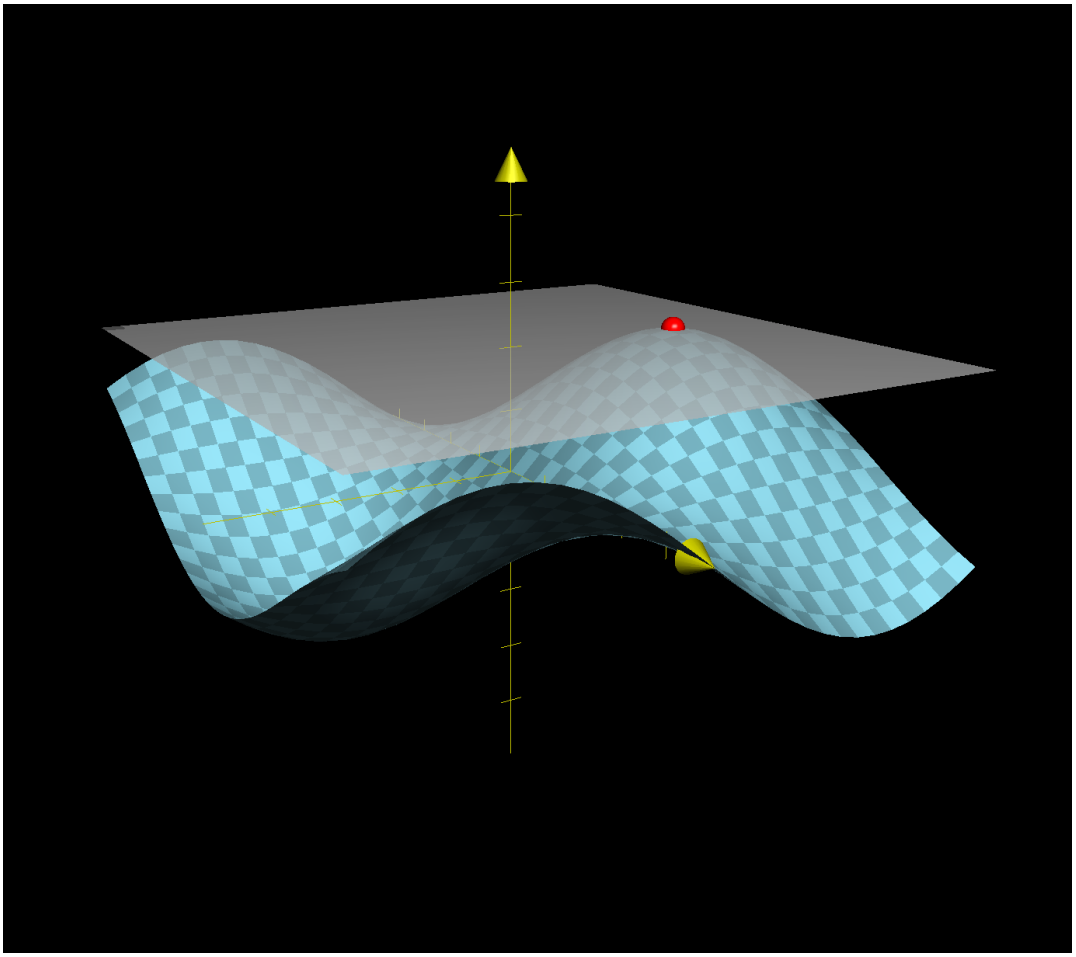
START STUDENT SOLUTION HW6.1.C

FOC

We can generalize the univariate discussion for multivariate optimization problem for first order conditions. If you are looking for the local maxima/minima of a two-variable function f(x, y) the first step is to find input points (x_0, y_0) where the gradient is the $\mathbf{0}$. $\nabla f(x_0, y_0) = \mathbf{0}$. These are basically points where the tangent plane on the graph of $\mathbf{f}$ is flat.

In [5]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image("maxima.png")
```

Out[5]:



What are the second order optimality conditions (SOC)?

The second partial derivative test tells us how to verify whether this stable point is a local maximum, local minimum, or a saddle point. Specifically, you start by computing this quantity:

$$H = f_{xx}(x_0, y_0) * f_{xx}(x_0, y_0) - f_{xy}(x_0, y_0)^2$$

Then the second partial derivative test goes as follows:

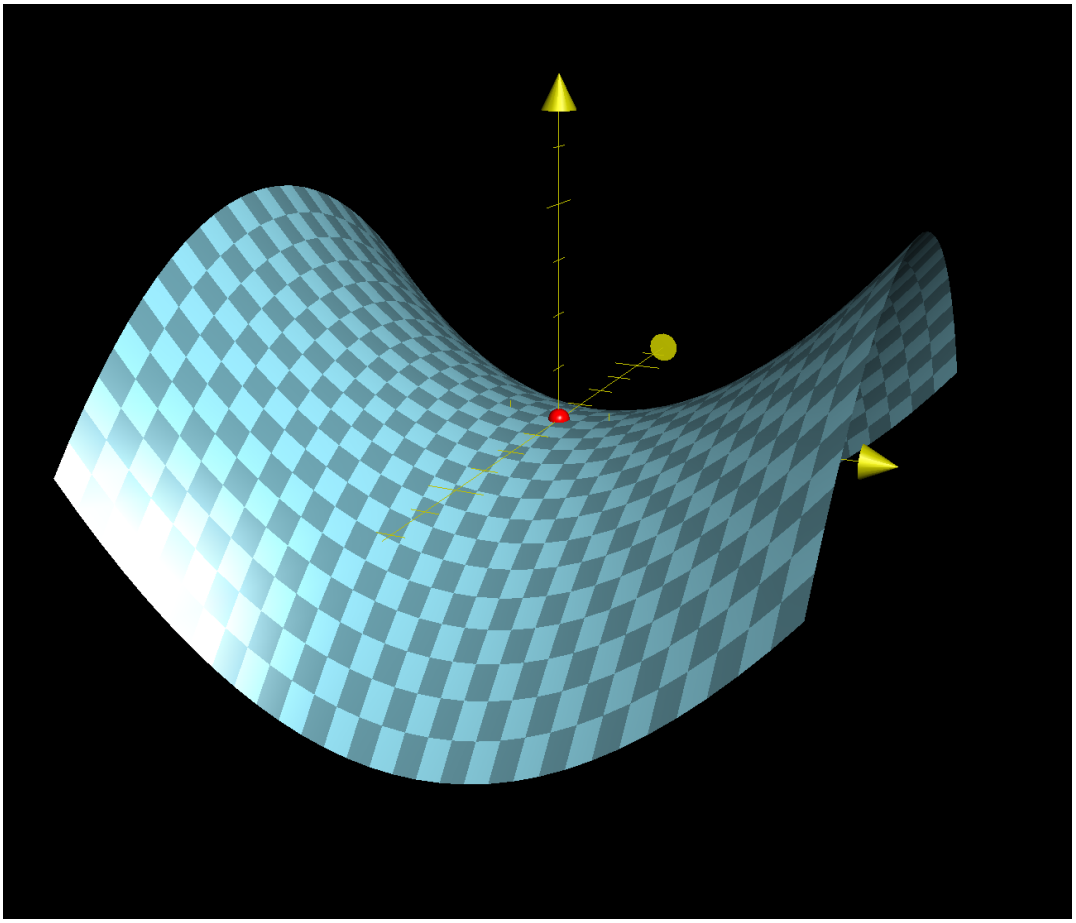
1. If H < 0 then (x_0, y_0) parenthesis is a saddle point.
2. If H > 0 and $f_{xx}(x_0, y_0) < 0$ then (x_0, y_0) is maximum point

```
if H > 0 and $f_{xx}(x_0,y_0) > 0$ then $(x_0, y_0)$ is minimu
m point
```

H is also called as hessian matrix .

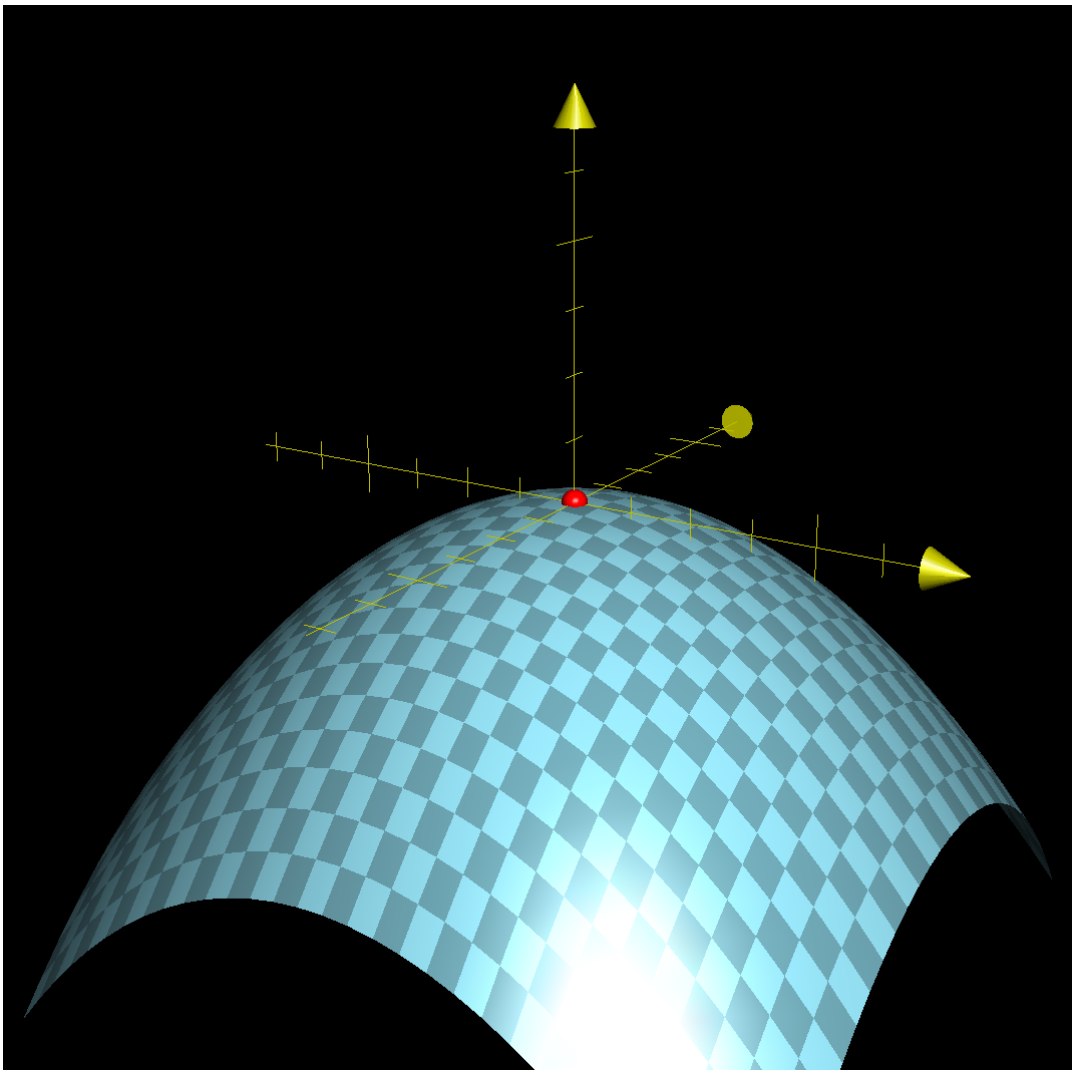
In [6]:
```
#saddle point
Image("saddlepoint.png")
```

Out[6]:



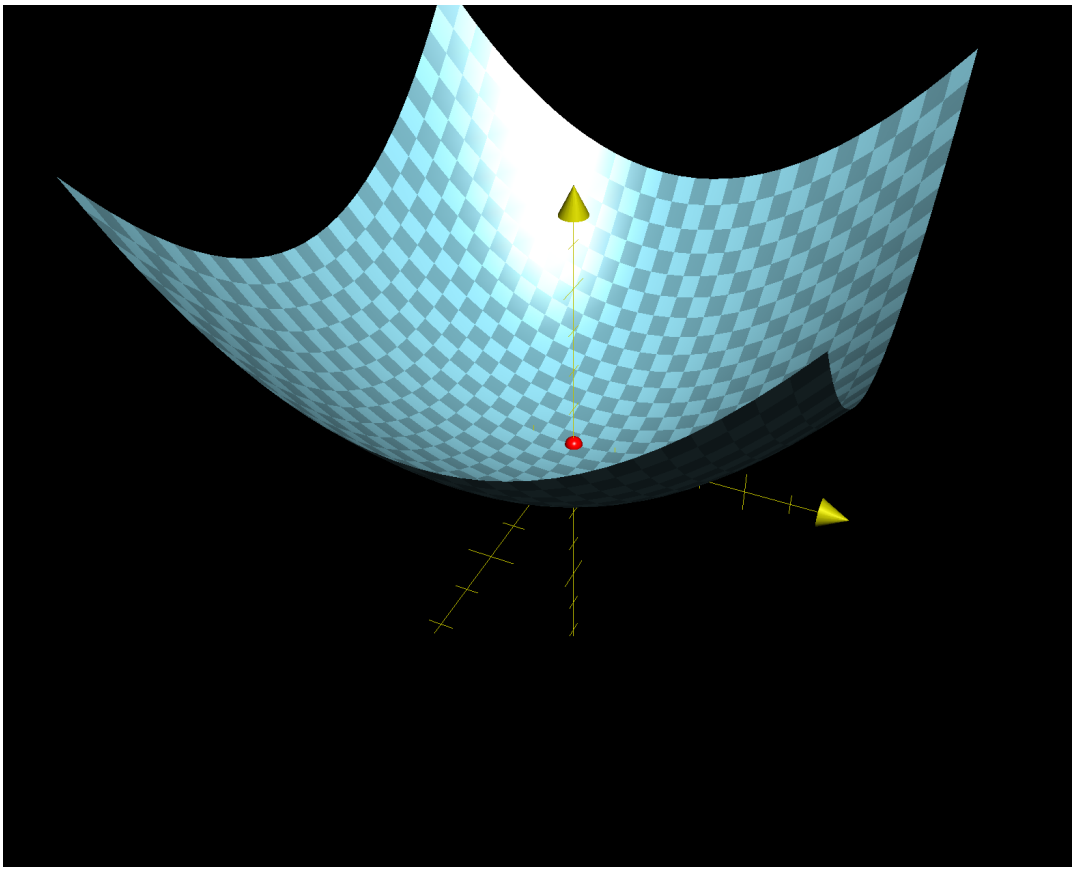
In [7]:
```
Image("localmaximum.png") #local maxium H > 0  and Fxx < 0 and x
0,y0
```

Out[7]:



In [8]:
```
Image("localminima.png") #local maxium H > 0  and Fxx > 0 and x0
,y0
```

Out[8]:



More information on below link https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/second-partial-derivative-test (https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/second-partial-derivative-test)

END STUDENT SOLUTION HW6.1.C

HW6.2

Taking $x = 1$ as the first approximation(xt1) of a root of $X^3 + 2x - 4 = 0$, use the Newton-Raphson method to calculate the second approximation (denoted as xt2) of this root.
(Hint the solution is $xt2 = 1.2$)

START STUDENT SOLUTION HW6.2

$$x_{(}n+1) = x_n - \frac{f^{'}(x)}{f(x)}$$

$$f(x) = x^3 + 2x - 4$$

$$f^{'}(x) = 3x^2 + 2$$

$$x_1 = 1$$

$$x_2 = 1 - \frac{1^3 + 2 - 4}{3 * 1^2 + 2}$$

$$x_2 = \frac{6}{5}$$

$$x_2 = 1.2$$

END STUDENT SOLUTION HW6.2

HW6.3 Convex optimization

- What makes an optimization problem convex?
- What are the first order Necessary Conditions for Optimality in convex optimization?
- What are the second order optimality conditions for convex optimization?
- Are both necessary to determine the maximum or minimum of candidate optimal solutions?

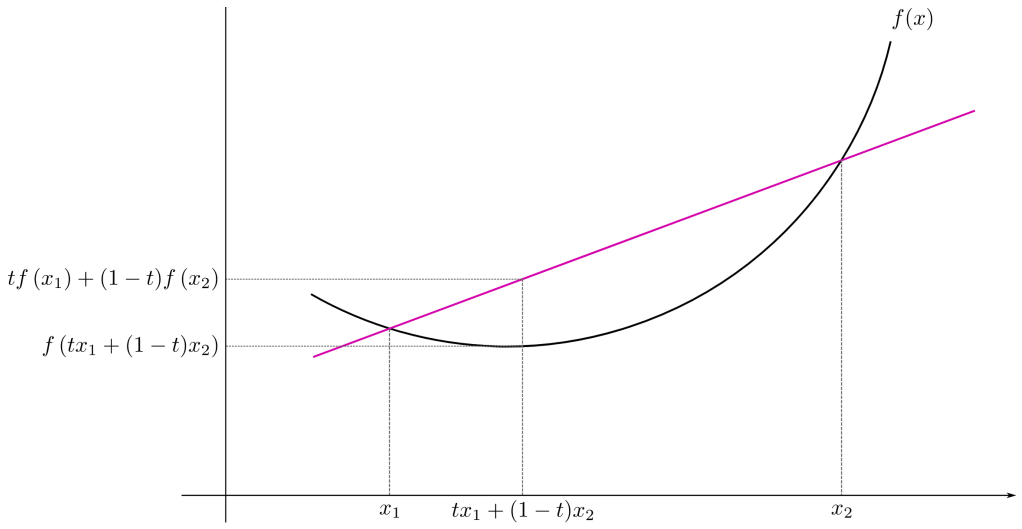
START STUDENT SOLUTION HW6.3

In mathematics, a real-valued function defined on an n-dimensional interval is called convex if the line segment between any two points on the graph of the function lies above or on the graph, in a Euclidean space (or more generally a vector space) of at least two dimensions.

Let X be a convex set in a real vector space and let $f: X \to \mathrm{R}$ be a function. f is called convex if:
$\forall x_1, x_2 \in X, \forall t \in [0, 1]: \quad f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$. .

In [9]:
```
Image("Convex.png")
```

Out[9]:



END STUDENT SOLUTION HW6.3

What are the first order Necessary Conditions for Optimality in convex optimization?

Let $\mathbf{x}_0$ be a global extrema similar to multi-variate optimization problem, $\mathbf{x}_0$ occurs when the first derivative has all entries equal to zero, or $J(\mathbf{x}_0) = \mathbf{0}$. In other words gradient should be zero.

What are the second order optimality conditions for convex optimization?

In a convex function, H matrix evaluated at any point has all entries non-negative (i.e., $H(\mathbf{x})$ is positive semidefinite). Therefore, once we identify an extrema $\mathbf{x}_0$ from the first order condition, we know that $\mathbf{x}_0$ is a minimum.

Are both necessary to determine the maximum or minimum of candidate optimal solutions?

If the objective function is convex, we only need the first order condition to know that an optimal solution is a minimum.

- Fill in the BLANKS here:
  Convex minimization, a subfield of optimization, studies the problem of minimizing **BLANK** functions over **BLANK** sets. The **BLANK** property can make optimization in some sense "easier" than the general case - for example, any local minimum must be a global minimum.

HW 6.4

The learning objective function for weighted ordinary least squares (WOLS) (aka weight linear regression) is defined as follows:

$$0.5 * \sum_{i=1}^{n} (weight_i * (W * X_i - y_i)^2)$$

Where training set consists of input variables X (in vector form) and a target variable y, and W is the vector of coefficients for the linear regression model.

Derive the gradient for this weighted OLS by hand; showing each step and also explaining each step.

START STUDENT SOLUTION HW6.4

cost function is given as

$$J(W) = 0.5 * \sum_{i=1}^{n} (weight_i * (W * X_i - y_i)^2)$$

we need to minimize this cost functon consider the gradient descent algorithm, which starts with some initial θ, and repeatedly performs the update as:

$$W_{i+1} = W_i + \nabla J(W)$$

We have to take partial derivative of cost function for finding minima

$$\frac{\partial J(W)}{\partial W} = \frac{\partial 0.5 * \sum_{i=1}^{n} (weight_i * (W * X_i - y_i)^2)}{\partial W}$$

$$\frac{\partial J(W)}{\partial W} = 2 * 0.5 * weight_i * (W * X_i - y_i) * \frac{\partial \sum_{i=1}^{n} (W * X_i - y_i)}{\partial W}$$

$$\frac{\partial J(W)}{\partial W} = \sum_{i=1}^{n} weight_i * (W * X_i - y_i) * X_i$$

$$\frac{\partial J(W)}{\partial W} = \sum_{i=1}^{n} (W * X_i - y_i) * weight_i * X_i$$

for single training example this gives update rule as where n=1

$$W_{i+1} = W_i + \alpha * \sum_{i=1}^{n} (W * X_i^i - y_i) * weight_i * X_i^i$$

Reference http://cs229.stanford.edu/notes/cs229-notes1.pdf (http://cs229.stanford.edu/notes/cs229-notes1.pdf)

END STUDENT SOLUTION HW6.4

HW 6.5

Write a MapReduce job in MRJob to do the training at scale of a weighted OLS model using gradient descent.
Generate one million datapoints just like in the following notebook: http://nbviewer.ipython.org/urls/dl.dropbox.com/s/kritdm3mo1daolj/MrJobLinearRegressionGD.ipynb (http://nbviewer.ipython.org/urls/dl.dropbox.com/s/kritdm3mo1daolj/MrJobLinearRegressionGD.ipynb)

Weight each example as follows:

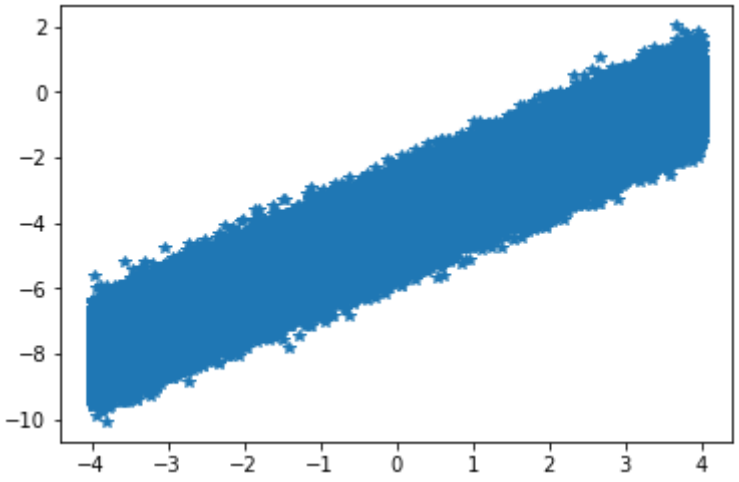
$$weight(x) = abs(1/x)$$

Sample 1% of the data in MapReduce and use the sampled dataset to train a (weighted if available in SciKit-Learn) linear regression model locally using SciKit-Learn (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html))

Plot the resulting weighted linear regression model versus the original model that you used to generate the data. Comment on your findings.

In [10]:
```
# START STUDENT SOLUTION HW6.5
# insert cells as needed
#generate data just as provided in notebook


%matplotlib inline
import numpy as np
import pylab
size = 1000000
x = np.random.uniform(-4, 4, size)
y = x * 1.0 - 4 + np.random.normal(0,0.5,size)
data = zip(y,x)
np.savetxt('LinearRegression.csv',data,delimiter = ",")



# END STUDENT SOLUTION HW6.5
```

In [11]:
```
pylab.plot(x, y,'*')
pylab.show()
```



In [12]:
```
%%writefile MrJobBatchGDUpdate_LinearRegression.py
from mrjob.job import MRJob

# This MrJob calculates the gradient of the entire training set
#     Mapper: calculate partial gradient for each example
#
class MrJobBatchGDUpdate_LinearRegression(MRJob):
    # run before the mapper processes any input
    def read_weightsfile(self):
        # Read weights file
        with open('weights.txt', 'r') as f:
            self.weights = [float(v) for v in f.readline().split
(',')]
        # Initialze gradient for this iteration
        self.partial_Gradient = [0]*len(self.weights)
        self.partial_count = 0

    # Calculate partial gradient for each example
    def partial_gradient(self, _, line):
        D = (map(float,line.split(',')))

        if D[1] != 0:
            weight_x = abs(1.0/D[1])
        else:
            weight_x = 0 #divide by zero

        # y_hat is the predicted value given current weights
        y_hat = self.weights[0]+self.weights[1]*D[1]
        # Update parial gradient vector with gradient form curre
nt example
        self.partial_Gradient =  [self.partial_Gradient[0]+ D[0]
-y_hat, self.partial_Gradient[1]+(D[0]-y_hat)*D[1]]
        self.partial_count = self.partial_count + 1
        #yield None, (D[0]-y_hat,(D[0]-y_hat)*D[1],1)

    # Finally emit in-memory partial gradient and partial count
    def partial_gradient_emit(self):
        yield None, (self.partial_Gradient,self.partial_count)

    # Accumulate partial gradient from mapper and emit total gra
dient
    # Output: key = None, Value = gradient vector
    def gradient_accumulater(self, _, partial_Gradient_Record):
        total_gradient = [0]*2
        total_count = 0
        for partial_Gradient,partial_count in partial_Gradient_R
ecord:
            total_count = total_count + partial_count
            total_gradient[0] = total_gradient[0] + partial_Grad
ient[0]
            total_gradient[1] = total_gradient[1] + partial_Grad
ient[1]
        yield None, [v/total_count for v in total_gradient]

    def steps(self):
        return [self.mr(mapper_init=self.read_weightsfile,
                        mapper=self.partial_gradient,
                        mapper_final=self.partial_gradient_emit,
```

```
Overwriting MrJobBatchGDUpdate_LinearRegression.py
```

In [13]:
```
from numpy import random,array
from MrJobBatchGDUpdate_LinearRegression import MrJobBatchGDUpda
te_LinearRegression

learning_rate = 0.05
stop_criteria = 0.000005

# Generate random values as inital weights
weights = array([random.uniform(-3,3),random.uniform(-3,3)])
# Write the weights to the files
with open('weights.txt', 'w+') as f:
    f.writelines(','.join(str(j) for j in weights))

# create a mrjob instance for batch gradient descent update over
all data
mr_job = MrJobBatchGDUpdate_LinearRegression(args=['LinearRegres
sion.csv',"--file",'weights.txt'])
# Update centroids iteratively
i = 0
while(1):
    print "iteration ="+str(i)+"  weights =",weights
    # Save weights from previous iteration
    weights_old = weights
    with mr_job.make_runner() as runner:
        runner.run()
        # stream_output: get access of the output
        for line in runner.stream_output():
            # value is the gradient value
            key,value =  mr_job.parse_output_line(line)
            # Update weights
            weights = weights + learning_rate*array(value)
    i = i + 1
    # Write the updated weights to file
    with open('weights.txt', 'w+') as f:
        f.writelines(','.join(str(j) for j in weights))
    # Stop if weights get converged
    if(sum((weights_old-weights)**2)<stop_criteria):
        break

print "Final weights\n"
print weights
```

```
iteration =0  weights = [ 0.1079431  -2.59810934]
No handlers could be found for logger "mrjob.job"
```

```
iteration =1  weights = [-0.09794304 -1.63631567]
iteration =2  weights = [-0.29340903 -0.93148414]
iteration =3  weights = [-0.4790095  -0.41496809]
iteration =4  weights = [-0.65526236 -0.0364593 ]
iteration =5  weights = [-0.82265306  0.24091094]
iteration =6  weights = [-0.98163793  0.44416212]
iteration =7  weights = [-1.13264696  0.59309563]
iteration =8  weights = [-1.27608605  0.70222302]
iteration =9  weights = [-1.41233891  0.78217918]
iteration =10  weights = [-1.54176866  0.84075788]
iteration =11  weights = [-1.66471927  0.88367083]
iteration =12  weights = [-1.78151672  0.91510385]
iteration =13  weights = [-1.8924702   0.93812452]
iteration =14  weights = [-1.99787298  0.95498089]
iteration =15  weights = [-2.09800342  0.96732044]
iteration =16  weights = [-2.19312573  0.97635047]
iteration =17  weights = [-2.28349074  0.98295576]
iteration =18  weights = [-2.36933663  0.98778468]
iteration =19  weights = [-2.4508896   0.99131235]
iteration =20  weights = [-2.52836445  0.99388697]
iteration =21  weights = [-2.60196523  0.99576367]
iteration =22  weights = [-2.67188572  0.99712939]
iteration =23  weights = [-2.73831001  0.99812112]
iteration =24  weights = [-2.80141296  0.99883923]
iteration =25  weights = [-2.86136066  0.99935724]
iteration =26  weights = [-2.91831091  0.99972903]
iteration =27  weights = [-2.9724136   0.99999404]
iteration =28  weights = [-3.02381112  1.00018118]
iteration =29  weights = [-3.07263874  1.00031161]
iteration =30  weights = [-3.11902497  1.0004008 ]
iteration =31  weights = [-3.16309187  1.00046009]
iteration =32  weights = [-3.20495541  1.00049778]
iteration =33  weights = [-3.24472578  1.00051993]
iteration =34  weights = [-3.28250762  1.00053095]
iteration =35  weights = [-3.31840037  1.00053409]
iteration =36  weights = [-3.35249848  1.00053169]
iteration =37  weights = [-3.38489169  1.00052547]
iteration =38  weights = [-3.41566523  1.00051667]
iteration =39  weights = [-3.44490011  1.0005062 ]
iteration =40  weights = [-3.47267323  1.0004947 ]
iteration =41  weights = [-3.49905771  1.00048264]
iteration =42  weights = [-3.52412296  1.00047035]
iteration =43  weights = [-3.54793495  1.00045806]
iteration =44  weights = [-3.57055634  1.00044594]
iteration =45  weights = [-3.59204667  1.0004341 ]
iteration =46  weights = [-3.61246248  1.00042261]
iteration =47  weights = [-3.6318575   1.00041151]
iteration =48  weights = [-3.65028277  1.00040085]
iteration =49  weights = [-3.66778678  1.00039062]
iteration =50  weights = [-3.68441559  1.00038083]
iteration =51  weights = [-3.70021296  1.00037149]
iteration =52  weights = [-3.71522046  1.00036257]
iteration =53  weights = [-3.72947759  1.00035407]
iteration =54  weights = [-3.74302187  1.00034598]
iteration =55  weights = [-3.75588893  1.00033827]
iteration =56  weights = [-3.76811264  1.00033094]
iteration =57  weights = [-3.77972516  1.00032397]
iteration =58  weights = [-3.79075706  1.00031734]
iteration =59  weights = [-3.80123737  1.00031104]
```

In [14]:
```
%%writefile buildSampleData.py
#!~/anaconda2/bin/python
# -*- coding: utf-8 -*-
from __future__ import division
import re
import random
import sys
import mrjob
from mrjob.job import MRJob
from mrjob.step import MRStep
from mrjob.protocol import RawValueProtocol

class MRbuildSample(MRJob):

    OUTPUT_PROTOCOL = RawValueProtocol
    INPUT_PROTOCOL = RawValueProtocol
    def configure_options(self):
        super(MRbuildSample, self).configure_options()

        self.add_passthrough_option(
            '--sampling-factor', type = 'float', default = 0.01)

    def mapper(self, _, line):
        if random.random() <= self.options.sampling_factor:
            yield None, line
    def steps(self):

        return [MRStep( mapper=self.mapper)
               ]

if __name__ == '__main__' and sys.argv[0].find('ipykernel') == -
1:
    MRbuildSample().run()
```

```
Overwriting buildSampleData.py
```

In [15]:
```
!rm -rf mr65ouput
!python buildSampleData.py -r local \
    --strict-protocols \
    --sampling-factor=0.01 \
    --output-dir=mr65ouput\
    LinearRegression.csv \
    --no-output \
    > /dev/null 2>&1
```

In [16]:
```
!cat mr65ouput/* > mrsample.csv
```

In [17]:
```
!wc -l mrsample.csv
```

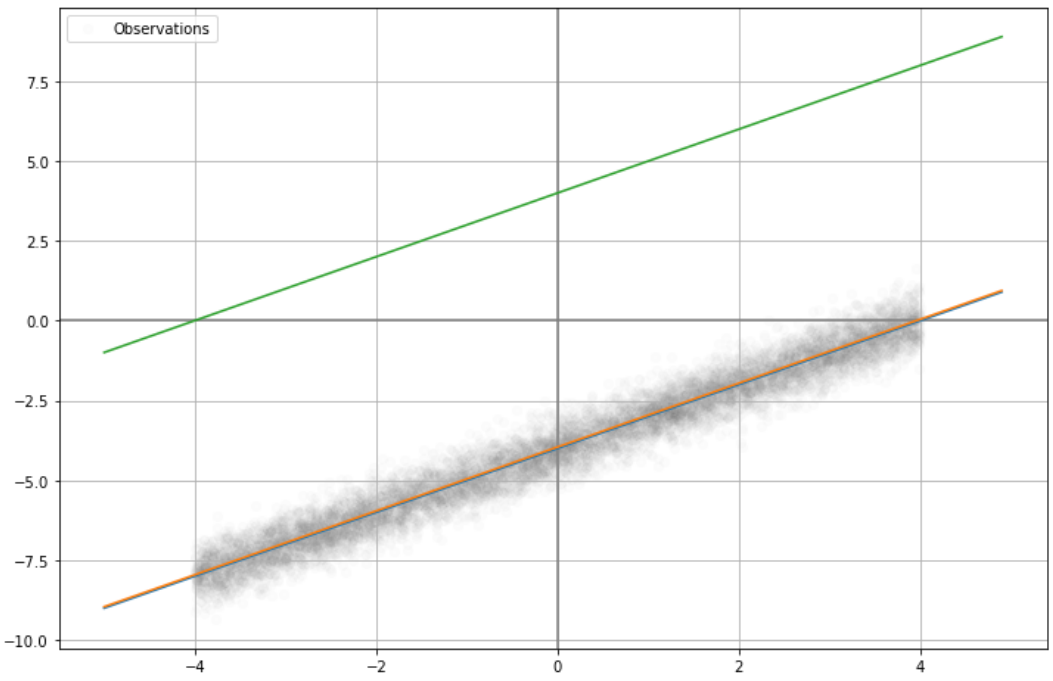
```
10135 mrsample.csv
```

In [18]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression

y_sample, x_sample = np.loadtxt('mrsample.csv', delimiter = ',')
.T



lr = LinearRegression()
lr.fit(x_sample.reshape(-1, 1), y_sample)

# The coefficients
print('Coefficients:', lr.intercept_,lr.coef_[0])
```

```
('Coefficients:', -4.0052780620689932, 0.99917985030323198)
```

In [19]:
```
x = np.arange(-5, 5, 0.1)
y_hat=x*1.0 - 4
mrjob = x * 1.00011967 -3.95658282
sklrn = x * 0.99888757487522928 - -4.0010069449567096

plt.figure(figsize=(12,8))
plt.grid(True)
plt.axhline(0, color='grey')
plt.axvline(0, color='grey')
plt.scatter(x_sample, y_sample, marker='o', color='grey', alpha=
0.02, label='Observations')
plt.plot( x, y_hat, x, mrjob,x,sklrn )
plt.legend(loc='upper left')
```

Out[19]: <matplotlib.legend.Legend at 0x7f76f6d43f90>



HW6.5.1 (OPTIONAL)

Using MRJob and in Python, plot the error surface for the weighted linear regression model using a heatmap and contour plot. Also plot the current model in the original domain space. (Plot them side by side if possible) Plot the path to convergence (during training) for the weighted linear regression model in plot error space and in the original domain space. Make sure to label your plots with iteration numbers, function, model space versus original domain space, etc. Comment on convergence and on the mean squared error using your weighted OLS algorithm on the weighted dataset versus using the weighted OLS algorithm on the uniformly weighted dataset.

In [20]:
```
# START STUDENT SOLUTION HW6.5.1
# insert cells as needed


# END STUDENT SOLUTION HW6.5.1
```

HW6.6 Clean up notebook for GMM via EM

Using the following notebook as a starting point:

http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/0t7985e40fovlkw/EM-GMM-MapReduce%20Design%201.ipynb (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/0t7985e40fovlkw/EM-GMM-MapReduce%20Design%201.ipynb)

Improve this notebook as follows:

- Add in equations into the notebook (not images of equations)
- Number the equations
- Make sure the equation notation matches the code and the code and comments refer to the equations numbers
- Comment the code
- Rename/Reorganize the code to make it more readable
- Rerun the examples similar graphics (or possibly better graphics)

Submission Instructions

Create a new notebook in your github repo and name it: MIDS-W261-HW-06-GMM-{your student id}.ipynb

HW6.7 Implement Bernoulli Mixture Model via EM

Implement the EM clustering algorithm to determine Bernoulli Mixture Model for discrete data in MRJob.

As a unit test use the dataset in the following slides:

https://www.dropbox.com/s/maoj9jidxj1xf5l/MIDS-Live-Lecture-06-EM-Bernouilli-MM-Systems-Test.pdf?dl=0 (https://www.dropbox.com/s/maoj9jidxj1xf5l/MIDS-Live-Lecture-06-EM-Bernouilli-MM-Systems-Test.pdf?dl=0)

Cross-check that you get the same cluster assignments and cluster Bernouilli models as presented in the slides after 25 iterations. Dont forget the smoothing.

As a full test: use the same dataset from HW 4.5, the Tweet Dataset. Using this data, you will implement a 1000-dimensional EM-based Bernoulli Mixture Model algorithm in MrJob on the users by their 1000-dimensional word stripes/vectors using K = 4. Use the same smoothing as in the unit test.

Repeat this experiment using your KMeans MRJob implementation fron HW4. Report the rand index score using the class code as ground truth label for both algorithms and comment on your findings.

Here is some more information on the Tweet Dataset.

Here you will use a different dataset consisting of word-frequency distributions for 1,000 Twitter users. These Twitter users use language in very different ways, and were classified by hand according to the criteria:

0: Human, where only basic human-human communication is observed.

1: Cyborg, where language is primarily borrowed from other sources (e.g., jobs listings, classifieds postings, advertisements, etc...).

2: Robot, where language is formulaically derived from unrelated sources (e.g., weather/seismology, police/fire event logs, etc...).

3: Spammer, where language is replicated to high multiplicity (e.g., celebrity obsessions, personal promotion, etc...)

Check out the preprints of recent research, which spawned this dataset:

http://arxiv.org/abs/1505.04342 (http://arxiv.org/abs/1505.04342) http://arxiv.org/abs/1508.01843 (http://arxiv.org/abs/1508.01843)

The main data lie in the accompanying file:

topUsers_Apr-Jul_2014_1000-words.txt

and are of the form:

USERID,CODE,TOTAL,WORD1_COUNT,WORD2_COUNT,... . .

where

USERID = unique user identifier CODE = 0/1/2/3 class code TOTAL = sum of the word counts

In [21]:
```
# START STUDENT SOLUTION HW6.7
# insert cells as needed


# END STUDENT SOLUTION HW6.7
```

HW6.8 (OPTIONAL) 1 Million songs

Predict the year of the song. Ask Jimi

In [ ]:

In [ ]: