Gradient Descent

Saul Garcia

November 22, 2015

Summary

The purpose of this project is to utilize the Gradient Descent Method with `Backtracking line search` and `Exact line search` in order to find the local minimum of a logarithmic function and compare the results. For this we will consider the following function with R^n dimensions.

$$f(x) = C^T X - \sum_{i=1}^{m} log_{10}(b_i - a_i^T X)$$

Intro

`Gradient descent` is a first-order optimization algorithm. In order to find a `local minimum` of a function, it takes `steps` proportional to the negative of the gradient until it reaches the minimum. There are two methods in which the Gradient descent achieves to find an appropriate step, or footstep, which are called: `Backtracking line search` and `Exact line search`.

Backtracking line search

Bakctracking line search is like most line searches used in practice, *inexact*. Backtracking is known for being quite simple, and it basically depends on two constants α, β with $0 < \alpha < .5$ and $0 < \beta < 1$.

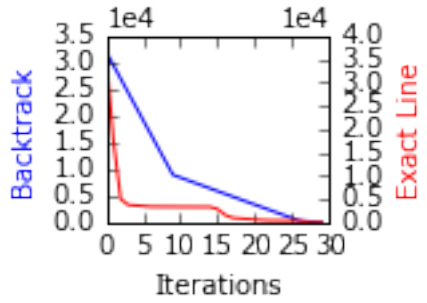
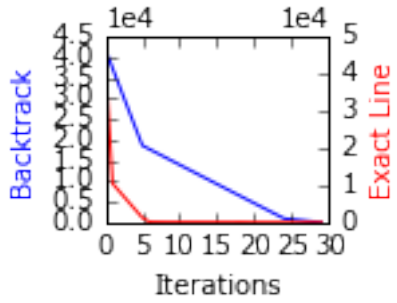
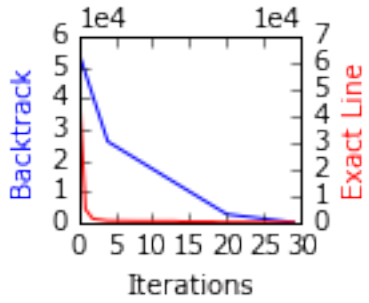
Basically the function will go through iterations starting with a footstep of $t := 1$, which will be updating **while** $f(x + t\Delta x) > f(x) + \alpha t \bigtriangledown f(x)^T \Delta x, t := \beta t$. When t is found for the given iteration, x will be updated with the following function: $x = x(t + \Delta x)$ where Δx is the negative gradient of our function, in order to keep advancing towards the local minimum (The function will be evaluated with each x for graphing purposes).

Exact line search

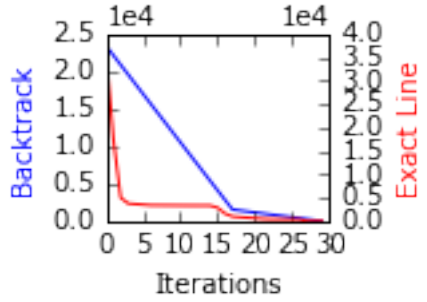
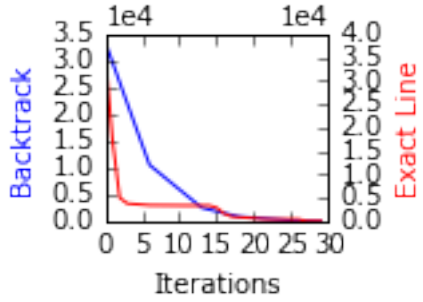
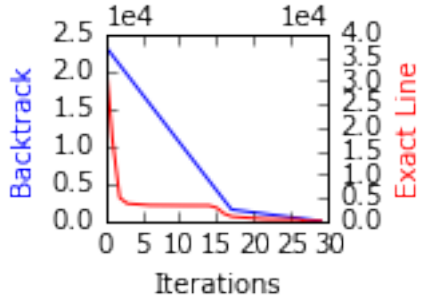
`Exact line search` is another method often used in optimization practice, in which t is chosen to minimize f along the ray $x + t\Delta x | t >= 0$: where $t = argmin_s >= 0 f(x + s\Delta x)$. Basically for this exact line search, an array of t was chosen with the following values: [.0001, .0005, .001, .005, .01, .05, .1, .5, 1, 1.5] and for each iteration, the value of x is updated by the same exact formula: $x = x(t + \Delta x)$ where t will be found throughout the minimization.

Experiment & Results

In order to compare both methods, the methods will be evaluated with by using different α, β, m and n. The experiments will be divided into groups, 1st to 3rd will be modifying m and n, and the 4th to 6th will be changing values for α and β. For generating the plots, we will be computing $f(x)$ in each iteration against the $f(x)_m in$ in order to vizualise when our function reaches its minimum.

Experiment 1	Experiment 2	Experiment 3
$m = 500$ $n = 100$ $\alpha = .1$ $\beta = .2$	$m = 400$ $n = 200$ $\alpha = .1$ $\beta = .2$	$m = 250$ $n = 250$ $\alpha = .1$ $\beta = .2$
		

By decreasing m and increasing n, several changes are noticable; first the logarithmic function will be taking higher amount of values and reducing the logarithmic summatory which results in our y axis being higher each time. For our `backtracking search`, we notice how the steps taken in order to reach the minimum are slower, but it reaches the local minimum around the 25th iteration for every different experiment. On the contrary, the `exact line search` takes a more direct path towards the minimum, each time being closer to the optimal value within less iterations: 25th, 5th and below 5.

Experiment 4	Experiment 5	Experiment 6
$m = 500$ $n = 100$ $\alpha = .2$ $\beta = .1$	$m = 500$ $n = 100$ $\alpha = .3$ $\beta = .3$	$m = 500$ $n = 100$ $\alpha = .4$ $\beta = .1$
		

For the next set of experiments the results are different. By modifying only α and β, the `exact line search` goes unmodified. But it is noticeable that a higher α leads to no change while a higher β makes the backtracking method get closer to the minimum for less iterations, even though it reaches the local minimum around the same iteration, around 25th.

(The graphs with bigger resolution may be found at the Appendix).

Conclusion

To conclude, the gradient descent method is a simple method which allows us to find the local minimum of a function. By comparing both methods `Backtracking` and `Exact Line`, it seems that the second one has

a more efficient path in order to find the local minimum, at least for a logarithmic function. It would still be interesting to see weather increasing the number of experiments lead to a different result, and possibly having different values for t in `Exact line Search` could be significant aswell.

Code

Importing the packages

```
# Import packages
import cvxpy as cvx
import scipy.optimize
import numpy as np
import random
import pandas as pd
import math
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from decimal import Decimal
from ggplot import *
```

Define the constants

```
# Problem Data
m = 500    #Subject to change upon experiment
n = 100    #Subject to change upon experiment
np.random.seed(1235813)

#Constants
iterations = 30
A = np.asmatrix(np.random.randint(low=-1,high=1,size=(n,m)))     #Very small number
b = np.asmatrix(np.random.randint(low=1000,high=2000,size=(m,1)))  #Very large number
c = np.asmatrix(np.random.randint(low=1,high=20,size=(n,1)))
x = np.asmatrix(np.random.randint(low=1,high=20,size=(n,1)))     #Very small number
x2= x
print A
```

Function $f(x)$

```
def function(x,A,b,c):
    fx = np.dot(c.T,x) - sum(np.log10((b - np.dot(A.T, x))))
    return fx
```

Calculate the Gradient Δx

```
def gradient(x,A,b,c):
    gradient = A * (1.0/(b - A.transpose()* x)) + c
    return gradient
```

Function for the footstep for Backtrack line search

```
#Backtracking footstep
def backtrack(x,A,b,c,t):
    alpha = 0.4
    beta = 0.1
    t=1
    while True:
        backtracking1 =function((x + t*-gradient(x,A,b,c)),A,b,c)
        backtracking2 = function(x,A,b,c) + alpha * t * gradient(x,A,b,c).transpose() * -(gradient(x,A,
        t=t*beta
        if backtracking1 < backtracking2:
            break

    return t
```

Function for the footstep for Exact line search

```
#Exact line search footstep
def exactline_t(x,A,b,c):
    t = np.array([.0001,.0005,.001,.005,.01,.05,.1,.5,1,1.5])
    tx = function((x + t.item(0)*-gradient(x,A,b,c)),A,b,c)
    tmin= t.item(0)
    for i in range(0,len(t)):
        if(tx >= function((x + t.item(i)*-gradient(x,A,b,c)),A,b,c)):
            tmin = t.item(i)
    tx =function((x + t.item(i)*-gradient(x,A,b,c)),A,b,c)
    return tmin
```

Iterations for Backtrack Gradient Descent

```
#BACKTRACKING
#Constanst
iterations = 30
t = 1

# Construct the problem for Backtracking
iterk = np.array([np.arange(iterations)]).transpose()
#Fn = np.array([np.arange(iterations)]).transpose()
Diff = np.array([np.arange(iterations)]).transpose()


for i in range(0, iterations):
    #Fn[i] = function(x,A,b,c)
    Diff[i] = function(x,A,b,c)
    t = backtrack(x,A,b,c,t)
    x = x + (t * -gradient(x,A,b,c))
    iterk[i] = i    #Need to update X = X + t * gradient
```

Iterations for Exact Line Search Gradient Descent

```
#EXACT LINE SEARCH

itere = np.array([np.arange(iterations)]).transpose()
Diffe = np.array([np.arange(iterations)]).transpose()
```

```
for i in range(0, iterations):
    Diffe[i] = function(x2,A,b,c)
    x2 = x2 + (exactline_t(x2,A,b,c) * -gradient(x2,A,b,c))
```

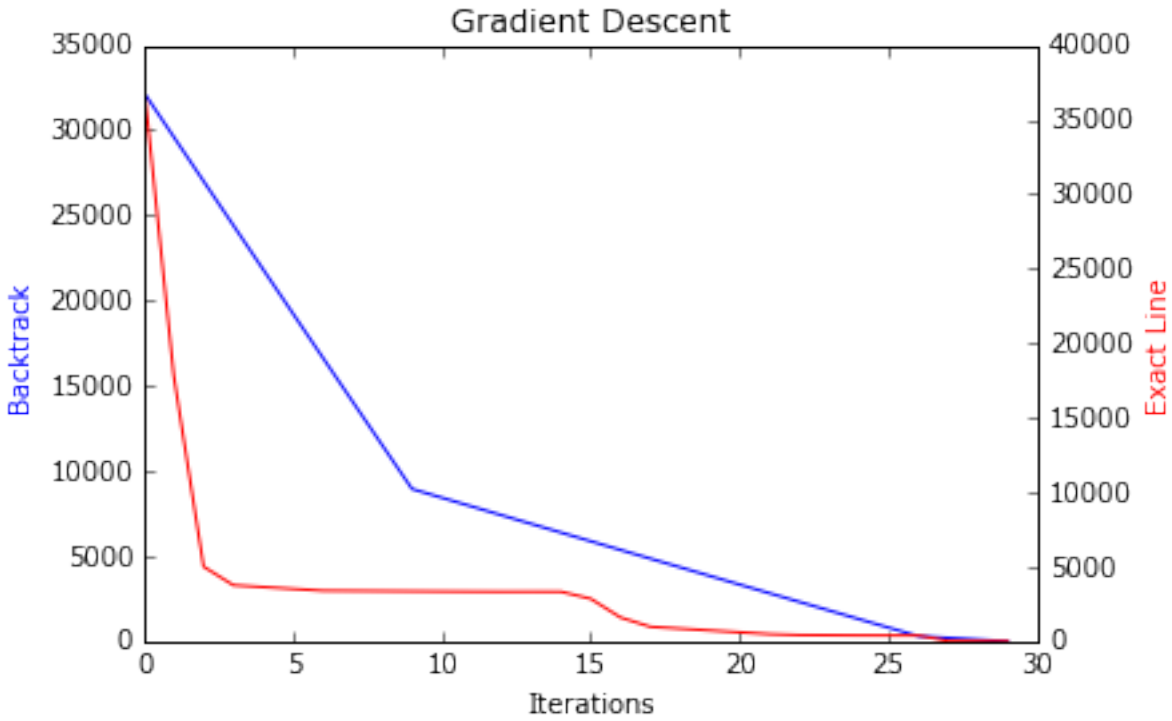
Plotting Backtracking and Exact line individually

```
#Dataframe for plotting
Dataframe = np.concatenate((itere,Diff-min(Diff),Diffe-min(Diffe)),axis=1)
df = pd.DataFrame(Dataframe)
df.columns = ['Iterations', 'Backtrack', 'Exact Line']

#Plotting Backtrack
btplot= ggplot(df, aes(x='Iterations',y='Backtrack')) + \
        geom_line(color='blue') + \
        geom_point() + \
        theme_bw()+ \
        ggtitle('Backtrack Gradient Descent')
btplot

#Plotting Exact Line
elplot =ggplot(df, aes(x='Iterations',y='Exact Line')) + \
        geom_line(color='red') + \
        geom_point(color='red') + \
        theme_bw() + \
        ggtitle('Exact Line Gradient Descent')
elplot
```

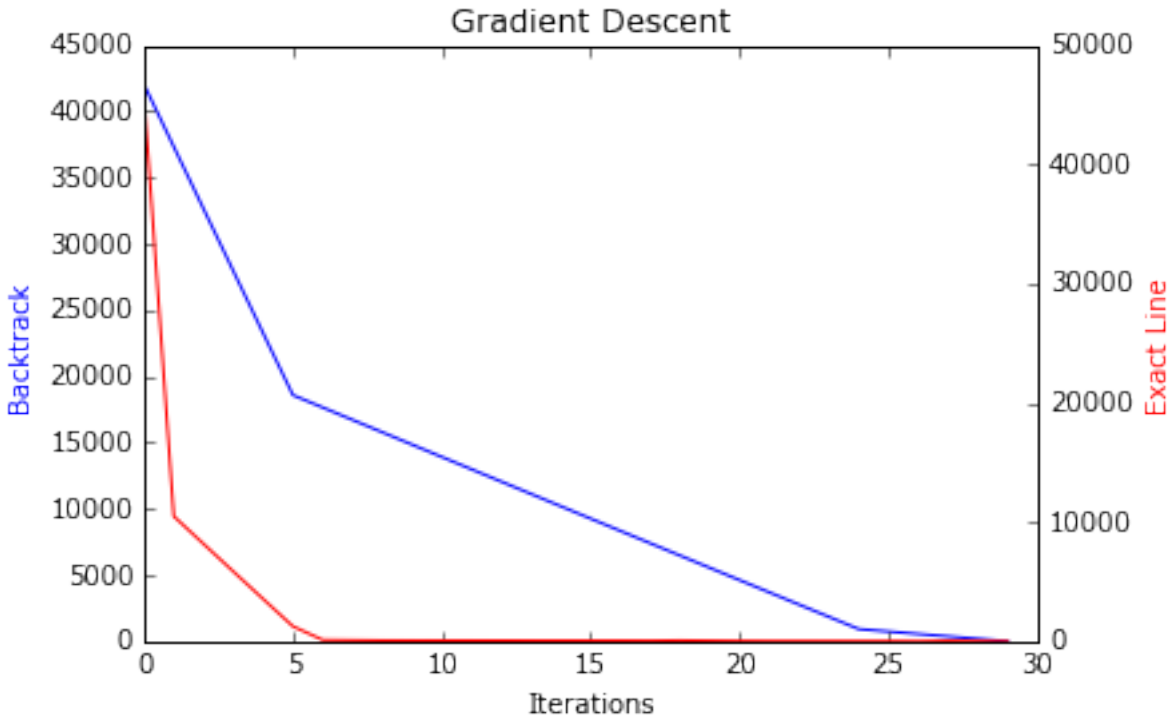
Plotting both methods together

```
#Plotting both Gradient Descent methods together
fig, ax1 = plt.subplots()
plt.title('Gradient Descent')
ax1.plot(iterk, Diff-min(Diff), 'b-')
ax1.set_xlabel('Iterations')
ax1.set_ylabel('Backtrack', color='b')



ax2 = ax1.twinx()
ax2.plot(iterk, Diffe-min(Diffe), 'r-')
ax2.set_ylabel('Exact Line', color='r')

plt.show()
```

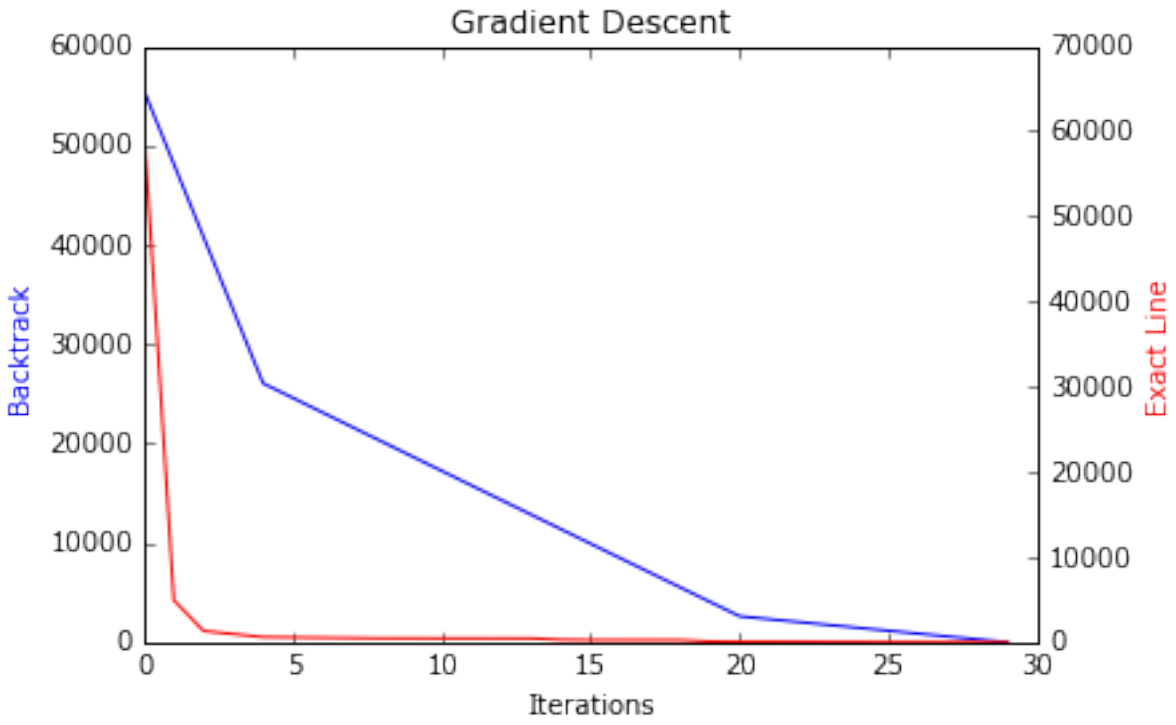
Appendix



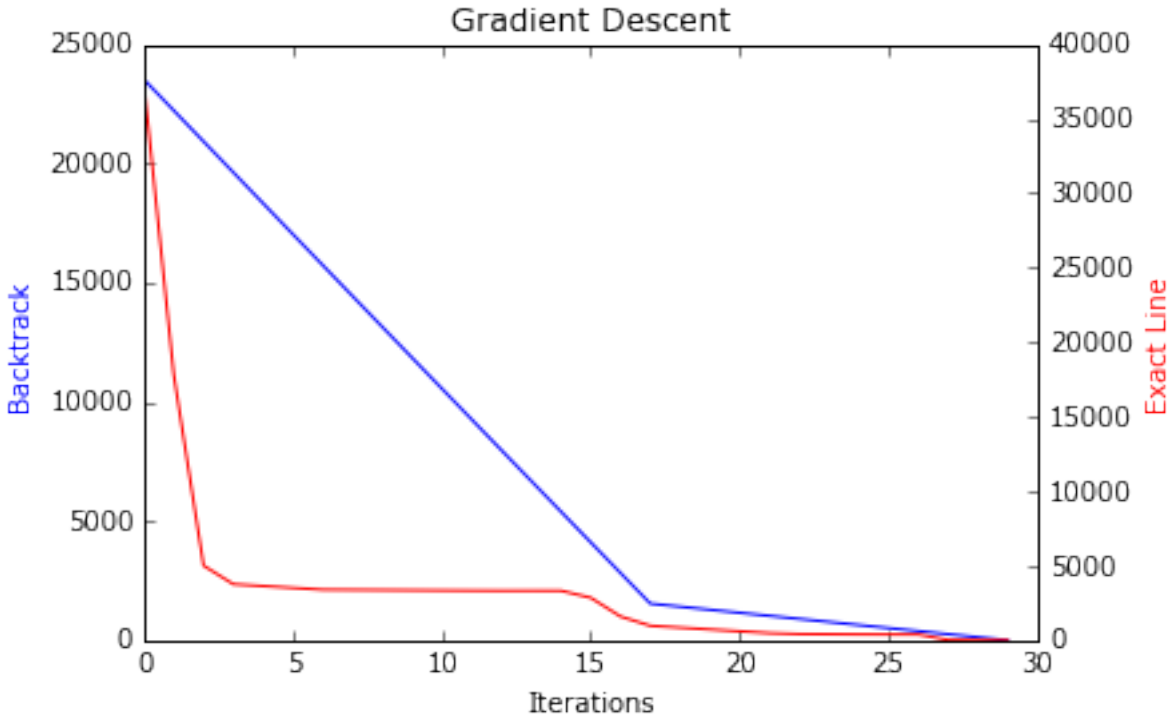
Experiment 1



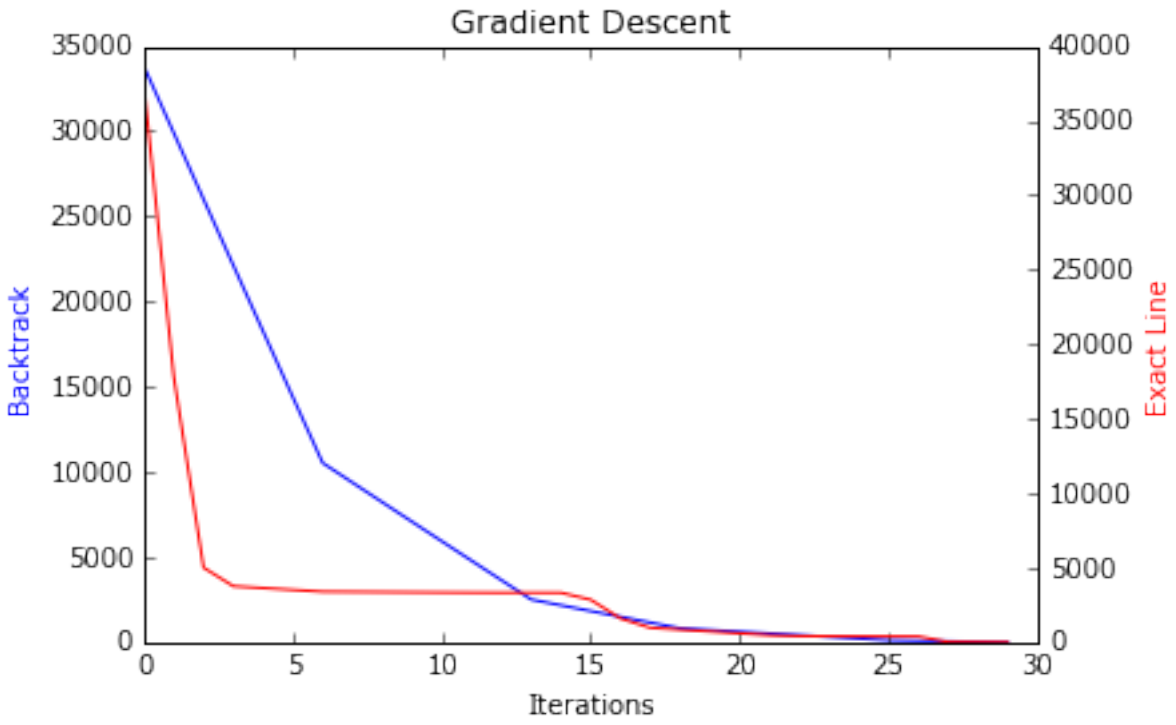
Experiment 2



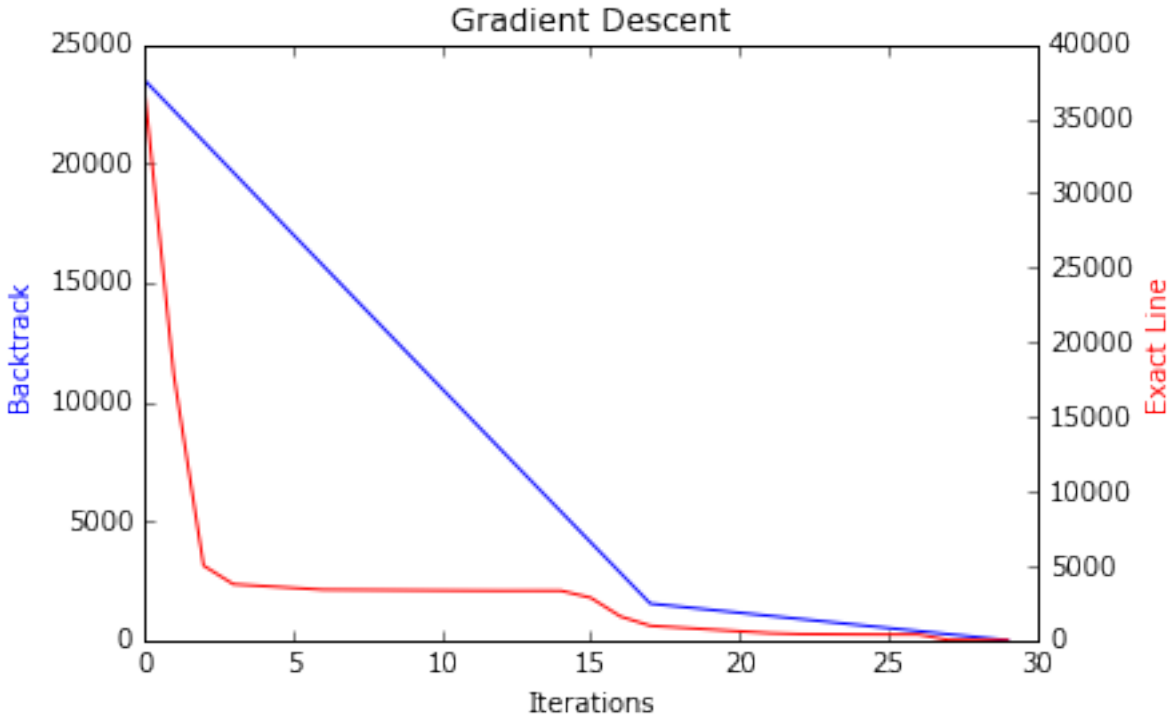
Experiment 3



Experiment 4



Experiment 5



Experiment 6