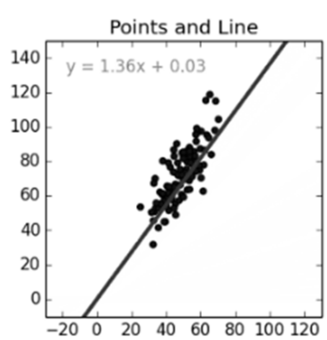
Linear Regression

Computation vs Gradient Descent

Linear Regression

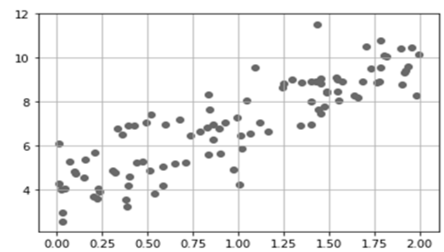


To do this we'll use the standard $y = mx + b$ slope equation where m is the line's slope and b is the line's y-intercept. To find the best line for our data, we need to find the best set of slope m and y-intercept b values.

Computational Approach

```
import matplotlib, matplotlib.pyplot as plt
%matplotlib inline

# To get the most random numbers for each run, call numpy.random.seed(
# To reproduce  experiments  use: numpy.random.seed(<some_constant>)

# Create a Matrix for X: 100 rows, one column
np.random.seed(666)
X = 2 *        np.random.rand(100,1)
Y = 4 + 3 * X+ np.random.randn(100,1)

print ("Both X and Y have type = ",type(X), "  shape = ", X.shape)

plt.scatter(X,Y)
plt.grid(True)
plt.show()
```

```
Both X and Y have type =  <class 'numpy.ndarray'>   shape =  (100, 1)
```



Computational Approach using good ol Linear Algebra

```
# Linear algebra solution -
# Values are close to actual 4 and 3
X_b = np.c_[np.ones((100,1)),X]
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(Y)
print (theta_best)
b_intercept = theta_best[0,0]
m_slope = theta_best[1,0]
```

```
[[4.02369667]
 [3.00517447]]
```

Computational Approach using sklearn

sklearn -- using randomized values for y = 4 + 3x

- works with our (100,1) X and Y matricies

```
from sklearn.linear_model import LinearRegression
# create an instance
lin_reg = LinearRegression()
reg = lin_reg.fit(X,Y)

# use attributes of lin_reg object
print("reg coeff=", reg.coef_ )
print ("intercept=", reg.intercept_ )
print ("predict x=72 : ", reg.predict(np.array([[72]])) )
```

```
reg coeff= [[3.00517447]]
intercept= [4.02369667]
predict x=72 :  [[220.39625844]]
```

Computational Approach using scipy

scipy - with simple lists

```
# do with scipy - need simple lists/arrays

# get simple lists out of matrix col 1
scix = X[:,0]
sciy = Y[:,0]

# Easy with scipy.stats.linregress
from scipy.stats import linregress
linregress(scix, sciy)
```

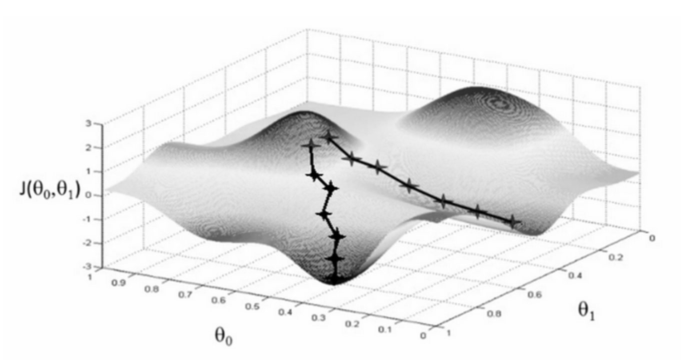
```
LinregressResult(slope=3.005174469063187, intercept=4.023696672103901,
stderr=0.17448712737736757)
```

Gradient Descent

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

To understand in an simpler way,let's us take the example Suppose you are at the top of a mountain, and you have to reach a lake which is at the lowest point of the mountain. A twist is that you are blindfolded and you have zero visibility to see where you are headed. So, what approach will you take to reach the lake?

The best way is to check the ground near you and observe where the land tends to descend. This will give an idea in what direction you should take your first step. If you follow the descending path, it is very likely you would reach the lake.



Sum of Squares Error Equation

Sum of squared distances formula (to calculate our error)

$$\text{Error}_{(\text{m,b})} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

Exercise 1

- Write the error function
  compute_error_for_line_given_points(b, m, points):

- where points is a 2D numpy array:

2 3

5 6

8 9

12 19

$$\text{Error}_{(\text{m,b})} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

Partial derivative with respect to b and m (to perform gradient descent)

$$\frac{\partial}{\partial \mathrm{m}} = \frac{2}{N}\sum_{i=1}^{N} - x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial \mathrm{b}} = \frac{2}{N}\sum_{i=1}^{N} - (y_i - (mx_i + b))$$

Exercise 2

```
def step_gradient(b_current, m_current, points, learning_rate):
    #gradient descent
    b_gradient = 0
    m_gradient = 0
    N = float(len(points))
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        b_gradient +=
        m_gradient +=
        new_b = b_current -
        new_m = m_current -
    return [new_b,new_m]
```

compute partial derivatives, add in negative to go downhill

adjust your new values based on learning rate

Expected Answer:

```
newvals = step_gradient(3, 4, points, .001)
print (newvals)
```

```
[2.9585, 3.6535]
```

Exercise 3

```
def gradient_descent_runner(points, starting_b, starting_m, learning_rate, num_iterations):
    b = starting_b
    m = starting_m
    for
    return [b,m]
```

iterate and compute new values for b and m based on learning rate

Test with these values

```
finvals = gradient_descent_runner(points, 1, 2, .001, 10000)
print (finvals)
```

```
[-1.3594932419233494, 1.5725899018407037]
```

Values obtained computationally

```
# compute values using formula
myx = [2,5,8,12]
myy = [3,6,9,19]
linregress(myx, myy)
```

```
LinregressResult(slope=1.5753424657534247, intercept=-1.3835616438356162,
3, stderr=0.2847206806962537)
```