**Linear Models- Pseudocode Gradient Descent**

```
initialize thetas to random values between [-1, 1]
previous_error := 0.0
current_error = calculate_error( thetas, data)
until abs( current_error – previous_error) < epsilon
    new_thetas = []
    for j = 0 to m
        new_thetas[ j] = thetas[ j] -
                           alpha * derivative( j, thetas, data)
    thetas = new_thetas
    previous_error = current_error
    current_error = calculate_error( thetas, data)
```

*Comments*

1. `epsilon` needs to be surprisingly small…probably around 1E-07.
2. `alpha` can be problematic. Too big and you "jump out of the cup"; too small and it'll take *forever*, to converge (defined by your `epsilon`). The best thing to do is to make `alpha` *adaptive*. If your error is increasing instead of decreasing that is a sign that your `alpha` is too big, reduce it. So you might start with `alpha` = 0.1 and then check to see if error is decreasing or not, if it is *increasing* then divide `alpha` by 10.
3. If you assume this code is wrapped in a function as described in the self-check, then "data" doesn't need to have any particular implementation…use one that you like. It should probably have $x_o = 1$ in it already (you don't want to do that more than once). You may want to strip off the "y" into a separate List

**Math**

The pseudocode above works for both linear regression and logistic regression.

It's worth repeating, exactly, what the formulas are for these. For linear regression, we have:

$$\hat{y} = \theta x$$

and for logistic regression we have:

$$\hat{y} = \frac{1}{1 + e^{-\theta x}}$$

Remember that for linear regression, y-hat can be anything but for logistic regression, y-hat is constrained to be between 0 and 1, exclusive.

Each has a different implementations for their `calculate_error()` functions (this actually is true *only* if you are using logistic regression for classification. If you're trying to actually use logistic regression as a numerical model, you need to use the other error function but don't worry about that now).

Remember that the Error for Linear Regression is:

$$J(\theta) = \frac{1}{2n}\sum_i (\hat{y}_i - y_i)^2$$

Quite honestly, it doesn't matter if the "2" is in there as it doesn't make one iota of difference for a *minimization* problem if successive values are multiplied by ½ or not. This entire summation is capsulated in the `calculate_error()` function. (Summation is a loop, you loop over the *n* data points…unless you're pythonic and then you just use a list comprehension and sum at the end).

The Error for Logistic Regression is trickier but again, it's going to be inside the `calculate_error()` function:

$$J(\theta) = -\frac{1}{n}\sum_i y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)$$

Again, the summation just indicates there is a loop in your calculate_error() function. If you're using NumPy, I assume you're figured out that that isn't true for you.

*Theoretically*, the logs should never cause a problem if y is 0, then you should avoid calculating log(y_hat). If (1 – y) is 0, the same is true. However….in *practice*, you can have underflow errors and log(y_hat) throw an error because y_hat will be 0 when it *theoretically* cannot be. You should check for those errors and just make log(y_hat) return a nice default value.

Both of these error functions have the same derivative. It's just that "y hat" is a different function (from above):

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{n}\sum_i (\hat{y}_i - y_i) x_{ij}$$

So what does this mean? Let’s skip the summation for now…that’s just to calculate an average.

For a specific data point, the adjustment to the j-th theta should be the actual error (y-hat – y) times that data point’s contribution to the error which is, the j-th x. Taking in (1/n) and the summation just says, “adjust the j-th theta *by the average of those weighted errors.*”

Raw LaTex:
J(\theta)=\frac{1}{2n}\sum_i(\hat{y_i} - y_i)^2
J(\theta)=-\frac{1}{n}\sum_i y_i log(\hat{y_i}) + (1 - y_i)log(1 - \hat{y_i})
\frac{\partial J}{\partial \theta_j} = \frac{1}{n}\sum_i(\hat{y_i} - y_i)x_{ij}