Question 1: Lake Mendota Ice [100 points]

Can you predict how harsh this Wisconsin winter will be?

The Wisconsin State Climatology Office keeps a record on the number of days Lake Mendota was covered by ice at `http://www.aos.wisc.edu/~sco/lakes/Mendota-ice.html`. For this homework, we will use the data.txt provided. Each line in the file contains two numbers, the first indicating the year and the second indicating the number of days the lake is covered by ice.

Write a program **Ice.java** with the following command line format:

```
$java Ice FLAG [arg1 arg2]
```

Where the two optional arguments are real valued.

Questions 5 is 20 points; other questions 10 points each.

1. As with any real problems, the data is not as clean or as organized as one would like for machine learning. Curate a clean data set starting from 1855-56 and ending in 2016-17. Let x be the year: for 1855-56, $x = 1855$; for 2016-17, $x = 2016$; and so on. Let y be the ice days in that year: for 1855-56, $y = 118$; for 2016-17, $y = 65$; and so on. Some years have multiple freeze thaw cycles such as 2001-02, that one should be $x = 2001, y = 21$. Although we do not ask you to hand in any visualization code or figures, we strongly advise you to plot the data (using any plotting tool inside or outside Java) and see what it is like.

   For simplicity, hard code the data set in your program. When FLAG=100, print out the data set. One year per line with the x value first, a space, then the y value. For example,

   ```
   $java Ice 100
   1855 118
   ...
   2001 21
   ...
   2016 65
   ```

2. When FLAG=200, print n the number of data points, the sample mean $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$, and the sample standard deviation $\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2}$, on three lines. For real values in this homework, keep two digits after decimal point. For example (the numbers are made-up):

   ```
   $java Ice 200
   162
   123.45
   32.10
   ```

3. We will perform linear regression with the model

   $$f(x) = \beta_0 + \beta_1 x.$$

   We first define the mean squared error as a function of β_0, β_1:

   $$MSE(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_i - y_i)^2.$$

When FLAG=300, arg1=β_0 and arg2=β_1. Print the corresponding MSE.

```
$java Ice 300 0 0
10897.85
$java Ice 300 100.00 0
386.74
$java Ice 300 300.00 -0.10
333.08
$java Ice 300 400 0.1
241561.79
$java Ice 300 200 -0.2
84199.70
```

4. We perform gradient descent on MSE. At current parameter (β_0, β_1), the gradient is defined by the vector of partial derivatives

$$\frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_0} = \frac{2}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_i - y_i) \quad (1)$$

$$\frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_1} = \frac{2}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_i - y_i)x_i. \quad (2)$$

When FLAG=400, arg1=β_0 and arg2=β_1. Print the corresponding gradient as two numbers on separate lines.

```
$java Ice 400 0 0
-205.11
-396150.93
$java Ice 400 100 0
-5.11
-9050.93
$java Ice 400 300 -0.1
7.79
15479.64
$java Ice 400 400 0.1
981.99
1901918.51
$java Ice 400 200 -0.2
-579.31
-1121289.79
```

5. Gradient descent starts from initial parameter $(\beta_0^{(0)}, \beta_1^{(0)})$, and iterates the following updates at time $t = 1, 2, \ldots, T$:

$$\beta_0^{(t)} = \beta_0^{(t-1)} - \eta \frac{\partial MSE(\beta_0^{(t-1)}, \beta_1^{(t-1)})}{\partial \beta_0} \quad (3)$$

$$\beta_1^{(t)} = \beta_1^{(t-1)} - \eta \frac{\partial MSE(\beta_0^{(t-1)}, \beta_1^{(t-1)})}{\partial \beta_1}. \quad (4)$$

When FLAG=500, arg1=η and arg2=T. Start from initial parameter $(\beta_0^{(0)}, \beta_1^{(0)}) = (0, 0)$. Perform T iterations of gradient descent. Print the following in each iteration on a line, separated by space: $t, \beta_0^{(t)}, \beta_1^{(t)}, MSE(\beta_0^{(t)}, \beta_1^{(t)})$. For example,

```
$java Ice 500 1e-7 5
1 0.00 0.04 1086.78
2 0.00 0.05 471.97
3 0.00 0.05 433.44
4 0.00 0.05 431.03
5 0.00 0.05 430.88

$java Ice 500 1e-8 5
1 0.00 0.00 9387.32
2 0.00 0.01 8094.78
3 0.00 0.01 6988.77
4 0.00 0.01 6042.37
5 0.00 0.02 5232.56

$java Ice 500 1e-9 5
1 0.00 0.00 10741.50
2 0.00 0.00 10587.49
3 0.00 0.00 10435.78
4 0.00 0.00 10286.34
5 0.00 0.00 10139.12

$java Ice 500 1e-6 5
1 0.00 0.40 442211.52
2 -0.00 -2.18 18646694.58
3 0.01 14.54 787004462.48
4 -0.05 -94.08 33217129357.42
5 0.32 611.63 1401997461432.22
```

Note with $\eta = 1e-6$ gradient descent is diverging.

The following is not required for hand in, but try different initial parameters, η, and much larger T and see how small you can make MSE.

6. Instead of using gradient descent, we can compute the closed-form solution for the parameters directly. For ordinary least squared in 1D, this is

$$\hat{\beta}_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

where $\bar{x} = (\sum x_i)/n$ and $\bar{y} = (\sum y_i)/n$. Required: When FLAG=600, print $\hat{\beta}_0, \hat{\beta}_1$ (note the order), and the corresponding MSE on a single line separated by space.

The following is not required for hand in, but give it some thought: what does a negative $\hat{\beta}_1$ mean?

7. Use $\hat{\beta}_0, \hat{\beta}_1$ you can predict the number of ice days for a future year. Required: When FLAG=700, arg1=year. Print a single real number which is the predicted ice days for that year. For example,

```
$java Ice 700 2020
86.30
```

   The following is not required for hand in, but give it some thought:

   - What's the prediction for this winter?
   - Which year will the predicted ice day become negative?
   - What does that say about the model?

8. If you are agonizing over your inability to get gradient descent to match the closed-form solution in question 5, you are not alone. The culprit is the scale of input x compared to the scale of the implicit offset value 1 (think $\beta_0 = \beta_0 \cdot 1$). Gradient descent converges slowly when these scales differ greatly, a situation known as bad condition number in optimization. When FLAG=800, we ask you to first normalize the input x (note: not the labels y):

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5}$$

$$std_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{6}$$

$$x_i \leftarrow (x_i - \bar{x})/std_x. \tag{7}$$

   Then proceed exactly as when FLAG=500. The two arguments are again arg1=η and arg2=T. For example,

```
$java Ice 800 0.1 5
1 20.51 -1.79 7082.51
2 36.92 -3.23 4640.63
3 50.05 -4.38 3077.80
4 60.55 -5.31 2077.56
5 68.95 -6.05 1437.39

$java Ice 800 1 5
1 205.11 -17.94 10895.86
2 0.00 -0.22 10893.93
3 205.11 -17.72 10892.04
4 -0.00 -0.44 10890.19
5 205.11 -17.51 10888.39

$java Ice 800 0.01 5
1 2.05 -0.18 10478.17
2 4.06 -0.36 10075.10
```

```
3 6.03 -0.53 9687.99
4 7.96 -0.70 9316.22
5 9.85 -0.86 8959.16
```

With $\eta = 0.1$ you should get convergence within 100 iterations.

(Note the β's are now for the normalized version of x, but you can easily translate them back for the original x with a little algebra. This is not required for the homework.)

9. Now we implement Stochastic Gradient Descent (SGD). With everything the same as part 8 (including x normalization), we modify the definition of gradient in equations (1) and (2) as follows. In iteration t we randomly pick one of the n items. Say we picked (x_{j_t}, y_{j_t}). We approximate the gradient using that item only:

$$\frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_0} \approx 2(\beta_0 + \beta_1 x_{j_t} - y_{j_t}) \quad (8)$$

$$\frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_1} \approx 2(\beta_0 + \beta_1 x_{j_t} - y_{j_t}) x_{j_t}. \quad (9)$$

When FLAG=900, print the same information as in part 8. For example (your results will differ because of randomness in the items picked):

```
$java Ice 900 0.1 5
1 24.60 -36.45 7123.53
2 42.85 -40.53 4850.61
3 49.01 -47.42 4631.96
4 45.74 -42.09 4614.13
5 67.18 -8.49 1550.91
```

With $\eta = 0.1$ you should approximately converge within a few hundred iterations.

Since n is small in our data set, there is little advantage of SGD over gradient descent. However, on large data sets SGD becomes more desirable.

10. Hints:
    1. Update β_0, β_1 simultaneously in an iteration. Don't use a new β_0 to calculate β_1.
    2. Use double instead of float in Java.
    3. Don't round the variables themselves to 2 digits in the middle stages. That is for printing only.