Homework 1

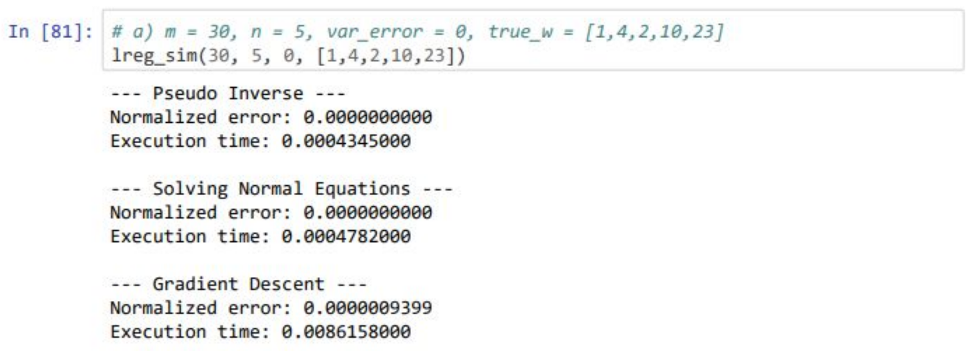
EE425X - Machine Learning: A signal processing perspective

An Nguyen

Python version: 3.8

1 Simulated Data

The independent variables are generated from a standard normal distribution. All entries of the data are independent identically distributed. The dependent variable (y) is derived from the independent variables plus a random error term. The error term follows a normal distribution with mean 0 and a given variance.

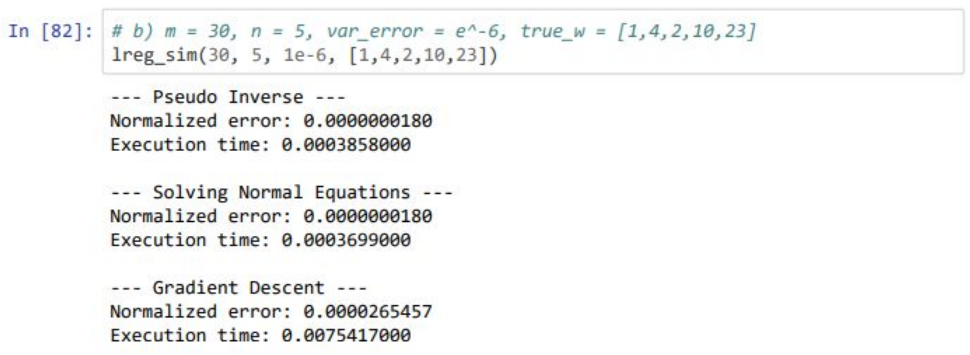


```
In [81]: # a) m = 30, n = 5, var_error = 0, true_w = [1,4,2,10,23]
         lreg_sim(30, 5, 0, [1,4,2,10,23])

--- Pseudo Inverse ---
Normalized error: 0.0000000000
Execution time: 0.0004345000

--- Solving Normal Equations ---
Normalized error: 0.0000000000
Execution time: 0.0004782000

--- Gradient Descent ---
Normalized error: 0.0000009399
Execution time: 0.0086158000
```

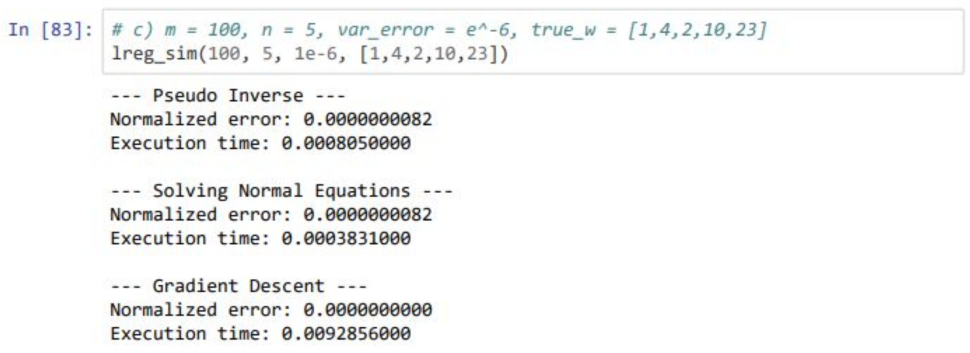


```
In [82]: # b) m = 30, n = 5, var_error = e^-6, true_w = [1,4,2,10,23]
         lreg_sim(30, 5, 1e-6, [1,4,2,10,23])

--- Pseudo Inverse ---
Normalized error: 0.0000000180
Execution time: 0.0003858000

--- Solving Normal Equations ---
Normalized error: 0.0000000180
Execution time: 0.0003699000

--- Gradient Descent ---
Normalized error: 0.0000265457
Execution time: 0.0075417000
```

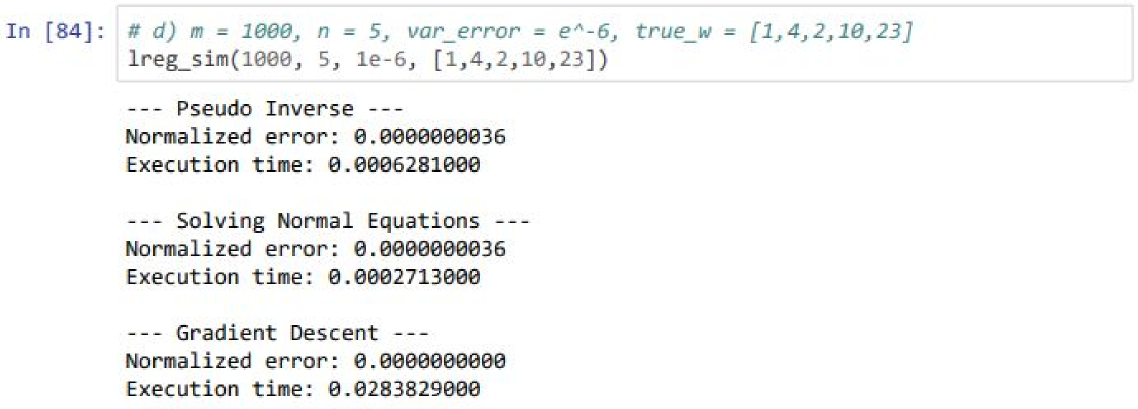


```
In [83]: # c) m = 100, n = 5, var_error = e^-6, true_w = [1,4,2,10,23]
         lreg_sim(100, 5, 1e-6, [1,4,2,10,23])

--- Pseudo Inverse ---
Normalized error: 0.0000000082
Execution time: 0.0008050000

--- Solving Normal Equations ---
Normalized error: 0.0000000082
Execution time: 0.0003831000

--- Gradient Descent ---
Normalized error: 0.0000000000
Execution time: 0.0092856000
```

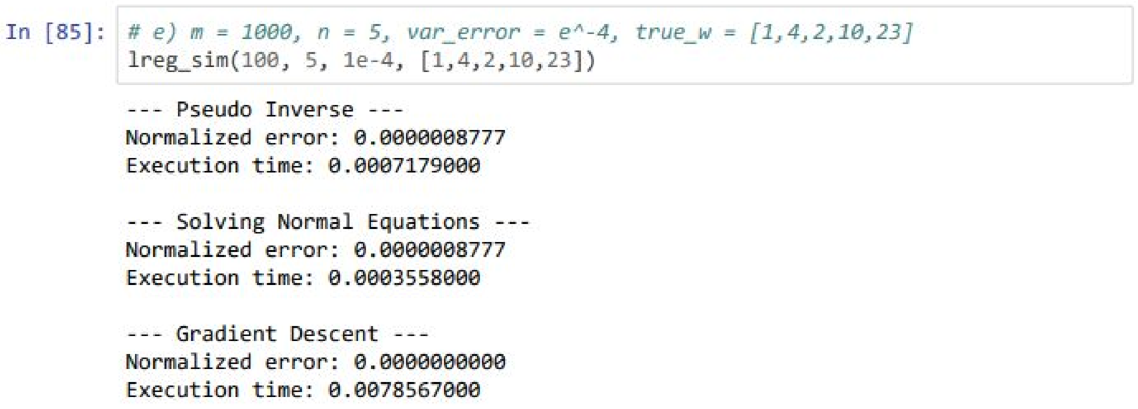


```
In [84]: # d) m = 1000, n = 5, var_error = e^-6, true_w = [1,4,2,10,23]
         lreg_sim(1000, 5, 1e-6, [1,4,2,10,23])

         --- Pseudo Inverse ---
         Normalized error: 0.0000000036
         Execution time: 0.0006281000

         --- Solving Normal Equations ---
         Normalized error: 0.0000000036
         Execution time: 0.0002713000

         --- Gradient Descent ---
         Normalized error: 0.0000000000
         Execution time: 0.0283829000
```



```
In [85]: # e) m = 1000, n = 5, var_error = e^-4, true_w = [1,4,2,10,23]
         lreg_sim(100, 5, 1e-4, [1,4,2,10,23])

         --- Pseudo Inverse ---
         Normalized error: 0.0000008777
         Execution time: 0.0007179000

         --- Solving Normal Equations ---
         Normalized error: 0.0000008777
         Execution time: 0.0003558000

         --- Gradient Descent ---
         Normalized error: 0.0000000000
         Execution time: 0.0078567000
```

The normalized errors and execution times are reported above. Here are a few observations from our results:

- The normalized error is 0 when the error terms are all 0 (we get a perfect fit of the regression line).
- The normalized error is negatively correlated with the sample size (more data points give us better estimates of the parameters vector)
- Pseudo Inverse method is faster than Solving Normal Equations method for smaller sample size. Both methods are much faster than the Gradient Descent method in our experiment.

2 Real Data

With the following parameters, we apply the Gradient Descent method to estimate θ

- max_iter: 10000

- learning_rate: 10^{-11}

The choices of max_iter and learning_rate are arbitrary (trial and error) so that learning rate is not too large (learning rate > 10^{-10} provides a NaN result as its product with the gradient exceeds the limit of python). Meanwhile, larger number of iterations only produces minimally better result.

- The mean square error: 1344.51
- Estimated parameters vector: [2.6e-3, 3.6e-1, 6.6e-3, 1.95, 6.0e-4, 4.9e-2]
- Solution of the least square error: [-1.3e-03, -4.2e-01, 36, 0.1, -1.5e02, 1.3e+02]

The last term of the parameters vector is the intercept of the regression function.

3 Extra Credit

With standardized features the mean square error is reduced to 23.03

4 Extra Credit

We implement the (batch) stochastic gradient descent on the standardized dataset. We can see that as batch size increases, the convergent rate also increases as first. But after a certain threshold the convergent rate decreases again (as the number of iterations gets smaller and smaller)

Batch size	**Mean square error**
1	732.63
10	688.27
50	569.14
200	350.97
400	91.30
600	664.07
800	14703.17