Sigurður Ágúst Jakobsson

Project: Assignment 3 - Sequential Estimation

Section 1.2

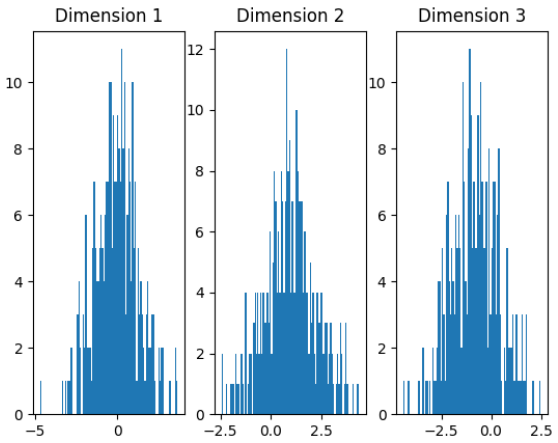
Create 300 3-dimensional data points sampled from N_3([0, 1, -1], sqrt(3))

Do you expect the batch estimate to be exactly (0, 1, -1) ? Which two parameters can be used to make this estimate more accurate?

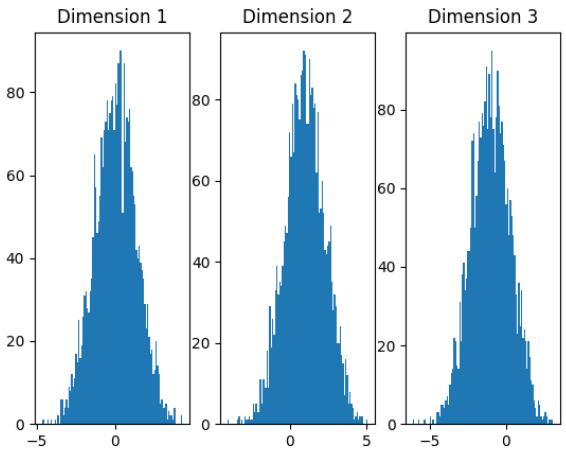
I would not expect the batch estimate to be exactly (0, 1, -1) from this generator. We are sampling random points from a distribution with a variance, so it is to be expected that they do not reflect the 'true' mean. We have to take an estimate of it after the fact and some error is to be expected. Parameters **n** and **var** (I think this is actually standard deviation in the function) can be used to make the estimate more accurate. Increasing **n** and decreasing **var** can make the estimate more accurate. Please see examples below. A deviation table at the bottom supports the statement above about increasing n or decreasing var.

Means

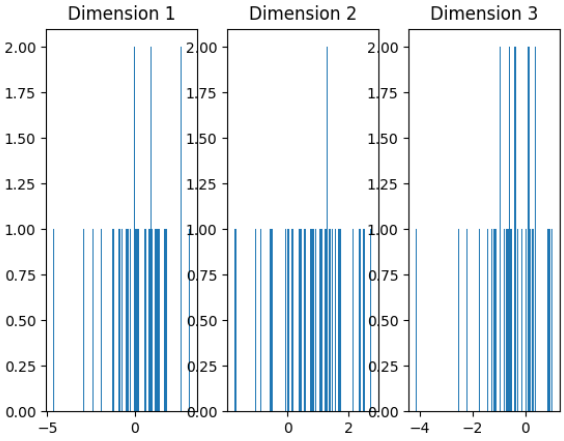
[-0.04813382 0.94639623 -0.83822799] – Benchmark: n=300, var=3^0,5 (sd = 3^0,25)



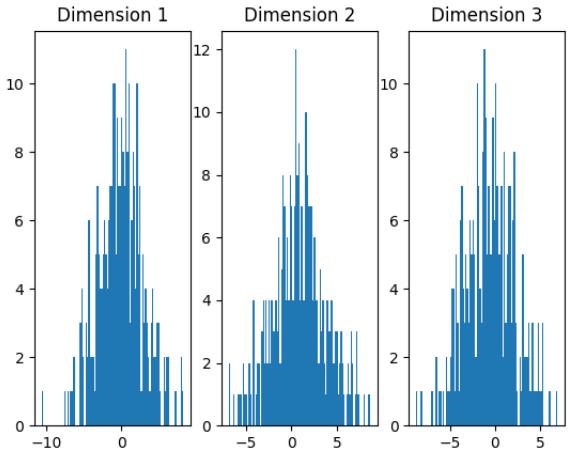
[0.04878703 0.99735841 -0.97989567] – N+: n=3000, var=3^0,5 (sd = 3^0,25)



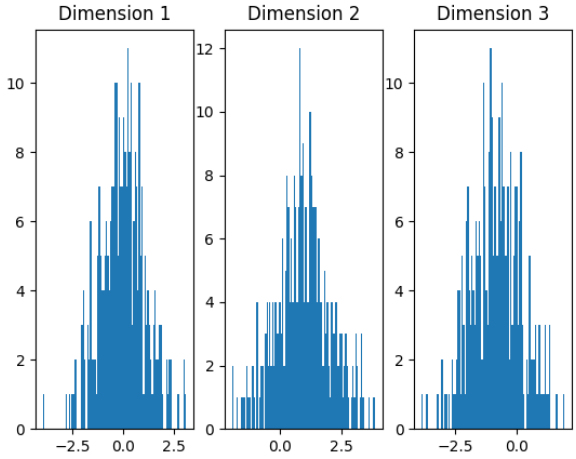
[0.14953321 0.73435783 -0.61599889] – N-: n=30, var=3^0,5 (sd = 3^0,25)



[-0.10972138 0.87780983 -0.63123956] – var+: n=300, var=9 (sd = 3^1)



[-0.04082086 0.95454024 -0.86280598] – var-: n=300, var=3^0,2 (sd = 3^0,1)



Deviation from given means:

[-0.04813382 -0.05360377 0.16177201] – Benchmark: n=300, var=3^0,5 (sd = 3^0,25)

[0.04878703 -0.00264159 0.02010433] – N+: n=3000, var=3^0,5 (sd = 3^0,25)

[0.14953321 -0.26564217 0.38400111] – N-: n=30, var=3^0,5 (sd = 3^0,25)

[-0.10972138 -0.12219017 0.36876044] – var+: n=300, var=9 (sd = 3^1)

[-0.04082086 -0.04545976 0.13719402] – var-: n=300, var=3^0,2 (sd = 3^0,1)

This table shows that the estimates generally increase as stated above.

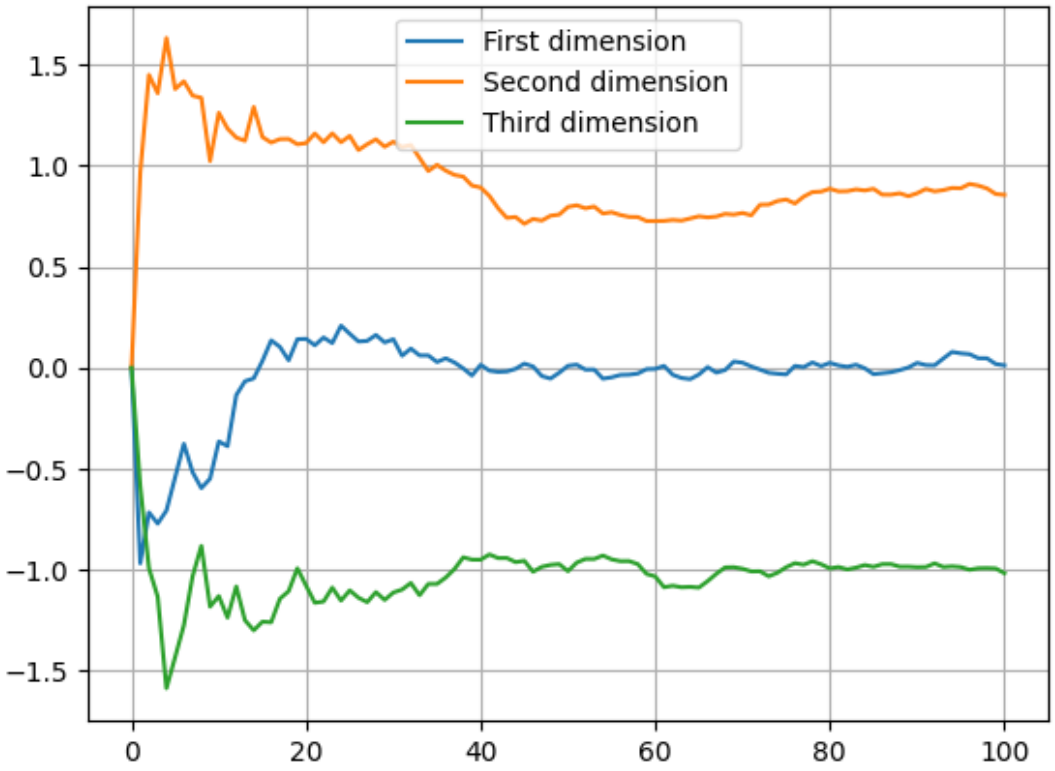
Section 1.5

Generate 100 3-dimensional points with the same mean and variance as above.

Set the initial estimate as (0, 1, -1)

And perform `update_sequence_mean` for each point in the set.

Collect the estimates as you go

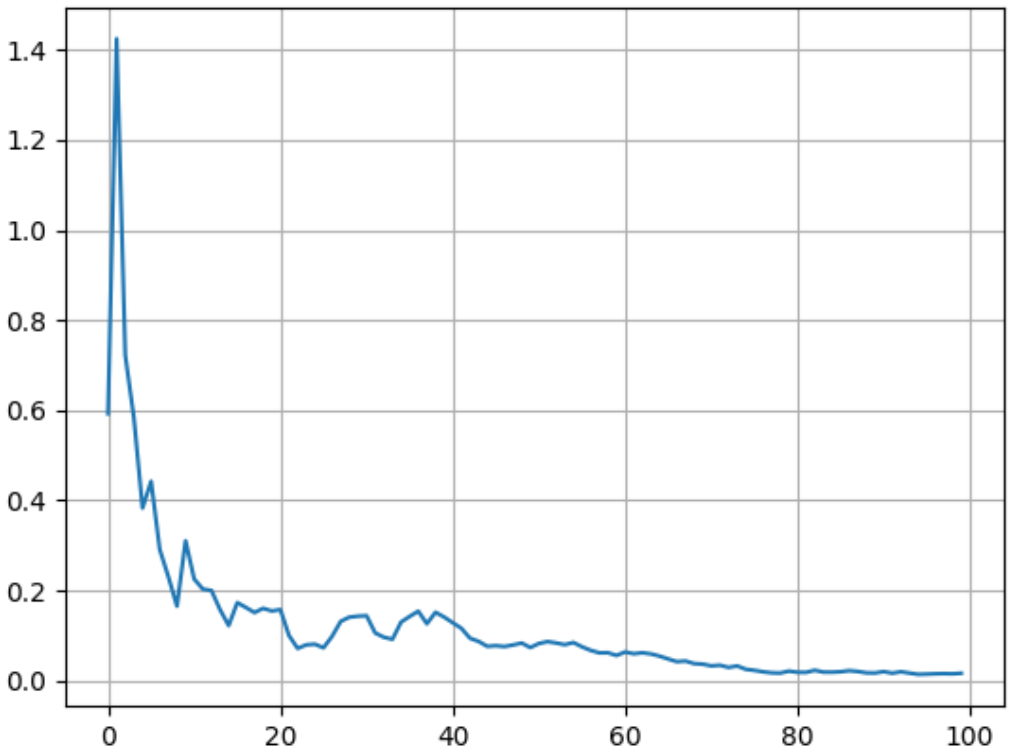


Section 1.6

Lets now plot the squared error between the estimate and the actual mean after every update.

The squared error between e.g. a ground truth y and a prediction y^ is (y-y^)^2.

Of course our data will be 3-dimensional so after calculating the squared error you will have a 3-dimensional error. Take the mean of those three values to get the average error across all three dimensions and plot those values.



Independent Section

What happens if the mean value changes (perhaps slowly) with time? What if μ=(0,1,−1) moves μ=(1,−1,0) in 500 time ticks? How would we track the mean? Some sort of a forgetting could be added to the update equation. How would that be done?

Create this type of data and formulate a method for tracking the mean.

Plot the estimate of all dimensions and the mean squared error over all three dimensions. Turn in these plots as `indep_1.png` and `indep_2.png`.

Write a short summary how your method works.

In thinking about this I typed some thoughts into a search engine and landed at an Investopedia article about simple moving averages: https://www.investopedia.com/terms/s/sma.asp

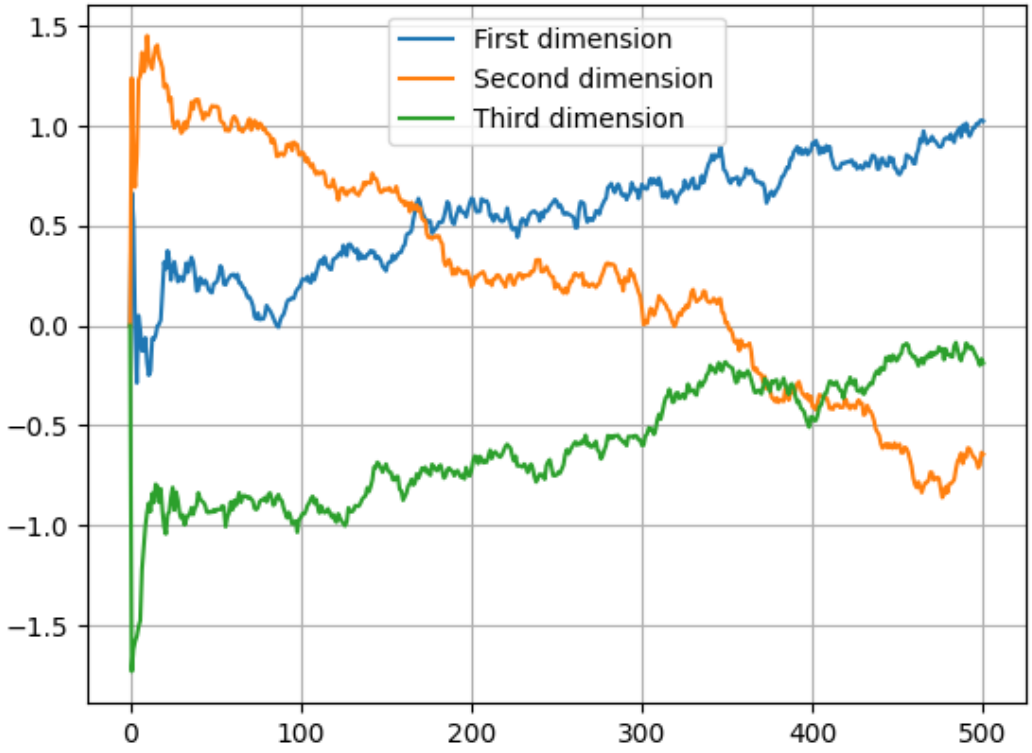
I tried to implement something that followed these ideas without having to recalculate the mean over a large set of data.

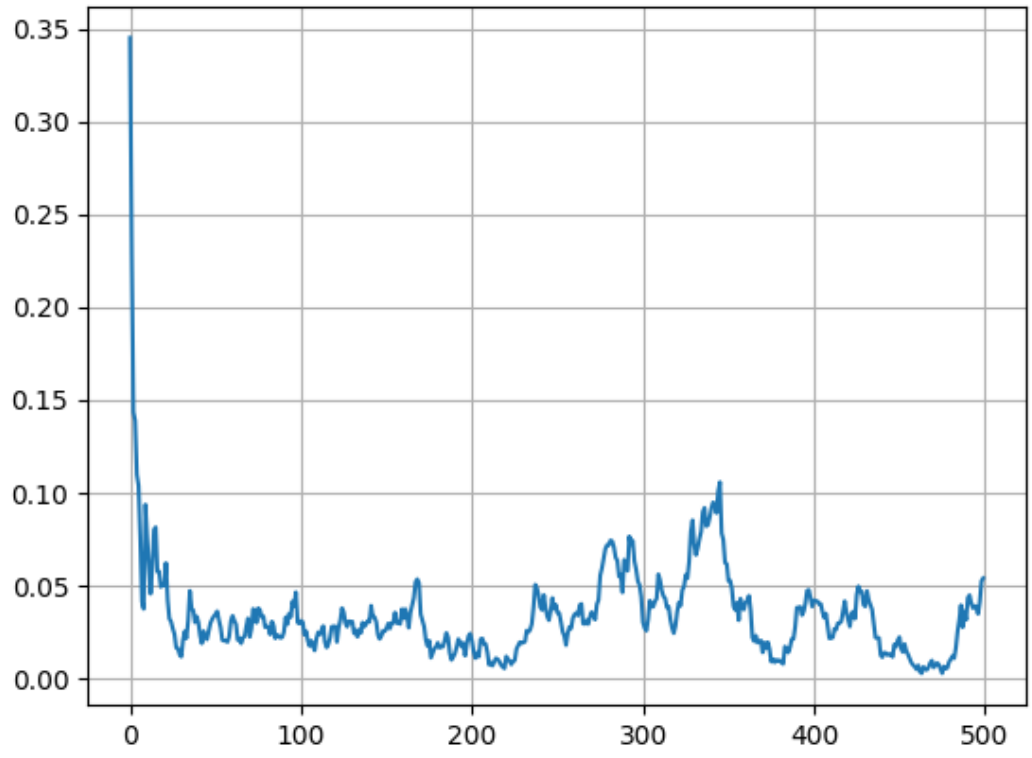
I arrived at the following function:

```
def update_sequence_mean_Mpoint_moving_avg(
    mu: np.ndarray,
    x: np.ndarray,
    n: int,
    m: int
) -> np.ndarray:
    '''Performs the mean sequence estimation update
    '''
    #Approximate moving average
    #Lowering n to a constant after x time gives more weight to newer points
    if n > m:
        n = m

    #Formula from book
    mu_new = mu + 1/(n+1)*(x-mu)

    return mu_new
```

This caps **n** at the value **m**, that is passed to the function. This gives each new mean more weight than it would get if it was divided my all **n**'s in the dataset and moves the mean faster. I think it approximates calculating the mean of the last **m** points. I didn't do the math to prove it, and the estimator might be biased but it gives good experimental results as shown below. The means trend in the right direction.

The squared error is not as good as for a fixed mean, but its not outrageous and I would always expect some error in this since it is a mean of many points over a moving mean. I used **m**=50 for the pictures shown below:


First dimension
Second dimension
Third dimension
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
100
200
300
400
500


0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
100
200
300
400
500