Preliminary results of optimal filtering problem with stochastic gradient descent

Ziyu Lu

June 2019

1 Gradient testing with finite difference approximation

We tested gradient computations with first order finite difference approximations. We measure the difference between the result given by gradient formulas and that given by finite difference approximation by the Frobenius norm of the gradient matrix. Despite of the noises in the simulations, we ensure that identical trajectories are generated for comparing different gradient computation methods by using numpy.seed. As the difference between the results of different computing methods are small (relative to the gradient itself) for small ΔK, we conclude that our implementation of gradient formulas is correct. If the finite difference approximation and exact gradient computation both work properly, we expect to see a linear growth in the difference as ΔK increases, and we are happy to report that this is exactly what is reflected by the graphs. Since it has been verified that the gradient computation using a forward recurrence relation and that using a backward recurrence relation produce same results, we only applied this finite difference approximation test to the backward computation.

Sample tests:

General settings:

- Mass of the object $m = 1$
- Spring constant $k = 0.5$
- Friction coefficient $\gamma = 0.1$
- Number of time steps in one path $N = 100$
- Step size $\Delta t = 0.05$
- Noise coefficient in SDE $\sigma = 0.1$
- Observation noise variance $Q = 0.1$
- Initial state $X_0 = [1.0, 0.0]^T$
- Observation matrix $C = [1.0, 0.0]$

When the Kalman gain K is set to $[1.0, 1.0]^T$, run

```
K = np.array([[1.0], [1.0]])
check_order(K, 0.0001, 1, 10)
```

to produce the following results:

- Gradient computed by formulas is $[0.0117978, 0.00778067]$
- Gradient computed by finite difference approximation with $\Delta K = 0.0001$ is $[0.0117994, 0.00778111]$
- Norm of difference/norm of exact gradient with $\Delta K = 0.0001$ is 0.0001
- The linear growth of the norm of difference as ΔK increases can be seen in Figure 1:

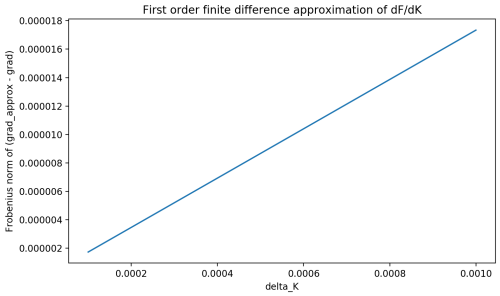

Figure 1

When the Kalman gain K is set to $[2.0, 2.0]^T$, run

```
K = np.array([[2.0], [2.0]])
check_order(K, 0.0001, 1, 10)
```

to produce the following results:

- Gradient computed by formulas is $[228529.64, 6511.34]$
- Gradient computed by finite difference approximation with $\Delta K = 0.0001$ is $[230574.95, 6513.00]$
- Norm of difference/norm of exact gradient with $\Delta K = 0.0001$ is 0.0089
- The linear growth of the norm of difference as ΔK increases can be seen in Figure 2:

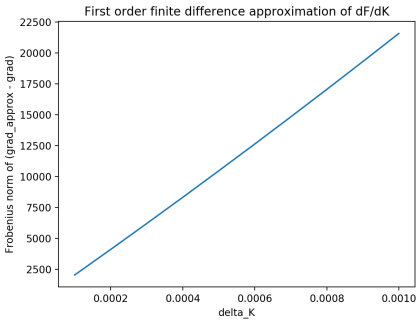

Figure 2

2 Stochastic gradient descent testing with small N

2.1 Testing with $N = 4$

2.1.1 The influence of the initialization of K

General settings:

- Mass of the object $m = 1$
- Spring constant $k = 0.5$
- Friction coefficient $\gamma = 0.1$
- Step size $\Delta t = 0.05$
- Noise coefficient in SDE $\sigma = 0.1$
- Observation noise variance $Q = 0.1$
- Initial state $X_0 = [1.0, 0.0]^T$
- Observation matrix $C = [1.0, 0.0]$

In the following discussion, we will use K_n, $n \geq 1$ to denote K in the nth iteration of the stochastic gradient descent algorithm, and K_0 to denote the initialization of K.

We observe that certain initializations can lead to very large gradients at the beginning of the training. In particular, the magnitude of gradient seems to be proportional to the magnitude of K_0: When $K_0 = [1.0, 1.0]^T$, the gradient is at order 0.01; when $K_0 = [5.0, 5.0]^T$, gradient at order 10^3 or 10^5 can appear during the first few iterations; when $K_0 = [10.0, 10.0]^T$, the gradient can start at order 10^5 or 10^6. Therefore, in order to avoid overflow and perform effective gradient descent, we have to adjust the learning rate according to the magnitude of gradient. For example, generally 0.1 can work for $K_0 = [1.0, 1.0]^T$, 0.001 can work for $K_0 = [5.0, 5.0]^T$, but for $K_0 = [10.0, 10.0]^T$, a

learning rate of 10^{-5} has to used.

Despite that it is very easy to have overflow, once the learning rate is properly chosen so that the overflow is avoided, we see that the algorithm is quite effective in reducing the loss. Some results are included below in the sample tests. In each graph, the results were averaged over 5 random seeds (1, 5, 10, 20, 27).

Sample tests

When $K_0 = [1.0, 1.0]^T$, run

```
K = np.array([[1.0], [1.0]])
K_, err_L = Stochastic_gradient_descent(K, 1000, 0.1, [1, 5,
10, 20, 27])
```

to produce the following results:

Averaging over 5 random seeds, K11 is 0.438, K12 is 0.345. The final loss is 0.002.

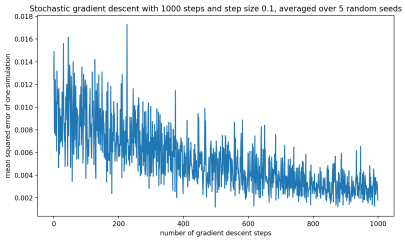

Figure 3

When $K_0 = [5.0, 5.0]^T$, run

```
K = np.array([[5.0], [5.0]])
K_, err_L = Stochastic_gradient_descent(K, 5000, 0.001, [1, 5,
10, 20, 27])
```

to produce the following results:

Averaging over 5 random seeds, K11 is 1.144, K12 is 4.194. The final loss is 0.101.

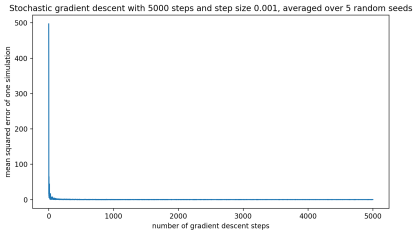

Figure 4

When $K_0 = [10.0, 10.0]^T$, run

```
K = np.array([[10.0], [10.0]])
```

`K_, err_L = Stochastic_gradient_descent(K, 5000, 1e-5, [1, 5, 10, 20, 27])`
to produce the following results:
Averaging over 5 random seeds, K11 is 2.149, K12 is 8.729. The final loss is 2.960.

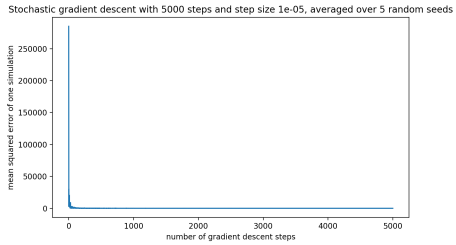

Figure 5

2.1.2 The influence of step size in one path

We observe that large step size will make the program more prone to overflow. Therefore, when we increase the step size, we have to decrease the learning rate accordingly.

2.1.3 The influence of observation noise variance

Similarly, we observe that large observation noise variance will make the program more prone to overflow as well. Also, when compared to the influence of step size, it seems that the effect of noise variance is harder to offset. That is, even though we can decrease the learning rate and increase training iterations, systems with larger observation noise variance tend to end up with larger loss.