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 in the graphs. Since it has been verified that the gradient computation using a forward recurrence relation and that using a backward recurrence relation produce identical 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 $[0.0117994, 0.00778111]$
- The ratio of the norm of difference/norm of exact gradient with $\Delta K = 0.0001$ is 0.0001226
- The linear growth of the norm of difference as ΔK increases can be seen in Figure 1:

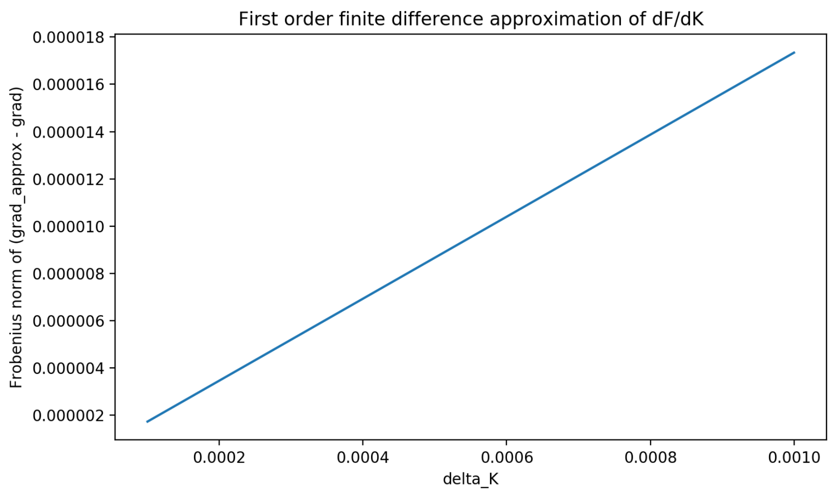

Figure 1

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 $[228529.6429463, 6511.3410305]$
- Gradient computed by finite difference approximation with $\Delta K = 0.0001$ is $[230574.95343332, 6513.00081057]$
- The ratio of the norm of difference/norm of exact gradient with $\Delta K = 0.0001$ is 0.0089462

- The linear growth of the norm of difference as ΔK increases can be seen in Figure 2:

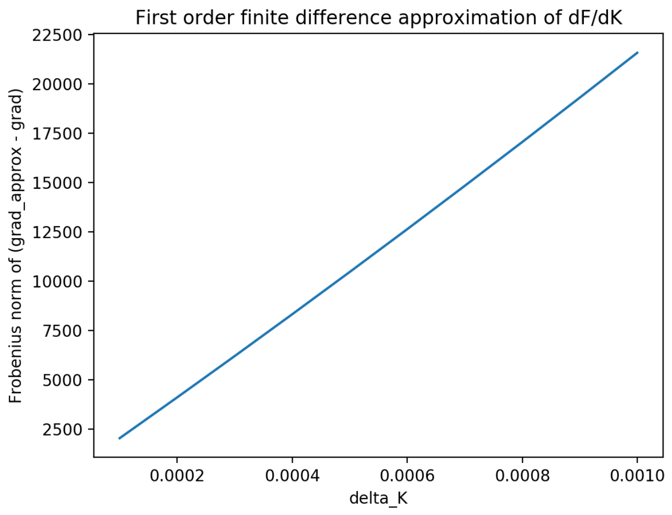

Figure 2

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 at the nth gradient descent step, and K_0 to denote the initialization of K.
We observe that K with different initializations can converge to different values. For example, after 20000 gradient descent steps with a learning rate of 0.1, when $K_0 = [1.0, 1.0]^T$, K converges to $[-0.02, 0.10]^T$ and the loss converges to 0.0002; meanwhile, when $K_0 = [5.0, 5.0]^T$, K converges to $[5.5, 1.5]^T$ and the loss converges to 0.02. In the case where $K_0 = [1.0, 1.0]^T$, it is clearly well-trained since the loss barely change in the last 100 gradient steps. In the case where $K_0 = [5.0, 5.0]^T$, we also experimented with 40000 gradient steps with the same learning rate to see whether it can be better trained. It turned out that K converged to $[5.50, 1.15]^T$, and there was only little change in the loss. So we may believe that they will never converge to the same point. One possible explanation of this phenomenon is that since the problem is multi-dimensional, it could be that there are several local optima. And in the above cases, the two K's have converged to different local optima, even though one is not as "optimal" as the other.

To be continued...