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:

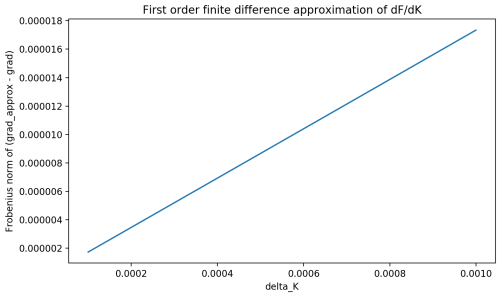

Figure 1

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:

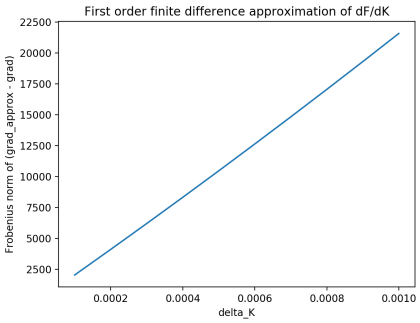

Figure 2

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 K with different initializations can converge to different values. For example, after 20000 iterations 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 iterations. In the case where $K_0 = [5.0, 5.0]^T$, we also experimented with 40000 iterations 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.

We also observe that certain initializations can lead to very large gradients. 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 starts at order 0.01; when $K_0 = [5.0, 5.0]^T$, the gradient starts at order 1; when $K_0 = [10.0, 10.0]^T$, the gradient starts at order 10^5. 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, 0.1 can work for $K_0 = [1.0, 1.0]^T$ and $[5.0, 5.0]^T$, but for $K_0 = [10.0, 10.0]^T$, a learning rate of 0.001 has to used.

Furthermore, we observe that a dramatic change in the magnitude of the gradient during gradient descent can affect the performance of the algorithm. For example, for $K_0 = [1.0, 1.0]^T$, the gradient starts at order 0.01, and after 1000 iterations with learning rate 0.1, the gradient is at order 0.001, and the loss is around 0.001. But for $K_0 = [10.0, 10.0]^T$, the gradient starts at order 10^5, and after 1000 iterations with learning rate 0.001, the gradient is at order 0.01, and the loss is 0.047. And even if we take 9000 more iterations, the loss only decreases by 0.001. We conjecture that this is because the learning rate has been chosen to be too small in order to dilute the large gradients during the first few iterations. After 1000 iterations, the change in the magnitude of gradient between each iteration is at order 10^{-5}, which is about 1% of the gradient, and thus leads to very slow improvement after 1000 iterations. One possible solution of this problem is to increase the learning rate to balance the shrinking of the gradient. If the case where $K_0 = [10.0, 10.0]^T$, if we raise the learning rate to 0.01 after 1000 iterations, then the loss decreases to 0.037 after 10000 iterations.

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

```
K = np.array([[1.0], [1.0]])
K_, err_L, grad_L = Stochastic_gradient_descent(K, 1000, 0.1,
1)
```

to produce the following results:

- After 1000 iterations, K11 becomes 0.119, K12 becomes 0.145. The final loss is 0.001.
- The change of the loss is shown in Figure 3:
- The change of the loss during the last 100 iterations is shown in Figure 4 (if the loss starts to increase, it is likely that we have missed the optimum):
- The change of the magnitude of the gradient is shown in Figure 5:

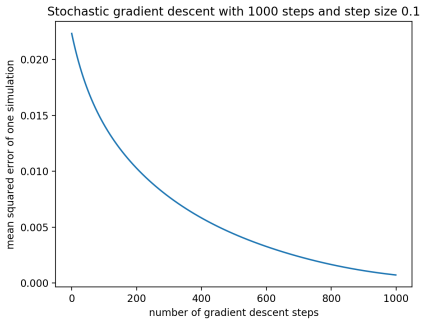

Figure 3

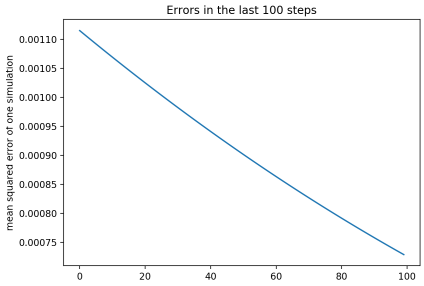

Figure 4

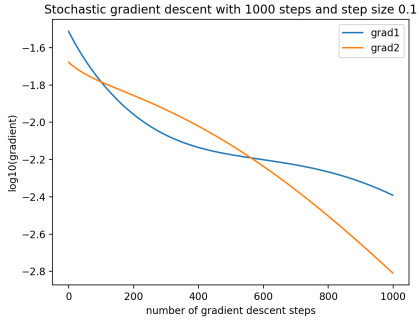

Figure 5

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

```
K = np.array([[10.0], [10.0]])
K_, err_L, grad_L = Stochastic_gradient_descent(K, 1000, 0.001,
1)
```

to produce the following results:

- After 1000 iterations, K11 becomes 5.318, K12 becomes 7.650. The final loss is 0.047.
- The change of the loss is shown in Figure 6:

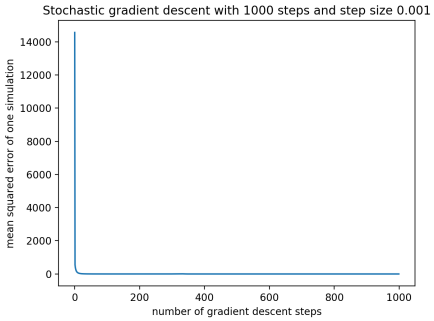

Figure 6

- The change of the loss during the last 100 iterations is shown in Figure 7:

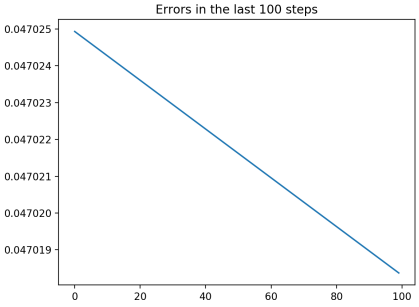

Figure 7

- The change of the magnitude of the gradient is shown in Figure 8:

2.1.2 The influence of step size in one path

General settings:

- Mass of the object $m = 1$
- Spring constant $k = 0.5$
- Friction coefficient $\gamma = 0.1$

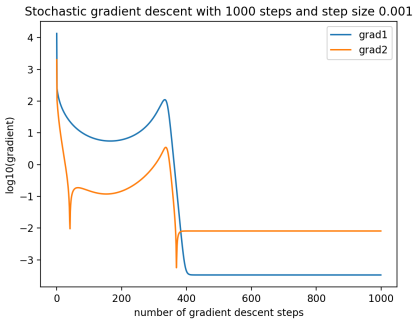

Figure 8

- Noise coefficient in SDE $\sigma = 0.1$
- Observation noise variance $Q = 0.1$
- Initial state $X_0 = [1.0, 0.0]^T$
- Observation matrix $C = [1.0, 0.0]$
- Initial Kalman gain $K_0 = [10.0, 10.0]^T$

We observe that the magnitude of the gradient grows proportionally to the step size in simulations. For example, when $\Delta t = 0.005$, the gradient starts at order 10^4, the loss after 1000 iterations with a learning rate of 0.0015 is 0.044; when $\Delta t = 0.05$, the gradient starts at order 10^5, the loss after 1000 iterations with a learning rate of 0.001 is 0.047; when $\Delta t = 0.5$, the gradient starts at order 10^6, and the loss after 1000 iterations with a learning rate of 0.00001 is 0.051. Despite the growth in the magnitude of the gradient, it seems that as long as we can choose a proper learning rate to dilute the large gradient, large step size in simulations won't leave any critical damage.

2.1.3 The influence of observation noise variance

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 $\sigma = 0.1$
- Initial state $X_0 = [1.0, 0.0]^T$
- Observation matrix $C = [1.0, 0.0]$

- Initial Kalman gain $K_0 = [10.0, 10.0]^T$

We observe that the magnitude of the gradient also grows proportionally to the variance of the observation noise. For example, when $Q = 0.005$, the gradient starts at order 10^2, the loss after 1000 iterations with a learning rate of 0.1 is less than 0.0005; when $Q = 0.05$, the gradient starts at order 10^4, the loss after 1000 iterations with a learning rate of 0.1 is 0.014; and when $Q = 0.5$, the gradient starts at order 10^6, and the loss after 1000 iterations with a learning rate of 10^{-5} is 1.716. When compared to the influence of step size, it seems that the effect of noise variance is harder to offset.