

Statistical Data Analysis Problem sheet 4

1. Exercise 1

- Here random variables are iid and our unknown parameter is θ .
- Then the maximum spacing estimator of θ is defined as a value that maximizes the logarithm of the geometric mean of sample spacings.
- As it is a uniform distribution, here the sample spacing of CDF is

$$D = \frac{x_i - a}{\theta - a}$$

- By Calculating Geometric Mean of the sample spacing and then taking logarithm we get

$$S_n(a, \theta) = \frac{1}{n+1} \sum_{i=1}^{n+1} \ln \frac{x_i - a}{\theta - a}$$

- As we know the left limit $a = 0$, by differentiating the statistics with respect to θ we get the estimator for theta using MSE

$$\theta_{MSE} = \frac{nx_n - x_1}{n - 1}$$

```
1 # Importing libraries
2 import numpy as np
```

```
1 # getting the data
2 dfx = np.loadtxt("/content/drive/MyDrive/1-DS/ps4/sampleset_1_problemsheet4_ex1.txt")
3 dfy = np.loadtxt("/content/drive/MyDrive/1-DS/ps4/sampleset_2_problemsheet4_ex1.txt")
4 dfz = np.loadtxt("/content/drive/MyDrive/1-DS/ps4/sampleset_3_problemsheet4_ex1.txt")
```

```
1 #defining theta estimator for uniform distribution
2 theta = lambda x,samplesize: (samplesize*x[-1] - x[0])/(samplesize-1)
```

```
1 mseThetax = np.array(theta(np.sort(dfx),dfx.size))
2 mseThetay = np.array(theta(np.sort(dfy),dfy.size))
3 mseThetaz = np.array(theta(np.sort(dfz),dfz.size))
4 print("Theta using MSE of \n sampleset 1 : {} \n sampleset 2 : {} \n sampleset 3 : {}".format(mseThetax, mseThetay, mseThetaz))
```

```
Theta using MSE of
sampleset 1 : 4.01135
sampleset 2 : 3.9147483673469385
sampleset 3 : 4.029571428571429
```

There was no lecture or slides in the moodle about this estimator, so we have used standard uniform distribution PDF and CDF for the exercise 1.

2. Exercise 2

- Here the evolution equation is

$$z_n = 0.99z_{n-1} + \xi_{n-1}$$

- The relative function is

$$y_n = z_n + \eta_n$$

- We have to implement kalman filter and ensemble kalman filter and then compare all estimations by mean squared error or **MSE**.

```
1 # defining standard kalman filter
2 def kalman(data):
3     me_var = 0.3
4     de_var = 0.5
5     mi_var = 0
6     n = data.shape[0]
7     emean = np.zeros(n+1)
8     evar = np.zeros(n+1)
9     emean[0] = 0
10    evar[0] = mi_var
11    for i in range(1, n+1):
12        fmean = 0.99*emean[i-1]
13        fvar = 0.99**2 * evar[i-1] + me_var
14        #calculating kalman gain
15        kalman_gain = fvar/(fvar+de_var)
```

```

16 # calculating analysis mean and variance
17 emean[i] = fmean + kalman_gain*(data[i-1] - fmean)
18 evar[i] = (1 - kalman_gain)*fvar
19 return emean, evar

```

```

1 # loading the data and reference signal values
2 ref = np.loadtxt("/content/drive/MyDrive/1-DS/ps4/reference_signal.txt")
3 mdata = np.loadtxt("/content/drive/MyDrive/1-DS/ps4/data.txt")

```

```

1 #applying kalman filter on the data
2 kfmean, kfvar = kalman(mdata)

```

```

1 print(kfmean)
2 print(kfvar)

```

```

[ 0.          0.1496025  0.48710132 ... -0.58526321 -0.77782534
 -0.47559205]
[0.          0.1875      0.24587524 ... 0.26383579 0.26383579 0.26383579]

```

```

1 # calculating mse for standard kalman filter
2 kfmse = np.mean((kfmean - ref)**2)
3 print(kfmse)

```

```
0.11351651539008914
```

```

1 # defining ensemble kalman filter
2 def enskalman(data, ens):
3     np.random.seed(404)
4     me_var = 0.3
5     de_var = 0.5
6     mi_var = 0
7     n = data.shape[0]
8     ens_ini = np.random.normal(0, mi_var**0.5, ens)
9     ensem = np.zeros((n + 1, ens))
10    ensem[0] = ens_ini
11    ens_mean = np.zeros(n+1)
12    ens_covar = np.zeros(n+1)
13
14    for i in range(1, n+1):
15        m_err = np.random.normal(0, me_var**0.5, ens)
16        fensem = 0.99*ensem[i-1] + m_err
17        f_mean = np.mean(fensem)
18        matx = fensem - f_mean
19        f_covar = (matx.dot(np.transpose(matx)))/(ens - 1)
20        kalman_gain = f_covar / (f_covar + de_var)
21        ensem[i] = fensem + kalman_gain*(perturbed_data(data[i-1],de_var,ens)-fensem)
22        ens_mean[i] = np.mean(ensem[i])
23    return ens_mean
24
25

```

```

1 #calculating perturbed data for correcting analysis variance
2
3 def perturbed_data(data, de_var, ens):
4     m_err_hat = np.random.normal(0,de_var**0.5,ens)
5     m_err_hat -= np.mean(m_err_hat)
6     return data - m_err_hat

```

```

1 # applying ensemble kalman filter
2 ens = [5,10,25,50]
3 for m in ens:
4     ens_mean = enskalman(mdata, m)
5     ens_kf_mse = np.mean((ens_mean - ref)**2)
6     print(f"For ensemble size {m} , MSE is : {ens_kf_mse}")
7

```

```

For ensemble size 5 , MSE is : 0.15886513895276547
For ensemble size 10 , MSE is : 0.1266817408205558
For ensemble size 25 , MSE is : 0.11970358943085674
For ensemble size 50 , MSE is : 0.11632277074743622

```

Here we can see that the standard kalman filter is better with respect to MSE values. But with ensemble kalman filter, upon increasing the ensemble, the error is decreasing. The ensemble kalman filter is computationally less expensive for higher dimensions.