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 Claculating Geomateric 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 repect to θ we get the estimator for theta using MSE

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

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

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

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

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

```
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***.

```
# defining standard kalman filter
def kalman(data):
  me_var = 0.3
  de_var = 0.5
  mi_var = 0
  n = data.shape[0]
  emean = np.zeros(n+1)
  evar = np.zeros(n+1)
  emean[0] = 0
  evar[0] = mi_var
  for i in range(1, n+1):
    fmean = 0.99*emean[i-1]
    fvar = 0.99**2 * evar[i-1] + me_var
    #calculating kalman gain
    kalman_gain = fvar/(fvar+de_var)
    # calculating analysis mean and variance
    emean[i] = fmean + kalman_gain*(data[i-1] - fmean)
    evar[i] = (1 - kalman_gain)*fvar
  return emean, evar
```

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

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

```
print(kfmean)
print(kfvar)
```

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

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

```
0.11351651539008914
```

```
# defining ensemble kalman filter
def enskalman(data, ens):
  np.random.seed(404)
  me_var = 0.3
  de_var = 0.5
  mi_var = 0
  n = data.shape[0]
  ens_ini = np.random.normal(0, mi_var**0.5, ens)
  ensem = np.zeros((n + 1, ens))
  ensem[0] = ens_ini
  ens_mean = np.zeros(n+1)
  ens_covar = np.zeros(n+1)

  for i in range(1, n+1):
    m_err = np.random.normal(0, me_var**0.5, ens)
    fensem = 0.99*ensem[i-1] + m_err
    f_mean = np.mean(fensem)
    matx = fensem - f_mean
    f_covar = (matx.dot(np.transpose(matx)))/(ens -1)
    kalman_gain = f_covar / (f_covar + de_var)
    ensem[i] = fensem + kalman_gain*(perturbed_data(data[i-1],de_var,ens)-fensem)
    ens_mean[i] = np.mean(ensem[i])
  return ens_mean
```

```
#calculating perturbed data for correcting analysis variance

def perturbed_data(data, de_var, ens):
    m_err_hat = np.random.normal(0,de_var**0.5,ens)
    m_err_hat -= np.mean(m_err_hat)
    return data - m_err_hat
```

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

```
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.