# APPM 5510 HW 6

Zane Jakobs

## 1(a)

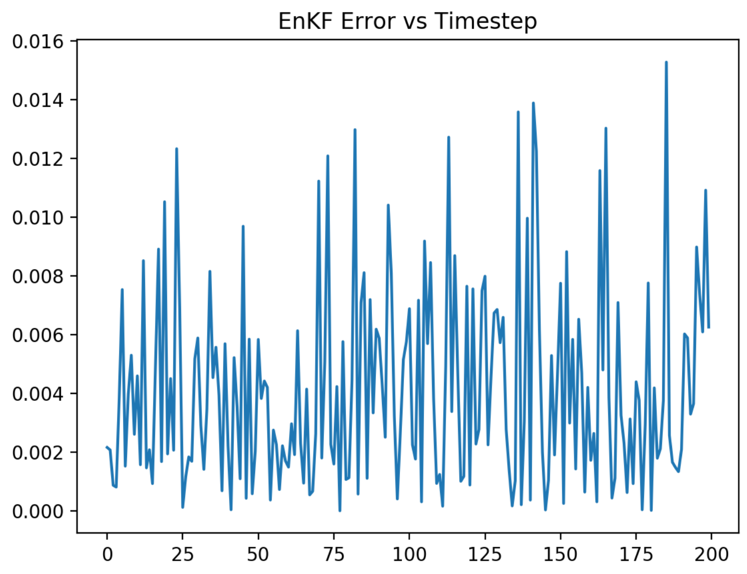
Since the samples are i.i.d. and have finite variance, the sample mean follows a Student t-distribution with expected value $\mu$, and the expected value of a sample from the posterior is $\mu$, so the expected error is zero (i.e. the estimator is unbiased).

## 1(b)

As $N \to \infty$, the variance of the sample mean, $\frac{\sigma^2}{N}$, goes to zero, but, independently of the draws used to form the estimate, the true value has a variance of $\sigma^2$, so since the error is the difference between the sample mean and the true value, which are independent, its variance is the sum of the variances of the sample mean and true value, $\sigma^2(1 + \frac{1}{N})$. Thus, the variance does not go to zero as $N \to \infty$.

## 2(a)

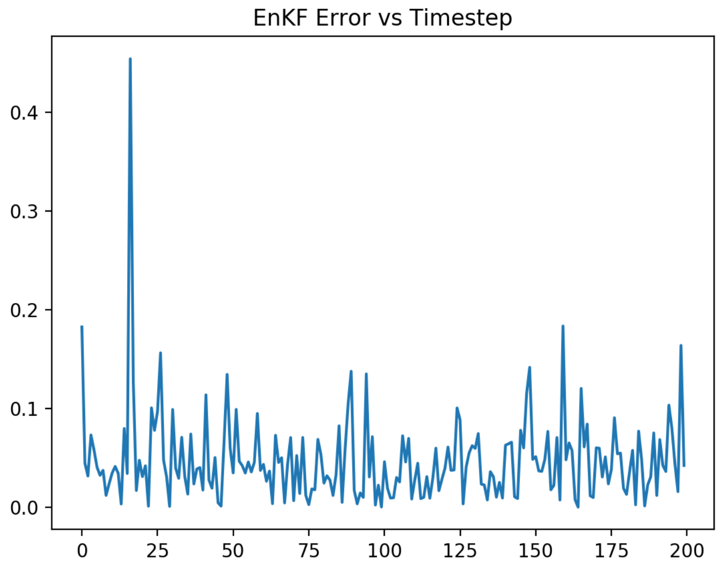
NOTE: all code is attached to the end of this assignment. We first get this plot:



and we have non-Gaussianity on 12% of timesteps, and error greater than observation error on 14%.

## 2(b)

Running with the observation error variance set to 0.01, we get this plot:



with non-Gaussianity 60% of the time, and error greater than observation error 8.5% of the time.

## 3(a)

Letting $\mathbf{1}$ be a vector of ones and $\boldsymbol{\varepsilon}$ be the vector of perturbations, condition (i) implies that $\mathbf{1}^T\boldsymbol{\varepsilon} = 0$. Now, given condition (i), the expected values of the perturbation and state ensembles are both $\mu\mathbf{1}$, so the cross-covariance matrix is

$$\begin{aligned}\mathbb{E}[(\boldsymbol{y}-\mu\mathbf{1})(\boldsymbol{y}+\boldsymbol{\varepsilon}-\mu\mathbf{1})^T] &= \mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T + \boldsymbol{y}\boldsymbol{\varepsilon}^T - \mu\boldsymbol{y}\mathbf{1}^T - \mu\mathbf{1}\boldsymbol{y}^T - \mu\mathbf{1}\boldsymbol{\varepsilon}^T + \mu^2\mathbf{1}\mathbf{1}^T] \\ &= \boldsymbol{y}\boldsymbol{y}^T - \mu\boldsymbol{y}\mathbf{1}^T - \mu\mathbf{1}\boldsymbol{y}^T + \mu^2\mathbf{1}\mathbf{1}^T + \mathbb{E}[\boldsymbol{y}\boldsymbol{\varepsilon}^T - \mu\mathbf{1}\boldsymbol{\varepsilon}^T] \\ &= \boldsymbol{y}\boldsymbol{y}^T - \mu\boldsymbol{y}\mathbf{1}^T - \mu\mathbf{1}\boldsymbol{y}^T + \mu^2\mathbf{1}\mathbf{1}^T + (\boldsymbol{y}-\mu\mathbf{1})\boldsymbol{\varepsilon}^T.\end{aligned}$$

In order for the cross-covariance to be zero, we need this matrix to equal the identity matrix $\boldsymbol{I}$–that is, we need

$$(\boldsymbol{y}-\mu\mathbf{1})\boldsymbol{\varepsilon}^T = \boldsymbol{I} - \boldsymbol{y}\boldsymbol{y}^T + \mu\boldsymbol{y}\mathbf{1}^T + \mu\mathbf{1}\boldsymbol{y}^T - \mu^2\mathbf{1}\mathbf{1}^T.$$

To simultaneously satisfy conditions (i) and (ii), we need for the following two equations to hold:

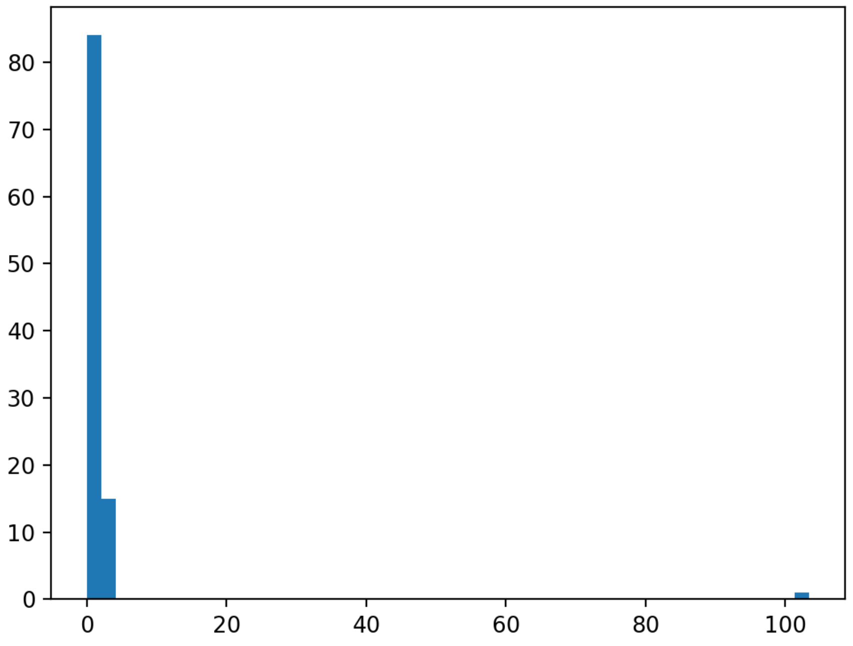
$$\begin{aligned}\mathbf{1}^T\boldsymbol{\varepsilon} &= 0 \\ (\boldsymbol{y}-\mu\mathbf{1})\boldsymbol{\varepsilon}^T &= \boldsymbol{I} - \boldsymbol{y}\boldsymbol{y}^T + \mu\boldsymbol{y}\mathbf{1}^T + \mu\mathbf{1}\boldsymbol{y}^T - \mu^2\mathbf{1}\mathbf{1}^T\end{aligned}$$

## 3(b)

Solutions to this system are not unique–that is, there are multiple combinations of actual values of perturbations that each independently satisfy the constraints.

## 4

All of the eigenvalues fit on one histogram, shown here:



The largest eigenvalue is well-approximated, but the smaller ones are not quite as well-approximated (they skew upwards).

```
import numpy as np
from scipy.stats import anderson
import matplotlib.pyplot as plt


def advance_and_observe(forecast, x0, obs_var):
    obs_error = np.random.normal(0.0, np.sqrt(obs_var))
    return forecast(x0) + obs_error


def forward_model(step_model, x0, n):
    obs = [x0]
    for i in range(n):
        obs.append(step_model(obs[-1]))

    return np.array(obs)

def draw_ensemble(mean, var, n):
    return np.random.normal(mean, np.sqrt(var), n)

def create_ensemble(obs, ensemble_mean, ensemble_var, n):
    return obs + draw_ensemble(ensemble_mean, ensemble_var, n)

def EnKF_Step(forecast, ensemble, obs, obs_var, n, inflation=1.0, H=None, adthreshold=0.05):

    for i in range(n):
        #forecast ensemble
        ensemble[i] = forecast(ensemble[i])
    #check if prior is Gaussian with Anderson-Darling
    stat, crit_vals, sig_lvls = anderson(ensemble, 'norm')
    #5% is the third critical value
    if stat >= crit_vals[2]:
        print("Ensemble fails Anderson-Darling test (is rejected as Normal) with test statistic
        failed=True
    else:
        failed=False
    if not inflation == 1.0:
        ensemble *= np.sqrt(inflation)
    #covariance
    amean = np.mean(ensemble) #analysis mean

    C = np.sum((ensemble - amean) ** 2) / (n-1)


    #update ensemble
    if H is None:
        K = C / (C + obs_var)

        ensemble += K * (obs + draw_ensemble(0.0, obs_var, n) - ensemble)

        C -= K * C

        amean = np.mean(ensemble) #analysis mean
        aspread = np.sqrt(C)


    return amean, aspread, ensemble, failed


def EnKF_Run(forecast, init_ensemble, obs, obs_var, timesteps, n,inflation=1.0, H=None, adthres
    ameans = []
    apreads = []
    nfailed = 0
    for t in range(timesteps):
        am, asp, init_ensemble, failed = EnKF_Step(forecast, init_ensemble, obs[t], obs_var, n,
        if failed:
            nfailed += 1
        ameans.append(am)
        apreads.append(asp)

    return ameans, apreads, nfailed

if __name__ == '__main__':

    def advance(x):
        xp = 3.95 * x * (1-x)
        if xp < 0.0:
            xp = 0.0
        elif xp > 1.0:
            xp = 1.0

        return xp


    #generate obs
```

```
n = 200
x0 = 0.25
x = forward_model(advance, x0, n-1)

obs_var = 0.01
ensemble_size = 50
xerr = draw_ensemble(0.0, obs_var, n)

obs = x + xerr

iensemble = draw_ensemble(1.0/3.0, 0.01, ensemble_size)

emeans, evars, nfailed = EnKF_Run(advance, iensemble, obs, obs_var, n, ensemble_size)

print("non-gaussian on fraction of timesteps" , (float(nfailed)/200.0), '\n')

err = np.abs(obs - emeans)

bigerr=0

for i in range(n):
    if err[i] > np.sqrt(obs_var):
        bigerr += 1

print("Error greater than observation error on fraction of timesteps", float(bigerr)/200.0,


plt.plot(np.array(list(range(n))), err)
plt.title('EnKF Error vs Timestep')


#random matrix
A = np.random.rand(100,100)

Q, R = np.linalg.qr(A)

D = np.identity(100)
D[0,0] = np.sqrt(100)

Broot = Q @ D @ Q.T
```

```
x = np.empty((100,110))

for i in range(110):
    x[:,i] = Broot @ np.random.standard_normal(100)

C = np.cov(x)

lbda, v = np.linalg.eig(C)

lbda = lbda[np.argsort(lbda)[-100:]] #10 largest


plt.figure()
plt.hist(lbda, bins=50)


plt.show()
```