# hw6p3

June 2, 2022

## 0.1 Homework 6, Problem 3 on Wiener filter

ECE C143A/C243A, Spring Quarter 2022, Prof. J.C. Kao, TAs T. Monsoor and W. Yu

Total: 15 points. In this notebook, you will implement an optimal linear estimator decoder.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.special
import scipy.io as sio
import math
import nsp as nsp
# Load matplotlib images inline
%matplotlib inline
# Reloading any code written in external .py files.
%load_ext autoreload
%autoreload 2
data = sio.loadmat('JR_2015-12-04_truncated2.mat') # load the .mat file.
R = data['R'][0,:]
```

### 0.1.1 (a) (4 points) Preparing training data for a Wiener filter. [Code solution provided; please understand it and answer the question.]

Now that we've built an optimal linear estimator, we'll extend it to incorporate history. We will train a Wiener filter, again using just the first 400 trials of the R-struct. We will use 100 ms worth of history in our Wiener filter. As each bin is 25 ms long, this corresponds to using data up to and including 4 bins in the past, i.e. $P = 3$. Make the large matrix $Y_W$ discussed in Lecture. To be clear, we are talking about the matrix:

$$\begin{bmatrix} y_{P+1} & y_{P+2} & y_{P+3} & \cdots & y_K \\ y_P & y_{P+1} & y_{P+2} & \cdots & y_{K-1} \\ \cdots & & & & \\ y_1 & y_2 & y_3 & \cdots & y_{K-P} \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}$$

Calculate this matrix and output its dimension.

[3]:
```
#=====================================================#
# YOUR CODE HERE:
#   Create the Y_W matrix and print its dimensions.
#=====================================================#
```

```
dt = 25
train_num = 400
Y =scipy.sparse.hstack(R[0:train_num]['spikeRaster'])
X =scipy.sparse.hstack(R[0:train_num]['cursorPos'])
X= scipy.sparse.csc_matrix(X)
Y_bin = nsp.bin(Y, dt,'sum')

X_bin = nsp.bin(X, dt,'first')
X_bin = np.diff(X_bin[0:2,:])/dt*1000

X_small_bin = np.matrix(X_bin[:,3:],dtype=float)
last_index = np.size(X_bin,1)
Y1= np.matrix(Y_bin[:,3:],dtype= float)
Y2 =np.matrix(Y_bin[:,2:-1],dtype= float)
Y3 =np.matrix(Y_bin[:,1:-2],dtype= float)
Y4 =np.matrix(Y_bin[:,0:-3],dtype= float)
Y_large = np.vstack((Y1,Y2,Y3,Y4))

Y_w_bin = np.vstack((Y_large,np.ones(np.size(Y_large,1))))

print("The dimension for the large matrix",np.size(Y_w_bin,0),np.
 ↪size(Y_w_bin,1))
#=====================================================#
# END YOUR CODE
#=====================================================#
```

```
The dimension for the large matrix 385 16462
```

[9]:
```
R.shape
```

[9]:
```
(506,)
```

Question: What are the dimensions of $Y_W$?

Answer: (385, 16462)

### 0.1.2 (b) (7 points) Fit a Wiener filter and decode activity.

Fit the Wiener filter using the data matrix you generated in part(a). Plot, as you did in question 2(g), the decoded trajectories for each test trial. (Hint: to decode the first 3 velocities of trial i at times 25 ms, 50 ms, and 75 ms, since 100 ms have not yet occurred in the trial, you will have to use the last bins of spiking activity in trial $i-1$.)

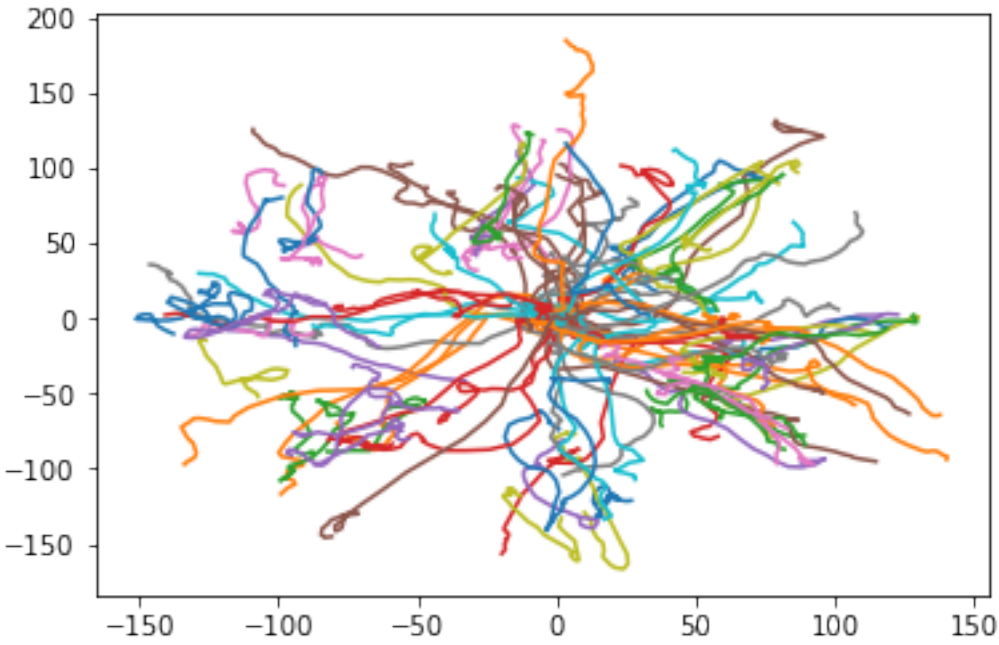
[42]:
```
#=====================================================#
# YOUR CODE HERE:
#   Fit the Wiener filter and decode the trajectories for
#   each test trial.
#=====================================================#
L=np.matmul(X_small_bin,scipy.linalg.pinv( Y_w_bin))
```

```
L.shape
mean_distances=[]
for i in range(train_num,R.shape[0]):
    Y_test = R[i]['spikeRaster']
    Y_bin_test=nsp.bin(Y_test, dt,'sum').astype('float64')
    Y_test_prev=R[i-1]['spikeRaster']
    Y_bin_test_prev=nsp.bin(Y_test_prev, dt,'sum').astype('float64')
    Y_bin_test=np.hstack((Y_bin_test_prev[:,-3:],Y_bin_test))
    Y1= np.matrix(Y_bin_test[:,3:],dtype= float)
    Y2 =np.matrix(Y_bin_test[:,2:-1],dtype= float)
    Y3 =np.matrix(Y_bin_test[:,1:-2],dtype= float)
    Y4 =np.matrix(Y_bin_test[:,0:-3],dtype= float)
    Y_bin_test = np.vstack((Y1,Y2,Y3,Y4))
    Y_bin_test=np.vstack((Y_bin_test, np.ones(np.size(Y_bin_test,1))))\

    predV=np.matmul(L,Y_bin_test)
    #print(R[train_num+i]['cursorPos'][0:2,0])
    predLoc=np.asarray(np.cumsum(predV,axis=1).T*25/1000+R[i]['cursorPos'][0:
 ↪2,0]).T
    #print(predLoc.shape)
    #print(predLoc)
    plt.plot(predLoc[0,:],predLoc[1,:])


    t=np.arange(0,predLoc.shape[1]*dt,dt)+dt-1

    distance=(R[i]['cursorPos'][:2,t]-predLoc).astype(float)
    distance=np.sum(distance**2,axis=0)
    mean_distances.append(np.mean(distance))
#=========================================================#
# END YOUR CODE
#=========================================================#
```




### 0.1.3 (c) (4 points) Calculate the mean-square error for the Wiener filter.

What is the average mean-square error in position per trial? Did the Wiener filter perform better or worse than the optimal linear estimator?

[43]:
```
#=====================================================#
# YOUR CODE HERE:
#   Calculate the mean-squared error between the decoded
#   hand position and the true hand position.  Average
#   the squared errors across time; then average the squared
#   errors across trials.
#=====================================================#
print(f"MSE={np.mean(mean_distances)}")
#=====================================================#
# END YOUR CODE
#=====================================================#
```

```
MSE=3343.9026185629855
```

Question: Does the WF do better or worse than the OLE?

Answer: Yes it does