Exam 3

Ryan Branagan

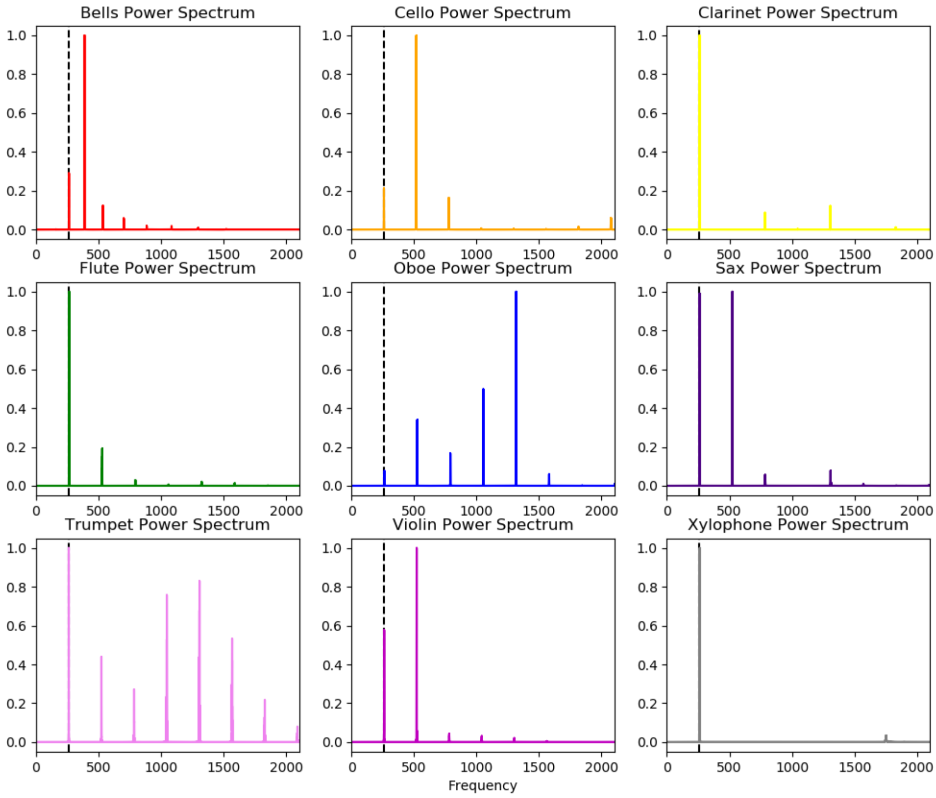
April 23, 2019

Problem 1

```
#Ryan Branagan
#Collaborators: Jack Featherstone, John Paul Habeeb, Mason Lovejoy-Johnson, and Grant She
#Branagan_ex3_p1.py
#4/17/19

import numpy as np
import pylab as p
import wavio as wav
#%%
Bells = wav.read("C:\\Users\\ryan-\\Documents\\251\\Exam 3\\Data\\bells.wav")
BellDat = Bells.data.flatten()
BellDat = BellDat.astype(float)
BellRate = Bells.rate

Cello = wav.read("C:\\Users\\ryan-\\Documents\\251\\Exam 3\\Data\\cello.wav")
CelloDat = Cello.data.flatten()
CelloDat = CelloDat.astype(float)
CelloRate = Cello.rate

Clar = wav.read("C:\\Users\\ryan-\\Documents\\251\\Exam 3\\Data\\clarinet.wav")
ClarDat = Clar.data.flatten()
ClarDat = ClarDat.astype(float)
ClarRate = Clar.rate

Flute = wav.read("C:\\Users\\ryan-\\Documents\\251\\Exam 3\\Data\\flute.wav")
FluteDat = Flute.data.flatten()
FluteDat = FluteDat.astype(float)
FluteRate = Flute.rate

Oboe = wav.read("C:\\Users\\ryan-\\Documents\\251\\Exam 3\\Data\\oboe.wav")
```

```
OboeDat = Oboe.data.flatten()
OboeDat = OboeDat.astype(float)
OboeRate = Oboe.rate

Sax = wav.read("C:\\Users\\ryan-\\Documents\\251\\Exam 3\\Data\\sax-alto.wav")
SaxDat = Sax.data.flatten()
SaxDat = SaxDat.astype(float)
SaxRate = Sax.rate

Trumpet = wav.read("C:\\Users\\ryan-\\Documents\\251\\Exam 3\\Data\\trumpet.wav")
TrumpetDat = Trumpet.data.flatten()
TrumpetDat = TrumpetDat.astype(float)
TrumpetRate = Trumpet.rate

Violin = wav.read("C:\\Users\\ryan-\\Documents\\251\\Exam 3\\Data\\violin.wav")
ViolinDat = Violin.data.flatten()
ViolinDat = ViolinDat.astype(float)
ViolinRate = Violin.rate

Xylo = wav.read("C:\\Users\\ryan-\\Documents\\251\\Exam 3\\Data\\xylophone.wav")
XyloDat = Xylo.data.flatten()
XyloDat = XyloDat.astype(float)
XyloRate = Xylo.rate
#%%
def Transform(Data,Rate):
    dt = (1/Rate)
    A = np.abs(np.fft.fft(Data))
    A = A/max(A)
    B = ((np.abs(A))**2)
    C = np.abs(np.fft.fftfreq(len(Data),dt))
    return A,B,C
#%%
BellSig,BellPower,BellFreq = Transform(BellDat,BellRate)
CelloSig,CelloPower,CelloFreq = Transform(CelloDat,CelloRate)
ClarSig,ClarPower,ClarFreq = Transform(ClarDat,ClarRate)
FluteSig,FlutePower,FluteFreq = Transform(FluteDat,FluteRate)
OboeSig,OboePower,OboeFreq = Transform(OboeDat,OboeRate)
SaxSig,SaxPower,SaxFreq = Transform(SaxDat,SaxRate)
TrumpetSig,TrumpetPower,TrumpetFreq = Transform(TrumpetDat,TrumpetRate)
ViolinSig,ViolinPower,ViolinFreq = Transform(ViolinDat,ViolinRate)
XyloSig,XyloPower,XyloFreq = Transform(XyloDat,XyloRate)
#%%
fig1,((Bellax,Celloax,Clarax),(Fluteax,Oboeax,Saxax),(Trumpetax,Violinax,Xyloax)) = p.sul
```

```
Bellax.axvline(261.6,color='k',linestyle='--')
Bellax.plot(BellFreq,BellPower,'r')
Bellax.set_title("Bells Power Spectrum")
Bellax.set_xlim(0,((8*261.6)+10))

Celloax.axvline(261.6,color='k',linestyle='--')
Celloax.plot(CelloFreq,CelloPower,'orange')
Celloax.set_title("Cello Power Spectrum")
Celloax.set_xlim(0,((8*261.6)+10))

Clarax.axvline(261.6,color='k',linestyle='--')
Clarax.plot(ClarFreq,ClarPower,'yellow')
Clarax.set_title("Clarinet Power Spectrum")
Clarax.set_xlim(0,((8*261.6)+10))

Fluteax.axvline(261.6,color='k',linestyle='--')
Fluteax.plot(FluteFreq,FlutePower,'g')
Fluteax.set_title("Flute Power Spectrum")
Fluteax.set_xlim(0,((8*261.6)+10))

Oboeax.axvline(261.6,color='k',linestyle='--')
Oboeax.plot(OboeFreq,OboePower,'b')
Oboeax.set_title("Oboe Power Spectrum")
Oboeax.set_xlim(0,((8*261.6)+10))

Saxax.axvline(261.6,color='k',linestyle='--')
Saxax.plot(SaxFreq,SaxPower,'indigo')
Saxax.set_title("Sax Power Spectrum")
Saxax.set_xlim(0,((8*261.6)+10))

Trumpetax.axvline(261.6,color='k',linestyle='--')
Trumpetax.plot(TrumpetFreq,TrumpetPower,'violet')
Trumpetax.set_title("Trumpet Power Spectrum")
Trumpetax.set_xlim(0,((8*261.6)+10))

Violinax.axvline(261.6,color='k',linestyle='--')
Violinax.plot(ViolinFreq,ViolinPower,'m')
Violinax.set_title("Violin Power Spectrum")
Violinax.set_xlabel("Frequency")
Violinax.set_xlim(0,((8*261.6)+10))

Xyloax.axvline(261.6,color='k',linestyle='--')
Xyloax.plot(XyloFreq,XyloPower,'gray')
Xyloax.set_title("Xylophone Power Spectrum")
```




```
Xyloax.set_xlim(0,((8*261.6)+10))
```

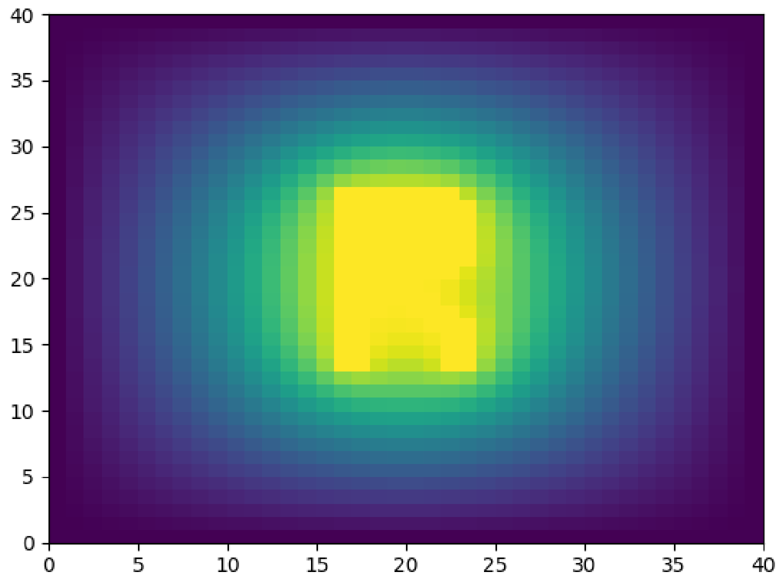
For this problem I imported the data, normalized it, took the power spectrum, got a frequency axis, then plotted it.

Problem 2

```
#Ryan Branagan
#Collaborators: Jack Featherstone, John Paul Habeeb, Mason Lovejoy-Johnson, and Grant She
#Branagan_ex3_p2.py
#4/17/19

import numpy as np
import pylab as p
#%%
```

```
#Make an R
Coords = np.array([[7,4],[8,4],[9,4],[10,4],[11,4],[12,4],[13,4],[7,5],[8,5],[12,5],[13,5
Coords = Coords + 9

Alf = 1.
dx = 1.
dy = 1.
dt = 0.25
X = 40
Y = 40
t0 = 0
tf = 300
T = int(tf/dt)+2
Tol = 10**-6
Temp = np.zeros([T+1,X,Y])

def Bound(Arr,t):
    for C in Coords:
        x = C[0]
        y = C[1]
        Arr[t,y,x] = 1.
    Arr[t,0] = 0
    Arr[t,-1] = 0
    Arr[t,:,0] = 0
    Arr[t,:,-1] = 0
    return Arr

Temp = Bound(Temp,0)
Temp = Bound(Temp,1)
for t in range(1,T):
    for x in range(1,X-1):
        for y in range(1,Y-1):
            Temp[t+1,y,x] = ((1.*Alf*dt)*((((Temp[t,y,x+1])-(2.*Temp[t,y,x])+(Temp[t,y,x-
    Temp = Bound(Temp,t+1)
    if (np.mean(np.abs(Temp[t+1]))-np.mean(np.abs(Temp[t]))) < Tol:
        print(t)
        break

p.pcolor(Temp[t,::-1,:])
```

I used Minecraft to make my R then converted the coordinates to Python. This actually made my boundary conditions very easy. I really do not like this problem because I wanted to use a centered stencil for the first derivative because it is more accurate and makes more sense to me but he code just breaks if you try to use it. Additionally the



code just breaks if you have a dt or 0.5 or higher. Even though there should be many different ways to do this question, most of them do not work. Discretized Equation: $T[t+1,y,x] = \alpha * dt * (\frac{T[t,y,x+1-2T[t,y,x]+T[t,y,x-1]}{dx^2} + \frac{T[t,y+1,x]-2T[t,y,x]+T[t,y-1,x]}{dy^2}) + T[t,y,x]$

Problem 3

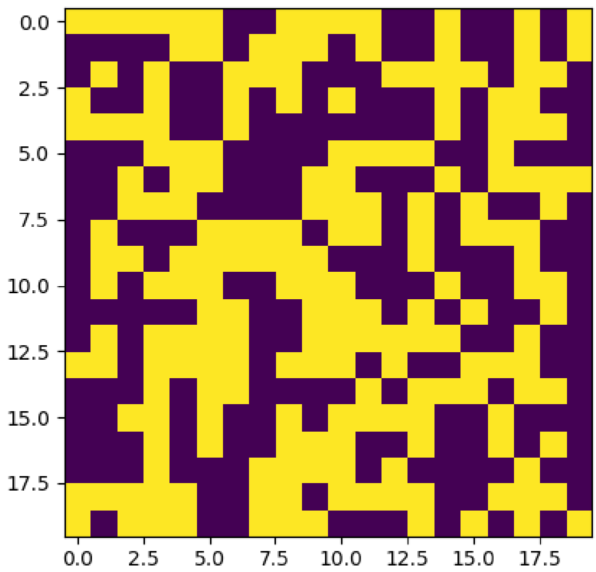
```
#Ryan Branagan
#Collaborators: Jack Featherstone, John Paul Habeeb, Mason Lovejoy-Johnson, and Grant She
#Branagan_ex3_p3.py
#4/17/19

import numpy as np
import pylab as p
#%%
#Dogs = 1
#Cats = 0
#Yes = 1
#No = 0
#Make neighborhood
Neigh = np.zeros([20,20],dtype=int)
for i in range(20):
    for j in range(20):
        Neigh[i,j] = np.random.randint(0,2)
```

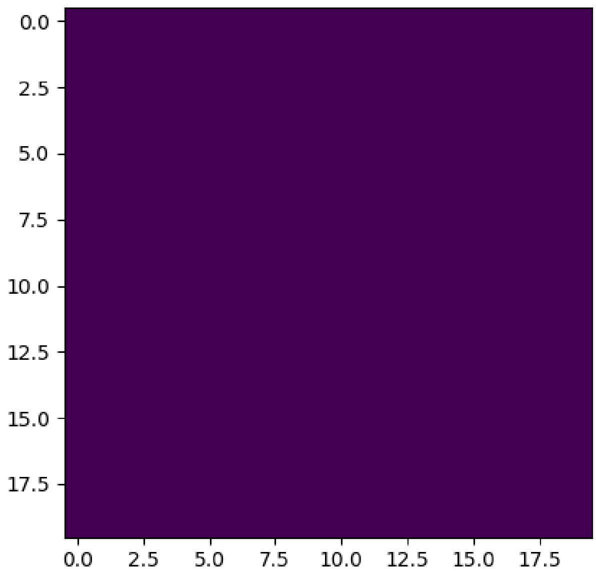
```
#Define Process of testing and changing animal
def Change(Arr,k):
    #Pick a house
    x = np.random.randint(0,20)
    y = np.random.randint(0,20)
    house = Arr[y,x]

    #Periodic Boundary and neighbors
    if x < 1:
        left = -1
    else:
        left = x - 1
    if x > 18:
        right = 0
    else:
        right = x + 1
    if y < 1:
        up = -1
    else:
        up = y - 1
    if y > 18:
        down = 0
    else:
        down = y + 1
    uph = Arr[up,x]
    righth = Arr[y,right]
    downh = Arr[down,x]
    lefth = Arr[y,left]

    #Logic Processes
    #Logic 1
    if int(uph+righth+downh+lefth) == 3 or 4 and house == 0:
        ans = 1
    if int(uph+righth+downh+lefth) == 0 or 1 and house == 1:
        ans = 1
    #Logic 2
    a = np.random.uniform(0,2)
    S = 0
    D = 0
    hlist = np.array([uph,righth,downh,lefth])
    for h in hlist:
        if not h == house:
            S = S + 1
```

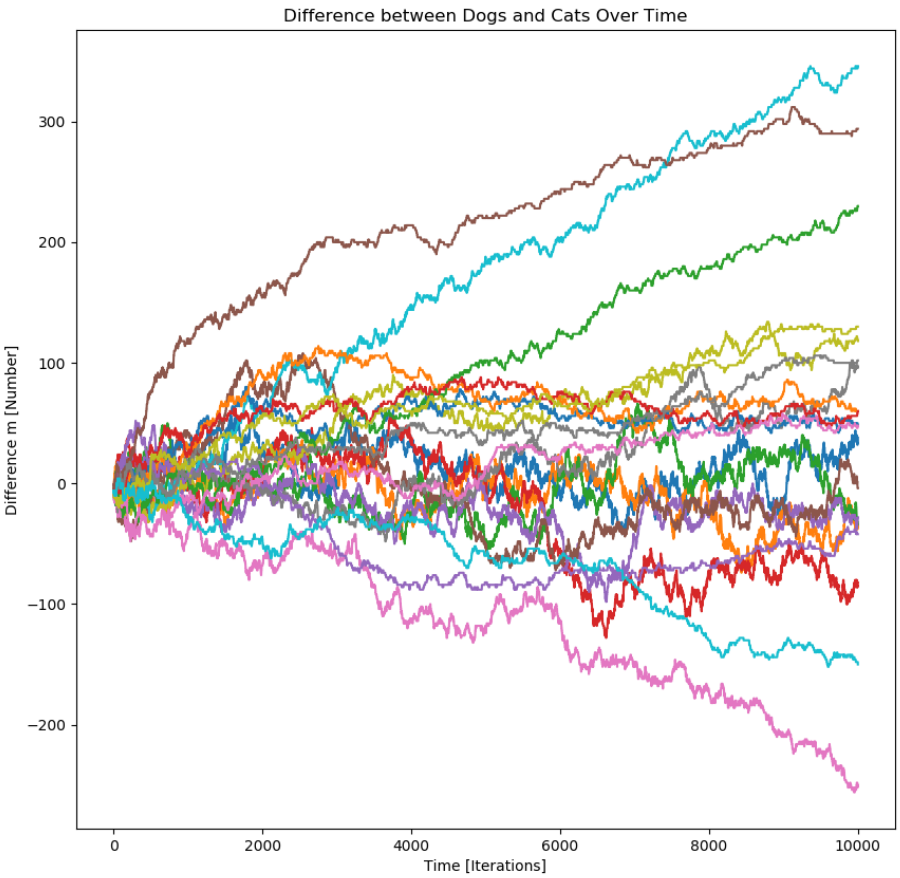
```
        if h == house:
            D = D + 1
    factor = np.exp(-k*(D-S))
    if a < factor:
        L2 = 1
    else:
        L2 = 0
    if int(uph+righth+downh+lefth) == 3 or 4 and house == 1:
        ans = L2
    if int(uph+righth+downh+lefth) == 0 or 1 and house == 0:
        ans = L2
    if int(uph+righth+downh+lefth) == 2:
        ans = L2

    #Change the animal
    if ans == 1:
        if house == 1:
            Arr[y,x] = 0
        if house == 0:
            Arr[y,x] = 1
    return Arr
#%%
#Part 1
ks = np.array([0.1,10.])
Days = 5*(10**4)
for k in ks:
    for i in range(Days):
        Neigh = Change(Neigh,k)
    p.figure(k)
    p.imshow(Neigh)
#%%
#Part 2
ks2 = np.linspace(0.,2.,20)
fig,ax = p.subplots(1,1,figsize=(10,10))
ts = np.array([0])
ms = np.array([(2*np.sum(Neigh)-400)])
for k in ks2:
    NewN = np.copy(Neigh)
    Newt = np.copy(ts)
    Newm = np.copy(ms)
    for i in range(int(Days/5)):
        Newt = np.append(Newt,Newt[i]+1)
        NewN = Change(NewN,k)
        m = 2*np.sum(NewN)-400
```

```
        Newm = np.append(Newm,m)
    ax.plot(Newt,Newm,label="k = "+str(k))
#ax.legend()
ax.set_title("Difference between Dogs and Cats Over Time")
ax.set_xlabel("Time [Iterations]")
ax.set_ylabel("Difference m [Number]")
#%%
ks3 = np.linspace(0.,2.,20)
SDs = np.array([])
Goal = np.array([])
ms2 = np.array([])
for k in ks3:
    NewN2 = np.copy(Neigh)
    for sim in range(10):
        SimSDs = np.copy(SDs)
        Simm = np.copy(ms2)
        for i in range(int(10**5)):
            NewN2 = Change(NewN2,k)
            m = 2*np.sum(NewN2)-400
            Simm = np.append(Simm,m)
        SD = np.std(Simm[50000:])
        SimSDs = np.append(SimSDs,SD)
    F = np.mean(SimSDs)
    Goal = np.append(Goal,F)

fig2,ax2 = p.subplots(1,1)
ax2.plot(ks3,Goal)
ax2.set_title("Mean Standard Deviation of m vs k")
ax2.set_xlabel("k")
ax2.set_ylabel("Mean Standard Deviation of m")
```

I know my code is very inefficient and may not even make sense but I think it works and is complete.


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Difference between Dogs and Cats Over Time
Difference m [Number]
Time [Iterations]
300
200
100
0
−100
−200
0
2000
4000
6000
8000
10000

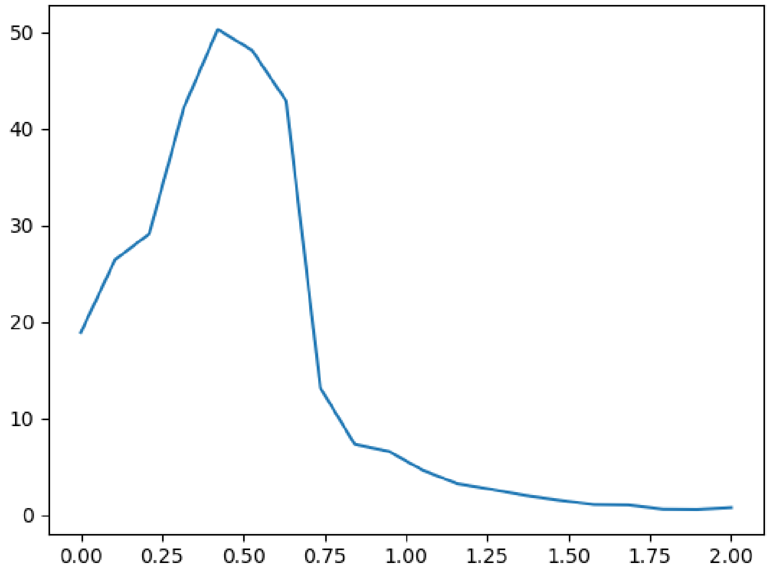


Figure 1: This graph is a little nicer but is missing the proper labels.

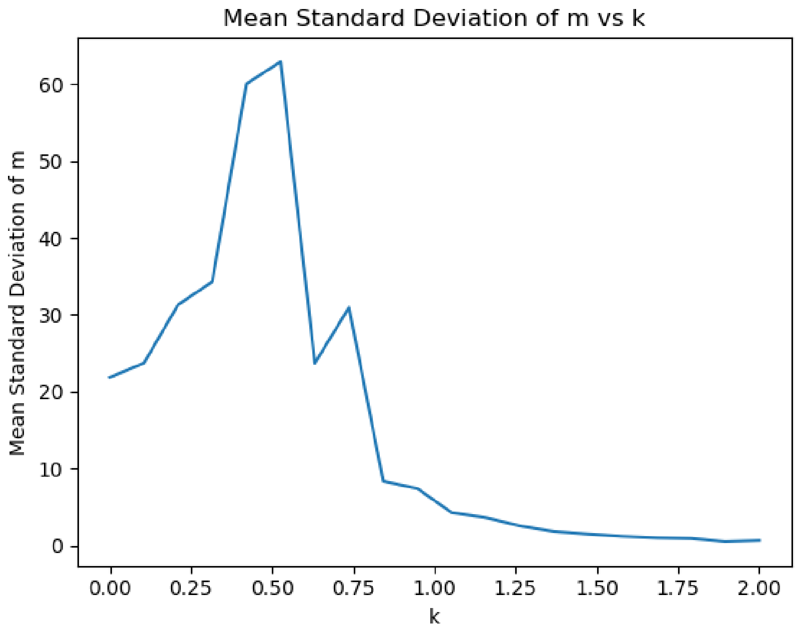


Figure 2: This graph has the proper labels but doesn't look as nice.