CSI 758 Spring 2017 Final

Bruce Goldfeder

May 11, 2017

You may use the computers, previous homeworks, the PDFs I handed out, and personal notes. You may not use other websites or communications with the outside world. (They would probably be wrong anyway.) Start a Word or LibreOffice document. Put your name on the document. Please start each question on a new page. Below is the table of operators and there are more there than you will use. The last two columns mean nothing (I snarfed this from another document and am too lazy to edit it.)

Table 1: List of Defined Operators

Symbol	Name	Op. Type	Function
$\{\}$	Set		
$\mathbf{a}[\alpha\vec{x}]$	Scaling		
$\square$	Window		
$\left\{ \begin{matrix} a \\ b \\ c \end{matrix} \right\}$	Selection		
$\otimes$	Correlation	2	
$\triangle$, $\triangleright$	Erosion and Dilation	2	
$\bigvee$, $\bigwedge$	Max, Min	1	
D	Shift	3	scipy.ndimage
E	Edge	2	edge.DervEdge
$\mathfrak{F}$	Fourier transform	5	
$\mathfrak{G}$	Gabor	2,4	
Γ	Threshold	2	
$\mathcal{L}$	Color Model	5	
L	Intensity Change	2	
$\mathcal{R}$	Rotation	3	
$\mathcal{S}$	Simple smooth		scipy.misc.imfilter
$\Sigma_{\vec{x}}$	Summation over all pixels		
U	Plop in Frame		

Figure 1: Cool Operators

1 Question 1

a is original (b,2), (c,8) (d,3),(e,4), (f,1), (g,6)

2 Question 2

Recall that in the spring (lung) problem we used $F = -k\Delta x$ to be the force on the nodes. Start with a simple architecture of one spring. The left end is tied to the wall and will not move. The right end is attached to a node. So this is a 1 node, 1 spring problem. However, there is a resistance to motion, so there is a second force F2 = -bv where b is a constant and v is the velocity that is related to the position by $v = \Delta x/\Delta t$. Here Δt is the time step in our simulation and it is a constant (so use $\Delta t = 1$). The total force is the sum of all forces. Your task is to write Python script that will simulate the motion of this node that now experiences friction as described. The output should be the position at time steps that increment by Δt.

I would augment the code below to include the counter force this would be added into the calculation for forces.

```
def Start1( springs, nodes ):
# use fudge=0.9995
   springs[0] = [0.5, 1.0, [0,1], 0 ] # k, olength, nodes, dx
   nodes[0] = [np.array((0.5,0)), np.zeros(2), [0] ] # posit,vel,spring
   nodes[1] = [np.array((1,0)), np.zeros(2), [0] ]

def ComputeForces( springs, nodes, dt=0.1 ):
  # NEED to add in the counter force into this calculation TODO
   NS =len( springs )
   forces = np.zeros((len(nodes),2) ) ### 2 = dim
   for i in range( NS):
      n1, n2 = springs[i][2] # node IDs
      p1 = nodes[n1][0]
      p2 = nodes[n2][0]
      currentlen = np.sqrt( (p1-p2)**2).sum()
      dx = springs[i][1] - currentlen
      uvec = (p2 - p1)/np.sqrt( (p1-p2)**2).sum()
      dx = dx * uvec
      springs[i][3] = dx + 0 # store dx
      F = - springs[i][0] * dx
      # accumuluate forces
      forces[n1] += F*dt
      forces[n2] -= F*dt
   return forces

def ComputeMotion( springs, nodes, forces, dt=0.1, fudge=0.9995 ):
   accs = forces/1.0
   NN = len( nodes )
   for i in range( NN ):
      posit,vel = nodes[i][0], nodes[i][1]
      newvel = vel + accs[i] * dt
      newvel *= fudge
      newpos = posit + vel *dt + 0.5*accs[i] * dt*dt
      nodes[i][:2] = newpos,newvel

def ComputeEnergies( springs, nodes ):
   NN = len( nodes )
   NS = len( springs )
   # compute potential
```

```
   U = 0
   for i in range( NS ):
      dx = np.sqrt( (springs[i][3]**2).sum())
      U += 0.5 * springs[i][0] * dx
   # compute kinetic
   K = 0
   for i in range( NN ):
      v2 = (nodes[i][1]**2).sum()
      K += 0.5 * v2# mass =1
   return U, K, U+K

def Iterate(springs,nodes):
   forces = ComputeForces(springs,nodes,dt=0.1)
   ComputeMotion(springs,nodes,forces,dt=0.1,fudge=0.999)
   K,U,T = ComputeEnergies(springs,nodes)
   return K,U,T

def DoRun():
   springs,nodes = Init()
   Start3(springs,nodes)
   x=[]
   for i in range( 5000 ):
      K,U,T= Iterate( springs,nodes)
      x.append(np.array((nodes[0][0][0],nodes[1][0][0],nodes[2][0][0],nodes[3][0][0]) ))
      if i%1000==0: print(K,U,T)

   return x
```

3 Question 3

Below are two sets of images. The first are the input images and the second set is several Fourier transforms after modifications. These modifications are to apply fftpack.fftshift and to compute log(—X— + 1 where X represents the Fourier data after the shift. Your job is to match the Fourier transform image to the input image. There are more Fourier images than originals so not all of the Fourier images will be matched. You may use Python to determine the correct answer if you need.

(b,e) (c,d) (d,f) (e,g (f,h)

4 Question 4

Consider the tachometer problem again. In the text that I handed out the job was to identify all of the numbers on the dial. In the tachometer homework you built an FPF that identified the location of some of the numbers (and rejected others). The term classification means that the individual targets can be distinguished. Thus it is possible to indicate that a certain peak in the output corresponds to a named target. In the original problem we identified that a target exists but did not know what that target was. In this new problem you are to build a system that also identifies which target is at each location. Your job is to design an FPF or FPF system that can identify each target. A system of 9 different filters will not be sufficient for a perfect grade. You will have to perform this task in fewer filters. Again, your job is to design the system, but you do not have to provide code or actual results. So, be specific and if you use equations (always a good idea) then please identify all variables. Basically, I should be able to write complete code from your descriptions. You can start with a matrix X in which each column contains the raveled, Fourier transform of a cutout target image (just as we did in the homework). Note: The descriptions I saw in HW7 were for the most part poorly written. Please be specific when answering this question. Vagueness will count against you. Equations offer better descriptions than text.

The Fractional Power Filter formula is:

$$D_{i,j} = \frac{\delta_{i,j}}{N} \sum_k |\widehat{x}_i^{(k)}|^\alpha \tag{1}$$

where $0 < \alpha < 2$
A setting of $\alpha = 1$ worked well previously and I would use this again.

The set of filters will be images that start as black and white but are smoothed at the edge of the white and black areas to avoid the hard edge issue. I will use four geometric components that comprise each letter.

1. A straight line up and down such as in the 4 and the 1
2. A half sized oval as in the 8 and the 6
3. A half sized half circle as in the 5 and the 3
4. A diagonal line as in the 2 and 7

I would create the filters using code similar to:

```
filt = fpf.FPF( X, cst, 0)
filt = ft.ifft2( filt.reshape((V,H)))
```

As these are all positive correlations I would use a constraint filter of four 1's.
Next I would edge enhance both the filters and the image

Then I would convert each of these into frequency space using fft

Then conjugate the images.

Then convert back using inverse fft.

The resulting image will have hotspots. Graph the image to determine these hotspots.

Repeat this process using one or more filters that are contradictory such as filter one and two and three and four to identify each number individually. These images support this concept. Put them together with a constraint matrix of one negative and one positive 1. For example Filter 2 as a positive and filter 3 as a negative will identify 8.

5 Question 5

We have in our possession alien DNA which has the strange characteristic that the letter 'A' appears twice as often as each of the other letters. So, 40% of the DNA is 'A' and 20% for 'C', 'G' and 'T' respectively. In the HMM homework you computed 52 different matrices. In this case you are only going to calculate the first matrix. We have collected 124 DNA samples from the alien. Given below is the counts of the first two letters of these 124 sequences. Your job is to calculate the following:

1. The count matrix. The letters across the top of the matrix should be ACGT and down the column should also be ACGT. (Please don't change the order).
2. The frequency matrix.
3. The odds matrix.
4. The log odds matrix.

If you can't get one of the other matrices then replace it with believable numbers and proceed. Tell me that is what you are doing. For example, if you don't know how to get the odds matrix but know how to get the log-odds from the odds matrix, then replace the odds matrix with believable numbers and proceed to compute the log-odds. That way you can get credit for the log-odds computation. The data:

- The number of sequences that begin with 'AA': 16
- The number of sequences that begin with 'AC': 10
- The number of sequences that begin with 'AG': 13
- The number of sequences that begin with 'AT': 10
- The number of sequences that begin with 'CA': 8
- The number of sequences that begin with 'CC': 5
- The number of sequences that begin with 'CG': 8
- The number of sequences that begin with 'CT': 1
- The number of sequences that begin with 'GA': 10
- The number of sequences that begin with 'GC': 1
- The number of sequences that begin with 'GG': 5
- The number of sequences that begin with 'GT': 4
- The number of sequences that begin with 'TA': 10
- The number of sequences that begin with 'TC': 8
- The number of sequences that begin with 'TG': 2

- The number of sequences that begin with 'TT': 3

1. $$\begin{bmatrix} 16 & 10 & 13 & 10 \\ 8 & 5 & 8 & 1 \\ 10 & 1 & 5 & 4 \\ 10 & 8 & 2 & 3 \end{bmatrix}$$

2. $$\begin{bmatrix} \frac{16}{49} & \frac{10}{49} & \frac{13}{49} & \frac{10}{49} \\ \frac{8}{22} & \frac{5}{22} & \frac{8}{22} & \frac{1}{22} \\ \frac{10}{20} & \frac{1}{20} & \frac{5}{20} & \frac{4}{20} \\ \frac{10}{23} & \frac{8}{23} & \frac{2}{23} & \frac{13}{23} \end{bmatrix}$$

This is equivalent to the numbers (I switched to python)

```
array([[ 0.32653061,  0.20408163,  0.26530612,  0.20408163],
       [ 0.36363636,  0.22727273,  0.36363636,  0.04545455],
       [ 0.5       ,  0.05      ,  0.25      ,  0.2       ],
       [ 0.43478261,  0.34782609,  0.08695652,  0.13043478]])
```

3. Divide by the probabilities so the vector to divide by is [.4,.2,.2.,2]

```
array([[ 0.81632653,  0.51020408,  0.66326531,  0.51020408],
       [ 1.81818182,  1.13636364,  1.81818182,  0.22727273],
       [ 2.5       ,  0.25      ,  1.25      ,  1.        ],
       [ 2.17391304,  1.73913043,  0.43478261,  0.65217391]])
```

4. Now just take the log of the matrix

```
array([[-0.20294084, -0.67294447, -0.41058021, -0.67294447],
       [ 0.597837  ,  0.12783337,  0.597837  , -1.48160454],
       [ 0.91629073, -1.38629436,  0.22314355,  0.        ],
       [ 0.77652879,  0.55338524, -0.83290912, -0.42744401]])
```