

```

1 #!/usr/bin/python
2 from common import *
3 import pdb
4 def p1(L):
5     #Given:
6     #   L, a list of 1xN vectors with len(L) = M
7     #Return:
8     #   MxN matrix of all the vectors stacked together
9     #Par: 1 line
10    #Instructor: 1 line
11    #Hint: vstack/hstack/dstack, don't use a for loop
12    P = np.vstack(L)
13
14    return P
15
16 def p2(M):
17     #Given:
18     #   M, a n x n matrix
19     #Return:
20     #   v the eigenvector corresponding to the smallest eigenvalue
21     #Par: 5 lines
22     #Instructor: 3 lines
23     #Hints:
24     #   1) np.linalg.eig
25     #   2) np.argmin
26     #   3) Watch rows and columns!
27     wm,vm = np.linalg.eig(M)
28     sm = np.argmin(wm)
29     P=vm[:,sm]
30
31     return P
32
33 def p3(M):
34     #Given:
35     #   a matrix M
36     #Return:
37     #   M, but with all the negative elements set to 0
38     #Par: 3 lines
39     #Instructor: 2 lines (there's a fairly obvious one line solution -- hint
40     #               np.minimum/maximum/np.clip)
41     #Hint:
42     #   1) if S is the same size as M and is True/False, you can refer to all
43     #       true entries via M[S]
44     #   2) if M[S] is the set of all entries, you can assign to them all with
45     #       M[S] = v for some value v
46     P = np.clip(M,0,np.amax(M))
47
48     return P
49
50
51 def p4(t):
52     #Given:
53     #   a tuple of 3x3 rotation matrix R
54     #   Nx3 matrix M
55     #Return:
56     #   a Nx3 matrix of the rotated vectors
57     #Par: 3 lines
58     #Instructor: 1 line
59     #Hint:
60     #   1) Applying a rotation to a vector is right-multiplying the rotation
61     #       matrix with the vector
62     #   2) .T transposes; this may make your life easier
63     #   3) np.dot matrix-multiplies
64     R, M = t #unpack
65     P=np.dot(M,R.T)
66

```

```

67     return P
68
69 def p5(M):
70     #Given:
71     #    a NxN matrix M
72     #Return:
73     #    the upper left 4x4 corner - bottom right 4x4 corner
74     #Par: 2 lines
75     #Instructor: 1 line
76     #Hint:
77     #    M[ystart:yend,xstart:xend] pulls out the sub-matrix
78     #        from rows ystart to (but not including!) yend
79     #        from columns xstart to (but not including!) xend
80     P = M[0:4,0:4]-M[-4:-4:]
81
82     return P
83
84 def p6(n):
85     #Given:
86     #    n -- an integer
87     #Return:
88     #    a nxn matrix of 1s, except the first and last 5 columns and rows are 0
89     #Par: 5 lines
90     #Instructor: 5 lines (can you make it shorter with np.zeros)
91     #Hints:
92     #    np.ones/np.zeros, it's ok to double-write
93
94     M = np.zeros((n,n))
95     M[5:n-5,5:n-5] = 1
96
97     return M
98
99
100 def p7(M):
101     #Given:
102     #    a NxF matrix M
103     #return:
104     #    S -- the same matrix but where each row is scaled to have unit norm
105     #    (i.e., S[i,:] is unit norm, and S[i,j] = M[i,j]*a[i] for some a[i])
106     #Par: 3 lines
107     #Instructor: 1 line
108     #Hints:
109     #    1) The vector v / ||v|| is unit norm for v != 0
110     #    2) Compute the normalization factor as a Nx1 vector by N[i] = \sum_j M[i,j]^2
111     #    3) Elementwise divide a NxF matrix by a Nx1 vector -- see what happens!
112     #        (broadcasting)
113     #    4) If it won't go together -- np.expand_dims or keepdims in np.sum or doing
114     #        X[None,:] to add dimensions (try X[None,:] and X[None,:]) on a vector
115
116     # N=np.square(N)
117     # N=np.sum(M, axis=1)
118     # N = np.sqrt(N)
119     # P=M/N
120
121     norm=np.linalg.norm(M,axis=1)
122     norm = norm[None,:].T
123     P = M/norm
124
125
126     return P
127
128
129 def p8(M):
130     #Given:
131     #    a matrix M
132     #Return:

```

```

133 # the same matrix but where each row is normalized to have mean 0 and std 1
134 # (i.e. mean(S[i,:]) = 0, std(S[i,:]) = 1 and S[i,j] = M[i,j]*a[i]+b[i] for some a[i],b[i])
135 #Par: 3 lines
136 #Instructor: 2 lines (but you can make it one)
137 #Hints:
138 # 1) If it won't broadcast, do np.expand_dims, keepdims
139 M-=np.mean(M,axis=1,keepdims=True)
140 M/=np.std(M,axis=1,keepdims=True)
141
142 return M
143
144 def p9(t):
145     #Given a:
146     # query q -- (1xK)
147     # keys k -- (NxK)
148     # values v -- (Nx1)
149     #Return
150     # sum_i exp(-||q-k_i||^2) * v[i]
151     #Par: 3 lines
152     #Instructor: 1 incomprehensible line, not written in one go
153     #Hints:
154     # 1) Again A NxK matrix and a 1xK vector go together the way you think
155     # (broadcasting)
156     # 2) np.sum has an axis and keepdims arguments
157     # 3) np.exp, - and friends apply to matrices too
158     Q, K, V = t #unpack
159
160     J=np.exp(-np.linalg.norm((Q-K),axis=1)**2)
161     J = J[:,None]
162     P=np.sum(J*V)
163
164     return P
165
166 def p10(L):
167     #given:
168     # a list NxF matrices of length M
169     #return:
170     # a MxM matrix R where
171     # R_ij = distance between the F-dimensional centroid of each matrix
172     #Par: 12 lines
173     #Instructor: 7 lines (there's a 9 line solution that avoids double work
174     # and apparently a 4 line solution too)
175     #Hints:
176     # 1) For loop over M
177     # 2) Distances are symmetric, so don't double compute that
178     # 3) Go one step at a time
179     Cen = []*len(L)
180     R = np.zeros((len(L),len(L)))
181     for i in range(len(L)):
182         Cen.append(np.sum(L[i],axis=0)/np.size(L[i],0))
183
184     for i in range(len(L)):
185         for j in range(len(L)):
186             if j>=i:
187                 R[i,j] = np.linalg.norm(Cen[i]-Cen[j])
188                 R[j,i] = R[i,j]
189
190     return R
191
192
193 def p11(M):
194     #given:
195     # a NxF matrix M
196     #compute the NxN matrix D
197     # D[i,j] = distance between M[i,:] and M[j,:]
198     # using ||x-y||^2 = ||x||^2 + ||y||^2 - 2x^T y

```

```

199 #Par: 3 lines
200 #Instructor: 2 lines (you can do this in one but it's wasteful compute-wise)
201 #Hints:
202 # 1) If I add a Nx1 vector and a 1xN vector, what do I get?
203 # 2) Look at the definition of matrix multiplication for the second bit
204 # 3) transpose is your friend
205 # 4) Note the square! -- square root it at the end
206 # 5) On some computers, you may have issues with  $||x||^2 + ||x||^2 - 2x^Tx$ 
207 # coming out as ever so slightly negative. Just make  $\max(0, \text{value})$  --
208 # note that the distance between x and itself should be exactly 0.
209 # Seems to occur on macs
210
211 XNorm = np.sum(M**2,axis=1,keepdims=True)
212 D_s = XNorm+XNorm.T - 2*np.dot(M,M.T)
213 D = np.maximum(0,D_s)**0.5
214
215 return D
216
217
218 def p12(t):
219     #Given:
220     # a NxF matrix A
221     # a MxF matrix B
222     #compute the NxM matrix D
223     #  $D[i,j]$  = distance between  $A[i,:]$  and  $B[j,:]$ 
224     #Par: 3 lines
225     #Instructor: 1 line
226     #Hints: same same but different; draw some boxes on a piece of paper
227     A,B = t #unpack
228     XNorm = np.sum(A**2,axis=1,keepdims=True)
229     YNorm = np.sum(B**2,axis=1,keepdims=True)
230     D = np.maximum(0,XNorm+YNorm.T - 2*np.dot(A,B.T))**0.5
231
232     return D
233
234
235 def p13(t):
236     #Given:
237     # a 1xF query vector q
238     # NxF matrix M
239     #Return:
240     # the index i of the row with highest dot-product with q
241     #Par: 1 line
242     #Instructor: 1 line
243     #Hint: np.argmax
244     q, M = t #unpack
245
246     ind=np.argmax(np.dot(M,q.T))
247
248     return ind
249
250 def p14(t):
251     #given a tuple of:
252     # X NxF matrix
253     # y Nx1 vector
254     #Return the w Fx1 vector such that
255     #  $||y-Xw||^2_2$  is minimized
256     #Par: 2 lines
257     #Instructor: 1 line
258     #Hint: np.linalg.lstsq or do it the old fashioned way  $(X^T X)^{-1} X^T y$ 
259     X , y = t #unpack
260
261     W,*_=np.linalg.lstsq(X,y,rcond=-1)
262
263
264

```

```

265     return W
266
267
268 def p15(t):
269     #Given a tuple of:
270     #   X: Nx3 matrix
271     #   Y: Nx3 matrix
272     #Return a matrix:
273     #   C such that  $C[i,:] = \text{the cross product between } X[i,:] \text{ and } Y[i,:]$ 
274     #Par: 1 line
275     #Instructor: 1 line
276     #Hint: np.cross and read the documentation or just try it
277     X, Y = t #unpack
278     C= np.cross(X,Y)
279
280     return C
281
282 def p16(X):
283     #Given:
284     #   a NxF matrix X
285     #Return a Nx(F-1) matrix Y such that
286     #    $Y[i,j] = X[i,j] / X[i,-1]$ 
287     #   for all i and j
288     #Par: 1 line
289     #Instructor: 1 line
290     #Hint: if it doesn't broadcast, np.expand_dims
291
292     Y=X/np.expand_dims(X[:,-1],axis=1)
293
294     return Y[:, :2]
295
296 def p17(X):
297     #Given:
298     #   a NxF matrix X
299     #Return a Nx(F+1) matrix Y such that
300     #    $Y[i,:F] = X[i,:]$  and
301     #    $Y[i,F] = 1$ 
302     #Par: 1 line
303     #Instructor: 1 lines
304     #Hint: np.hstack, np.ones
305
306     Y = np.hstack((X,np.ones((np.size(X,0),1))))
307
308     return Y
309
310
311 def p18(t):
312     #Given:
313     #   an integer n
314     #   a radius r
315     #   an x coordinate x
316     #   a y coordinate y
317     #Return:
318     #   An nxn image I such that
319     #    $I[i,j] = 1$  if  $||[j,i] - [x,y]|| < r$ 
320     #    $I[i,j] = 0$  otherwise
321     #Par: 3 lines
322     #Instructor: 2 lines
323     #Hint:
324     #   1) np.meshgrid and np.arange give you X,Y
325     #   2) watch the <
326     #   3) arrays have an astype method
327     n,r,x,y = t #unpack
328
329     X,Y = np.meshgrid(np.arange(n),np.arange(n),sparse=True)
330     I= ((X-x)**2 +(Y-y)**2<r**2).astype(float)

```

```

331     return I
332
333
334
335 def p19(t):
336     #Given:
337     #    an integer n
338     #    a float s
339     #    an x coordinate x
340     #    a y coordinate y
341     #Return:
342     #    a nxn image such that
343     #     $I[i,j] = \exp(-||[j,i]-[x,y]||^2 / s^2)$ 
344     #Par: 3 lines
345     #Instructor: 2 lines
346     #Hint: watch the types -- float and ints aren't the same!
347     n, s, x, y = t #unpack
348
349     X,Y = np.meshgrid(np.arange(n),np.arange(n),sparse=True)
350     I = np.exp(-((X-x)**2+(Y-y)**2)/s**2)
351
352     return I
353
354
355 def p20(t):
356     #Given:
357     #    an integer n
358     #    a vector v = [a,b,c]
359     #Return:
360     #    a matrix M such that  $M[i,j]$  is the distance from the line  $a*j+b*i+c=0$ 
361     #
362     #    Given a point (x,y) and line  $ax+by+c=0$ , the distance from x,y to the line
363     #    is given by  $\text{abs}((ax+by+c) / \sqrt{a^2 + b^2})$  (the sign tells you which side)
364     #Par: 4 lines
365     #Instructor: 2 lines
366     #Hints:
367     #    np.abs works on matrices too
368     n, v = t #unpack
369     X,Y = np.meshgrid(np.arange(n),np.arange(n),sparse=True)
370
371     M=np.abs((v[0]*X+v[1]*Y+v[2])/np.sqrt(v[0]**2 + v[1]**2))
372
373     return M

```

```
[iunghuiui-MacBook-Pro:mastery_assignment ungheele$ python3 run.py --alltests
Running p1
Running p2
Running p3
Running p4
Running p5
Running p6
Running p7
Running p8
Running p9
Running p10
Running p11
Running p12
Running p13
Running p14
Running p15
Running p16
Running p17
Running p18
Running p19
Running p20
Ran all tests
20/20 = 100.0
```

```

1  from common import *
2
3
4  def b1(M):
5      #Given:
6      #    a matrix M
7      #Return:
8      #    the matrix S such that  $S[i,j] = M[i,j]*10+100$ 
9      #Hint: Trust that numpy will do the right thing
10     S = M*10 + 100
11
12     return S
13
14  def b2(t):
15      #Given:
16      #    a nxn matrix M1
17      #    a nxn matrix M2
18      #Return:
19      #    the matrix P such that  $P[i,j] = M1[i,j]+M2[i,j]*10$ 
20      #Hint: Trust that numpy will do the right thing
21      M1, M2 = t #unpack
22      P = M1 + M2*10
23
24      return P
25
26  def b3(t):
27      #Given:
28      #    a nxn matrix M1
29      #    a nxn matrix M2
30      #Return:
31      #    the matrix P such that  $P[i,j] = M1[i,j]*M2[i,j]-10$ 
32      #Hint: By analogy to + , * will do the same thing
33      M1, M2 = t #unpack
34      P = M1*M2-10
35
36      return P
37
38  def b4(t):
39      #Given:
40      #    a nxn matrix M1
41      #    a nxn matrix M2
42      #Return:
43      #    the matrix product M1 M2
44      #Hint: Not the same as * !
45      M1, M2 = t #unpack
46      P = M1.dot(M2)
47
48      return P
49
50  def b5(M):
51      #Given:
52      #    a nxn matrix M of floats
53      #Return:
54      #    a nxn matrix M of integers
55      #Hint: astype
56      # M.astype(int)
57      M=np.int64(M)
58
59      return M
60
61  def b6(t):
62      #Given:
63      #    a nx1 vector M of integers
64      #    a nx1 vector D of integers

```



```

65     #Return:
66     # the ratio (M/D), treating them as floats (i.e., 1/5 => 0.2)
67     #Hint: dividing one integer by another is not the same as dividing two floats
68     M, D = t #unpack
69     P=M/D
70
71     return P
72
73 def b7(M):
74     #Given:
75     # a nxm matrix M
76     #Return:
77     # a vector v of size (nxm)x1 containing the entries of M, listed in row order
78     #Hint:
79     # 1) np.reshape
80     # 2) you can specify an unknown dimension as -1
81     P= np.reshape(M,(-1,1))
82
83     return P
84
85 def b8(n):
86     #Given:
87     # an integer n
88     #Return:
89     # a nx(2n) matrix of ones
90     #Hint:
91     # data type not understood with calling np.zeros/np.ones is guaranteed
92     # to be an issue where you passed in two arguments, not a tuple
93     P = np.ones((n,2*n), dtype=float)
94
95     return P
96
97 def b9(M):
98     #Given:
99     # a matrix M where each entry is between 0 and 1
100    #Return:
101    # a matrix S where S[i,j] = True if M[i,j] > 0.5
102    #Hint: Trust python to do the right thing
103    S = M > 0.5
104
105    return S
106
107 def b10(n):
108     #Given:
109     # an integer n
110     #Return:
111     # the n-entry vector of 0, ..., n-1
112     #Hint: range+np.array/np.arange
113     result=np.arange(n)
114
115     return result
116
117 def b11(t):
118     #Given:
119     # a NxF matrix A
120     # a Fx1 vector v
121     #Return:
122     # the matrix-vector product Av
123     A, v = t
124     P = A.dot(v)
125
126     return P
127
128 def b12(t):
129     #Given:

```

```

130     #   a NxN matrix A, full rank
131     #   a Nx1 vector v
132     #Return:
133     #   the inverse of A times v:  $A^{-1} v$ 
134     A, v = t
135     P = np.linalg.inv(A).dot(v)
136
137     return P
138
139
140 def b13(t):
141     #Given:
142     #   a Nx1 vector u
143     #   a Nx1 vector v
144     #Return:
145     #   the innner product  $u^T v$ 
146     #Hint:
147     #   .T
148     u, v = t
149     P=np.transpose(u).dot(v)
150
151     return P
152
153 def b14(v):
154     #Given:
155     #   a Nx1 vector v
156     #Return:
157     #   the L2-norm without calling np.linalg.norm
158     #norm = ( $\sum_{i=1}^N v[i]**2$ )**0.5
159     P = np.linalg.norm(v)
160
161     return P
162
163 def b15(t):
164     #Given:
165     #   a NxF matrix M
166     #   an integer i
167     #Return:
168     #   the ith row of M
169     M, i = t
170     P = M[i,:]
171
172     return P
173
174 def b16(M):
175     #Given:
176     #   a NxF matrix M
177     #Return:
178     #   the sum of all the entrices of the matrix
179     #Hint:
180     #   np.sum
181     P = np.sum(M)
182
183     return P
184
185 def b17(M):
186     #Given:
187     #   a NxF matrix M
188     #Return:
189     #   a N-entry vector S where  $S[i]$  is the sum along row i of M
190     #Hint:
191     #   np.sum has an axis optional arg; note keepdims if you already know this
192     P = np.sum(M, axis=1)
193
194     return P

```

```

195
196 def b18(M):
197     #Given:
198     #    a NxF matrix M
199     #Return:
200     #    a F-entry vector S where S[j] is the sum along column j of M
201     #Hint: same as above
202     P = np.sum(M, axis=0)
203
204     return P
205
206 def b19(M):
207     #Given:
208     #    a NxF matrix M
209     #Return:
210     #    a Nx1 matrix S where S[i,1] is the sum along row i of M
211     #Hint:
212     #    Watch axis, keepdims
213     P = np.sum(M, axis=1)
214     P = P.reshape(-1,1)
215
216     return P
217
218
219 def b20(M):
220     #Given:
221     #    a NxF matrix M
222     #Return:
223     #    a Nx1 matrix S where S[i] is the L2-norm of row i of M
224     #Hint:
225     #    Put it together
226     P = np.linalg.norm(M,axis=1)
227     P = P.reshape(-1,1)
228
229     return P

```

```
iunghuiui-MacBook-Pro:mastery_assignment ungheele$ python3 run.py --allwarmups
Running b1
Running b2
Running b3
Running b4
Running b5
Running b6
Running b7
Running b8
Running b9
Running b10
Running b11
Running b12
Running b13
Running b14
Running b15
Running b16
Running b17
Running b18
Running b19
Running b20
Ran warmup tests
20/20 = 100.0
```