Take Home Final Exam

MATH/COMP-365 Computational Linear Algebra
Monday, April 30, 2018 — Sunday, May 6, 2018

- **Please upload your answers to Moodle and turn in a hard copy to my office before I arrive on Monday morning (around 8:45am). If I'm not there, slide it under my door, but if I am, come say hi!**
- The exam is open notes and open book. You may use the internet and library to look things up, but of course you should cite any reference you use that is not the textbook.
- **It is not okay to talk to each other or anyone else about the exam! This of course includes discussing how you solve problems, but it even includes things like asking, "how it is going?" or "which problem you are working on?" or making comments like "problem 5 is easy."**
- Please show and explain all of your work so that I can give you as much partial credit as possible.
- You are welcome to use any of the `R` functions in `365Functions.r`
- The most convenient method to turn in your answers would be to do them in a single R Markdown file and submit the html file containing your code and your solutions. I have provided a starter file `2018.365.Final.Rmd`. **Make sure to insert your name and type out the academic honesty statement at the top of your Markdown file.**
- In addition to submitting your file on Moodle, you should turn in a hard copy of your answers (including both the typed section and any hand written comments) by sliding it under the door of my office.
- Hints: I may give a small hint or a little help with R for free. If you are stuck, I may sell a hint for points so that you can move forward. I will warn you of the point value before I sell you anything.
- I may reserve some "style" points for the clarity of your work and the quality of your computation. That is, I may possibly take some points off if you get the right answer, but do it in a confusing or clumsy or inefficient way. (It is still true, though, that it's okay to use a loop even if vectorization is possible).

Problem	Point Value	Your Score
1	15	
2	26	
3	18	
4	14	
5	17	
Total	90	
EC	2+5	

Please sign the following (if you are able) and be sure to turn in this cover sheet with your exam: *I pledge my honor that I have not participated in any dishonest work on this exam, nor do I know of dishonest work done by other students on this exam.*

_____
(signature)

As an alternative to turning this in, you may also type out this pledge as part of your exam.

1. (15 points)

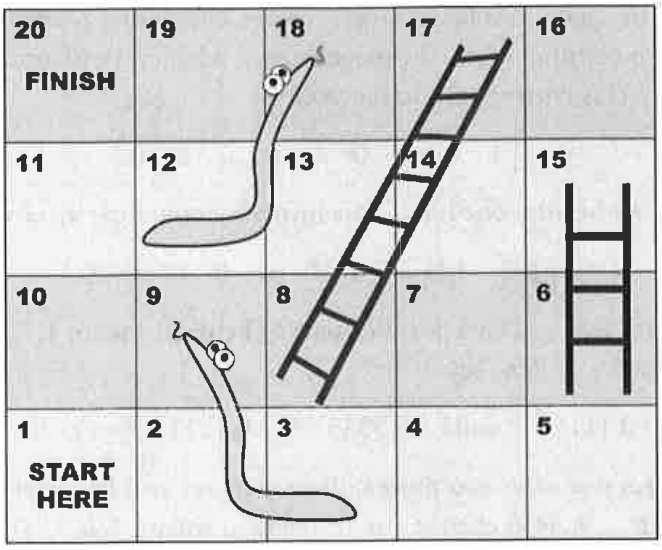


Figure 1: A Chutes and Ladders game board variant.

In this problem, we are going to model a variant of the Chutes and Ladders game as a Markov chain. We'll analyze some of the game dynamics **assuming there is just one player**. Here are some more details of the movement rules on the game board shown above in Figure 1:

- In this variant of the game, the single player roles a **three-sided** die each turn, which yields the possible values of 0, 1, or 2, each with probability equal to one third. The player starts at square 1, and moves the number of spaces rolled on the die. The direction of movement is according to the order indicated in the top left corner of each square. For example, if the player starts on square 5 and rolls a 2, she ends up on square 7. If she rolls a 0, she stays on square 5.

- If the player's roll leads to a space that is the bottom of the ladder, she is **instantaneously** transferred to the top of the ladder. For example, if she starts on square 5 and rolls a 1, she ends the turn on square 15.

- On the other hand, if the player's roll leads to a space that is the top of a chute, she instantaneously falls to the bottom of the slide. For example, if she starts on square 7 and rolls a 2, she ends the turn on square 3.

- Once she reaches square 20, she stays there and the game is over. If the player rolls 2 while starting on square 19, she moves to square 20 and the game is over.

(a) (5 points) Let the state of the Markov chain be the square on which the player starts her turn. First draw a state diagram and label the edges with the transition probabilities. Then construct the transition matrix P and check that the rows sum to 1.

(b) (5 points) Compute the probability that after 20 turns the player has reached the end (square 20), naturally she should start at square 1. Save this probability in a variable called *pend*. Are there any squares which are unreachable after 20 turns?

(c) (5 points) Now let's change the board slightly and add a chute from square 20 back to square 1, so the game never ends![1] You play the game for a very long time (e.g., while you are contemplating the beauty of linear algebra), and then I walk into the room. What is the probability that I find you on square 19 at the beginning of your next turn? How about on square 10? On which square(s) am I most likely to find you, and what is the probability of being there? Save these quantities in variables called `p19`, `p10`, `mostlikely`, and `plikely`.

[1] If you roll a 2 on 19, you also go back to square 1.

2. (26 points) Your local post office has asked you to use your computational linear algebra skills to help them recognize handwritten digits (i.e. the numbers 0-9) in images. You are given the data sets described below:

   `train_data`: a 256×4649 matrix that stores the data. Each column is a 16×16 image that has been flattened into a 256×1 vector. Note that the data has been sorted, starting with images corresponding to 0, then 1, then 2, up to 9.

   `indices`: a 10×1 vector which indicates the index where each new digit in `train_data` starts

   `train_data_1000`: a 256×1000 matrix that stores 100 samples of each digit in the columns, sorted in the order from 0 to 9.

   `test_data`: a 256×4649 matrix on which we would like to predict the digit

   `test_label`: a 4649-dimensional vector revealing the true label for each of the columns in `test_data` (this allows us to compare how accurate our prediction is).

   (a) (3 points) To investigate the data a bit, load each of the data sets above (prompts to do so can be found in the starter file). Then, one at a time, extract the columns of `train_data` corresponding to the first three values in `indices`, reform each column as a 16×16 matrix, and then use the `imPlot` command in `365Functions.r` to view an image of the transpose of each matrix. They should look like the numbers 0, 1, 2 but notice that the 2 is fairly hard to distinguish, even for a human!

   (b) We are going to classify each digit using a few different methods. First, we will explore the method of least squares. For each digit between 0 and 9, we will set up a system of equations

   $$A_0\mathbf{x} = \mathbf{b}, A_1\mathbf{x} = \mathbf{b}, \ldots, A_9\mathbf{x} = \mathbf{b},$$

   where A_i corresponds to the flattened images of the ith digit in the training data. Then, for each new flattened image $\mathbf{b}$ to be classified in `test_data`, we solve the system of equations $A_i\mathbf{x} = \mathbf{b}$ to find $\mathbf{x}^*$, find $\hat{\mathbf{b}} = A_i\mathbf{x}^*$, and compute the error as the norm of the residual. The target vector $\mathbf{b}$ should be classified as digit i with the smallest error.

      i. (2 points) Each of the systems with the full training data $A_i\mathbf{x} = \mathbf{b}$ are an:
      Overdetermined or Underdetermined system (choose one).
      Why should the least squares method not be used on the full training data?

      ii. (9 points) We will instead use the sampled data set `train_data_1000` to perform least squares. We have discussed three ways to solve least squares problems. Write a function for each method to find the digit with the minimum residual given a target vector.
      - Least squares with the normal equations, call this `digit_NormLS`
      - Least squares with the QR decomposition, call this `digit_QRLS`
      - Least squares with the SVD method, call this `digit_SVDLS`

      iii. (2 points) Test the accuracy and processing time of each of the methods above using the function provided in the starter file.[2] What do you notice?

   (c) Now, we would like to use the SVD to construct a 'best basis' for the images of the digits using the Eigenimages approach discussed in the S27 slides.

      i. (4 points) For each digit i, compute the best J-dimensional basis $U_J^{(i)}$ corresponding to digit i, project a new target vector $\mathbf{b}$ into that basis, and compute the norm of the residual (i.e. the error). Again, the target vector $\mathbf{b}$ should be classified as digit i with the smallest error. Write a function called `digit_SVD` which takes as input the target vector $\mathbf{b}$ and the matrix U_J which binds together all of the $U_J^{(i)}$ matrices for each digit into its columns.

      ii. (2 points) Use `train_data_1000` to build the matrix $U_J^{(i)}$ containing the best J-dimensional basis for each i and then bind it together into U_{100}. Choose $J = 17$.[3]

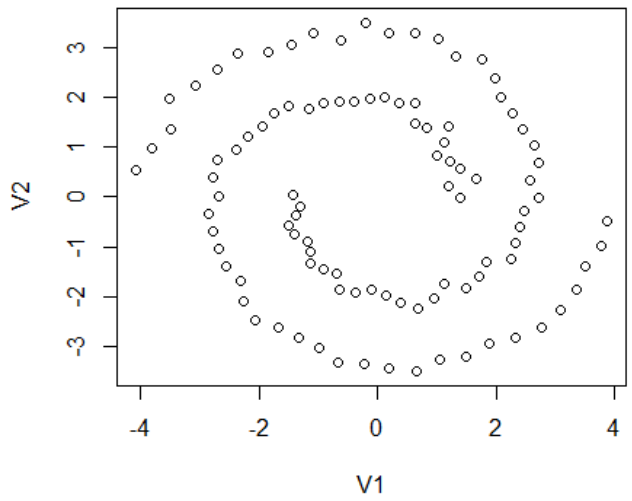
      iii. (2 points) Now, we do not run into the issue discussed above with using the full training data. Use `train_data` to build your matrix $U_J^{(i)}$ containing the best J-dimensional basis for each i and then bind it together into $Uall$. Again, choose $J = 17$.

      iv. (2 points) Test the accuracy and processing time of each of the methods above using the function provided in the starter file. What do you notice both in relation to each other and in relation to the least squares methods?

[2] It will take several minutes to perform this on the full testing data. While you are debugging your code, you may want to try your functions on, say, the first 200 columns.

[3] After experimenting with a few different choices, 17 was chosen as it gives good classification rates.

3. (18 points) In data analysis, we sometimes want to group similar data points together in one group, a cluster. Consider, for instance, the following data which is connected but not so easy to cluster.



Spectral clustering is a method that can detect each of these two spirals by looking at the eigenvalues[4] of a particular matrix following the steps below.

(a) (2 points) Spectral clustering uses a *similarity* measure to determine how close two points x_i and x_j are from one another. One measure of similarity, known as a Gaussian kernel is defined as $s(x_i, x_j) = exp(-\alpha||x_i - x_j||_2^2)$, which is ≈ 1 if two points are close and ≈ 0 if two points are far. Create a function `s` that computes the Gaussian kernel similarity between two input points, using a default value of $\alpha = 1$.

(b) (3 points) Denote the similarity matrix S as the matrix $S_{i,j} = s(x_i, x_j)$ which gives the similarity between observations x_i and x_j. Write a function `make.similarity` which takes as input the data and a similarity measure (in our case `s` defined above) and outputs the similarity matrix.

(c) (3 points) We expect that when two points are from different clusters they are far away. Notice, though, that in the spiral example two points from the same spiral can be farther away than points in different spirals. However, there is a sequence of points between any two points in one of the spirals creating a "path" between them. This is encoded in the *affinity* matrix A[5], which only defines a nonzero similarity between "nearest neighbors" of a data point, connecting just the closest points. I have provided code to compute the affinity matrix for you. Comment what each line does.

```
make.affinity <- function(S, n.neighbors=2) {
  N <- length(S[,1])

  if (n.neighbors >= N) {
    A <- S
  } else {
    A <- matrix(0, ncol=N, nrow=N)
    for(i in 1:N) {
      closest.similarities <- sort(S[i,], decreasing=TRUE)[1:n.neighbors]
      for (s in closest.similarities) {
        j <- which(S[i,] == s)
        A[i,j] <- S[i,j]
        A[j,i] <- S[i,j]
      }
    }
  }
  return(A)
}
```

[4]The eigenvalues of the Laplacian matrix are often called the spectrum of a graph, hence the name spectral clustering.
[5]An affinity matrix is based on a similarity matrix S but must be made of positive values and be symmetric.

Note that with this affinity matrix, clustering is replaced by a graph-partition problem, where connected graph components are interpreted as clusters.

(d) (10 points) Now, follow the algorithm outlined below:

i. Compute the similarity matrix, S, between all points in the spiral data.

ii. Calculate the affinity matrix, A, using $n.neighbors = 2$ for the spiral data.

iii. Compute the diagonal degree matrix, D, by summing the rows of A and then putting the row sum as the diagonal entry in matrix D.

iv. Calculate the Laplacian $L = D - A$. The graph Laplacian has several cool features. In particular, the number of connected components in a graph is the dimension of the null space of L.

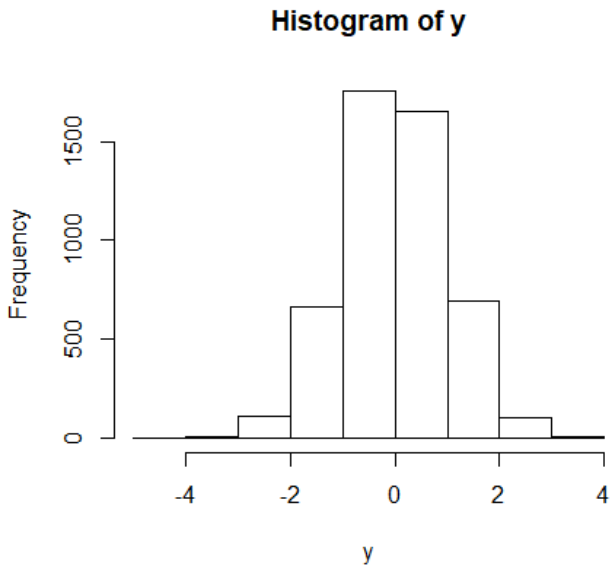
v. Compute the eigenvalues and eigenvectors of L. How many of the eigenvalues are 0? Does this agree with the statement above?

vi. Now, take the eigenvectors in the null space of L, call these Z. Perform k-means clustering with k=*number of eigenvectors in the null space of* L on Z using the following code and plot the spirals colored according to these clusters like so[6]:

```
library(stats)
km <- kmeans(Z, centers=k, nstart=5)
plot(spirals, col=km$cluster)
```

[6]The final step should be for you to say "Wow!"

4. (14 points) Suppose you have a histogram and you would like to fit a curve to it, so that you can understand the underlying distribution. Below is a randomly generated dataset (code found in starter file):



The variable *heights* is the height of each bar and *centers* is the center (x-position) of each bar. The histogram has bars that are equally spaced with t as the interval values (i.e. the breaks).

(a) (4 points) Fit a natural cubic spline directly using the centers and the heights as your data points. Plot this spline on top of the histogram. Do you think it is a good fit? Why or why not? (Hint: Think about the laws of probability.)

(b) Let's instead create a curve by using the cumulative density function (CDF), which cumulates the area under the probability density function (PDF). We would like to construct a spline function f whose average value over each interval equals the height of the corresponding bar. In other words, if h is the height of one of these bars, and its left and right edges are at L and R, then we want the spline f to satisfy

$$\frac{\int_L^R f(x)dx}{R-L} = h$$

Therefore, we would like to define a spline for the CDF (the definite integral of the function f), and then take the derivative of the CDF to get the function f. Here are steps to do so:

i. (2 points) Construct a variable $Fvals$, which cumulates the total area under the histogram for each of the break point values. For instance, at break point $t[1] = -5$, $Fvals[1] = 0$, at $t[2] = -4$, $Fvals[2] = 2$, at $t[3] = -3$, $Fvals[3] = 9$, and so on. You may find the function `cumsum` useful.

ii. (4 points) Now, fit a clamped cubic spline to the data points defined by t and $Fvals$ where the derivatives of the endpoints are defined to be 0 [7] This gives you a function for your CDF, call it F. Plot this function, and notice that it is always an increasing function over the domain.

iii. (4 points) Now, the CDF can be interpreted as an anti-derivative of our PDF. Therefore, computing the derivative[8] of the CDF will give us the PDF, which is our desired function f. Compute f and plot it on top of the histogram. Does it provide a better fit than the original natural cubic spline?

[7] Note the R splinefun command does not have a built-in method to do clamped cubic spline functions. Instead, use the cubicspline command in the pracma package. In the examples, you can see how to take the information given from cubicspline to make a function.

[8] Remember you've already written code to compute numerical derivatives earlier this semester.

5. (17 points) A Cryptogram

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	0	1	2	5	0	1	4	0	2	0	1	9	0	15	0	1	0	10	5	36	1	8	0	0	1	0
B	1	0	0	0	5	0	0	0	1	0	0	1	0	0	1	0	0	2	0	0	2	0	0	0	1	0
C	12	0	0	0	4	0	0	2	1	0	0	0	0	0	7	0	0	4	0	1	0	0	0	0	0	0
D	2	0	1	6	14	0	0	3	13	0	0	0	0	0	4	1	0	0	4	4	1	1	4	0	0	0
E	16	3	8	26	3	5	2	7	6	0	0	4	5	10	5	4	1	22	9	12	2	4	8	0	3	0
F	3	0	0	1	0	1	0	0	5	0	0	0	0	0	10	0	0	3	0	3	1	0	0	0	0	0
G	5	1	0	0	5	0	1	4	0	0	0	1	0	0	3	1	0	6	0	0	0	1	0	0	0	0
H	24	0	0	0	32	2	0	0	7	0	0	1	0	0	8	0	0	0	0	5	1	0	0	0	0	0
I	0	1	8	1	3	0	2	0	0	0	0	2	0	16	9	0	0	2	9	8	0	7	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
L	3	0	0	4	6	0	0	1	6	0	0	8	2	3	3	0	0	1	0	1	0	1	1	0	2	0
M	2	1	0	0	7	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
N	10	0	5	9	4	1	9	0	2	0	0	3	2	4	12	0	0	0	4	8	1	1	0	0	2	0
O	1	3	1	3	0	6	1	1	0	0	0	1	4	20	2	5	0	17	3	13	7	2	3	0	0	0
P	0	0	0	0	5	0	0	0	0	0	0	4	0	0	4	0	0	2	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
R	8	0	0	1	26	4	3	0	1	0	1	3	0	1	6	2	0	0	5	12	3	0	3	0	0	0
S	4	2	2	0	10	2	1	6	1	0	1	0	0	1	4	0	0	1	0	8	1	0	0	0	0	0
T	4	1	4	1	11	5	1	47	18	0	0	3	0	2	11	1	0	2	0	9	0	0	5	0	1	0
U	1	0	0	0	0	0	3	0	0	0	0	2	0	3	0	0	0	5	5	2	0	0	0	0	0	0
V	2	0	0	0	17	0	0	0	3	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
W	2	1	0	0	11	0	0	8	1	0	0	0	0	1	2	0	0	0	0	1	0	0	1	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y	2	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2: The digram frequency matrix for Lincoln's Gettysburg address.

In `2018.365.Final.Rmd`, you will find a 26×26 matrix G that represents the *digram frequency matrix* for Lincoln's Gettysburg address.[9] You can see this matrix in Figure 2 above. The ij entry of this matrix is g_{ij} = the number of times that the ith letter is followed immediately by the jth letter. Blank spaces and punctuation, if present, are ignored, and the first letter of the text is assumed to follow the last letter. For example, the pair "th" occurs 47 times in this text.

(a) (2 points) In `2018.365.Final.Rmd`, I have computed the vector $f = G1$, where 1 is the column vector of all ones. Describe what this vector tells us about the text.

(b) (3 points) Compute the SVD of the matrix G. Use it to give the numerical rank of G, call it $rankG$. Explain how the numerical rank "makes sense" from the given data.

(c) (4 points)

(i) Compute the best rank one approximation of G; i.e., find G_1 such that

$$||G - G_1||_2 = \inf_{B \in \mathbb{R}^{26\times 26} \text{ s.t. } \text{rank}(B) \leq 1} \{||G - B||_2\}.$$

Note: roughly speaking, the ij entry of G_1 is proportional to the product of the frequency of letter i with the frequency of letter j. It does not really account for the combinations/orders in which letters tend to appear most frequently.

(ii) Also compute the best rank two approximation of G; i.e., find G_2 such that

$$||G - G_2||_2 = \inf_{B \in \mathbb{R}^{26\times 26} \text{ s.t. } \text{rank}(B) \leq 2} \{||G - B||_2\}.$$

(d) (4 points) Computational linguists have noted that rank two approximations of digram frequency matrices help distinguish consonants from vowels.[10] Let $x = u_2$ and $y = v_2$ (where these are the u's and the v's from the SVD), and make a graph that plots the ith letter at the position (x_i, y_i). Describe how the vowels are grouped in this graph.[11]

(e) (4 points) This technique can be used in cryptography. Here is a simple example. In `2018.365.Final.Rmd`, I also give you the digram frequency matrix S from another text, but in this text the letters have been permuted (so as to secretly encrypt the message). Use the SVD to see if you can figure out which letters in this scrambled text are the vowels.

Hint: remember that if $U\Sigma V^T$ is an SVD of a matrix, then $(-U)\Sigma(-V^T)$ is also an SVD of the same matrix.

(f) **Extra credit** (2 points). Unscramble the following excerpt from the encrypted text and explain how you did it:

d lbyz b crzbt ulbu ti njvr xduuxz sldxcrzw hdxx jwz cbi xdyz dw b wbudjw hlzrz ulzi hdxx wju kz mvcozc ki ulz sjxjr jn ulzdr qadw kvu ki ulz sjwuzwu jn ulzdr slbrbsuzr

[9] Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal....

[10] One important point underlying the analysis is that vowels tend to follow consonants more often than other vowels.

[11] Note: the vowel "U" is a little harder to classify as it often follows other vowels, such as in the "OU" combination.

EC. (**Up to 5 more extra credit points**) Write a function called `myEigen` that uses the power iteration function `PI` in `365Functions.r` to compute ***all*** of the eigenvalues and eigenvectors of an $n \times n$ real, symmetric matrix A that has n eigenvalues with distinct magnitudes (i.e., $|\lambda_i| \neq |\lambda_j|$ if $i \neq j$).

- Your function should take the matrix A as input and output a list containing (i) an $n \times 1$ column vector called `vals` with the eigenvalues, and (ii) an $n \times n$ matrix `vecs` whose columns are the orthonormal eigenvectors of A
- Your function should call `PI()` multiple times
- **However, it should not call any other functions (i.e., no other functions from `R`'s built-in library such as `eigen`, `svd`, `solve` or `qr`, and no other functions from `365Functions.r` such as `IPI` or `RQI`); nor should you copy code from any of these functions into your function `myEigen`**
- Test your function on the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 6 & 5 \\ 3 & 6 & 1 & 7 \\ 4 & 5 & 7 & 0 \end{pmatrix}$$