University of Toronto
Department of Electrical and Computer Engineering

ECE367 MATRIX ALGEBRA AND OPTIMIZATION

Problem Set #2

Autumn 2020

Prof. S. C. Draper **Due:** 5pm (Toronto time) Friday, 09 October 2020

Homework policy: Problem sets must be turned by the due date and time. Late problem sets will receive deductions for lateness. See the information sheet for futher details. The course text "Optimization Models" is abbreviated as "OptM". Also, see PS01 for details of the "Non-graded", "graded" and "optional" problem categories.

Problem Set #2 problem categories: A quick categorization by topic of the problems in this problem set is as follows:

Non-graded:

- Taylor approximations and hyperplanes: Problems 2.1, 2.2
- Matrices as linear maps: Problems 2.3-2.5
- Eigenvalues and vectors: Problems 2.6

Optional non-graded:

- More on linear maps: Problem 2.7

Graded:

- Taylor approximations: Problems 2.8
- Eigenvalues and vectors: Problems 2.9

NON-GRADED PROBLEMS

Problem 2.1 (Taylor series expansion)
Consider the function $f(x) = -\sum_{l=1}^{m} \log(b_l - a_l^T x)$, where $x \in \mathbb{R}^n$, $b_l \in \mathbb{R}$ and $a_l \in \mathbb{R}^n$. Compute $\nabla f(x)$ and $\nabla^2 f(x)$. Write down the first three terms of the Taylor series expansion of $f(x)$ around some x_0.

Problem 2.2 (Distance between a pair of parallel hyperplanes)
Find the distance between the two parallel hyperplanes $\mathcal{H}_i$, $i \in [2]$ where $\mathcal{H}_i = \{x \in \mathbb{R}^n | a^T x = b_i\}$. Your solution should be expressed in terms of the problem parameters, i.e., the vector $a \in \mathbb{R}^n$ and the scalars $b_i \in \mathbb{R}$. (Note: this problem is from "Convex Optimization" by Boyd and Vandenberghe.)

Problem 2.3 (Practice computing rank, range, nullspaces)

(a) Find rank(A) and the dimension of, and a basis for, each of the four subspaces $\mathcal{R}(A)$, $\mathcal{R}(A^T)$, $\mathcal{N}(A)$, $\mathcal{N}(A^T)$ when

$$A = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

(b) Find rank(B) and the dimension of, and a basis for, each of the four subspaces $\mathcal{R}(B)$, $\mathcal{R}(B^T)$, $\mathcal{N}(B)$, $\mathcal{N}(B^T)$ when

$$B = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{bmatrix}.$$

(b) Find rank(C) and the dimension of, and a basis for, each of the four subspaces $\mathcal{R}(C)$, $\mathcal{R}(C^T)$, $\mathcal{N}(C)$, $\mathcal{N}(C^T)$ when

$$C = \begin{bmatrix} 1 & 2 & 1 & 3 \\ 3 & 4 & 6 & 9 \\ 1 & 4 & 4 & 4 \\ 1 & 0 & 10 & 4 \end{bmatrix}.$$

Problem 2.4 (Rank and nullspace)
OptM Problem 3.6.

Problem 2.5 (Linear dynamical systems)
OptM Problem 3.4.

Problem 2.6 (Practice computing eigenvalues and eigenvectors)

In this problem you consider the eigenvalues and eigenvectors of each of the following four matrices:

(a) $A = \begin{bmatrix} 3 & 1 \\ 2 & 2 \end{bmatrix}$, (b) $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, (c) $A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$, (d) $A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$.

For each of the matrices in parts (a)-(d) do the following:

(i) compute the eigenvalues of the matrix,

(ii) compute eigenvectors of the matrix,

(iii) specify both the algebraic and geometric multiplicity of each distinct eigenvalue, and

(iv) *if* the matrix is diagonalizable express the matrix in its diagonalized form. In other words, if A is diagonalizable express it as $A = V\Lambda V^{-1}$ where V is the matrix of eigenvectors and Λ is a diagonal matrix of eigenvalues. Specify V and Λ for each matrix that is diagonalizable.

OPTIONAL, NON-GRADED PROBLEMS

Problem 2.7 (The matrix of a linear map)
In the setup of this problem we prove that *any* linear map between vector spaces can be represented by a matrix. You will then extend that reason to show that any linear function on a vector space can be represented by a vector and you will show that the way matrix-matrix multiplication is defined follows from the concatenation of any pair of compatible linear maps.

In this problem we work with a triplet of vector spaces $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{U}$ where $\dim(\mathcal{V}) = n$, $\dim(\mathcal{W}) = m$, and $\dim(\mathcal{U}) = p$. Let $\{v^{(i)}\}_{i\in[n]} = \text{basis}(\mathcal{V})$, $\{w^{(i)}\}_{i\in[m]} = \text{basis}(\mathcal{W})$, and $\{u^{(i)}\}_{i\in[p]} = \text{basis}(\mathcal{U})$. We first recall the definition of a linear map, e.g,. from $\mathcal{V}$ to $\mathcal{W}$.

Definition 1. *A map $f : \mathcal{V} \to \mathcal{W}$ is "linear" if for any two points $x^{(i)} \in \mathcal{V}$, $i \in [2]$, and scalings α_i, $i \in [2]$*

$$f(\alpha_1 x^{(1)} + \alpha_2 x^{(2)}) = \alpha_1 f(x^{(1)}) + \alpha_2 f(x^{(2)}).$$

To introduce you to the perspectives we need in this problem we now show that *any* linear map f can be fully specified by an $m \times n$ matrix of coefficients. First note that any $x \in \mathcal{V}$ can be expressed in terms the the basis elements of $\mathcal{V}$ as $x = \sum_{i\in[n]} \alpha_i v^{(i)}$ for some choice of the α_i. Next, we have

$$f(x) = f\left(\sum_{i=1}^{n} \alpha_i v^{(i)}\right) \overset{(i)}{=} \sum_{i=1}^{n} \alpha_i f(v^{(i)}) \tag{1}$$

$$\overset{(ii)}{=} \sum_{i=1}^{n} \alpha_i \left[\sum_{k=1}^{m} \beta_k^{(i)} w^{(k)}\right], \tag{2}$$

where (i) follows by the linearity of f and the $\beta_k^{(i)}$ in (ii) are the coefficients associated with the basis elements of $\mathcal{W}$ that represent $f(v^{(i)}) \in \mathcal{W}$, the point in $\mathcal{W}$ to which the ith basis element of $\mathcal{V}$ maps. (Note that this point need not itself be a basis element of $\mathcal{W}$.) Expanding this out in matrix form we find that

$$f(x) = \left[w^{(1)} w^{(2)} \dots w^{(m)}\right] \begin{bmatrix} \beta_1^{(1)} & \beta_1^{(2)} & \dots & \beta_1^{(n)} \\ \beta_2^{(1)} & \beta_2^{(2)} & \dots & \beta_2^{(n)} \\ \vdots & & \ddots & \vdots \\ \beta_m^{(1)} & \beta_m^{(2)} & \dots & \beta_m^{(n)} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}. \tag{3}$$

Hence, any linear operator f can be represented by a matrix that relates the basis of the input space to the basis of the output space. If one changes the linear operator then, naturally, the matrix changes. However, one should also note that if one changes either the basis that represents the input space or the basis that represents the output space the matrix will also change. Thus the representation of a linear map by a matrix is not dependent solely on f but *also* on the bases used to describe points in the input and output spaces. To make this dependence explicit, one therefore denotes the matrix of the linear map f as $\mathcal{M}(f, \{v^{(1)}, \dots v^{(n)}\}, \{w^{(1)}, \dots, w^{(m)}\})$. Most of the time one writes down a matrix the bases are not specified which (typically) means that the standard basis is being assumed. Indeed, often when introducing matrices in linear algebra it is not even made explicit that one is using the standard basis to parameterize the input and output spaces of the mapping that the matrix describes. The problem framing above makes that choice explicit and, of course, and one need not have made that (perhaps unknowing) assumption.

The above was a bit hard to work into a problem for you to solve, so we simply provided the framework, derivation, and discussion. Now for the work. In the next part you consider linear functions (rather than maps) and then extend the above logic and see what happens when you concatenate a pair of linear maps.

(a) Based on the logic above argue why any linear function can be represented by a vector. (Hint: Yes, this part is as easy as it seems.)

(b) Now, let $f : \mathcal{V} \to \mathcal{W}$ be as in the problem introduction. In addition let $g : \mathcal{W} \to \mathcal{U}$. Following the same logic as in the problem introduction, find an expression for $g(f(x))$ of the form $g(f(x)) = \sum_{l=1}^{p} \zeta_l u^{(l)}$. (Note that $g(f(x)) \in \mathcal{U}$ and $\{u^{(1)}, \dots u^{(p)}\}$ is a basis for $\mathcal{U}$.) (Hint: Your expression should contain three sums, one of n terms, one of m terms, and one of p terms.)

(c) Look at your expressions for the ζ_l from part (b) and rewrite in a form that involves two vectors and two matrices. You should now see why matrix-matrix multiplication is defined in the way that it is.

GRADED PROBLEMS

Problem 2.8 (Second-order approximation of functions)
In this exercise you extend your code from the problem "First-order approximation of functions" to second order. (As before you are welcome to use Matlab or Python or the software of your choice.) The objective is, as before, to write code to plot approximations each of (the same) three functions $f_i : \mathbb{R}^2 \to \mathbb{R}$, $i \in [3]$, but now the approximation is quadratic rather than linear. To recall, the three functions are defined pointwise as

$$\begin{aligned} f_1(x,y) &= 2x + 3y + 1, \\ f_2(x,y) &= x^2 + y^2 - xy - 5, \\ f_3(x,y) &= (x-5)\cos(y-5) - (y-5)\sin(x-5). \end{aligned}$$

For each of the above function, do the following.

(a) Write down the gradient and Hessian with respect to x and y in closed form. (You don't need to rederive the gradient as you did that last time. To recall the gradient and Hessian are compactly denoted as ∇f_i and $\nabla^2 f_i$ for $i \in [3]$ where

$$\nabla f_i = \begin{bmatrix} \frac{\partial f_i}{\partial x} \\ \frac{\partial f_i}{\partial y} \end{bmatrix} \quad \nabla^2 f_i = \begin{bmatrix} \frac{\partial^2 f_i}{\partial^2 x} & \frac{\partial^2 f_i}{\partial x \partial y} \\ \frac{\partial^2 f_i}{\partial y \partial x} & \frac{\partial^2 f_i}{\partial^2 y} \end{bmatrix}.$$

(b) For each function produce do the following two things. First, produce a 2-D contour plot indicating the level sets of each function in the range $-2 \leq x, y \leq 3.5$ and plot the direction of the gradient and tanget to the level set at the point (x,y) = 1,0). (Note, you have already produced these plots in the previous problem, we are just reproducing them here to help with visualization). Second, For the same point $(x, y) = (1, 0)$ plot the 3-D quadratic approximation of the function.

(c) Now, repeat the previous plot for point $(x, y) = (-.7, 2)$ and $(x, y) = (2.5, -1)$.

(d) Comment on where your approximations are accurate and where they are not (if anywhere) for the three functions. Discuss what the reason is behind your observations.

Note: We recommend you design these plotting scripts as Matlab functions so that you can reuse them for to plot approximations for different non-linear functions (or for theses functions at different points). In either case make sure to attach your code.

Problem 2.9 (Google's PageRank algorithm)
In this problem you will implement and analyse approaches to the eigenvector computation that is at the heart of Google's PageRank algorithm (cf. discussion in Example 7.1 of OptM). Download `pagerank_urls.txt` and `pagerank_adj.mat` files from the course website. The first is a plain text

file in which each line consists of a URL of a web page. We refer to the web pages by their respective URL-line numbers starting from 1. For example, web page 9 is the page that the URL in line 9 points to. The URLs are of the internal web pages of Hollins University in Roanoke, Virginia. The provided data files consist of a modified version of web crawling data downloaded from `https://www.limfinity.com/ir`. The original dataset has been created in January, 2004 therefore you might notice that some of the links are inactive.

Let N be the number of URLs (therefore the number of web pages) in the first file and $i, j \in [N]$. Execute the command `load 'pagerank_adj.mat'` in MATLAB to load the content of the second file into your MATLAB workspace. You will see that a new $N \times N$ variable `J` has been imported. This variable represents an *adjacency matrix* $J \in \{0,1\}^{N \times N}$ which describes the relationships between the web pages. Specifically, the element in ith row and jth column $J_{i,j} = 1$ if there exists a link from web page j to web page i, and $J_{i,j} = 0$ otherwise. Data have been carefully filtered so that $J_{j,j} = 0$ and $\sum_{i=1}^{N} J_{i,j} > 0$ for all $j \in [N]$. In other words, we do not allow links from a web page to itself, and we do not allow for dangling pages, that is pages with no outgoing links.

Use J to obtain the *link matrix* A where

$$A_{i,j} = \frac{J_{i,j}}{\sum_{k=1}^{N} J_{k,j}}.$$

Also, let $x \in \mathbb{R}^N$ be a vector with all entries equal to 1. Use the matrix A and the vector x the same way as described in Example 7.1 to solve the following.

(a) Verify that each column in the provided matrix A sum to 1. What is the importance of this property for the Google PageRank algorithm?

(b) In the following we are consistent with the notation used in OptM. Let $x(k+1)$ be the approximation of the eigenvector in the $k+1$th iteration. We define the approximation error in the $k+1$th iteration as $e(k+1) = \|Ax(k+1) - x(k+1)\|_2$. Using MATLAB, implement the power iteration algorithm described in Section 7.1.1 in OptM. Run the algorithm for 10 iterations and plot $\log(e(k+1))$ versus k.

(c) Implement the shift-invert power iteration and Rayleigh quotient iteration algorithms presented in Sections 7.1.2 and 7.1.3 of OptM. For the shift-invert power method use $\sigma = 0.99$. In the Rayleigh quotient iteration method, use $\sigma_1 = \sigma_2 = 0.99$ for the first two iterations and $\sigma_k = \frac{x^*(k)Ax(k)}{x^*(k)x(k)}$ for $k > 2$, in a similar manner as the discussion in Example 7.1. Repeat your experiment of part (b) for these two algorithms. Plot $\log(e(k+1))$ for each of your three algorithms on a single plot. Check whether your results are consistent with Example 7.1. Include your MATLAB code with the answers.

(d) List the (page index, PageRank score) pairs of the top 5 and bottom 5 pages according to your PageRank scores. Compare them with the provided web page links and briefly explain whether the ranking seem intuitively correct.

(Note: While we write "MATLAB" in the above you are, as usual, welcome to use any software you would like to, just not to use built-in functions that accomplish the objective.)

Problem 2.8

a.

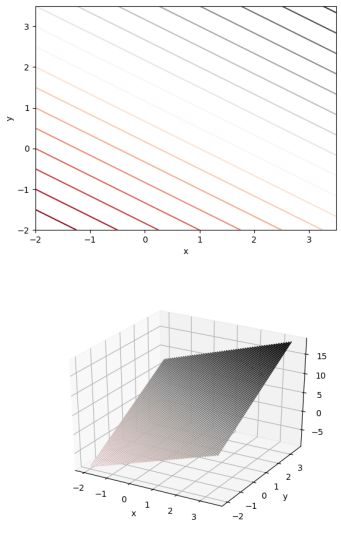
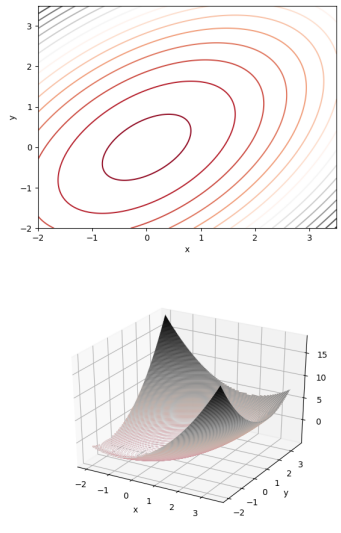
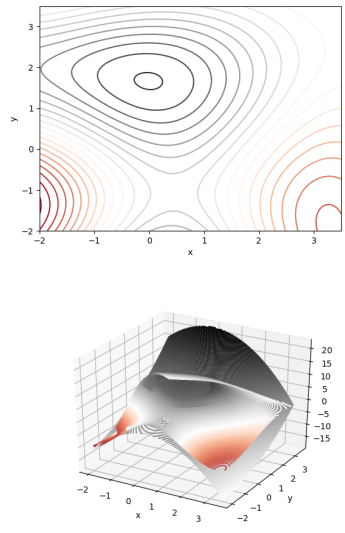
a) $\nabla f_1 = [2 \quad 3]^T$ $\quad \nabla^2 f_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

$\nabla f_2 = [2x - y \quad 2y - x]^T$ $\quad \nabla^2 f_2 = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$

$\nabla f_3 = [\cos(y-5) - (y-5)\cos(x-5) \quad (5-x)\sin(y-5) - \sin(x-5)]^T$

$\nabla^2 f_3 = \begin{bmatrix} (y-5)\sin(x-5) & -\sin(y-5) - \cos(x-5) \\ -\sin(y-5) - \cos(x-5) & (5-x)\cos(y-5) \end{bmatrix}$

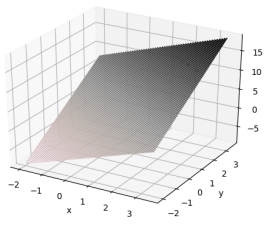
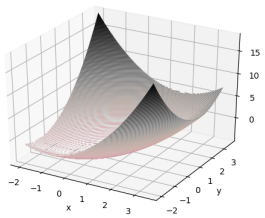
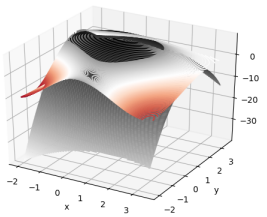
b.

f1	f2	f3
		

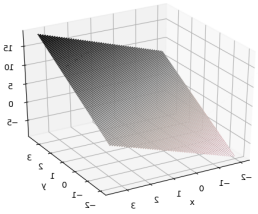
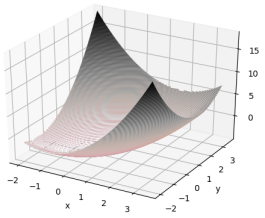
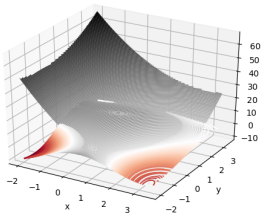
c.

(x, y) = (-0.7, 2)

f1	f2	f3

(x, y) = (2.5, -1)

f1	f2	f3
		

d. The approximations are identical to the actual graphs for both f1 and f2 since the approximations are second order and the functions are either second order or less. For f3, the approximation does a better job of predicting localized values at the point of interest, however, fails to accurately represent the graph as you move farther away from that point, which can be expected since the function is trigonometric.

Python Code:

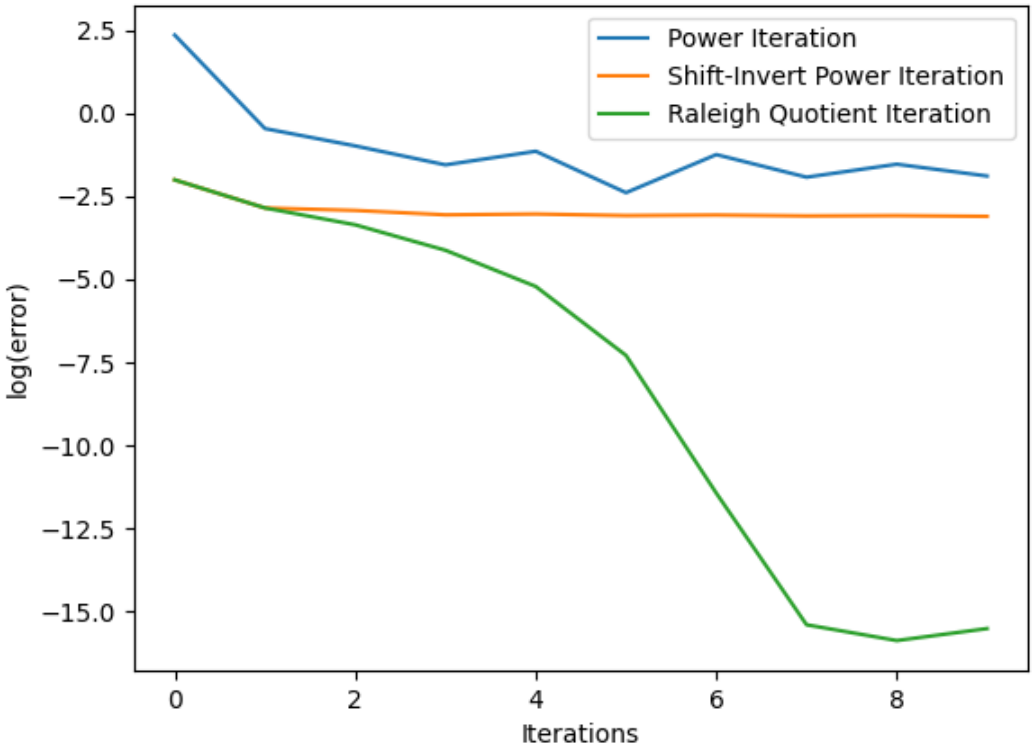
```
import numpy as np
import matplotlib.pyplot as plt

from sympy import *

x_s = Symbol('x')
y_s = Symbol('y')

f1_s = 2*x_s + 3*y_s + 1
f2_s = x_s**2 + y_s**2 - x_s*y_s - 5
f3_s = (x_s-5)*cos(y_s-5) - (y_s-5)*sin(x_s-5)

def f1(x,y):
    return 2*x + 3*y + 1

def f2(x,y):
    return x**2 + y**2 - x*y - 5

def f3(x,y):
    return (x-5)*np.cos(y-5) - (y-5)*np.sin(x-5)
```

```
x = np.linspace(-2,3.5,200)
y = np.linspace(-2,3.5,200)

X, Y = np.meshgrid(x,y)
Z1 = f1(X,Y)
Z2 = f2(X,Y)
Z3 = f3(X,Y)

def derivative(f,arg1):
    diff = f.diff(arg1)
    return diff

def derivative_2(f,arg1,arg2):
    diff = f.diff(arg1, arg2)
    return diff

def quadratic(f,x0,y0):
    f_x = derivative(f,x_s)
    f_y = derivative(f,y_s)
    f_xx = derivative_2(f,x_s,x_s)
    f_xy = derivative_2(f,x_s,y_s)
    f_yy = derivative_2(f,y_s,y_s)

    f = lambdify([x_s, y_s], f)
    f_x = lambdify([x_s, y_s], f_x)
    f_y = lambdify([x_s, y_s], f_y)
    f_xx = lambdify([x_s, y_s], f_xx)
    f_xy = lambdify([x_s, y_s], f_xy)
    f_yy = lambdify([x_s, y_s], f_yy)

    final = f(x0,y0) + f_x(x0,y0)*(x_s-x0) + f_y(x0,y0)*(y_s-y0) + (f_xx(x0,y0)*(x_s-
x0)*(x_s-x0))/2 + f_xy(x0,y0)*(x_s-x0)*(y_s-y0) + (f_yy(x0,y0)*(y_s-y0)*(y_s-y0))/2
    return final

x0 = 2.5
y0 = -1

f1_prime = quadratic(f1_s,x0,y0)
f1_prime = lambdify([x_s,y_s], f1_prime)
Z1_prime = f1_prime(X,Y)

f2_prime = quadratic(f2_s,x0,y0)
f2_prime = lambdify([x_s,y_s], f2_prime)
Z2_prime = f2_prime(X,Y)

f3_prime = quadratic(f3_s,x0,y0)
f3_prime = lambdify([x_s,y_s], f3_prime)
```

```
Z3_prime = f3_prime(X,Y)


fig = plt.figure()
ax = plt.axes(projection = '3d')
ax.contour3D(X,Y,Z1,150,cmap='RdGy')
ax.contour3D(X,Y,Z1_prime,150,cmap='binary')
plt.xlabel('x')
plt.ylabel('y')
plt.show()

fig = plt.figure()
ax = plt.axes(projection = '3d')
ax.contour3D(X,Y,Z2,150,cmap='RdGy')
ax.contour3D(X,Y,Z2_prime,150,cmap='binary')
plt.xlabel('x')
plt.ylabel('y')
plt.show()

fig = plt.figure()
ax = plt.axes(projection = '3d')
ax.contour3D(X,Y,Z3,150,cmap='RdGy')
ax.contour3D(X,Y,Z3_prime,150,cmap='binary')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Problem 2.9

a. By manipulating the matrices, it can be shown that each column sums up to 1. This is an important property for the PageRank algorithm since the final output of the function should be probabilities of being at a certain page. Therefore, each entry should be a percentage/fraction representing the probability of traveling to that specific website based on the column.
b. Code below



c.

d. Rank 1 is website #730 with a score of 0.18588265057984998
Rank 2 is website #731 with a score of 0.18588265057984998
Rank 3 is website #735 with a score of 0.18588265057984998
Rank 4 is website #736 with a score of 0.18588265057984998
Rank 5 is website #738 with a score of 0.18588265057984998
Rank 2570 is website #1 with a score of 4.4170189632794063e-75
Rank 2569 is website #48 with a score of 4.4170189632794063e-75
Rank 2568 is website #5 with a score of 6.428689430104313e-75
Rank 2567 is website #3 with a score of 6.428689430104315e-75
Rank 2566 is website #4 with a score of 6.428689430104315e-75

These results are based on the results of x from the Raleigh Quotient Iteration.

These results can also be normalized to represent actual percentages by dividing each element by the total sum of the elements in vector x.
Rank 1 is website #730 with a score of 0.026372816488283404
Rank 2 is website #731 with a score of 0.026372816488283404
Rank 3 is website #735 with a score of 0.026372816488283404
Rank 4 is website #736 with a score of 0.026372816488283404
Rank 5 is website #738 with a score of 0.026372816488283404
Rank 2570 is website #1 with a score of 6.266815659258909e-76
Rank 2569 is website #48 with a score of 6.266815659258909e-76
Rank 2568 is website #5 with a score of 9.120950560551506e-76

Rank 2567 is website #3 with a score of 9.12095056055151e-76
Rank 2566 is website #4 with a score of 9.12095056055151e-76

The rankings do seem intuitively correct. As shown, the first few websites all have a relatively low ranking, and this is expected because they represent either the home page or early links on the home page. Their low ranking is expected because they have relatively few incoming links because they represent home pages and early links. Websites in the range of 730 to 740 have very long URL's which represent that they are deep within the webpage. This often means that they are much more highly probable to be on as the time goes to infinity because they have a very high number of incoming links with few outgoing since they are so deep in the chain of URL's. Therefore, the behaviour of the models does seem intuitively correct.

Python Code:

```
import numpy as np
import math
import matplotlib.pyplot as plt

J = np.loadtxt('data.csv', delimiter=',')
A = np.zeros((len(J[0]), len(J[0])))

x = np.ones((len(J[0]),1))

for i in range(0, len(J[0]), 1):
    col = 0
    for j in range(0, len(J[0]), 1):
        col = col + J[j,i]
    for j in range(0, len(J[0]), 1):
        A[j,i] = J[j,i] / col

#for i in range(0, len(J[0]), 1):
#    col = 0
#    for j in range(0, len(J[0]), 1):
#        col = col + A[j,i]
#    print("Sum of column " + str(i) + ": " + str(col))

# POWER ITERATION METHOD
error = []
x_1 = x
for i in range(0, 10, 1):
    y_1 = np.matmul(A, x_1)
    x_1 = y_1 / np.linalg.norm(y_1)
    e_1 = np.linalg.norm(y_1 - x_1)
    error.append(math.log(e_1,10))

# SHIFT-INVERT POWER ITERATION
error_si = []
```

```
x_1 = x
I = np.identity(len(J[0]))
sigma = 0.99
inv = np.linalg.inv(A - sigma*I)

for i in range(0, 10, 1):
    y_1 = np.matmul(inv, x_1)
    x_1 = y_1 / np.linalg.norm(y_1)
    e_1 = np.linalg.norm(np.matmul(A, x_1) - x_1)
    error_si.append(math.log(e_1,10))

# RAYLEIGH QUOTIENT ITERATION
error_r = []
x_1 = x
sigma = 0.99
for i in range(0,10, 1):
    if i > 1:
        x_T = np.matrix.transpose(x_1)
        top = np.matmul(x_T, A)
        top = np.matmul(top, x_1)
        bottom = np.matmul(x_T, x_1)
        sigma = top / bottom
    inv = np.linalg.inv(A - sigma*I)
    y_1 = np.matmul(inv, x_1)
    x_1 = y_1 / np.linalg.norm(y_1)
    e_1 = np.linalg.norm(np.matmul(A, x_1) - x_1)
    error_r.append(math.log(e_1,10))

plt.plot(error, label = "Power Iteration")
plt.plot(error_si, label = "Shift-Invert Power Iteration")
plt.plot(error_r, label = "Raleigh Quotient Iteration")
plt.legend()
plt.xlabel("Iterations")
plt.ylabel("log(error)")
plt.show()

x = x_1.tolist()
sum = 0
for i in range(0, len(x),1):
    sum = sum + x[j][0]

for i in range(1, 6, 1):
    max_idx = 0
    for j in range(0, len(x), 1):
        if x[j][0] > x[max_idx][0]:
            max_idx = j
    print("Rank " + str(i) + " is website #" + str(max_idx + 1) + " with a score of "
+ str(x[max_idx][0]/sum))
```

```
    x[max_idx] = [0]

x = x_1.tolist()

for i in range(1, 6, 1):
    min_idx = 0
    for j in range(0, len(x), 1):
        if x[j][0] < x[min_idx][0]:
            min_idx = j
    print("Rank " + str(len(x) - i) + " is website #" + str(min_idx + 1) + " with a
score of " + str(x[min_idx][0]/sum))
    x[min_idx] = [1]
```