

# Lab 5

## Contents

1. [Objectives](#)
2. [Pre-lab Reading](#)
3. [Pre-lab Reading: Google PageRank Algorithm](#)
4. [Pre-lab Reading: MATLAB](#)
5. [In-Lab Exercises](#)
6. [Take-home Assignment](#)
7. [Deliverables](#)
8. [Helpful Hints](#)

## 1. Objectives for this lab

1. Recognize the utility of using different programming tools to solve different problems
2. Appreciate the relative strengths and weaknesses of programming in compiled vs scripting languages
3. Gain familiar and programming experience with MATLAB, a computational tool commonly used in scientific/engineering careers
4. Develop an understanding of matrices as a fundamental data-storage unit
5. Gain continuing experience with pointers and their tradeoffs and risks (dangling pointers, memory leaks)
6. Learn about the Google PageRank algorithm and how to implement a simple version in MATLAB
7. Gain experience using a nonstandard C library (the MathWorks MATLAB C library)
8. Write and execute a program written in C which invokes the MATLAB engine to process data.

## 2. Pre-lab reading

TO-DO The Pre-lab section of the lab summarizes the preparations you need to make **prior** to each lab, so you are ready to begin when you arrive at the start of your lab session. It is important to be prepared for the lab because you will be working in pairs.

DO You should always read the **entire** lab prior to your lab session, but often there are other things you will need to do in addition to just reading the description. Look for the TO-DO for the major tasks.

## I guess this is goodbye (almost)

Congratulations! You've made it to the final lab. It hasn't always been easy, but we hope you've had fun.

Your final CPSC 259 lab will investigate two very interesting topics: the **Google PageRank algorithm**, and how to implement it using **MATLAB**. MATLAB is a programming language and a programming environment in one, and it's a lot of fun to use. Many of you have used MATLAB already (like in MATH 152), but this lab assumes that you have never seen it. We'll take you on a whirlwind tour of MATLAB, and teach you just enough to implement PageRank.


This take-home lab is different from the others because there is no code framework. You'll write your own program from scratch. That's right, we're taking off the training wheels. You're ready. You and your partner will **write a short C program that invokes the MATLAB engine to calculate the PageRank** for a given 'web.' You will apply what you have learned during class and the first 4 labs to implement most of the C portion of your assignment, and this lab will teach you how to use MATLAB to perform the math, and how to invoke MATLAB from a C program running inside Visual Studio. Sound interesting? Read on!

Note: This lab seems long, but there are many, many screenshots. There's not as much reading as you might think, but the pre-reading contains some linear algebra, so go make a cup of tea and tuck in.

In this lab, you will learn how to:

1. Model the Google PageRank algorithm using introductory linear (matrix-based) algebra
2. Solve the Google PageRank algorithm using MATLAB
3. Configure a new Visual Studio project so it can communicate with the MATLAB engine
4. Write a simple C program that sends raw data to MATLAB, asks MATLAB to analyze the data, and receives the solution.

## To C, or not to C, that is the question

A [wise man](http://en.wikipedia.org/wiki/Law_of_the_instrument)  ([http://en.wikipedia.org/wiki/Law\\_of\\_the\\_instrument](http://en.wikipedia.org/wiki/Law_of_the_instrument)) once said, "When all you have is a hammer, everything looks like a nail." This term we have used C to implement software solutions to some interesting problems in fields like bioinformatics and aviation. As fantastic as C is, it's not the only tool you want to have hanging from your programming tool belt.

When all you have is imperative languages like C or Java (another popular programming language) everything looks like a problem to be solved by moving values in and out of memory locations. This isn't always the most efficient way to do things. Sometimes we want to **abstract away the implementation details** and map a solution onto a collection of math-like functions that someone else has written. The gory details of the typical computer architecture (pointers, variables etc) are abstracted away, leading to greater clarity in the problem solving process and more elegant solutions.

Enter your newest tool: MATLAB. MATLAB is a high-level interactive environment and language which lets us perform computationally intensive tasks faster than we can with languages such as C or Java. MATLAB can be used by itself directly, or **we can write C programs that invoke the MATLAB engine to process data that we send it**. In this lab, we'll be doing both, but focus more on the latter.

Let's get started. We'll start by introducing a simplified version of the Google PageRank algorithm and the math behind it, and then we'll dive right in and show you how to use MATLAB. In this lab we are going to use MATLAB and the power method to solve a simple matrix-based dynamical system and, specifically, to **implement a simplified version of Google's PageRank algorithm**. You should recognize the math from your linear algebra courses.

### 3. Google PageRank Algorithm

Here's an **implementation-free description** of a simplified version of the Google PageRank algorithm. This is the version of the PageRank algorithm which you will use in your in-lab and take-home assignments. In order to understand the algorithm, you should be familiar with basic matrix operations and definitions from your linear algebra class. Don't be alarmed if you haven't taken algebra in a while, this is surprisingly and pleasantly easy:

1. Let **W** be a set of webpages (a web) and let the set have size  $n$ .
2. Let **G** be the square  $n$  by  $n$  **connectivity matrix** for the web where:

$g_{ij} = 1$  if there is a hyperlink from page  $j$  to page  $i$ , where  $i$  is the row and  $j$  is the column, and

$g_{ij} = 0$  otherwise

For example, suppose we have a tiny web of 4 pages: A, B, C, and D, and apart from page D which has no links to it or from it, each page contains one link to each of the other pages, but not to itself. That is, page A contains a link to page B, and to page C,

but not to page A or to page D. Page B contains a link to page A, and to page C, but not to page B or page D. And so on. We say that the **jth column of the connectivity matrix contains the outbound links from page j**. Our connectivity matrix will look like this :

|     |   |   |   |   |
|-----|---|---|---|---|
| G = | A | B | C | D |
| A   | 0 | 1 | 1 | 0 |
| B   | 1 | 0 | 1 | 0 |
| C   | 1 | 1 | 0 | 0 |
| D   | 0 | 0 | 0 | 0 |

- Let's **define the importance of each page by the importance of the pages that link to it**. (Hey wait, that sounds a little like recursion!). If we do this, we can use the connectivity matrix as a launching pad for determining the relative importance of each page in our web. Note that a typical connectivity matrix **G** is probably very **big** (how many billions of web pages are there these days?), but very **sparse** (lots of zeros). Its jth column shows the links on the jth page, and the total number of nonzero entries in the entire matrix is the total number of links in our web **W**. In fact, we can define  $r_i$  and  $c_j$  to be the row and column sums of **G** respectively. These quantities are the **in-degree** and **out-degree**. The in-degree is the number of pages that have links to our ith page, and the out-degree is the number of links on our jth page to other pages.
- Mathematically-speaking, the "**importance**" of page i ( $x_i$ ) equals the sum over all pages j that link to i of the importance of each page j divided by the number of links in page j:

$$x_i = \sum_{j \in W} \frac{1}{N_j} x_j$$

- We can modify our connectivity matrix to show us this "importance" if we **divide each value in each column by the sum of each column**. Since it is a connectivity matrix, the values the cells are either 0 or 1, so each cell containing a 1 is divided by the number of 1s in the column. For example, in our connectivity matrix G, the sum in the first column (column A) is 2, so we divide each element by 2 to get 1/2. We can observe that every non-zero column now adds up to 1. Let **S** be the matrix constructed according to this rule:

|     |     |     |     |   |
|-----|-----|-----|-----|---|
| S = | A   | B   | C   | D |
| A   | 0   | 0.5 | 0.5 | 0 |
| B   | 0.5 | 0   | 0.5 | 0 |
| C   | 0.5 | 0.5 | 0   | 0 |
| D   | 0   | 0   | 0   | 0 |

- In matrix **S**, the value in **S[i][j]** is the "**probability**" of going from page j to page i. Note that every non-zero column adds up to 1, but we have that pesky final column of zeros to worry about. There are no links to page D, and there are no links away from page

D. You may remember from your linear algebra that we have to do something about this, otherwise page D is going to end up at the bottom of the PageRank no matter how interesting its contents (and if you don't remember this, trust us!). We want to replace every column of zeros with a column whose elements equal  $1/n$  (recall  $n$  is the dimension of our square connectivity matrix):

$$S = \begin{array}{ccccc} & A & B & C & D \\ A & 0 & 0.5 & 0.5 & 0.25 \\ B & 0.5 & 0 & 0.5 & 0.25 \\ C & 0.5 & 0.5 & 0 & 0.25 \\ D & 0 & 0 & 0 & 0.25 \end{array}$$

7. Are you still with us? Good! The matrix **S** is now a **stochastic matrix** [⌨\(http://en.wikipedia.org/wiki/Stochastic\\_matrix\)](http://en.wikipedia.org/wiki/Stochastic_matrix), or to be more specific, a left stochastic matrix because the columns all sum to 1. In a **left stochastic matrix**, the elements are all strictly between 0 and 1 and its columns all sum to 1. We're almost finished. We can also call **S** a **probability matrix**, and we will use our **probability matrix** to construct a **transition matrix** for a **Markov process**. This sounds much more complicated than it really is. There are two steps:

1. We need to introduce the notion of a random walk. We need to multiply our probability matrix by a **random walk probability factor**. For our lab, we will designate this variable **p**, and set **p = 0.85**.
2. If we multiply our probability matrix by **p**, then you might think we need to add something to compensate for the overall reduction in value. You're right. We need to add **(1 - p)** multiplied by a rank 1 matrix **Q**, whose every element =  $1 / n$ . (recall that a rank 1 matrix has a single linearly independent column). Let's call the **transition matrix** that results **M**:

$$M = 0.85 * S + (1 - 0.85) * Q = 0.85 * S + (1 - 0.85) * \begin{array}{cccc} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{array} = \begin{array}{cccc} 0.0375 & 0.4625 & 0.4625 & 0.2500 \\ 0.4625 & 0.0375 & 0.4625 & 0.2500 \\ 0.4625 & 0.4625 & 0.0375 & 0.2500 \\ 0.0375 & 0.0375 & 0.0375 & 0.2500 \end{array}$$

8. The rest is easy. Let's create a column vector (or we can call it an array) of length  $n$  called **rank**. We'll initialize each element to 1:

$$\text{rank} = \begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \end{array}$$

9. The final 2 steps are the Markov process (aka the dynamical system aka the power method):

1. Multiply the transition matrix **M** by our column vector (array) **rank**, and then multiply **M** by the result and then keep doing this until the and rank stops changing (**result converges**), e.g.,  $M * \text{rank} = \text{rank}$ . In this case, we get:

$$\text{rank} = \begin{array}{c} 1.2698 \\ 1.2698 \end{array}$$

```
1.2698
0.1905
```

2. Divide each element in rank by the sum of the values in rank (scale rank so its elements sum to 1):

```
rank = 1.2698 / 3.999 = 0.3175
       1.2698 / 3.999    0.3175
       1.2698 / 3.999    0.3175
       0.1905 / 3.999    0.0476
```

**And that's all there is to it!** We've ranked the pages in our little web. The first row in the rank corresponds to the first column of our transition matrix, the second row to the second column, and so on.

The result makes intuitive sense. Each of pages A, B, and C has a rank of about 32%, and page D ranks fourth with about 5%. Keeping in mind that we haven't considered how a user's query will affect the rank, you now have a basic understanding of how Google's PageRank works.

In the next section, we will show you how to do this in MATLAB. We'll also teach you how to send the instructions (and data!) from your C program operating in Visual Studio to MATLAB, and how to accept and display the result on the command line using your C program.

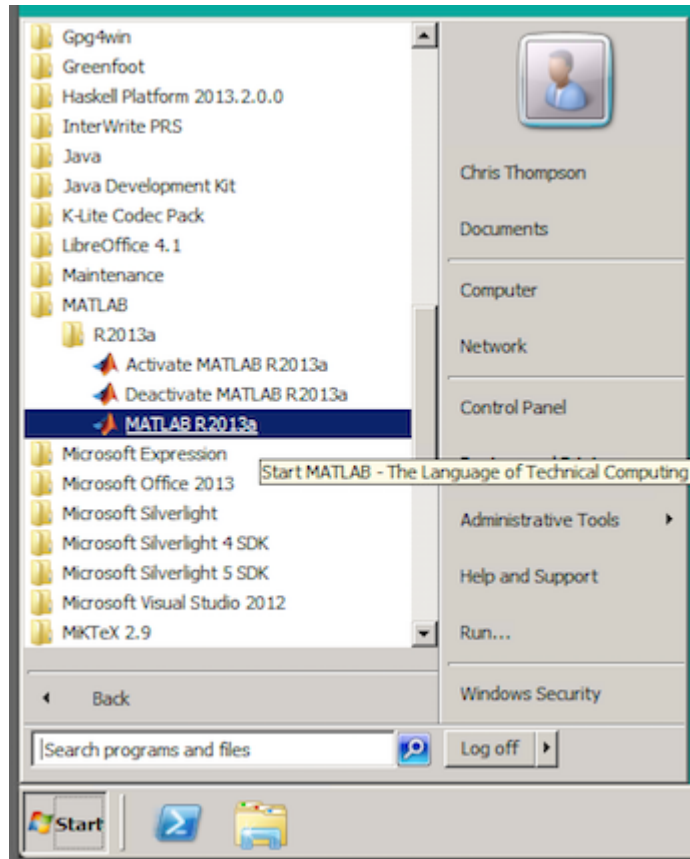
## 4. MATLAB

Before you begin, please download and install MATLAB. UBC provides you with a license to use the software. See [MATLAB | UBC Information Technology \(https://it.ubc.ca/services/desktop-print-services/software-licensing/matlab\)](https://it.ubc.ca/services/desktop-print-services/software-licensing/matlab) for details about acquiring and licensing MATLAB for your personal machine.

The installation has several optional components. For this lab, we only require MATLAB to be installed (requires approximately 3GB of disk space).

We're going to show you how to set up and perform the Google PageRank algorithm in MATLAB now. As we demonstrate this, you'll pick up enough MATLAB to complete the MATLAB portion of the in-lab and take-home programming assignments. It's very likely that you will use MATLAB in future courses, and possibly in your engineering career, so if you have time we encourage you to explore the MATLAB environment. Note: if any of these screen images are too small, you may click on them to view the full-size version. Note that there may be a different version of MATLAB available than what is shown on these images/instructions.

1. Let's start by opening MATLAB. Choose Start -> All Programs -> MATLAB -> R2020b -> MATLAB R2020b:



[\(images/openmatlab.png\)](#)

2. When MATLAB opens, the window should look like this. We'll **use the centre pane (the Command Window) to enter commands**. Commands can also be saved in a .m source file, but we won't be creating or using any MATLAB source files today:

3. Let's **define a variable**. The **matrix is the basic data structure in MATLAB**, so let's define a simple 2D matrix.

In fact, in MATLAB *everything* is a matrix (MATLAB stands for Matrix Laboratory, after all). The data elements in the matrix can be numbers, boolean values, characters, or other structs. In MATLAB, a single number is stored as a 1 x 1 matrix. A vector (array) is stored as a 1 x n matrix.

We'll start by creating our original connectivity matrix, **G**. Recall that **G** is a 4 x 4 matrix of 1s and 0s. We define a matrix variable in MATLAB almost the same way we do in C, except we don't need to declare its type.

Type in the following declaration command, and then press enter. When you are done, MATLAB should look like this:

```
ConnectivityMatrix = [ 0 1 1 0; 1 0 1 0; 1 1 0 0; 0 0 0 0]
```






There are 4 things to note here:

1. When we declare a matrix, we use square brackets, and we separate each row with a semi-colon (;)
2. MATLAB prints the results of your command in the command window. It does this when we do NOT end the command with a semi-colon (;)
3. The `ConnectivityMatrix` variable appears in our Workspace pane
4. Our command is entered in the Command History pane.
4. If you double click on the `ConnectivityMatrix` variable in the Workspace pane, MATLAB will open a detailed Variables pane, like this (you can close it when you're done looking at it by clicking the X in the upper right corner of the pane, because we don't need it right now):
5. We need to **store the size of the connectivity matrix** in order to proceed with the Google PageRank calculation.

The easiest way to do this is with the MATLAB `size` command. The `size` command accepts a single parameter, the matrix we are measuring. Since a matrix has length and width, we assign the result to two new (undeclared) variables.

MATLAB is produced by MathWorks, and MathWorks maintains a very good reference API for their C library. You can read more about the `size` command at <http://www.mathworks.com/help/matlab/ref/size.html>   
(<http://www.mathworks.com/help/matlab/ref/size.html>)

Type in the following command, and then press enter. When you are done, MATLAB should look like this:

```
[rows, columns] = size(ConnectivityMatrix)
```

Since this is a square matrix, we can also use a MATLAB shortcut, because both of the dimensions are the same. Instead of one parameter (the connectivity matrix), and instead of assigning the result to two values (rows and columns), we can use two parameters and assign the result to a single variable. You can read more about this at the [MathWorks size reference page](#).

Type in the following command, and then press enter. When you are done, MATLAB should look like this:

```
dimension = size(ConnectivityMatrix, 1)
```

6. Let's **calculate the sum of each column** in the `ConnectivityMatrix`, and store these sums in an ordered vector (array). We will need the sum of each column when we start to create our stochastic matrix (remember we create a connectivity matrix first, and from that a stochastic matrix, and from that a transition matrix) (see the [Pre-lab Reading: Google PageRank Algorithm](http://www.mathworks.com/help/matlab/ref/sum.html) section), so it's best that we do this now before we start manipulating the matrix.

You can read more about the `sum` command at <http://www.mathworks.com/help/matlab/ref/sum.html>   
(<http://www.mathworks.com/help/matlab/ref/sum.html>).

Type in the following command, and then press enter. When you are done, MATLAB should look like this:

```
columnsums = sum(ConnectivityMatrix, 1)
```

It is important to note here that when we are using the `sum` command, when the second parameter equals 1, it refers to columns, and when the second parameter equals 2, it refers to rows. When we are using the `size` command, when the second parameter equals 1, it refers to rows, and when the second parameter equals 2, it refers to columns. Don't get confused!

7. Do you remember the **random walk probability factor**? This introduces the notion of a random walk. Let's set it now.

Type in the following command, and then press enter. When you are done, MATLAB should look like this:

```
p = 0.85
```

8. **We will use 5 commands to generate our stochastic matrix.** We generate our stochastic matrix from our connectivity matrix, and we will use our stochastic matrix to create our transition matrix. So remember: connectivity  $\rightarrow$  stochastic  $\rightarrow$  transition. Remember that we will use the transition matrix as our initial state in the Markov chain process.

These steps may not seem intuitive at first, but they are an efficient way to generate the matrix we need. This is not the only way to generate the matrix we need, and you are encouraged to experiment.

1. We need to **find the indices of the columns whose sums are not zero**. The easiest way to do this is with the MATLAB `find` command. The version of the `find` command which we will use (and there are a few!) accepts a single parameter, the vector (array) we are investigating. It locates all of the nonzero elements of the parameter array, and returns the indices of those elements in a vector (array) of equal length.


We're going to modify this command just a little, and ask it to find all the columns which do *not* add to zero. In MATLAB, we use `~=` for 'not equal to' (e.g., C's `!=`). Maybe the best explanation is a demonstration:

You can read more about the `find` command at <http://www.mathworks.com/help/matlab/ref/find.html>   
(<http://www.mathworks.com/help/matlab/ref/find.html>)

Type in the following command, and then press enter. When you are done, MATLAB should look like this:

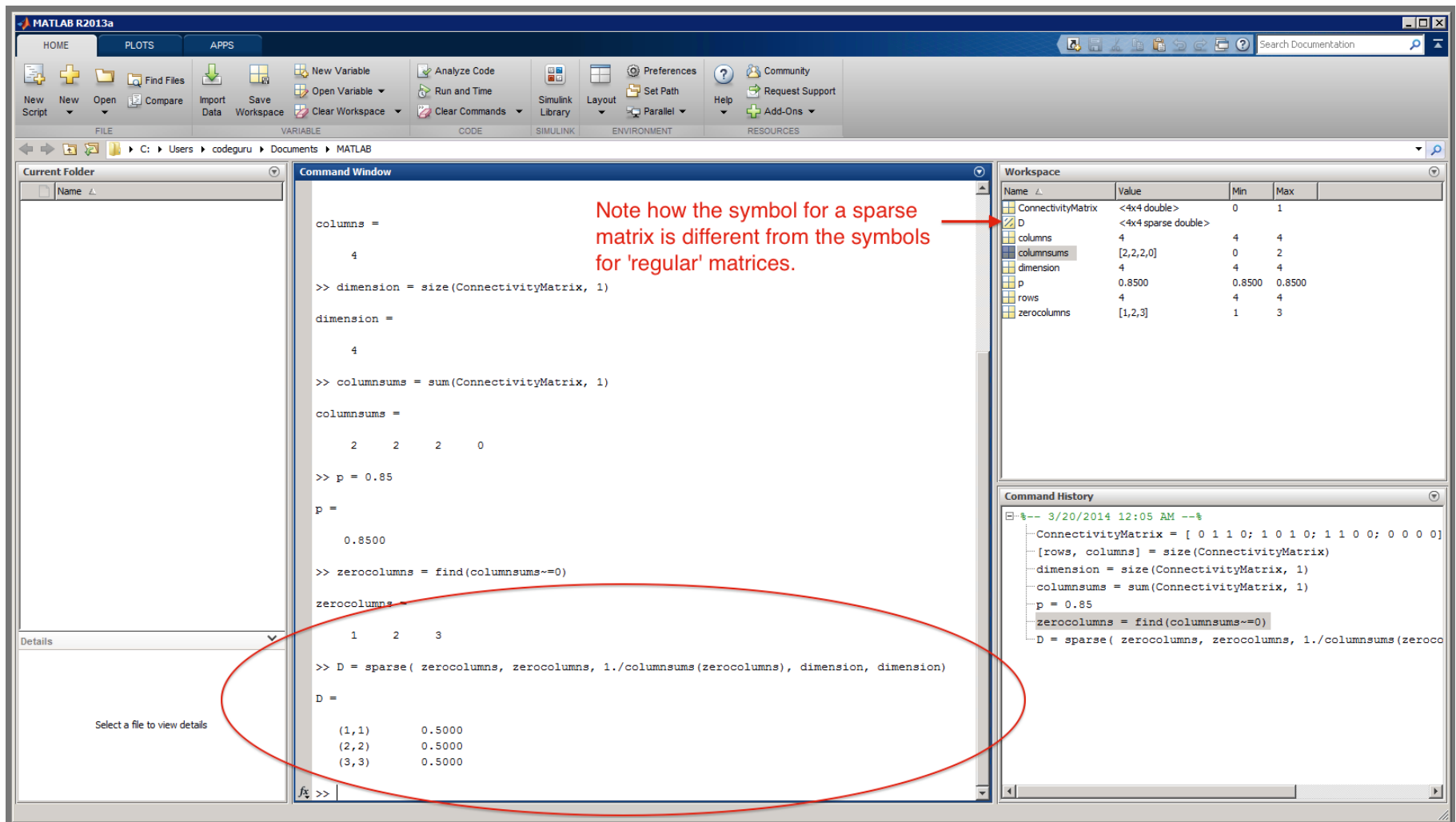
```
zerocolumns = find(columnsums~=0)
```

2. **Generate a (sparse) matrix whose diagonal components are equal to the inverse of each column sum**. That's a mouthful, isn't it! A sparse matrix is a matrix whose elements are mostly zero. In this case, we will use the 5-parameter version of MATLAB's `sparse` command. The `sparse` command uses the first 2 parameters (which are all vectors) to generate a list of `ij` indices, and the 3rd parameter contains the values to place in those indices in the sparse matrix. The sparse matrix's size is determined by the final 2 parameters. The sparse matrix is all zeros, except for the diagonal elements. Maybe the best explanation is a demonstration:

You can read more about the `sparse` command at <http://www.mathworks.com/help/matlab/ref/sparse.html>   
(<http://www.mathworks.com/help/matlab/ref/sparse.html>)

Type in the following command, and then press enter. When you are done, MATLAB should look like this:

```
D = sparse( zerocolumns, zerocolumns, 1./columnsums(zerocolumns), dimension, dimension)
```



Perhaps the most challenging thing to understand here is the code fragment `1./columnsums(zerocolumns)`. What does this do?

It takes the values in the `columnsums` array which correspond to the indices stored in `zerocolumns`, and creates a new vector of the inverses of these sums. The `.` (dot) means we are using element-wise multiplication here, e.g., we apply the calculation to each element of the array. Pretty neat! Also, look at the interesting way that we get our sparse matrix output, as a set of indices and corresponding values.

3. **Start to construct our stochastic matrix: multiply our connectivity matrix by the sparse matrix.** This will create a partial (incomplete) stochastic matrix; only the non-zero columns are correct after this command.

Type in the following, and then press enter. When you are done, MATLAB should look like this:

```
StochasticMatrix = ConnectivityMatrix * D
```

4. **Find the zero columns of our original connectivity matrix.** The easiest way to do this is with the MATLAB `find` command, which we used a few moments ago. The version of the `find` command which we will use this time accepts a single parameter again, but we are going to accept 2 return values, the same way we accepted 2 return values when using the `size`.

Type in the following command, and then press enter. When you are done, MATLAB should look like this:

```
[row, column] = find(columnsums==0)
```


5. **Generate the stochastic matrix.**

All we need to do now is replace the zero column of our stochastic matrix with a column whose elements equal  $1 / \text{dimension}$ . We want to apply our transformation to the zero columns only.

We've already stored the the columns which are zero in our `column` variable. In MATLAB, we can use this to designate which columns of the matrix to work with. We do this by using some MATLAB shorthand with the colon (`:`) to designate the columns of `StochasticMatrix` that correspond to the indices stored in `column`.

You can read more about how to use the MATLAB `colon` at <http://www.mathworks.com/help/matlab/ref/colon.html>  (<http://www.mathworks.com/help/matlab/ref/colon.html>).

We're going to use element-wise division here, too, which you can read about at

<http://www.mathworks.com/help/matlab/ref/rdivide.html>  (<http://www.mathworks.com/help/matlab/ref/rdivide.html>). The best example is a demonstration, so let's finish this stochastic matrix!


Type in the following command, and then press enter. When you are done, MATLAB should look like this:

```
StochasticMatrix(:, column) = 1./dimension
```

9. Now that we have our stochastic matrix, we need to **generate our transition matrix**. Recall from the [Pre-lab Reading: Google PageRank Algorithm](#) section that we want to generate a matrix Q, whose dimension matches the stochastic matrix, and whose entries are all equal to the inverse of the dimension.

We will use the MATLAB `ones` command to generate a matrix of 1s of appropriate size. The version of the `ones` command we will use accepts two parameters, for the two dimensions of the matrix to generate.

Once we have this matrix, we can use some math fun to invert its elements and complete the transition matrix calculation.

You can read more about the `ones` command at <http://www.mathworks.com/help/matlab/ref/ones.html>  (<http://www.mathworks.com/help/matlab/ref/ones.html>)

Type in the following 2 commands, pressing enter after each command. When you are done, MATLAB should look like this:

```
Q = ones(dimension, dimension)
```

```
TransitionMatrix = p * StochasticMatrix + (1 - p) * (Q/dimension)
```




We can use this transition matrix now to find the PageRank. We have had to use 11 commands to get where we are. There are only 3 more things to do.

10. Remember from the [Pre-lab Reading: Google PageRank Algorithm](#) section that we need a column vector whose length equals our transition matrix's dimension. Let's generate a column vector of ones. Can you guess how to do this (hint: what about the `ones` command?).

Type in the following command, and then press enter. When you are done, MATLAB should look like this:

```
PageRank = ones(dimension, 1)
```

11. And now our penultimate step. Remember from the [Pre-lab Reading: Google PageRank Algorithm](#) section that we need to **multiply our column vector by the transition matrix over and over again until the column vector stops changing**. There are some very clever ways to do this, but let's start with a simple option. Let's use a for-loop and just loop over so many times that it's bound to converge. We can use a for-loop in MATLAB the same way we use for-loops in C. The only difference is the syntax.

You can read more about the `for` command, and other loop control statements, at [http://www.mathworks.com/help/matlab/matlab\\_prog/loop-control-statements.html](http://www.mathworks.com/help/matlab/matlab_prog/loop-control-statements.html)   
([http://www.mathworks.com/help/matlab/matlab\\_prog/loop-control-statements.html](http://www.mathworks.com/help/matlab/matlab_prog/loop-control-statements.html))

Type in the following command, and then press enter. Be very careful not to mix up the colon (:) and the semi-colon (;). When you are done, MATLAB should look like this:

```
for i = 1:100 PageRank = TransitionMatrix * PageRank; end
```

12. **Normalize the PageRank vector values.** We can do this by dividing each value in the vector by the sum of the values, to determine which proportion each element occupies. We can use the `sum` command here again.

Type in the following command, and then press enter. When you are done, MATLAB should look like this:

```
PageRank = PageRank / sum(PageRank)
```

And that's it! That was a lot of reading, but there are only 14 lines of code to implement Google PageRank. Only 14!

You will use MATLAB during the in-lab exercise to complete this calculation for a different web, so you may want to highlight the 14 lines of code you will need to use (or save them in a `.m` file).

## 5. In-lab Exercises

**Select a partner quickly and remember to exchange contact information right away** if you have not worked together before.

If you're not sure about something, check the lecture slides, the textbook, and the web, or ask a TA. Remember that copying code, even snippets, is plagiarism. If you do find some code on the net for something common (like a snippet of code), remember to include the website's URL in your function comments. Make sure you understand how it works. You will answer questions about your lab individually.

### TO-DO #1: Configure a Visual Studio Project to Invoke the MATLAB Engine

0. Before you do anything else, choose a partner and **exchange contact information**. Get his or her first and last name, email address (one that they use at least once or twice a day), and cell number. You will need to coordinate schedules in order to meet for the take-home component. It would be a good idea to set aside about 4 or so hours for this, in 1.5 hour 'chunks'. Start early, and don't wait until the night before it's due.
1. For the first exercise in the lab, you will create a new Visual Studio project and tinker with its configuration settings so the C program you write can communicate with MATLAB. We need to ensure that the program you write has access to the C libraries

provided with the MATLAB software. **These steps are finicky, and if you miss a single step it will not work.** Navigators: pay very close attention to what your Driver is doing!

2. Start Visual Studio. If you are asked to choose your default environment settings, select **Visual C++ Development Settings**.
3. **Let's create a new project.** Choose **File -> New -> Project**.
4. When the New Project dialog box opens, select **Visual C++** from the left sidebar (Installed Templates) and then **Empty Project** from the middle panel.
5. Choose a short, descriptive name for the project, like **CPSC259\_Lab5**.
  - Make sure that **Create directory for solution** option is checked (in the lower right corner of the dialog box), and **save your project**.
  - Click OK. Ignore any warnings about the project location and the CLR runtime.
6. You must create a C source file in the project in order for Visual Studio to recognize that the project is a C/C++ project. This lets us 'tamper' with project properties that are specific to C programs, e.g., how to communicate with the MATLAB engine installed in the lab. Right-click the Source Files folder and select **Add/New Item** from the pop-up menu. **DO NOT SKIP ANY STEPS!**
7. When the New Item dialog box opens, make sure **Visual C++** is selected in the left panel and then choose **C++ file (.cpp)** from the options in the right-hand panel. C++ is a superset of C, and we use the same compiler.
8. Enter `pagerank.c` for your source file name. *Please be sure to use the ".c" suffix, which denotes a C source file, and not ".cpp", which denotes a C++ file.*
9. Leave the specified location unchanged and click Add.
10. An empty source file will be added to your project and the file will be open for editing.
11. In the Solution Explorer, right-click the project name and choose Properties from the dropdown list
12. When the project's Property Pages opens, click the Configuration Manager button in the upper right corner of the window.
13. In the new window you will see column headings Project, Configuration, Platform and Build. From the dropdown menu under 'Platform', choose <New...>. A new window shows up with 'New Platform' as 'ARM.' Change 'ARM' to 'x64' by choosing 'x64' from the drop down list, and 'Copy Settings from' as 'Win32'. Select OK followed by Close.
14. Under 'C/C++ General', add the directory `C:\Program Files\MATLAB\R2020b\extern\include` to the field 'Additional Include Directories'. Click anywhere inside the empty field beside 'Additional Include Directories', and choose <Edit...>, enter the name of the directory,

and then click on "OK." Note that this path may be different depending on where MATLAB is installed on your/the lab system.


15. Under 'C/C++ Precompiled Headers', select 'Not Using Precompiled Headers'.
16. Under 'Linker General', add the directory `C:\Program Files\MATLAB\R2020b\extern\lib\win64\microsoft` to the 'Additional Library Directories' field. Note that this path may be different depending on where MATLAB is installed on your/the lab system.
17. Under 'Linker Input', add the following names to 'Additional Dependencies' one per line: `libmx.lib libmat.lib libeng.lib`.
18. Under 'Configuration Properties Debugging' add `PATH=C:\Program Files\MATLAB\R2020b\bin\win64` to 'Environment'. Make sure there are not any spaces around the = between PATH and C:, otherwise you will receive this error: 'The program can't start because libmx.dll is missing from your computer. Try reinstalling the program to fix this problem.' Note that this path may be different depending on where MATLAB is installed on your/the lab system.
19. Don't forget to click "OK" when you're done.

Pretty easy, right? That's all there is to it. Every time you want to create a project in Visual Studio that communicates with MATLAB, you must make these modifications to the project properties **after** creating a .c source file. If you try to do this before creating a .c source file, the configuration options for C/C++ will not be available in the Properties Manager.

**You do not need to demonstrate anything to your TA until the end of the To-Do #2, because To-Do #2 will not work if you do not complete this section correctly.**

## TO-DO #2: Learn how a C program in Visual Studio can invoke the MATLAB Engine

20. Now let's create a simple C program that uses MATLAB to analyze a simple matrix. All that configuration we did allows us to use pre-written MATLAB-specific C functions that are installed alongside MATLAB. They're there for people like us who want to use MATLAB with other programming languages. The functions will be unfamiliar, but you will use them every time you want to communicate with MATLAB from a C program running in Visual Studio. We're going to walk you through each step of the process. Here's what we're going to do:
  - i. include some header files
  - ii. create a single function called main
  - iii. declare some variables using some special MATLAB types and some familiar C types


- iv. start a MATLAB "process"
  - v. create a MATLAB variable for our C-array test data
  - vi. copy the raw data from our C-array to a MATLAB variable
  - vii. place the MATLAB variable in the MATLAB workspace
  - viii. executes a MATLAB command that asks the MATLAB engine to calculate some eigenvalues using the MATLAB variable we copied over
  - ix. retrieve the result from the MATLAB workspace
  - x. learn how to use a buffer and echo MATLAB output to our C program
  - xi. close the connection to the MATLAB engine and free our memory.
21. You can view the API we will be using for the in-lab and take-home lab at <https://www.mathworks.com/help/matlab/calling-matlab-engine-from-c-programs-1.html>  (<https://www.mathworks.com/help/matlab/calling-matlab-engine-from-c-programs-1.html>) . In fact, we encourage you refer to this API while you are completing the take-home portion.
22. At the top of your `pagerank.c` source file, add the following **preprocessor directives**. The final header file, `engine.h`, contains the prototypes for the functions and types we will be using to communicate with MATLAB from C.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "engine.h"
```

23. We will use a **single main function** in this exercise: `int main( void )`. Go ahead and create it now. Inside our main function let's declare the variables we will use:

```
/* Variables */
Engine *ep = NULL; // A pointer to a MATLAB engine object
mxArray *testArray = NULL, *result = NULL; // mxArray is the fundamental type underlying MATLAB data
double time[3][3] = { { 1.0, 2.0, 3.0 }, {4.0, 5.0, 6.0 }, {7.0, 8.0, 9.0 } }; // Our test 'matrix', a 2-D array
```

24. Now let's **start a MATLAB process**. We will use the function `engOpen`. On Windows systems, `engOpen` opens a COM channel to the MATLAB software registered during installation.

You can read more about the `engOpen` function at <http://www.mathworks.com/help/matlab/apiref/engopen.html>  (<http://www.mathworks.com/help/matlab/apiref/engopen.html>) .

```

/* Starts a MATLAB process */
if ( !(ep = engOpen(NULL)) ) {
    fprintf(stderr, "\nCan't start MATLAB engine\n");
    system("pause");
    return 1;
}

```

25. Let's create a MATLAB-friendly variable for our test data. We will use the `mxCreateDoubleMatrix` function, which creates a 2-D double-precision floating-point array initialized to 0, or returns NULL on failure.

You can read more about the `mxCreateDoubleMatrix` function at

<http://www.mathworks.com/help/matlab/apiref/mxcreatedoublematrix.html>   
[\(<http://www.mathworks.com/help/matlab/apiref/mxcreatedoublematrix.html>\)](http://www.mathworks.com/help/matlab/apiref/mxcreatedoublematrix.html).

```
testArray = mxCreateDoubleMatrix(3, 3, mxREAL);
```

26. Our next task is to **copy the data** from our local 2-D array `time` to the MATLAB variable `testArray`. We do this using the `memcpy` function (which is why we included `string.h` in our preprocessor directives). We will also use `mxGetPr` which returns a pointer to the first element of our mxArray. Recall that this is just the starting address of the mxArray.

You can read more about the `mxGetPr` function at <http://www.mathworks.com/help/matlab/apiref/mxgetpr.html>   
[\(<http://www.mathworks.com/help/matlab/apiref/mxgetpr.html>\)](http://www.mathworks.com/help/matlab/apiref/mxgetpr.html).

```
memcpy((void*) mxGetPr(testArray), (void *)time, 9 * sizeof(double));
```

27. Now we can **place the test array into the MATLAB workspace**, and we will use the function `engPutVariable`. `engPutVariable` writes the mxArray to the engine and gives it the specified variable name. If the mxArray does not exist in the workspace, the function creates it. If an mxArray with the same name exists in the workspace, the function **replaces** the existing mxArray with the new mxArray.

Do not use MATLAB function names for variable names. Common variable names that conflict with function names include `i`, `j`, `mode`, `char`, `size`, or `path`. **To determine whether a particular name is associated with a MATLAB function, use the which function at the MATLAB command prompt.** The engine application owns the original mxArray and is responsible for freeing its

memory. Although the `engPutVariable` function sends a copy of the mxArray to the MATLAB workspace, the engine application does not need to account for or free memory for the copy. `engPutVariable` returns 0 if successful and 1 if an error occurs.

You can read more about the `engPutVariable` function at <http://www.mathworks.com/help/matlab/apiref/engputvariable.html>  (<http://www.mathworks.com/help/matlab/apiref/engputvariable.html>).

```
if ( engPutVariable(ep, "testArray", testArray) ) {  
    fprintf(stderr, "\nCannot write test array to MATLAB \n");  
    system("pause");  
    exit(1); // Same as return 1;  
}
```

28. So far, so good, right? Now it's finally time to **invoke the power of the MATLAB engine**. Let's calculate the **eigenvalues** of our matrix. We can do this using the MATLAB command `eigenvectorVariableName = eig(testArray)`. Recall that the `eig(A)` MATLAB function returns a vector of the eigenvalues of matrix A.

We send this command to the MATLAB engine from our C program using a helpful function called `engEvalString`. This function accepts two parameters: a) the first parameter is the pointer to the engine; b) the second is a string representing the command we wish the engine to evaluate. `engEvalString` returns 0 if the MATLAB engine session evaluated the command and 1 if an error occurs.

You can read more about the `engEvalString` function at <http://www.mathworks.com/help/matlab/apiref/engevalstring.html>  (<http://www.mathworks.com/help/matlab/apiref/engevalstring.html>).

```
if ( engEvalString(ep, "testArrayEigen = eig(testArray)") ) {  
    fprintf(stderr, "\nError calculating eigenvalues \n");  
    system("pause");  
    exit(1);  
}
```

29. Your next question is **how do we send the result of this calculation to our C program**, right? Well, we put our variable into the MATLAB workspace with the `engPutVariable` function, and we can get a variable from the MATLAB workspace with the `engGetVariable` function.

`engGetVariable` reads the named mxArray from the MATLAB engine session associated with `ep` (our C program pointer to the

MATLAB engine), and returns a pointer to a newly allocated `mxArray` structure, or NULL if the attempt fails. We will also use `mxGetNumberOfElements` to find out how large the returned data is. `mxGetNumberOfElements` tells us how many elements an array has. For example, if the dimensions of an array are 3-by-5-by-10, it returns 150.

You can read more about the `engPutVariable` function at <http://www.mathworks.com/help/matlab/apiref/engputvariable.html>  <http://www.mathworks.com/help/matlab/apiref/engputvariable.html>).

You can read more about the `engGetVariable` function at <http://www.mathworks.com/help/matlab/apiref/enggetvariable.html>  <http://www.mathworks.com/help/matlab/apiref/enggetvariable.html>).

You can read more about the `mxGetNumberOfElements` function at <http://www.mathworks.com/help/matlab/apiref/mxgetnumberofelements.html>  <http://www.mathworks.com/help/matlab/apiref/mxgetnumberofelements.html>).

```
printf("\nRetrieving eigenvector\n");
if ((result = engGetVariable(ep,"testArrayEigen")) == NULL) {
    fprintf(stderr, "\nFailed to retrieve eigenvalue vector\n");
    system("pause");
    exit(1);
}
else {
    size_t sizeOfResult = mxGetNumberOfElements(result);
    size_t i = 0;
    printf("The eigenvalues are:\n");
    for (i = 0; i < sizeOfResult; ++i) {
        printf("%f\n", *(mxGetPr(result) + i) );
    }
}
```

30. What if we want to **capture MATLAB output directly in order to echo it**? It's simple. We need to define a local character buffer for `engEvalString` to return any output which would ordinarily appear in MATLAB. At the top of your source file, after the preprocessor directives and before the main method, define a buffer size:

```
#define BUFSIZE 256
```

Inside the main method, in the variable list, declare a buffer whose size is 1 larger than our defined size:



```
char buffer[BUFSIZE + 1];
```

Why do we want to do this? The buffer returned by `engEvalString` is not **NULL terminated**, so we ensure that the buffer is NULL-terminated by inserting a NULL into the final cell of the buffer. We create the buffer using the function `engOutputBuffer`.

`engOutputBuffer` defines a character buffer for `engEvalString` to return any output that ordinarily appears on the screen. It returns 1 if we pass it a NULL engine pointer, else it returns 0.

You can read more about the `engOutputBuffer` function at <http://www.mathworks.com/help/matlab/apiref/engoutputbuffer.html> . (<http://www.mathworks.com/help/matlab/apiref/engoutputbuffer.html>)

```
if ( engOutputBuffer(ep, buffer, BUFSIZE) ) {  
    fprintf(stderr, "\nCan't create buffer for MATLAB output\n");  
    system("pause");  
    return 1;  
}  
buffer[BUFSIZE] = '\0';
```

31. Once we have our MATLAB output buffer, how do we use it? Simple! After each call to function `engEvalString`, the **buffer is automatically cleared** and refilled with the results of the command contained in the string passed to the engine. We can try it like this:

```
engEvalString(ep, "whos"); // whos is a handy MATLAB command that generates a list of all current variables  
printf("%s\n", buffer);
```

32. That's it. All we need to do now is **manage (free) our memory and close our connection to MATLAB**. We can free memory associated with MATLAB mxArray's using `mxDestroyArray`. This function deallocates memory occupied by a specified mxArray. The last thing we want to do is close the engine. We can do that using the `engClose` function. `engClose` sends a quit command to the MATLAB engine session and closes the connection. It returns 0 on success, and 1 otherwise.

You can read more about the `mxDestroyArray` function at <http://www.mathworks.com/help/matlab/apiref/mxdestroyarray.html> . (<http://www.mathworks.com/help/matlab/apiref/mxdestroyarray.html>)

You can read more about the `engClose` function at <http://www.mathworks.com/help/matlab/apiref/engclose.html> . (<http://www.mathworks.com/help/matlab/apiref/engclose.html>)

```
mxDestroyArray(testArray);
mxDestroyArray(result);
testArray = NULL;
result = NULL;
if ( engClose(ep) ) {
    fprintf(stderr, "\nFailed to close MATLAB engine\n");
}
```

33. Don't forget to add these two lines to the very end of your program!

```
system("pause"); // So the terminal window remains open long enough for you to read it
return 0; // Because main returns 0 for successful completion
```

34. When you execute your program, don't be worried if it takes a few moments for Visual Studio to open the MATLAB engine. Though it may seem like it takes longer than it would to simply solve this using C (or by hand!), the computation time we save by using MATLAB for problems of a larger scale is worth it. Your output should look like this:

35. Copy all the contents of your `.c` file and save it as `eigenvalues.c`. This will be submitted via PL and manually graded.

## TO-DO #3: Invoke MATLAB from a C program to find the product of 2 matrices

36. Now that you've learned the basics, it's time to apply them. You will edit the demonstration code to perform a matrix operation, and show your correct answer to a TA.
37. In our first example, we declared a 3 x 3 matrix called time. In this exercise, modify your existing code to **declare two 3 x 3 matrices**, and populate each of them with some doubles. For simplicity, we called ours `matrixOne` and `matrixTwo`. Remember to declare two variables of type `mxArray` too (refer to step 22 if you're not sure what we mean).
38. Use the MATLAB API to **multiply the two matrices**, and then retrieve the product from the MATLAB engine.
39. Print both of the original matrices, and the product. Keep the elements of your matrix simple, and verify your result manually. Make sure you **print the result in matrix form**, and not just a column of numbers (e.g., you should print a 3 X 3 matrix of numbers like this, not a list of 9 numbers):
40. If you are having problems here, look carefully at how you are representing your matrices in C vs MATLAB. C represents 2D arrays in "row-major order", while MATLAB uses "column-major order". This can cause problems when you copy the matrix from C to MATLAB and may be critical for your assignment. You will need to represent your matrix in C in such a way that it is correct when copied to MATLAB.

41. Copy all the contents of your `.c` file and save it as `multmatrix.c`. This will be submitted via GradeScope and manually graded.

## TO-DO #4: Use MATLAB to calculate a PageRank

42. We're going to return to MATLAB for the rest of the in-lab work. In fact, you can close Visual Studio now.

43. We'd like you to calculate the PageRank using MATLAB for the following web graph. You and your partner should determine the connectivity matrix for this web, and then use the steps from the [pre-lab reading about MATLAB](#) (those 14 lines of code plus one more line for your connectivity matrix declaration) to determine the PageRank. Do not use Visual Studio or any C, just use MATLAB directly. **Save the commands in a new file** `pagerank.m`. This is a MATLAB script file, and you can add comments by preceding the comment with a "%" character. When you have the result, show a TA for your marks.

44. Consider a web graph with nodes 1 through 5 inclusive:

1. No node contains links to itself
2. Node 1 contains links to all of the other nodes
3. Node 2 contains links to node 1 and node 4
4. Node 3 contains no links
5. Node 4 contains links to nodes 1, 2, and 3
6. Node 5 contains links to nodes 2 and 4.

45. **When you have determined the PageRank of the 5 nodes in this web, save your** `pagerank.m` **file and include it in your GradeScope submission.. Congratulations. You have completed your final CPSC 259 in-lab exercise. You're almost there!**

## 6. Take-Home Assignment

For your take-home lab, you and your partner will modify your in-lab project and write a C program that uses MATLAB to **compute the PageRank of a web of hyperlinked pages**. You have the tools you need. Everything you need to know about using MATLAB to calculate the PageRank is in this lab. Everything you need to know about sending information from a C program to MATLAB is in this lab. You and your partner just need to put it all together.

You may wish to review the frameworks from your previous labs, specifically the parts about opening and reading files. You are encouraged to copy snippets of code from these labs to your final programming lab. Make sure you document the sources in a comment above each snippet. It's okay to reuse code instead of reinventing the wheel over and over again.

## TO-DO #5: Write a C program that uses MATLAB to calculate a PageRank

46. Your program will assume that the web you are analyzing is a text file called `web.txt`. You can hard-code this into your program.
47. The file is a text file which contains an **unspecified** number of lines of the form:

```
0 1 1 0 1 0
1 0 1 1 1 1
0 0 0 0 0 1
1 1 1 0 1 1
0 1 0 1 0 1
1 1 1 1 1 0
```

Each line represents a row of a connectivity matrix. Since it is a connectivity matrix, the number of rows will be equal to the number of 0s and 1s in the first row. **You do not need to worry about pages which link to themselves, e.g., the main diagonal will always be zeros.**

48. Use C to extract the information in the file and store it in some simple and useful structure (a 2D connectivity matrix is a good first step). If you are not sure how to proceed, remember to review the frameworks from the previous labs. Do not hard-code the dimension of the connectivity matrix because we may test your program with a variety of `web.txt` files containing square connectivity matrices of all sorts of sizes. Use dynamic memory allocation.
49. Use your new C skills to copy the connectivity matrix to the MATLAB workspace, and then use the 14 MATLAB commands to calculate the PageRank.
50. Print a ranked list of the pages to standard output (the command line). Your output should look a little like this (this is the PageRank for the example `web.txt` file in step 47):

```
NODE  RANK
---  ----
1      0.1226
2      0.2149
3      0.0707
4      0.2030
5      0.1739
6      0.2149
```

Press any key to continue . . .

51. The TAs who mark your assignment will test your program with a variety of `web.txt` files containing square connectivity matrices of all sorts of sizes, so don't hard-code a fixed web size.

## 7. Deliverables

There are in-lab deliverables and 1 take-home deliverable for this lab:

### In Lab Deliverable Due Date and Mark Distribution:

**You have until the end of the second Lab 5 meeting to demonstrate your in-lab deliverables to a TA.**

VISUAL STUDIO: Property set-up walkthrough, and test program (eigenvalues.c, not autograded) 10 marks

VISUAL STUDIO AND MATLAB: Multiplying two matrices (multmatrix.c, not autograded) 10

MATLAB: Calculating the page rank (pagerank.m, not autograded) 10

**TOTAL: 30 marks**

**Be aware that pagerank.m must be a Matlab-executable script containing the Matlab commands for computing a pagerank - it is NOT a text copy of the Matlab output window upon running your pagerank commands.**

### Take Home Deliverable You need to submit these file(s):

1. pagerank.c (not autograded)

### Take Home Deliverable This is when and how you must submit your work:

1. **Your take-home assignment is due at 11:59 PM by the deadline indicated in PrairieLearn .**
2. Ensure that both of your names, student numbers and CWLs appear in the comments at the top of the files.
3. For this and every take-home lab, **BOTH partners must submit individually on PrairieLearn.**
4. Submit the deliverable files to the appropriate activity on PrairieLearn.

**Marks** will be assigned based on:

- correctness of code (does your program compile and generate the correct response, and is recursive where applicable)
- clarity of code (are your variable names self-explanatory, have you written code that someone else can understand, etc.)
- code style (is your indentation correct, is your code consistent and well-formatted, are your comments complete and correct, etc.)

Work with your partner, not alone. We're encouraging you to program in pairs because it makes programming easier. **Remember our plagiarism rules.**

## Take Home Deliverable Marks:

**Marks** will be assigned based on:

- correctness of code (does your program compile and generate the correct response)
- clarity and readability of code (are your variable names self-explanatory, have you written code that someone else can understand, etc.)
- code style (is your indentation correct, is your code consistent and well-formatted, are your comments complete and correct, etc.)
- submission components (did you include **all** of the required elements in your readme.txt, etc).

Work with your partner, not alone. We're encouraging you to program in pairs because it makes programming easier. **Remember our plagiarism rules.**

## 8. Hints

1. Keep it simple. For your take-home assignment, you're welcome to add one or more header and source files, but this is not necessary. You can also decompose the program into separate functions. The most important thing is that your code is correct and *readable*.
2. The example earlier in the lab which shows how to read a matrix into MATLAB uses a static 2D matrix in C, and later casts it as a 1D matrix when the memcpy function is used. In your own assignment you will need to use a dynamic matrix, and this casting **will not work**. It works for the static case because the rows are contiguous in memory, and this is not true for a dynamically allocated matrix. You will need to figure out another way to get your C matrix into the proper form.
3. There are different ways to read and copy the connectivity matrix text file into dynamic memory management before sending it to MATLAB. One possible approach is to count the number of characters in the first line of the file which you can acquire using fgets and sscanf. Since the lines of file always follow the pattern `number space number space ... number`, followed by a new-line character, the dimension of the matrix will always be  $(\text{\# of characters in the first line of the file excluding any new-line characters} - 1) / 2 + 1$ , and the numbers will always be at indices 0, 2, 4, ..., so you could approach the task with some creative control loops. Check the

last several lines of the `get_maze_dimension` function in lab 4's take-home code for a good method to get the length of the string while excluding any new-line and carriage-return characters.

4. To generate your own `web.txt` file, right click the **Resource Files** folder in your Visual Studio project, and Add -> New Item -> Utility -> Text File. Remember to call your file `web.txt`, and go ahead and hard-code the name into your program.
5. Remember to use the debugger. The debugger is your friend and helper. Set breakpoints and execute the debugger after you write a block of code. Remember that you can add and remove breakpoints during a debugging session? You can also restart a debugging session by pressing **Control+Shift+F5**, or by selecting the little blue square with the white arrow in it on the debugging section of the Visual Studio menu bar.
6. When you're debugging code that contains pointers, remember to set Watches (see lab 2) in order to examine dynamically allocated memory easily.
7. Some of you will likely notice that there is some overhead when we invoke the MATLAB engine from C, e.g., it takes a while. You're right, it does. You've identified a trade-off: is it faster to try to code this up in C, or is it easier to just invoke MATLAB and abstract away the math? For a small, simple program like your lab assignment, it could be more cost-efficient to simply write your program in C (or, frankly, just do it by hand). Imagine an enormous web with billions of links. As our problem scales up in size, using MATLAB becomes a more efficient way to approach these problems. Scalability is an issue that computer scientists consider when approaching problems, and the methods you are learning in today's lab scale very well to large, complex problems (and not so well to smaller, easier problems!).
8. When we send a command to the MATLAB engine, MATLAB will print out the answer for any command not terminated by a semi-colon. MATLAB will not print out the result if the line ends in a semi-colon.