ENGR 695 Advanced Topics in Engineering Mathematics Fall 2024

Lab #1: Introduction to *Matlab* and Solving Linear Systems

Introduction

The purpose of this first lab experience is to familiarize you with and/or reintroduce you to *Matlab* so that you can use it for matrix and array operations and other computational tasks to come. The specific goals are to:

1. Familiarize you with the *Matlab* interface and its use and syntax.
2. Begin applying some of the built-in functions in *Matlab* to solve and assess the solvability of linear systems.

Many of the items in this handout are meant to be prompts for discussions during the lab session. You should take notes on topics and techniques that are unfamiliar to you and ask questions when clarifications are needed.

Tutorial Activities

We will work through a few group exercises together to make sure that everyone knows and/or becomes reacquainted with basic *Matlab* operations. If you are an intermediate or expert *Matlab* user, much of this will seem very elementary to you. If so, please be patient since other students might be less familiar with the software. Feel free to help other students if it looks like they might be struggling a bit, or just speak up during the lab session and we can discuss questions as a group. I don't know *everything* about *Matlab*!

The lists below outline basic information and commands to help you get started. We will work through this section at a pace appropriate for the relative levels of expertise in the class. Note that there is a list of frequently used *Matlab* commands on the last page of this handout. You might want to keep it handy as a reference today and throughout the semester.

The *Matlab* interface and environment

- Start-up
- Where you are and where you want to be
- The command window and the command prompt (`>>`)
- Variables and the meaning of =
- Workspace window and other windows
- Working directory (Current Folder, where files are stored and read from)
- Data types: numeric, character, complex numbers (`whos`)
- Help (`help`) and documentation (`doc`)
- Script files (also known as m-files): a way to record command sequences; like a program
- `clear`, `close all`, and `home`: three recommended commands for beginning scripts
- Dealing with errors

Arrays in *Matlab*

Default behaviors:

- Most functions create row vectors; use transpose (`'` or `.'`) to get column vectors
- Complex numbers are the basic numerical type
- The asterisk (*) means matrix multiplication; size, shape, and direction matter
- *Matlab* uses scalar extension; statements like $A + a$, where A is a matrix and a is a scalar, are allowed. Purists beware.
- See the handout on the last page

Syntax – key points and hints:

- Define arrays with [and].
- Use parentheses liberally to ensure proper evaluation.
- Separate array elements in the same row with spaces or commas.
- Use continuation ellipses (…) to break up long expressions into two or more lines.
- Use a semicolon (;) to force a new row within a matrix or column vector.
- The colon (:) is a shorthand way to indicate all of the elements in a row or column (1 through M or 1 through N).
- Array indices can be vectors – for example, `b([2 3 4])` – but those vectors must contain integers.
- Make liberal use of comment lines, which start with the symbol %, in m-files (scripts).
- Keep it rectangular when constructing arrays from subarrays; that is, make sure that the matrix dimensions match.

Guided Exercises

The following exercises are for practice and instruction and will not be graded. We will work on them together to reinforce skills related to constructing and manipulating matrices in *Matlab*.

First, we will work through how to define a matrix equation and solve it. This section will cover a lot of the key commands needed for matrix definition and manipulation, but there are many more useful commands summarized on the last page of this handout. We will start by considering the linear system of equations

$$\begin{aligned} -2x_1 + 2x_2 - 4x_3 - 6x_4 &= -4 \\ -3x_1 + 6x_2 + 3x_3 - 15x_4 &= -3 \\ 5x_1 - 8x_2 - x_3 + 17x_4 &= 9 \\ x_1 + x_2 + 11x_3 + 7x_4 &= 7 \end{aligned}$$

Enter the elements of the coefficient matrix A and the right-hand-side vector **b** into *Matlab* variables:

```
>> clear
>> A = [-2 2 -4 -6 ; -3 6 3 -15 ; 5 -8 -1 17 ; 1 1 11 7]
>> b = [-4 -3 9 7]'
```

Use the `whos` command or the Workspace window to check that both quantities have been saved and are the correct sizes.

Solve the system using the recommended method (\ or backslash):

```
>> x = A\b
```

It is worth assessing the answer generated by *Matlab* after using this and similar commands. It is not so important for a small and well-defined problem like this one, but for larger problems or ones involving coefficients and other values of widely varying sizes, checking solutions could be very important. For now, check the solution x by premultiplying it by the matrix `A` (i.e., performing the "forward" multiplication) and compare it to the vector that it should have calculated. The variable `bcheck` below is the result of the forward multiplication `A*x`. (`b` is the previously defined right-hand-side vector.) The vector `error` saves the computation errors:

```
>> bcheck = A*x
>> error = b - bcheck
```

The `error` vector should be full of zeros. What does it mean if it has nonzero entries?

Solve for the solution vector `x` using the `inv` (inverse) command and matrix multiplication:

```
>> x = inv(A)*b
>> bcheck = A*x
>> error = b - bcheck
```

Are there differences between this result and the previous one?

Now compute the inverse of A using two different methods. You do not usually have to compute the inverse of a matrix in order to solve a system of linear equations, but there are times when it is useful to know. The `eye` function generates an identity matrix of the specified size.

```
>> Ainv1 = inv(A)
>> Ainv2 = A\eye(size(A))
```

Compare the two results by computing the difference:

```
>> Diff = Ainv1 – Ainv2
```

The differences in this case are small, but for larger or more complicated problems, the second method (using `eye`) is recommended.

Individual Exercises

The following exercises are meant to be completed on your own. Assistance will be provided as needed, but try to deduce on your own how to complete as much of the work as possible. Before you begin, activate the *Matlab* `diary` command to record everything that you type in the command window. After you complete the lab, you will submit your diary record as your deliverable for grading. Mistakes will not be counted against you. My primary concern is that you eventually figure out how to complete the following tasks correctly.

First, make sure that you are in the network folder in which you wish to store your ENGR 695 work. Next, to activate the diary, type:

```
>> diary Lab1_LName.txt,
```

where "LName" should be replaced with your last name (surname). After you have finished or if you need to turn off the recording temporarily, type:

```
>> diary off
```

To reactivate the diary, type:

```
>> diary on
```

Matlab will resume saving your commands and results in the original file you named when you first activated the diary. New text is appended to the old text; the file is not replaced or overwritten. To start a new diary, use the `diary` command with a new file name.

1. Find the solution to the following system of equations. Since this is the first problem, use matrix *A* to store the coefficients and vector **a** to store the right-hand side:

$$\begin{aligned} 3x_1 - x_2 + x_3 &= -1 \\ 9x_1 - 2x_2 + x_3 &= -9 \\ 3x_1 + x_2 - 2x_3 &= -9 \end{aligned}$$

2. That should have gone okay. Now try to solve the following system. Use matrix *B* to store the coefficients and vector **b** to store the right-hand side:

$$\begin{aligned} x + 2y - 5z &= 2 \\ 2x - 3y + 4z &= 4 \\ 4x + y - 6z &= 8 \end{aligned}$$

   You should get an error/warning message (which you should read) that indicates that you are working with a singular matrix.

3. Try applying elementary row operations (EROs) to the augmented matrix (*B* and **b** combined) for the second problem to reduce the augmented matrix to row echelon form. First, use the commands

```
>> B2 = B
>> b2 = b
```

   to preserve the original variables *B* and **b**, then type `aug2 = [B2 b2]` to form an augmented matrix called `aug2`. To perform an ERO on the augmented matrix, type something like the following:

```
>> aug2(3,:) = aug2(3,:) - 2*aug2(2,:)
```

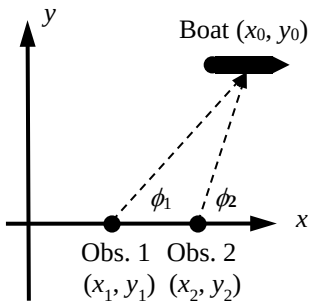
This command subtracts 2 times the second row of the augmented matrix `aug2` from the third row and stores the result in the third row of the same matrix. Note that the colon symbol (:) is used here to tell *Matlab* to apply the operation to every column in the indicated row. For example, the expression `aug2(3,:)` refers to the entire third row of the augmented matrix. After you perform the operation, *Matlab* should display the modified augmented matrix, which should have a zero in the first column of the third row. Continue using similar commands, modified as necessary, to reduce `aug2` to row echelon form. You should ultimately end up with at least one row full of zeros.

4. *Matlab* has tools for assessing the solvability of linear systems. The key commands are `rank` and `rref`. As you work through the following steps, keep two important questions in mind:

    a. What does `rref` "do" to an input? Use the `help` and/or `doc` commands if you would like more information.
    b. What characteristic of a row-reduced matrix is summarized by the command `rank`?

    Apply the `rank` and `rref` commands to the matrices `A` and `B` and to the augmented matrices `[A a]` and `[B b]` from the problems above. Do the results confirm the earlier results that you obtained when you tried to solve each problem?

5. Now imagine that observers on land are trying to determine the position of a boat offshore. As shown in the diagram below, the observers with their spotting scopes are located at positions 1 and 2. Each observer records an angle ϕ from the x-axis to the boat.



Expressing the tangent of each angle in terms of the boat's and observers' coordinates yields the relationships

$$\tan\phi_1 = \frac{\sin\phi_1}{\cos\phi_1} = \frac{y_0 - y_1}{x_0 - x_1} \quad \rightarrow \quad (x_0 - x_1)\sin\phi_1 = (y_0 - y_1)\cos\phi_1$$

and

$$\tan\phi_2 = \frac{\sin\phi_2}{\cos\phi_2} = \frac{y_0 - y_2}{x_0 - x_2} \quad \rightarrow \quad (x_0 - x_2)\sin\phi_2 = (y_0 - y_2)\cos\phi_2 ,$$

which can be expressed in matrix form as

$$\begin{bmatrix} \sin\phi_1 & -\cos\phi_1 \\ \sin\phi_2 & -\cos\phi_2 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} x_1\sin\phi_1 - y_1\cos\phi_1 \\ x_2\sin\phi_2 - y_2\cos\phi_2 \end{bmatrix}.$$

Only the boat's position (the coordinates x_0 and y_0) is unknown. The observers' locations and the angles that they are read are known quantities.

Modify the provided *Matlab* script (Lab1start.m) in the indicated location to determine the boat's location if $\phi_1 = 74°$ and $\phi_2 = 85°$. Record the result. Suppose that the boat moves so that $\phi_1 = 0°$ and $\phi_2 = 0°$. Use the script to determine the boat's location if you can. Explain what happens in this case and why.

Lab Submission and Scoring

Submit your *Matlab* diary via the link provided at the ENGR 695 course Moodle site. Please do not e-mail it directly to me. Make sure that your diary file includes a record of the Individual Exercises 1 through 5 above. Your score will be based primarily on the diary that you submit according to the rubric posted on the Laboratory page at the course web site.

If you do not complete the exercises during the lab session, you may submit your *Matlab* diary as late as 5:00 pm on Friday, September 6. If the file is submitted after the deadline, a 10% score deduction will be applied for every 24 hours or portion thereof that the item is late (not including weekend days) unless extenuating circumstances apply. No credit will be given five or more days after the deadline.

Important Matlab Commands for Linear Algebra

The table below summarizes some of the more common operations that are specific to the field of linear algebra and their *Matlab* equivalents. Unless otherwise specified, vectors are assumed to be column vectors and contain N elements while matrices are square and $N \times N$ in size.

Action	**Example**	**Command(s)**	**Comments**
Define a matrix	$A = \begin{bmatrix} 1 & 5 \\ 9 & 4 \end{bmatrix}$	`A = [1 5 ; 9 4]`	; forces a new row
Define a column vector	$\mathbf{b} = \begin{bmatrix} 5 \\ 9 \end{bmatrix}$	`b = [5 ; 9]`	
		`A = [5 9]'`	`'` is transpose; `.'` w/o conjugation
Get matrix size	M = rows(A) N = cols(A)	`[M,N] = size(A)`	see also `length`
$N \times N$ identity	I_N	`I = eye(N)`	`eye(M,N)` also works
Vector inner product	$s = \mathbf{x}^T\mathbf{y}$	`s = x'*y`	scalar result
Vector outer product	$A = \mathbf{x}\mathbf{y}^T$	`A = x*y'`	matrix result
Matrix transpose	A^T or A^H	`A'`	includes conjugation if complex
Matrix power	A^n	`A^n`	must be square
Matrix multiply	$C = AB$	`C = A*B`	must be conformable
Hadamard (array) product	$C = A \circ B$	`C = A.*B`	matrices same size `./` and `.^` are similar
Matrix inverse	$B = A^{-1}$	`B = inv(A)` `B = A\eye(size(A))`	use only for "small" problems recommended if inverse needed
Solve $A\mathbf{x} = \mathbf{b}$	$\mathbf{x} = A^{-1}\mathbf{b}$	`x = inv(A)*b` `x = A\b`	not recommended generally recommended ("A under $\mathbf{b}$")
Determinant	$\lvert A \rvert$	`det(A)`	mostly for "small" problems
Matrix rank	rank(A)	`rank(A)`	uses the SVD (see below)
LU decomposition	$A = LU$	`[L,U] = lu(A)`	see `help lu`
QR decomposition	$A = QR$	`[Q,R] = qr(A)`	see `help qr`
Singular value decomposition	$A = U\Sigma V^T$	`[U,S,V] = svd(A)`	see `help svd`
Compute eigensystem	solve $A\mathbf{x} = \lambda\mathbf{x}$	`[V,D] = eig(A)`	see `help eig`
Extract column k from A	$\mathbf{c} = \mathbf{a}_k$	`c = A(:,k)`	column vector
Extract row j from A	$\mathbf{r}^T = \mathbf{a}_j^T$	`r = A(j,:)`	row vector

 Adapted from handouts by Prof. Maneval, Fall 2019