

LAB 1: Matrix Algebra and LU Factorization

Before doing this assignment, please read the document *Notes on MATLAB Assignments* (available from the course web page). It describes how to record the results of your MATLAB session in a *diary file*, and then how to edit this file to create your Lab report.

Script Files: When you need to write a script file for this lab and subsequent labs, use the following procedure: Start MATLAB and click on *File*, then *New*. Move the pointer to the right and click on *m-Files*. This will open the MATLAB Editor/Debugger Window. Type the script commands in this window, then click on *File* in the Editor/Debugger toolbar and save your script on in your directory.

After you have created and saved an m-file, you must set the *Path* so that MATLAB can find this file. Click on *File* and then *Set Path* and follow the directions to add your directory to the list of path names.

Script Files for this Lab: Use the text editor in MATLAB to create the following MATLAB *function* m-files:

- (a) Create a *function* m-file with the commands

```
function v = rvect(m)
v = fix(10*rand(m,1));
```

(note the semicolon on the end of the second line). Save this file under the name `rvect.m` (be sure that you have set the *Path* as described above so that MATLAB can find this m-file). Test the file by clicking on the MATLAB window and typing `v = rvect(3)` at the MATLAB prompt. You should get a column vector $\mathbf{v} \in \mathbf{R}^3$ with entries that are (random) integers between 0 and 9. Now type `u = rvect(3)` You will get another random column vector $\mathbf{u} \in \mathbf{R}^3$. Type `v` at the prompt. You should get the *same* vector \mathbf{v} as before.

Note that the name \mathbf{v} in the `rvect` function file is a *local variable*; you can assign any name to the output. If you have already defined a vector \mathbf{v} in your local space, it is not changed when you generate \mathbf{u} by `rvect`.

- (b) Create another function m-file with the commands

```
function A = rmat(m,n)
A = fix(10*rand(m,n));
```

(note the semicolon on the end of the second line). Save this file under the name `rmat.m`. Then test the file by clicking on the MATLAB window and typing `A = rmat(3, 5)` at the MATLAB prompt. You should get a 3×5 matrix A with entries that are (random) integers between 0 and 9.

Random Seed: When you start your MATLAB session, initialize the random number generator by typing

```
rand('seed', abcd)
```

where *abcd* are the last four digits of your Student ID number. This will ensure that you generate your own particular random vectors and matrices.

BE SURE TO INCLUDE THIS LINE IN YOUR LAB WRITE-UP

The lab report that you hand in must be your own work. The following problems all use randomly generated matrices and vectors, so the matrices and vectors in your lab report will not be the same as those of other students doing the lab. Sharing of lab report files is not allowed in this course.

Question 1. Gaussian Elimination and Reduced Row-Echelon Form

The most basic algorithm in linear algebra is Gaussian elimination. In this question you will use MATLAB to carry out this algorithm and obtain the reduced row-echelon form of a matrix. Before starting work on this question, type `rrefmovie` at the MATLAB prompt. You will see a step-by-step example of the row operations that transform a matrix A into its reduced row echelon form $R = \text{rref}(A)$. In this demonstration, each pivot is chosen to be the *largest* in its column (for numerical stability), so extra row interchanges are used. Since $\text{rref}(A)$ is *uniquely* determined by A , this does not affect the final answer. (Do not include this part in your lab write-up.)

(a) Generate a 3×4 matrix $A = \text{rmat}(3,4)$. Now use MATLAB to transform A into $R = \text{rref}(A)$. Start with $R = A$ and normalize the first row of R to get $R(1,1) = 1$:

$$R = A; R(1,:) = R(1:)/R(1,1)$$

(note the use of the colon to operate on whole rows of the matrix). If your random matrix happens to have $A(1,1) = 0$, then interchange rows to get a nonzero entry in the $(1,1)$ position before doing the calculation above. Next subtract a multiple of the first row of R from the second row to make $R(2,1) = 0$:

$$R(2,:) = R(2,:) - R(2,1)*R(1,:)$$

Repeat this procedure to make $R(3,1) = 0$:

$$R(3,:) = R(3,:) - R(3,1)*R(1,:)$$

(you can minimize typing by using the up-arrow key and editing the command line). The first column of your matrix R should now be the same as the first column of $\text{ref}(A)$.

(b) Operate on R by the same method as in (a) to obtain the second column of $\text{rref}(A)$. First normalize row 2 of R , then subtract multiples of row 2 from rows 1 and 3 to put zeros in the $(1,2)$ and $(3,2)$ positions (you can minimize typing by using the up-arrow key and editing the commands used in part (a)).

(c) Operate on R by the same method as in (b) to obtain the third column of $\text{rref}(A)$. First normalize row 3 of R , then subtract multiples of row 3 from rows 1 and 2 to put zeros in the $(1,3)$ and $(2,3)$ positions.

(d) Your matrix R should now be transformed into $\text{rref}(A)$ (since A is a random 3×4 matrix, $\text{rref}(A)$ is (almost) sure to have rank 3). Check your answer by MATLAB with the command `rref(A)`. If the MATLAB answer is not the same as the current value of your R , go back and redo your calculations.

Question 2. Triangular Matrices

A square matrix is called *strictly upper triangular* if its entries on and below the main diagonal are all zero.

(a) Use the MATLAB editor to write the function m-file

```
function y = rup(n)
y = triu(rand(n), 1);
```

Save this file under the name `rup.m`. Then by entering the command `A = rup(3)` at the prompt, you will get a 3×3 random strictly upper triangular matrix. Repeat to generate random 3×3 strictly upper triangular matrices B and C . Then calculate $A * B$, $A * B * C$. What pattern do you observe about the forms of these matrix products? Do this again for 4×4 matrices (taking products of 2, 3, and 4 random strictly upper triangular matrices generated by `rup(4)`).

(b) Using the evidence from (a), formulate a theorem about the product of any k strictly upper triangular $n \times n$ matrices (for any integer n). Then prove your theorem (without MATLAB).

(c) Set $I = \text{eye}(4)$ (the 4×4 identity matrix). Generate a random 4×4 strictly upper triangular matrix A and calculate the matrix $U = I + A + A^2 + A^3$ and the products $U * (I - A)$ and $(I - A) * U$. What do

you observe? State your conclusion as a theorem that applies to *every* 4×4 strictly upper triangular matrix A and then prove it.

(d) Formulate and prove a result similar to what you observed in (c) that applies to *every* $n \times n$ strictly upper triangular matrix A .

Question 3. Row Operations and LU Factorization

In this problem you will use MATLAB to carry out elementary row operations and to obtain the matrix factorization $A = LU$ for a square 3×3 matrix A .

(a) Generate a random 3×3 matrix :

$$A = \text{rmat}(3,3), U = A$$

Here U has the initial value A , but at the end of the LU algorithm U will be upper triangular.

(b) Use the MATLAB editor to create an m-file called `col1.m` with the following MATLAB commands:

```
L1 = eye(3);
L1(2,:) = L1(2,:) + (U(2,1)/U(1,1))*L1(1,:);
L1(3,:) = L1(3,:) + (U(3,1)/U(1,1))*L1(1,:);
L1
```

(notice the use of `;` to suppress screen output of the intermediate results). Execute this file by typing `col1` at the MATLAB prompt. If you get a division by zero error message, go back to step (a) and generate another A and U . Describe in words how the matrix L_1 is obtained from row operations on the 3×3 identity matrix (remember that $U = A$ at the beginning of the algorithm). Use MATLAB to calculate L_1^{-1} (if M is a square matrix that has an inverse, then the MATLAB command `inv(M)` will calculate the inverse matrix M^{-1}). Describe in words how the matrix entries of L_1^{-1} are obtained from those of L_1 .

Now use MATLAB to replace the current value of the matrix U by the new value $L_1^{-1} * U$ (remember that the command $X = Y$ in MATLAB means to replace the current value of the variable X by the current value of the variable Y). Describe in words how the new value of U is obtained from the old value of U by row operations.

(c) Use the MATLAB editor to create an m-file called `col2.m` with the commands

```
L2 = eye(3);
L2(3,:) = L2(3,:) + (U(3,2)/U(2,2))*L2(2,:); L2
```

This will be used with the matrix U modified as in (b). Execute this file by typing `col2` at the MATLAB prompt. If you get a division by zero error message, go back to step (a) and start again (this is very unlikely to happen since A is completely random). Use MATLAB to calculate L_2^{-1} . Describe in words how the matrix entries of L_2^{-1} are obtained from those of L_2 .

Now use MATLAB to replace the current value of the matrix U by the new value $L_2^{-1} * U$. Describe in words how the new value of U is obtained from the old value of U by row operations.

(d) Use MATLAB to get $L = L_1 * L_2$. Verify by MATLAB that $A = L * U$ (where U has the value from (c)). Describe in words how the entries of L are obtained from those of L_1 and L_2 .

Question 4. Using LU Factorization to Solve $Ax = b$

(a) **Inverting A , L and U :** Use MATLAB to calculate the inverses of the matrices L and U that you obtained in Question #3. Which entries in `inv(L)` and `inv(U)` are *always* zero (no matter what random matrix A you generate)? Which entries in `inv(L)` are *always* 1? For the matrices L_1 and L_2 , you found in Question #3 that the inverse matrices are simply obtained by putting a minus sign in front of the entries below the main diagonal. Does this method give the correct inverse matrix for L ?

(b) **Solving $Ax = b$ using L and U** (See Equation (8) on page 35 of Strang's text): Generate a random vector $b = \text{rvect}(3)$. Calculate the solution

```
c = inv(L)*b
```

to the triangular system $Lc = \mathbf{b}$. Then calculate the solution

```
x = inv(U)*c
```

to the triangular system $Ux = \mathbf{c}$. Finally, calculate Ax and check that it is \mathbf{b} (since the entries in \mathbf{b} are integers, this should be obvious by inspection).

Question 5. LU vs. RREF for solving $Ax = b$

In this question you will compare the speed of two methods of solving the equation $Ax = \mathbf{b}$ when A is an invertible square matrix. You will measure the execution time of MATLAB commands using the internal computer clock and the MATLAB function `cputime`.

(a) Write an m-file called `ludata.m` to generate a random 100×100 matrix A , a vector $\mathbf{b} \in \mathcal{R}^{100}$, and calculate the LU decomposition of A by

```
A = rand(100) ; b = rand(100,1); [L U] = lu(A);
```

Important: Be sure to use the semicolon ; after each command so that these matrices and vector are *not* displayed or included in your diary file.

Now type `ludata`. MATLAB will carry out the operations, but not show any results on the screen.

DO NOT PRINT OR INCLUDE THESE 100×100 MATRICES IN YOUR LAB WRITE-UP

(b) You can solve $Ax = \mathbf{b}$ by using RREF (Gaussian elimination). The last column \mathbf{y} of the augmented matrix $R = \text{rref}([A \ \mathbf{b}])$ satisfies $A\mathbf{y} = \mathbf{b}$ because `rref(A)` is the identity matrix if A is a random square matrix. Write an m-file called `rrefmeth.m` with the commands

```
t = cputime; R = rref([A b]); y = R(:,101); rreftime = cputime - t
```

Important: Be sure to use the semicolon ; after the first three commands so that the matrix and vector are not displayed or printed in your diary file.

Now type `rrefmeth` to find the CPU time using RREF.

(c) You can also solve $Ax = \mathbf{b}$ by using the LU decomposition of A . Write an m-file called `lumeth.m` with the commands

```
t = cputime; c = inv(L)*b; x = inv(U)*c; lutime = cputime - t
```

Important: Be sure to use the semicolon ; after the first three commands so that the vectors are not displayed or printed in your diary file.

Now type `lumeth` to find the CPU time using the LU method.

(d) Compare the methods (b) and (c) for speed. Which method of solution was faster? Calculate the ratio `rreftime/lutime` of the computation times.

(e) For an $n \times n$ system, the number of floating point arithmetic operations (flops) needed for Gaussian elimination is approximately $2n^3/3$, while the the number of flops for the LU method (after the L , U factors are already calculated) is approximately $2n^2$ (solving two triangular systems). Compare the ratio `rreftime/lutime` that you observed with the flops ratio for the two methods when $n = 100$.

Final Editing of Lab Write-up:

After you have worked through all the parts of the lab assignment, edit your diary file. Include the MATLAB calculations, but remove errors that you made in entering commands and remove other material that is not directly related to the questions.