MAT 343 Laboratory 2
Matrix Computations and Programming in MATLAB

In this laboratory session we will learn how to

1. Solve linear systems with MATLAB
2. Create M-files with simple MATLAB codes

Backslash or Matrix Left Division

If A is an $n \times n$ matrix and $\mathbf{b}$ represents a vector in $\mathbb{R}^n$, the solution of the system $A\mathbf{x} = \mathbf{b}$ can be computed using MATLAB's backslash operator by setting

```
x = A\b
```

The "\" or "left matrix divide" command invokes an algorithm which depends upon the structure of the matrix A. Depending on the structure of the matrix, the algorithm employs different matrix factorization techniques to solve the system, some of which you will learn in later chapters (e.g., the LU decomposition and the QR factorization). Type `help mldivide` to learn more about this command.
For example, if we set `A = [1,1,1,1;1,2,3,4;3,4,6,2;2,7,10,5]` and `b = [3;5;5;8]`, then the command `x = A\b` will yield

```
x =
    1.0000
    3.0000
   -2.0000
    1.0000
```

In the case that the coefficient matrix is singular (or "close" to singular), the backslash operator will still compute a solution, but MATLAB will issue a warning. For example, the 4×4 matrix

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \tag{1}$$

is singular and the command `x = A\b` (with the same vector $\mathbf{b}$ as above) yields

```
Warning:  Matrix is close to singular or badly scaled.  Results may be inaccurate.
RCOND = 1.755481e-18.

x =
   1.0e+15 *
    2.2153
   -5.6648
    4.6837
   -1.2342
```

The `1.0e+15` indicates the exponent for each of the entries of `x`. Thus, each of the four entries listed is multiplied by 10^{15}. The value of `RCOND` is an estimate of the reciprocal of the *condition number* of the coefficient matrix. The *condition number* is a way to measure how sensitive the matrix is to round off error. Even if the matrix were nonsingular, with a condition number of the order of 10^{18}, one could expect to lose as much as 18 digits of accuracy in the decimal representation of the computed solution. Since the computer keeps track of only 16 decimal digits, this means that the computed solution may not have any digit of accuracy.

Remark: You can verify that the system $A\mathbf{x} = \mathbf{b}$, with A given by (1), is inconsistent by computing the RREF of the augmented matrix: `rref([A, b])`.

If the coefficient matrix for a linear system has more rows than columns, then MATLAB assumes that a *least squares solution* of the system is desired (you will learn about least squares in Chapter 5). If we set

```
C = A(:,1:2)
```

then C is a 4×2 matrix and the command `x = C\b` will compute the least squares solution

```
x =
   -2.2500
    2.6250
```

EXERCISES

Instructions: For the following three problems, follow the instructions in LAB 1 to create a `diary` text file.

1. (a) Set $n = 700$. The commands below generate an $n \times n$ random matrix with integer entries between 0 and 24, an $n \times 1$ vector $\mathbf{z}$ with all entries equal to 1, and an $n \times 1$ vector $\mathbf{b}$ given by the product $A\mathbf{z}$.
   Enter the following commands:

   ```
   A = floor(25*rand(n));
   z = ones(n,1);
   b = A*z;
   ```

   Note: the command `rand(n)` generates a random $n \times n$ matrix with entries uniformly distributed between 0 and 1. We multiply these entries by 25 and then use the `floor` function to round them to the greatest integer less than or equal to. This gives a matrix with integer entries between 0 and 24.
   Since A was generated randomly, it is likely to be nonsingular. The system $A\mathbf{x} = \mathbf{b}$ has a unique solution given by the vector $\mathbf{z}$.

   (i) One could compute the solution in MATLAB using the "\" operation or by computing A^{-1} and then multiplying A^{-1} times $\mathbf{b}$. Let us compare these two computational methods for both speed and accuracy. The inverse of the matrix A can be computed in MATLAB by typing `inv(A)`. One can use MATLAB `tic` and `toc` commands to measure the elapsed time for each computation. To do this, use the commands

   ```
   tic, x = A\b;   toc
   tic, y = inv(A)*b;   toc
   ```

   (Make sure the `'tic toc'` are on the same line, for otherwise, the elapsed time will include the time it takes you to type the commands. If MATLAB gives a warning about the matrix being close to singular, generate the matrix A and the vector $\mathbf{b}$ again and repeat the computations.
   Which method is faster?

   (ii) To compare the accuracy of the two methods, we can measure how close the computed solutions $\mathbf{x}$ and $\mathbf{y}$ are to the exact solution $\mathbf{z}$. One way to do this is to compute the sum of the absolute values of the components of the difference of the two vectors, that is, $\sum_{i=1}^{n} |x_i - z_i|$ and $\sum_{i=1}^{n} |y_i - z_i|$. To evaluate these sums in MATLAB we use the following commands:

```
sum(abs(x - z))
sum(abs(y - z))
```

Since we are finding the difference between the computed solution and the exact solution, the smaller number is associated with the method that is more accurate. Recall that $\mathbf{x}$ was computed using the "\" operation , while $\mathbf{y}$ was computed using the inverse.
Which method produces the more accurate solution?

(b) Repeat part (a) using $n = 1400$ and $n = 2800$.

(c) Explain why the exact solution of the system $A\mathbf{x} = \mathbf{b}$ is the vector $\mathbf{z}$.

2. The goal of this exercise is to emphasize how, when solving a linear system, the choice of which method to use is extremely important, especially in the case of matrices which are close to singular (these matrices are often refereed to as *ill conditioned.*) Similarly to Exercise 1, in this exercise you will solve a system using two methods: the "\" operator method and the inverse method. One of these two methods will produce extremely inaccurate results, while the other will produce the exact answer.

Consider the 80×80 matrix:

$$B = \begin{bmatrix} 1 & -1 & \dots & -1 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \dots & 0 & 1 \end{bmatrix}$$

Use the following MATLAB commands to generate the matrix B, the matrix $A = B^T B$ and two vectors $\mathbf{b}$ and $\mathbf{z}$

```
n = 80 ;
B = eye(n) - triu(ones(n),1);
A=B'*B;
z = ones(n,1);
b= A*z;
```

Similarly to the previous problem, the exact solution of the system $A\mathbf{x} = \mathbf{b}$ is the vector $\mathbf{z}$. Compute the solution using the "\" and using the inverse:

```
x = A\b;
y = inv(A)*b;
```

Compare the accuracy of the two methods as in the previous problem, that is, compute

```
sum(abs(x - z))
sum(abs(y - z))
```

Recall that the two commands above compute how close the computed solution is to the exact solution. Thus the smaller value is associated to the more accurate method.

Which method produces the more accurate solution?

If you solved this exercise correctly, you should have found that the inverse method produced a very inaccurate result. This is not a coincidence. In fact, in real applications, the inverse matrix is never used to solve linear systems. Besides being inaccurate, this method is also numerically inefficient since it requires a high number of computations.
Finding methods that are both efficient and accurate is one of the main goals of applied linear algebra. Just like the "\" operator, these methods are often based on matrix factorizations, which you will learn in later chapters.

3. Generate a random 7×7 matrix A with integer entries by setting
`A = floor(10*rand(7));`
and generate a 7×1 vector $\mathbf{b}$ by setting `b = floor(20*rand(7,1))-10;`
Note: the command `floor(20*rand(7,1))` generates a random 7×1 vector with entries between 0 and 19. We subtract 10 to each entry so that the vector $\mathbf{b}$ will have entries between -10 and 9.

   (a) Since A was generated randomly, we would expect it to be nonsingular. The system $A\mathbf{x} = \mathbf{b}$ should have a unique solution. Find the solution using the "\" operation (if MATLAB gives a warning about the matrix being close to singular, generate the matrix A again).

   (b) Enter

   ```
   U = rref([A, b])
   ```

   to compute the reduced row echelon form, U, of the augmented matrix `[A b]`.
   Note that, in exact arithmetic, the last column of U and the solution $\mathbf{x}$ from part (a) should be the same since they are both solutions to the system $A\mathbf{x} = \mathbf{b}$.

   (c) To compare the solutions from part (a) and part (b), compute the difference between the last column of U and the vector $\mathbf{x}$:

   ```
   U(:,8) - x
   ```

   (d) Let us now change A so as to make it singular. Set

   ```
    A(:,5) = 7*A(:,4)+4*A(:,3);
   ```

   (the above command replaces column 5 of A with a linear combination of columns 4 and 3: $\mathbf{a}_5 = 7\mathbf{a}_4 + 4\mathbf{a}_3$ where $\mathbf{a}_i$ is the ith column of A.)
   Use MATLAB to compute `rref([A b])`. How many solutions will the system $A\mathbf{x} = \mathbf{b}$ have? Explain.
   Hint: Look at the last row(s) of the RREF.

   (e) Generate two vectors, `y` and `c` by setting

   ```
   y = floor(20*rand(7,1)) - 10;
   c = A*y;
   ```

   (here `A` is the matrix from part (d)).
   The way the vector $\mathbf{c}$ is defined guarantees that the system $A\mathbf{x} = \mathbf{c}$ is consistent, that is, it has at least one solution. Explain why that is the case.

   (f) Compute the reduced row echelon form U of `[A c]`. How many solutions does the system $A\mathbf{x} = \mathbf{c}$ have? Explain.
   Hint: Look at the last row(s) of the RREF.

Relational and Logical Operators

MATLAB has six relational operators that are used for comparisons of scalars or for elementwise comparison of arrays, and three logical operators:

Relational Operators	
`<`	Less than
`<=`	Less than or equal to
`>`	Greater than
`>=`	Greater than or equal to
`==`	Equal
`~=`	Not Equal

Logical Operators	
`&`, `&&`	AND
`|`, `||`	OR
`~`	NOT

NOTE: The statement A & B is true if both statements A and B are true. The same is with

A && B, but, in this case, if A is false, MATLAB will not check whether B is true, as false A implies that (A AND B) is false. Similarly for the OR logical operators `|` and `||`.

Programming Features

MATLAB has all the flow control structures that you would expect in a high level language, including `for` loops, `while` loops, and `if` statements. To understand how loops work, it is important to recognize the difference between an algebraic equality and a MATLAB assignment. Consider the following commands:

```
>> y = 3
y =
     3
>> y = y + 1
y =
     4
```

The second command does **not** say that `y` is one more than itself. When MATLAB encounters the second statement, it looks up the present value of `y` (3), evaluates the expression `y+1` (4) and stores the result of the computation in the variable on the left, here `y`. Thus the effect of the statement is to add one to the original value of `y`.

When we want to execute a segment of a code a number of times it is convenient to use a `for` loop. One form of the command is as follows:

```
for k = kmin:kmax
     <list of commands>
end
```

The loop index or loop variable is `k`, and `k` takes on integer values from the loop's initial value, `kmin` through its terminal value, `kmax`. For each value of `k`, MATLAB executes the body of the loop, which is the list of commands.

Example 1: The following loop evaluates the sum of the first five integers, $1+2+3+4+5$, and stores the result in the variable `y`.

```
y = 0;     % initialize the value of y
for k = 1:5
   y = y+k;
end
y
```

The line beginning with % is a comment that is not executed.
Because we are not printing intermediate values of `y`, we display the final value by typing `y` after the end of the loop.
Note that the vector `y` can also be generated using the command `y=sum((1:5))`.

Example 2: The following loop generates the 4×1 vector $\mathbf{x} = (1^2, 2^2, 3^2, 4^2)^T = (1, 4, 9, 16)^T$

```
x = zeros(4,1);  % initialize the vector x
for i = 1:4
   x(i) = i^2;   % define each entry of the vector x
end
x
```

Note that the vector `x` can be generated using the single command `x = (1:4).^2`.

M-files

It is possible to extend MATLAB by adding your own programs. MATLAB programs are all given the extension `.m` and are referred to as *M-files*. There are two basic types of *M-files*: script files and function files.

Script files

Script files are files that contain a series of MATLAB commands. All the variables used in these commands are global, and consequently the values of these variables in your MATLAB session will change every time you run the script file. To create a new script, click on the *New Script* icon in the upper left corner of the Home toolbar. In the MATLAB text editor window enter the commands as you would in the Command window. To save the file click on the Save button.
For example, we know that the product $\mathbf{y} = A\mathbf{x}$ of an $m \times n$ matrix A times a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, can be computed column-wise as

$$\mathbf{y} = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \ldots + x_n\mathbf{a}_n \tag{2}$$

(here $\mathbf{a}_i$ is the ith column of A).
To perform the multiplication $A\mathbf{x}$ by column, we could create a script file `myproduct.m` containing the following commands:

```
[m,n] = size(A); % determine the dimension of A
y = zeros(m,1);  % initialize the vector y
for   i = 1:n
   y = y + x(i)*A(:,i);    % compute y
end
```

Note that, every time we go through the loop, we add a term of the sum (2) to the vector `y`.
Entering the command `myproduct` would cause the code in the script to be executed. The disadvantage of this script file is that the matrix must be named A and the vector must be named $\mathbf{x}$. An alternative would be to create a *function file*.

Function files

To create a *Function file*, click on the *New* button on the upper left corner of the Home toolbar and select *function* from the pulldown menu. *Function files* begin with a function declaring statement of the form

```
function [output_args] = fname(input_args)}
```

The "output_args" are the output arguments, while the "input_args" are input arguments. All the variables used in the function M-file are local. When you call a function file, only the values of the output variables will change in your MATLAB session.

We can change the Script file above into a function `myproduct.m` as follows:

```
function y = myproduct(A,x)
% The command myproduct(A,x) computes the product
% of the matrix A and the vector x by column.
[m,n] = size(A); % determine the dimension of A
y = zeros(m,1);  % initialize the vector y
for   i = 1:n
   y = y + x(i)*A(:,i);
end
end
```

The comment lines will be displayed whenever you type `help myproduct` in a MATLAB session. Once the function is saved, it can be used in a MATLAB session in the same way that we use built-in MATLAB functions.
For example, if we set

```
B=[1 2 3; 4 5 6; 7 8 9]; z=[4;5;6];
```

and then enter the command

```
y = myproduct(B,z)
```

MATLAB will return the answer:

```
y =
    32
    77
   122
```

Using an `if` statement

Both the script file and the function files are not complete since they do not check whether the dimension of the matrix and the vector are compatible for the multiplication. If the matrix is $m \times n$ and the vector is $q \times 1$ with $q < n$, MATLAB will give an error message when trying to run the M-file. For instance if we type

```
 A=rand(3);  x=rand(2,1);
 y = myproduct(A,x)
```

MATLAB will give the error message:

```
Index exceeds matrix dimensions.
Error in myproduct (line 7)
y = y + x(i)*A(:,i);
```

However, if $q > n$ or if the vector is a row vector, the file will run and produce the wrong output without giving any error message.
To fix this problem we can add an `if` statement to the code.

```
function y = myproduct(A,x)
% The command myproduct(A,x) computes the product
% of the matrix A and the vector x by column.
[m,n] = size(A);  % determine the dimension of A
[p,q] = size(x);  % determine the dimension of x
if (q==1 && p==n)   % check the dimensions
   y = zeros(m,1);  % initialize the vector y
   for  i = 1:n
      y = y + x(i)*A(:,i);
   end
else
   disp('dimensions do not match')
   y = [];
end
end
```

If the dimensions do not match, MATLAB will print a message and assign an empty vector to the output.

CAUTION:

- The names of script or function M-files must begin with a letter. The rest of the characters may include digits and the underscore character. You may not use periods in the name other than the last one in '.m' and the name cannot contain blank spaces.
- Avoid name clashes with built-in functions. It is a good idea to first check if a function or a script file of the proposed name already exists. You can do this with the command `exist('name')`, which returns zero if nothing with name *name* exists.
- *NEVER name a script file or function file the same as the name of a variable it computes.* When MATLAB looks for a name, it first searches the list of variables in the workspace. If a variable of the same name as the script file exists, MATLAB will never be able to access the script file.

EXERCISES

Instructions: For the following exercise, **copy and paste into a text document the function M-files and the output obtained by running them**.

4. The product $\mathbf{y} = A\mathbf{x}$ of an $m \times n$ matrix A times a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ can be computed *row-wise* as

```
y = [A(1,:)*x; A(2,:)*x; ... ;A(m,:)*x];
```

that is

```
y(1) = A(1,:)*x
y(2) = A(2,:)*x
       ...
y(m) = A(m,:)*x
```

Write a `function` M-file that takes as input a matrix A and a vector $\mathbf{x}$, and as output gives the product $\mathbf{y} = A\mathbf{x}$ *by row*, as defined above (Hint: use a `for` loop to define each entry of the vector `y`.)
Your M-file should perform a check on the dimensions of the input variables A and $\mathbf{x}$ and return a message if the dimensions do not match. Call the file `myrowproduct.m`.
Note that this file will NOT be the same as the `myproduct.m` example.
Test your function on a random 4×4 matrix A and a random 4×1 vector $\mathbf{x}$. Compare the output with `A*x`.
Repeat with a 2×7 matrix and a 7×1 vector and with a 2×7 matrix and a 1×7 vector. Use the command `rand` to generate the random matrices for testing.
Include in your lab report the function M-file and the output obtained by running it.

5. Recall that if A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the product $C = AB$ is defined if and only if $n = p$, in which case C is an $m \times q$ matrix.

   (a) Write a `function` M-file that takes as input two matrices A and B, and as output produces the product *by columns* of the two matrix.
   For instance, if A is 3×4 and B is 4×5, the product is given by the matrix

```
C = [A*B(:,1), A*B(:,2), A*B(:,3), A*B(:,4), A*B(:,5)]
```

The function file should work for any dimension of A and B and it should perform a check to see if the dimensions match (Hint: use a `for` loop to define each column of `C`). Call the file `columnproduct.m`.
Test your function on a random 4×5 matrix A and a random 5×2 matrix B . Compare the output with `A*B`.
Repeat with 3×6 and 6×5 matrices and with 3×6 and 5×6 matrices.
Use the command `rand` to generate the random matrices for testing.
Include in your lab report the function M-file and the output obtained by running it.

(b) Write a `function` M-file that takes as input two matrices A and B, and as output produces the product *by rows* of the two matrices.
For instance, if A is 3×4 and B is 4×5, the product AB is given by the matrix

```
C = [A(1,:)*B; A(2,:)*B; A(3,:)*B]
```

The function file should work for any dimension of A and B and it should perform a check to see if the dimensions match (Hint: use a `for` loop to define each row of `C`). Call the file `rowproduct.m.`
Test your function on a random 4×5 matrix A and a random 5×2 matrix B . Compare the output with `A*B`.
Repeat with 3×6 and 6×5 matrices and with 3×6 and 5×6 matrices.
Use the command `rand` to generate the random matrices for testing.
Include in your lab report the function M-file and the output obtained by running it.