MAT 343 Laboratory 2
Solving systems in MATLAB and simple programming

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
```

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

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \tag{L2.1}$$

is singular and the command `x = A\b` yields

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.387779e-018.
ans =
  1.0e+015 *
    2.2518
   -3.0024
   -0.7506
    1.5012
```

The `1.0e+15` indicates the exponent for each of the entries of $\mathbf{x}$. 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 (you will learn about this in later chapters). 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 (L2.1), 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 $n = 1000$ and generate an $n \times n$ matrix and two vectors in $\mathbb{R}^n$, all having integer entries, by setting

   ```
   A = floor(10*rand(n));
   b = sum(A')';
   z = ones(n,1);
   ```

   The exact solution of the system $A\mathbf{x} = \mathbf{b}$ is the vector $\mathbf{z}$. Why? Explain.
   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
   ```

   Which method is faster?
   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 $\mathbf{z}$. Do this with the commands:

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

   What method produces the most accurate solution?

   (b) Repeat part (a) using $n = 2000$ and $n = 5000$.

2. Consider the 100×100 matrix:

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

   Use the following MATLAB commands to generate the matrix A and two vectors $\mathbf{b}$ and $\mathbf{z}$

   ```
   n = 100;
   A = eye(n,n) - triu(ones(n,n),1);
   b = sum(A')';
   z = ones(n,1);
   ```

Similarly to the previous problem, the exact solution of the system $A\mathbf{x} = \mathbf{b}$ is the vector `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.
What method produces the most accurate solution?

3. Generate a matrix A by setting `A = floor(10*rand(6));`
and generate a vector $\mathbf{b}$ by setting `b = floor(20*rand(6,1))-10;`

   (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) Use MATLAB to compute the reduced row echelon form, U, of the augmented matrix `[A b]` (use the command `rref`).
   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 `U(:,7) - x` or examine both using `format long` (when you are done, type `format short` to go back to the original format).

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

   ```
   A(:,3) = A(:,1:2)*[4,3]'
   ```

   (the above command replaces the third column of A with a linear combination of the first two: $\mathbf{a}_3 = 4\mathbf{a}_1 + 3\mathbf{a}_2$ 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.

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

   ```
   y = floor(20*rand(6,1)) - 10;
   c = A*y;
   ```

   (here `A` is the matrix from part (d)).
   Why do we know that the system $A\mathbf{x} = \mathbf{c}$ must be consistent? Explain.
   Compute the reduced row echelon form U of `[A c]`. How many solutions does the system $A\mathbf{x} = \mathbf{c}$ have? Explain.

Relational and Logical Operators

MATLAB has six relational operators that are used for comparisons of scalars or for elementwise comparison of arrays;

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

These are MATLAB logical operators:

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

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

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 an M-file click on the File button on the upper left corner, select *New* and *Script*. 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

$$\mathbf{y} = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \ldots + x_n\mathbf{a}_n \tag{L2.2}$$

(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 (L2.2) 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 File button on the upper left corner, select *New* and *Function*. *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:

```
??? Attempted to access x(3); index out of bounds
because numel(x)=2.

Error in ==> myproduct at 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 `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* M-file example.
   Generate random matrices A and vectors `x`, and compare the output of your function file with the product `A*x`.
   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 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 the rows of `C`). Call the file `rowproduct.m`. Generate two random matrices A and B and compare the output of your function file with the product `A*B`.
   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 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 the columns of `C`). Call the file `columnproduct.m`. Generate two random matrices A and B and compare the output of your function file with the product `A*B`.
   Include in your lab report the function M-file and the output obtained by running it.