ENGINEERING MATHEMATICS
MATLAB

Department of Science and Humanities

Introduction to MATLAB

- MATLAB is a programming language developed by MathWorks.
- MATLAB stands for **MAT**rix **LAB**oratory.
- MATLAB is a program for doing numerical computation. It was originally designed for solving linear algebra type problems using matrices.
- While other programming languages mostly work with numbers one at a time, MATLAB is designed to operate primarily on whole matrices and arrays.

Introduction to MATLAB, Continued…

- Using MATLAB, an image (or any other data like sound, etc.) can be converted to a matrix and then various operations can be performed on it to get the desired results and values.
- MATLAB is a fourth-generation high-level programming language and interactive environment for numerical computation, visualization and programming.
- It has numerous built-in commands and math functions that help in mathematical calculations, generating plots, and performing numerical methods.

MATLAB's Power of Computational Mathematics

- MATLAB is used in every fact of computational mathematics. Following are some commonly used mathematical calculations where MATLAB is used:
- Dealing with Matrices and Arrays
- 2-D and 3-D Plotting and graphics
- Linear Algebra
- Algebraic Equations
- Non-linear Functions
- Statistics

MATLAB's Power of Computational Mathematics, Continued…

- Data Analysis
- Calculus and Differential Equations
- Numerical Calculations
- Integration
- Transforms
- Curve Fitting
- Special functions

Uses of MATLAB

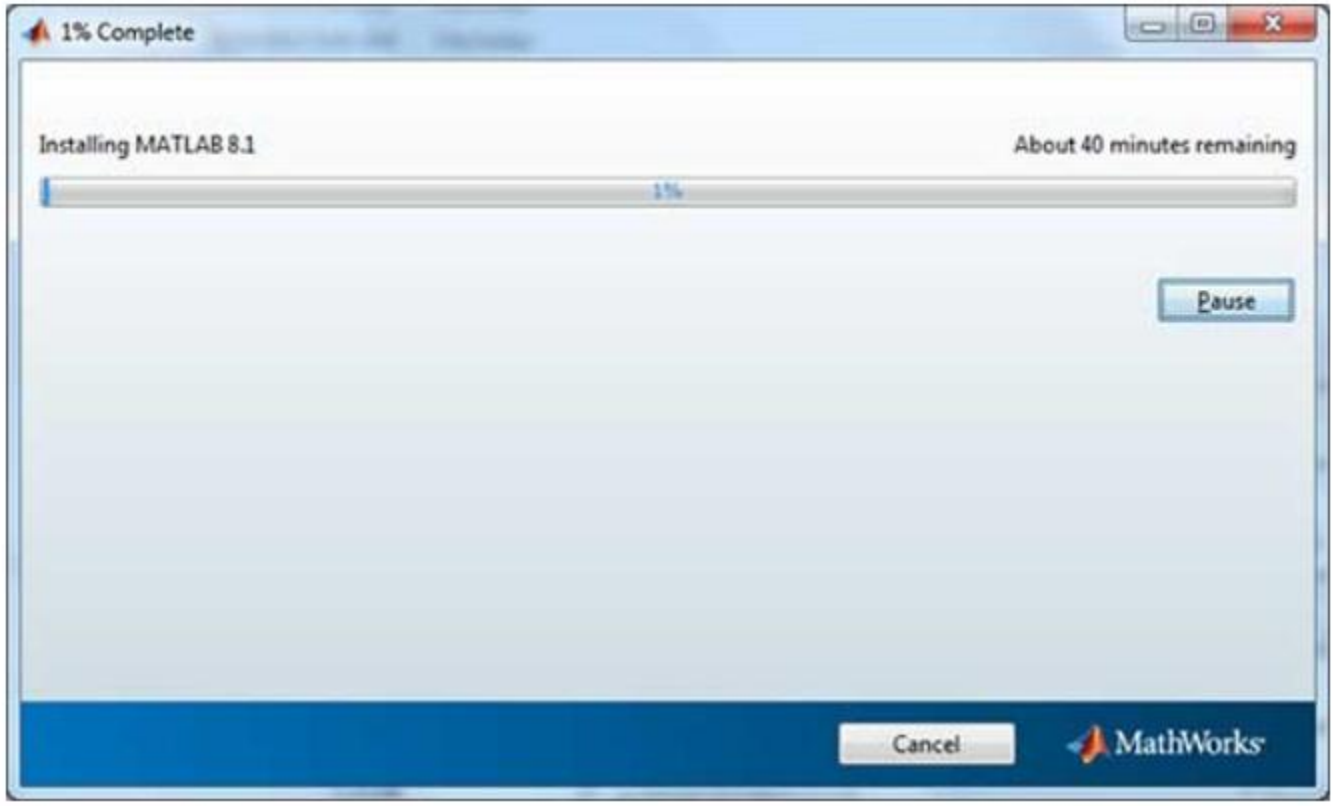
- MATLAB is widely used as a computational tool in science and engineering encompassing the fields of physics, chemistry, mathematics, and all engineering streams. It is used in a range of applications including
- Signal Processing and Communications
- Algorithm development
- Control Systems
- Computational Finance; Computational Biology

Local Environment Setup

- Setting up MATLAB environment is a matter of few clicks.
- MathWorks provides the licensed product, a trial version, and a student version as well. We need to log into the site and wait a little for their approval.
- After downloading the installer, the software can be installed through few clicks.

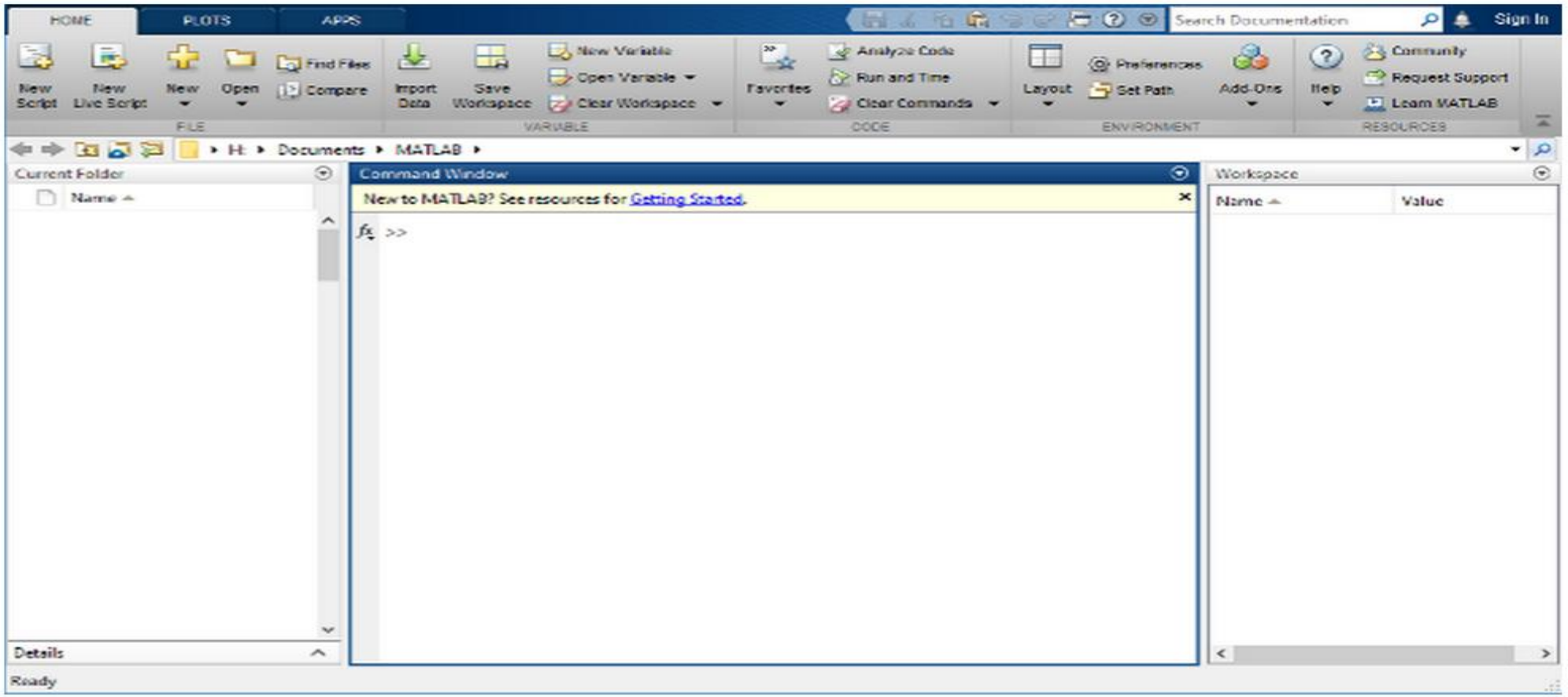
Local Environment Setup, Continued...

Local Environment Setup, Continued...



Understanding the MATLAB Environment

When you start MATLAB®, the desktop appears in its default layout.

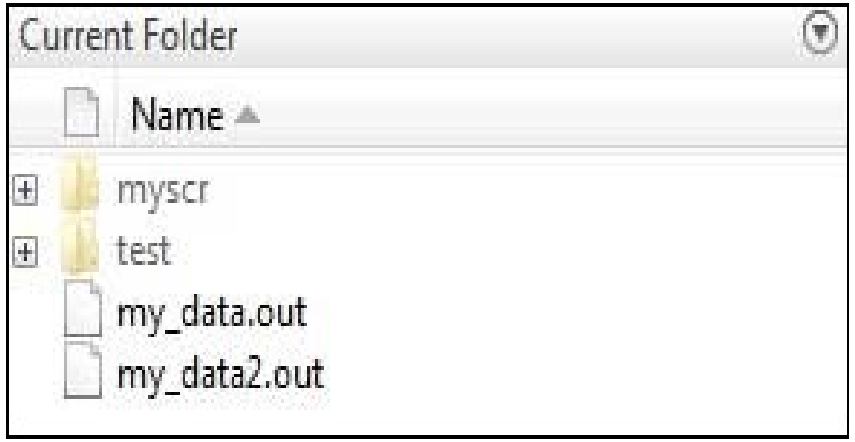


The desktop includes these panels:

- **Current Folder** — Access your files.
- **Command Window** — Enter commands at the command line, indicated by the prompt (>>).
- **Workspace** — Explore data that you create or import from files.

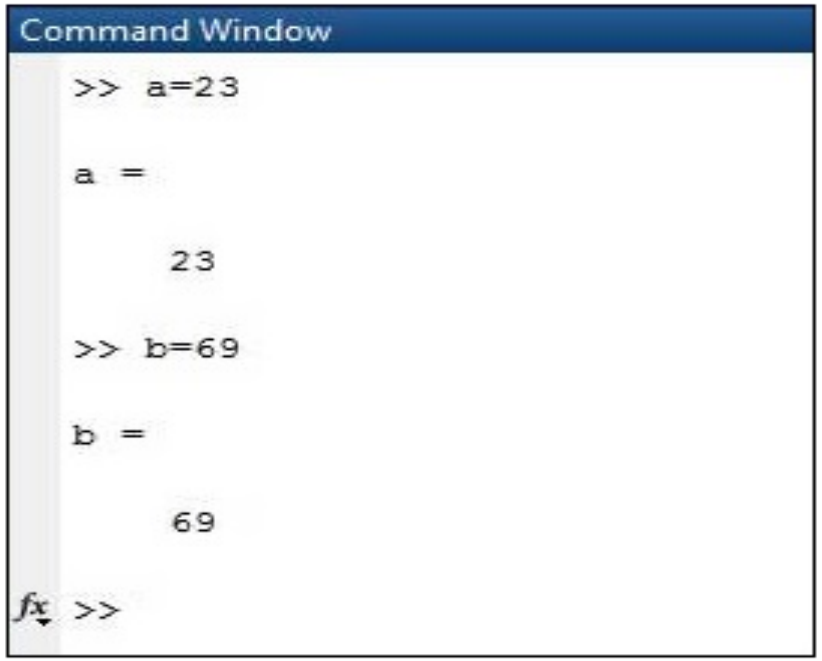
Understanding the MATLAB Environment, Continued…

- The desktop has the following panels:
- **Current Folder:** This panel allows us to access the project folders and files.



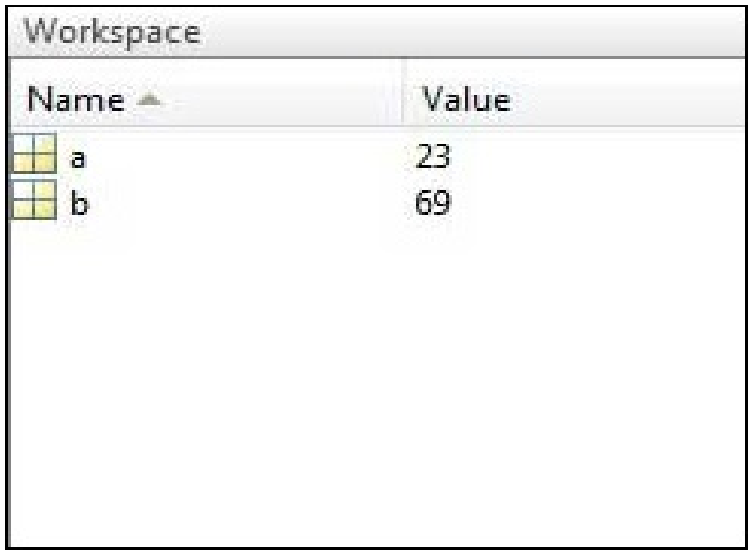
Understanding the MATLAB Environment, Continued…

- **Command Window:** This is the main area where commands can be entered at the command line. It is indicated by the command prompt (>>).



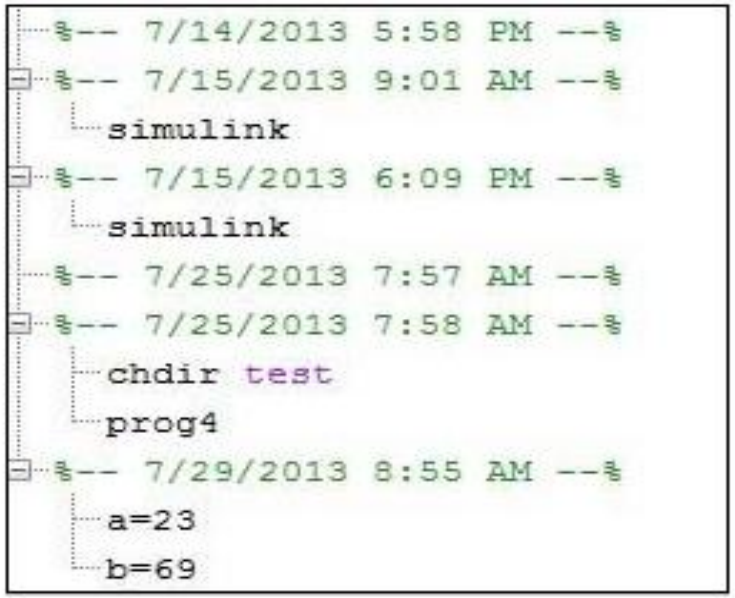
Understanding the MATLAB Environment, Continued…

- **Workspace:** The workspace shows all the variables created and/or imported from files.



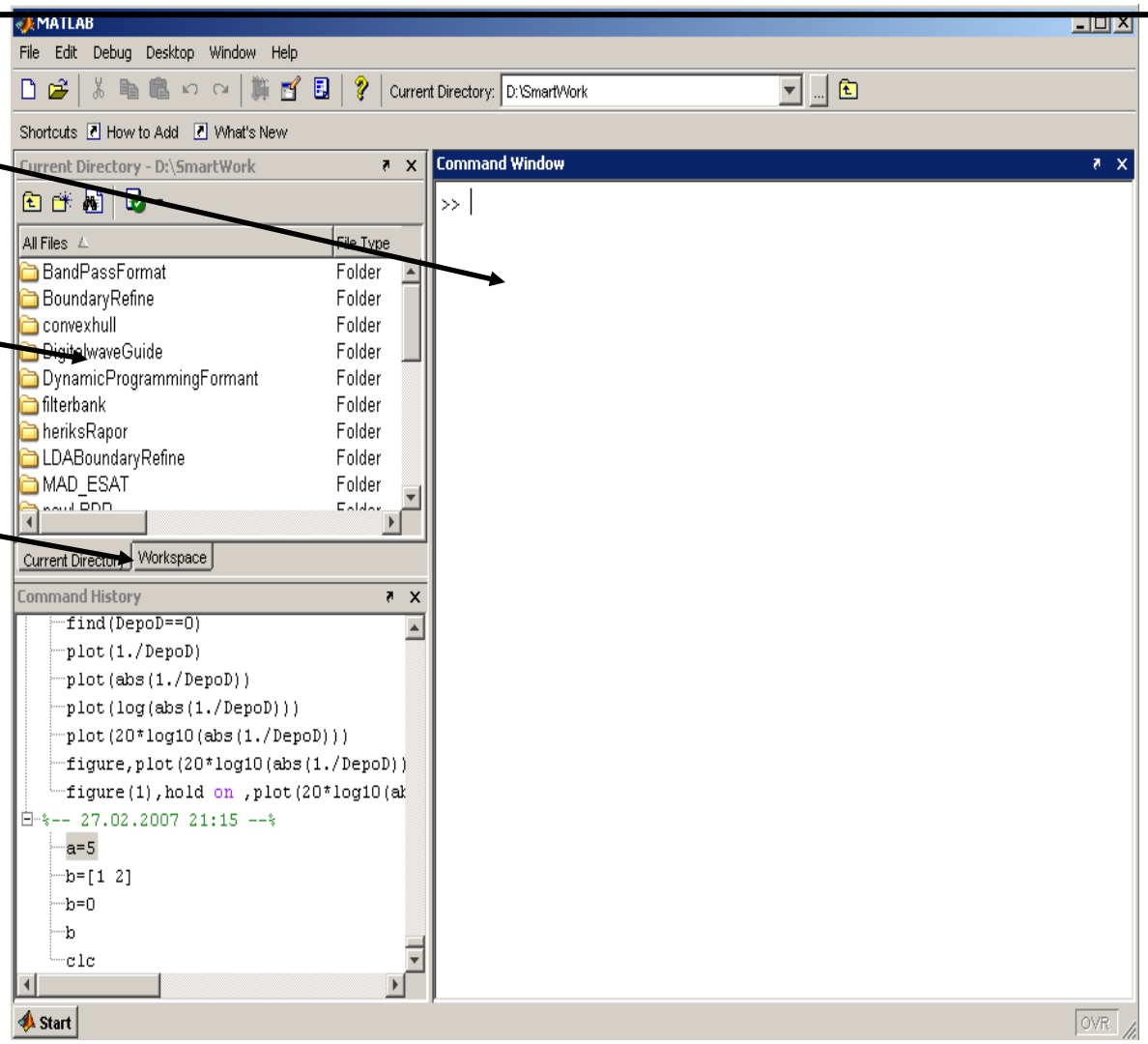
Understanding the MATLAB Environment, Continued…

- **Command History:** This panel shows or return commands that are entered at the command line.



MATLAB Screen

- Command Window
  - type commands
- Current Directory
  - View folders and m-files
- Workspace
  - View program variables
  - Double click on a variable to see it in the Array Editor
- Command History
  - view past commands
  - save a whole session using diary



Hands on Practice

- MATLAB environment behaves like a super-complex calculator. We can enter commands at the >> command prompt.
- MATLAB is an interpreted environment. In other words, we give a command and MATLAB executes it right away.
- Type a valid expression, for example, >>20 + 21; and press ENTER.
- When we click the Execute button, MATLAB executes it immediately and the result returned is ans=41.

Hands on Practice, Continued…

- Let us take up few more examples:
- >>3 ^ 2 (%3 raised to the power of 2). When we click the Execute button, MATLAB executes it immediately and the result returned is ans=9.
- >>sin(pi/2). (% sine of angle 90). When we click the Execute button, MATLAB executes it immediately and the result returned is ans=1.
- >>7/0 (% Divide by zero). When we click the Execute button, MATLAB executes it immediately and the result returned is ans=Inf.

Hands on Practice, Continued…

- >>732 * 20.3. When we click the Execute button, MATLAB executes it immediately and the result returned is ans=1.4860e+04.
- MATLAB provides some special expressions for some mathematical symbols, like pi for π, Inf for ∞, i (and j) for $\sqrt{-1}$.
- **Nan** stands for 'not a number'.

Use of Semicolon (;) in MATLAB

- Semicolon (;) indicates end of statement. However, if we want to suppress and hide the MATLAB output for an expression, add a semicolon after the expression.
- For example, >>x = 5; y = x + 5. When we click the Execute button, MATLAB executes it immediately and the result returned is y=8.

Adding Comments in MATLAB

- The percent symbol (%) is used for indicating a comment line.

  For example, x = 9 {% assign the value 9 to x}.
- We can also write a block of comments using the block comment operators % { and % }.
- The MATLAB editor includes tools and context menu items to help us add, remove, or change the format of comments.

Commonly used Operators and Special Characters

Operation	Symbol	Example
Addition	+	5+3
Subtraction	-	5-3
Multiplication	*	5*3
Right Division	\	5\3
Left Division	5\3	5\3=3\5
Exponentiation	^	5^3=125

Special Variables and Constants

MATLAB supports the following special variables and constants:

Name	Meaning
ans	Most recent answer.
eps	Accuracy of floating-point precision.
i,j	The imaginary unit √-1.
Inf	Infinity.
NaN	Undefined numerical result (not a number).
pi	The number π

Naming Variables

- Variable names consist of a letter followed by any number of letters, digits or underscore.
- MATLAB is **case-sensitive**.
- For example, >>x = 10 (% defining x and initializing it with a value). MATLAB will execute the above statement and return the following result x = 10.
- >>x = sqrt(25) (% defining x and initializing it with an expression) MATLAB will execute the above statement and return the result as x = 5.

Naming Variables, Continued…

- Note that once a variable is entered into the system, we can refer to it later. Variables must have values before they are used. When an expression returns a result that is not assigned to any variable, the system assigns it to a variable named answer, which can be used later.
- For example, >>x = 7 * 8; y = x * 7.89. MATLAB will execute the above statement and return the following result y = 441.84.

Multiple assignments

- We can have multiple assignments on the same line.
- For example,

```
>> a = 2; b = 7; c = a * b
```

- MATLAB will execute the above statement and return the result c = 14.

I have forgotten the Variables

- The **who** command displays all the variable names we have used.
- MATLAB will execute the above statement and return the result: Your variables are: ans - - - …
- The **whos** command. MATLAB will execute the above statement and return the following result.
- The **clear** command deletes all (or the specified) variable(s) from the memory.

I have forgotten the Variables

>> clear x

It will delete x, won't display anything.

>>clear

It will delete all variables in the workspace.

Long Assignments

- Long assignments can be extended to another line as follows:

```
>> initial_velocity = 0;
>> acceleration = 9.8;
>> time = 20;
>> Final_velocity = initial_velocity + acceleration * time
```

- MATLAB will execute the above statement and return the following result: final velocity = 196.

The format Command

- By default, MATLAB displays numbers with four decimal place values. This is known as **short format**.
- However, if we want more precision, then we need to use the **format** command.
- The **format long** command displays 16 digits after decimal.
- For example: >> format long

  >> x = 7 + 10/3 + 5 ^ 1.2
- MATLAB will execute the above statement and return the following result: x = 17.2319816406394.

The format Command, Continued…

Another example,

```
>> format short
>> x = 7 + 10/3 + 5 ^ 1.2
```

- MATLAB will execute the above statement and return the following result: x = 17.232
- The **format bank** command rounds numbers to two decimal places.

The format Command, Continued…

For example,

```
>> format bank
>> daily_wage = 177.45; weekly_wage = daily_wage * 6
```

- MATLAB will execute the above statement and return the following result: weekly_wage = 1064.70
- MATLAB displays large numbers using exponential notation.
- The **format short e** command allows displaying in exponential form with four decimal places plus the exponent.

The format Command, Continued…

- For example,

  >> format short e

  >> 4.678 * 4.9

- MATLAB will execute the above statement and return the following result: ans = 2.2922e+01
- The **format long e** command allows displaying in exponential form with four decimal places plus the exponent.

The format Command, Continued…

- For example, >> format long e
  >> x = pi
- MATLAB will execute the above statement and return the following result: x = 3.141592653589793e+00
- The **format rat** command gives the closest rational expression resulting from a calculation.
- For example, >> format rat
  >> 4.678 * 4.9
- MATLAB will execute the above statement and return the following result: ans = 34177/1491

Creating Vectors

- A vector is a one-dimensional array of numbers.
- MATLAB allows creating two types of vectors:
- Row vectors and Column vectors.
- **Row vectors** are created by enclosing the set of elements in square brackets, using space or comma.
- For example, >> X = [7 8 9 10 11]. MATLAB will execute the above statement and return the following result:

  X =

  7 8 9 10 11

Creating Vectors, Continued…

- Another example,

  >> X = [7 8 9 10 11]; >> Y = [2, 3, 4, 5, 6]; >> Z = X +Y

- MATLAB will execute the above statement and return the following result:

```
Z =

   9    11    13    15    17
```

Creating Vectors, Continued…

- **Column vectors** are created by enclosing the set of elements in square brackets, using semicolon(;).
- For example, >> X = [7; 8; 9; 10]
- MATLAB will execute the above statement and return the following result:

```
X =

     7

     8

     9

    10
```

Creating matrices

- A matrix is a two-dimensional array of numbers.
- In MATLAB, a matrix is created by entering each row as a sequence of space or comma separated elements, and end of a row is terminated by a semicolon.
- For example, let us create a 3-by-3 matrix as

```
>>A = [1 2 3 ; 4 5 6 ; 7 8 9]
```

Creating matrices, Continued…

- MATLAB will execute the above statement and return the following result:

```
A =

     1     2     3
     4     5     6
     7     8     9
```

Commands for Managing a Session

- MATLAB provides various commands for managing a session.

The following table provides all such commands:

Command	Purpose
clc	Clears command window.
clear	Removes variables from memory.
exist	Checks for existence of file or variable.
global	Declares variables to be global.
help	Searches for a help topic.
lookfor	Searches help entries for a keyword.
quit	Stops MATLAB.
who	Lists current variables.
whos	Lists current variables (long display).

Examples for Gauss Elimination

1) Solve the system of equations 2x + 4y + 6z = 14, 3x -2 y + z = -3 and 4x + 2y – z = -4 using Gaussian Elimination.

Matlab Code:

```
C = [2 4 6; 3 -2 1; 4 2 -1]
b= [14;-3;-4]
A = [C b]; %Augmented Matrix
n= size(A,1); %number of eqns/variables
x = zeros(n,1); %variable matrix [x1 x2 ... xn] coulmn
for i=1:n-1
for j=i+1:n
```

```
• m = A(j,i)/A(i,i)
• A(j,:) = A(j,:) - m*A(i,:)
• end
• end
• x(n) = A(n,n+1)/A(n,n)
• for i=n-1:-1:1
• summ = 0
• for j=i+1:n
• summ = summ + A(i,j)*x(j,:)
• x(i,:) = (A(i,n+1) - summ)/A(i,i)
• end
• end
```

Examples for Gauss Elimination Continuation.....

OUTPUT:

```
x =

    -1
     1
     2
```

Examples for Gauss Elimination

Matlab Code: (another way of solving)

```
function C = gauss_elimination(A,B) % defining the function
A= [2 4 6; 3 -2 1; 4 2 -1] % Inputting the value of coefficient matrix
B = [14;-3;-4] % % Inputting the value of coefficient matrix
i = 1; % loop variable
X = [ A B ];
[ nX mX ] = size( X); % determining the size of matrix
while i <= nX % start of loop
if X(i,i) == 0 % checking if the diagonal elements are zero or not
disp('Diagonal element zero') % displaying the result if there exists zero
return
end
```

Examples for Gauss Elimination Continuation.....

```
X = elimination(X,i,i); % proceeding forward if diagonal elements are non-zero
i = i +1;
end
C = X(:,mX);
function X = elimination(X,i,j)
% Pivoting (i,j) element of matrix X and eliminating other column
% elements to zero
[ nX mX ] = size( X);
a = X(i,j);
X(i,:) = X(i,:)/a;
for k = 1:nX % loop to find triangular form
if k == i
continue
end
X(k,:) = X(k,:) - X(i,:)*X(k,j); % final result
end
```

Examples for Gauss Elimination Continuation.....

OUTPUT:

```
x =

    -1
     1
     2
```

Solve the following system of equations by Gaussian Elimination.

$2x + 5y + z = 0$, $4x + 8y + z = 2$, $y - z = 3$

1. 2. $2x + 3y + z = 8$, $4x + 7y + 5z = 20$, $-2y + 2z = 0$

3. $2x - 3y = 3$, $4x - 5y + z = 7$, $2x - y - 3z = 5$.

THANK YOU

MATLAB

Gaussian Elimination

- Given a system of n equations in n unknowns we use the method of pivoting to solve for the unknowns. The method starts by subtracting multiples of the first equation from the other equations.
- The aim is to eliminate the first unknown from the second equation onwards. We use the coefficient of the first unknown in the first equation as the first pivot to achieve this. At the end of the first stage of elimination, there will be a column of zeros below the first pivot.
- Next, the pivot for the second stage of elimination is located in the second row second column of the system. A multiple of the second equation will now be subtracted from the remaining equations using the second pivot to create zeros just below it in that column.
- The process is continued until the system is reduced to an upper triangular one. The system can now be solved backward bottom to top.

Gaussian Elimination

1. Solve the following system of equations using Gaussian Elimination
   $2x + y - z = 8, -3x - y + 2z = -11 \; and \; -2x + y + 2z = -3$

Matlab Code:

```
A = [2 1 -1; -3 -1 2; -2 1 2];        % Coefficient matrix
b = [8; -11; -3];                     % Right-hand side vector
augMatrix = [A b];                    % Augment the matrix

% Forward elimination
n = size(augMatrix, 1);
for i = 1:n-1
    for j = i+1:n
        factor = augMatrix(j, i) / augMatrix(i, i);
        augMatrix(j, :) = augMatrix(j, :) - factor * augMatrix(i, :);
    end
end
```

Gaussian Elimination

```
% Back substitution
x = zeros(n, 1);
x(n) = augMatrix(n, end) / augMatrix(n, n);
for i = n-1:-1:1
    x(i) = (augMatrix(i, end) - augMatrix(i, i+1:n) * x(i+1:n)) / augMatrix(i, i);
end

% Display the solution
disp('Solution is :');
disp(x);
```

Gaussian Elimination

Output

Solution is :

2

3

-1

Gaussian Elimination

Practice Problems:

A = [1 1 1; 2 -6 -1; 3 4 2]

b= [11 0 0]

Ans:: x=-8, y=-7, z=26

A = [2 1 -1; 2 5 7; 1 1 1]

b= [0 52 9]’

Ans: x=1, y=3, z=5

THANK YOU

Gauss - Jordan Method To find Inverse

Find by Gauss Jordan Method:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{bmatrix}$$

```
A =[1,1,1;4,3,-1;3,5,3];
n =length(A(1,:));
Aug =[A,eye(n,n)]
for j=1:n-1
for i=j+1:n
Aug(i,j:2*n)=Aug(i,j:2*n)-
Aug(i,j)/Aug(j,j)*Aug(j,j:2*n)
end
end
for j=n:-1:2
Aug(1:j-1,:)=Aug(1:j-1,:)-Aug(1:j-
1,j)/Aug(j,j)*Aug(j,:)
end
```

Gauss - Jordan Method To find Inverse

```
for j=1:n
Aug(j,:)=Aug(j,:)/Aug(j,j)
end
B=Aug(:,n+1:2*n)
```

Gauss - Jordan Method To find Inverse

OUTPUT:

Aug =

1 1 1 1 0 0
4 3 -1 0 1 0
3 5 3 0 0 1

Aug =

1 1 1 1 0 0
0 -1 -5 -4 1 0
3 5 3 0 0 1

Aug =

1 1 1 1 0 0
0 -1 -5 -4 1 0
0 2 0 -3 0 1

Aug =

1 1 1 1 0 0
0 -1 -5 -4 1 0
0 0 -10 -11 2 1

Aug =

1.0000 1.0000 0 -0.1000 0.2000 0.1000
0 -1.0000 0 1.5000 0 -0.5000
0 0 -10.0000 -11.0000 2.0000 1.0000

Aug =

1.0000 0 0 1.4000 0.2000 -0.4000
0 -1.0000 0 1.5000 0 -0.5000
0 0 -10.0000 -11.0000 2.0000 1.0000

Aug =

1.0000 0 0 1.4000 0.2000 -0.4000
0 -1.0000 0 1.5000 0 -0.5000
0 0 -10.0000 -11.0000 2.0000 1.0000

Aug =

1.0000 0 0 1.4000 0.2000 -0.4000
0 1.0000 0 -1.5000 0 0.5000
0 0 -10.0000 -11.0000 2.0000 1.0000

Aug =

1.0000 0 0 1.4000 0.2000 -0.4000
0 1.0000 0 -1.5000 0 0.5000
0 0 1.0000 1.1000 -0.2000 -0.1000

B =

1.4000 0.2000 -0.4000
-1.5000 0 0.5000
1.1000 -0.2000 -0.1000

Gauss - Jordan Method To find Inverse

Practice Problems:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 4 \\ 1 & 1 & 5 \end{bmatrix}$$

$$A = \begin{bmatrix} -1 & 2 & 6 \\ -1 & -2 & 4 \\ -1 & 1 & 5 \end{bmatrix}$$

LU Decomposition Method:

```
%LU Decomposition
Ab = [1 1 -1;3 5 6;7 8 9];
%% Forward Elimination
n= length(A);
L = eye(n);
% With A(1,1) as pivot Element
for i =2:3
alpha = Ab(i,1)/Ab(1,1);
L(i,1) = alpha;
Ab(i,:) = Ab(i,:) - alpha*Ab(1,:);
end
% With A(2,2) as pivot Element
i=3;
alpha = Ab(i,2)/Ab(2,2);
L(i,2) = alpha
Ab(i,:) = Ab(i,:) - alpha*Ab(2,:);
U = Ab(1:n,1:n)
```

OUTPUT

L =

1.0000	0	0
6.0000	1.0000	0
9.0000	1.0909	1.0000

U =

1	2	3
0	-11	-11
0	0	-17

LU Decomposition Method:

Practice Problems:

$$A = \begin{bmatrix} 1 & 1 & 3 \\ 1 & 2 & 4 \\ 1 & 1 & 5 \end{bmatrix}$$

$$A = \begin{bmatrix} -1 & 4 & 6 \\ 0 & -2 & 4 \\ 0 & 0 & 5 \end{bmatrix}$$

```
clc;
clear all;
close all;

% Bases of four fundamental vector spaces of matrix A.

A=[1,2,3;2,-1,1];

% Row Reduced Echelon Form
[R, pivot] = rref(A)

% Rank
rank = length(pivot)

% basis of the column space of A
columnsp = A(:,pivot)

% basis of the nullspace of A
nullsp = null(A,'r')

% basis of the row space of A
rowsp = R(1:rank,:)'

% basis of the left nullspace of A
leftnullsp = null(A','r')
```

```
R =

     1     0     1
     0     1     1


pivot =

     1     2


rank =

     2


columnsp =

     1     2
     2    -1


nullsp =

    -1
```

```
    -1
     1


rowsp =

     1     0
     0     1
     1     1


leftnullsp =

  2×0 empty double matrix
```

Published with MATLAB® R2021b

DATA STRUCTURES AND ITS APPLICATIONS UE21MA241B

Dr. Roopa Ravish
Department of Computer Science & Engineering

LINEAR ALGEBRA

Pseudo Inverse

Dr. Roopa Ravish
Department of Computer Science & Engineering

Pseudo Inverse

The Pseudo-inverse A^+ of a mxn matrix A is an extension of the inverse of a square matrix to non-square matrices and to singular(non-invertible) square matrices.

The Pseudo-inverse matrix A^+ is a nxm matrix with the following properties:

1) If $m \geq n$, then A^+A is invertible and $A^+=(A^TA)^{-1}A^T$ and so $A^+A=I$, A^+ is left inverse of A.
2) If $m \leq n$, then AA^+ is invertible and $A^+=A^T(AA^T)^{-1}$ and so $AA^+=I$, A^+ is right inverse of A.

Pseudo Inverse

In other words,

Case 1: If $\rho(A) = n$ (n is the number of columns), Ax = b has at most one solution x for every b if and only if the columns are linearly independent. Then A will have left inverse of order n x m such that $B_{nxm}A_{mxn}=I_{nxn}$
Thus, $B=(A^TA)^{-1}A^T$

Case 2: If $\rho(A) = m$ (m is the number of rows), Ax = b has at least one solution x for every b if and only if the columns span R^m. Then A will have right inverse of order m x n such that $A_{mxn}C_{nxm}=I_{mxm}$
Thus, $C=A^T(AA^T)^{-1}$

Pseudo Inverse

Example 1:

$$A=\begin{bmatrix}2 & 0 & 0\\ 0 & 4 & 0\end{bmatrix}_{2x3}$$

Rank r=2=m (m<n)

$$C=A^T(AA^T)^{-1} = \begin{bmatrix}2 & 0\\ 0 & 4\\ 0 & 0\end{bmatrix}_{3x2} \begin{bmatrix}1/4 & 0\\ 0 & 1/16\end{bmatrix}_{2x2} = \begin{bmatrix}1/2 & 0\\ 0 & 1/4\\ 0 & 0\end{bmatrix}_{3x2}$$

Pseudo Inverse

Example 2:

$$A=\begin{bmatrix}1 & 0\\ 1 & 1\\ 0 & 1\end{bmatrix}_{3x2}$$

Rank r=2=n (n<m)

$$B=(A^TA)^{-1}A^T=\begin{bmatrix}2/3 & -1/3\\ -1/3 & 2/3\end{bmatrix}_{2x2}\begin{bmatrix}1 & 1 & 1\\ 0 & 1 & 1\end{bmatrix}_{2x3}=\begin{bmatrix}2/3 & 1/3 & -1/3\\ -1/3 & 1/3 & 2/3\end{bmatrix}_{2x3}$$

Pseudo Inverse

Matlab code using in-built function:

```
% Define a matrix
A = [1, 2, 3; 4, 5, 6];

% Compute the pseudo-inverse of A
A_pseudo_inv = pinv(A);

% Display the pseudo-inverse
disp('Pseudo-inverse of A:')
disp(A_pseudo_inv)
```

THANK YOU

Dr. Roopa Ravish

Department of Computer Science & Engineering

rooparavish@pes.edu

ENGINEERING MATHEMATICS-IV
LINEAR ALGEBRA
MATLAB

PES UNIVERSITY ONLINE

Department of Science and Humanities

Projection matrices and least squares:

Projections:

We know that $P = A(A^TA)^{-1}A^T$ is a matrix that projects a vector b onto the space spanned by the columns of A. If b is perpendicular to the column space, then it's in the left nullspace $N(A^T)$ of A and Pb = 0. If b is in the column space then b = Ax for some x, and Pb = b.

A typical vector will have a component p in the column space and a component e perpendicular to the column space (in the left nullspace); its projection is just the component in the column space. The matrix projecting b onto $N(A^T)$ is I − P:

e = b − p

e = (I − P)b.

Naturally, I − P has all the properties of a projection matrix.

Projection matrices and least squares:

Least squares:

We want to find the closest line b = C + Dt to the points (1, 1), (2, 2), and (3, 2). The process we're going to use is called linear regression; this technique is most useful if none of the data points are outliers. By "closest" line we mean one that minimizes the error represented by the distance from the points to the line. We measure that error by adding up the squares of these distances. In other words, we want to minimize $\|Ax-b\|^2 = \|e\|^2$.

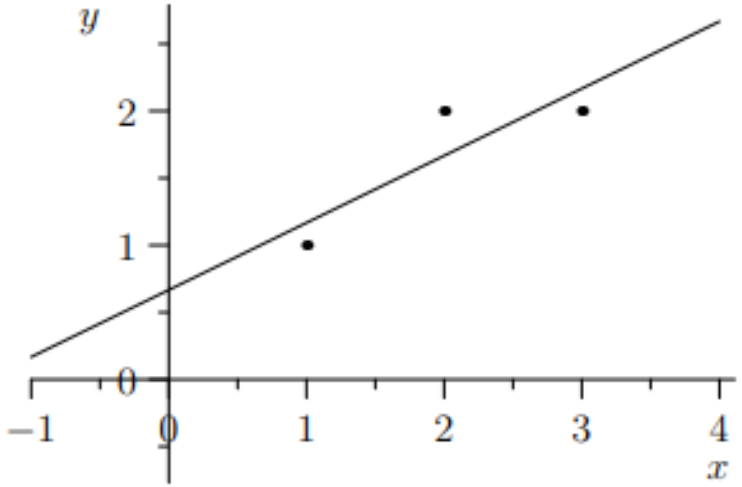


Figure 1: Three points and a line close to them.

Projection matrices and least squares:

Find the projection for the matrix $A=\begin{pmatrix}1 & 0\\ 0 & 1\\ 1 & 1\end{pmatrix}$ **;** $x=\begin{pmatrix}u\\ v\end{pmatrix}$ **and** $b=\begin{pmatrix}1\\ 3\\ 4\end{pmatrix}$.

Code:

```
A=[1,0;0,1;1,1]
b=[1;3;4]
x = lsqr(A,b)
```

Projection matrices and least squares:

Output:

```
A =
     1   0
     0   1
     1   1
b =
     1
     3
     4
lsqr converged at iteration 2 to a solution with relative residual 6.7e-17.
x =
    1.0000
    3.0000
```

Projection matrices and least squares:

Find the projection for the matrix $A=\begin{pmatrix}1 & 0\\ 0 & 2\\ 3 & 1\end{pmatrix}$; $x=\begin{pmatrix}u\\ v\end{pmatrix}$ **and**

$$b=\begin{pmatrix}1\\ 0\\ 4\end{pmatrix}.$$

Code:

```
A=[1,0;0,2;3,1]
b=[1;0;4]
x = lsqr(A,b)
```

Projection matrices and least squares:

OUTPUT:

```
A =
        1        0
        0        2
        3        1
b =
        1
        0
        4
lsqr converged at iteration 2 to a solution with relative residual 0.076.
x =
        1.2927
        0.0244
```

Projection matrices and least squares:

Find the point on a plane x+y-z=0 that is closest to (2,1,0)

Code:

```
syms   c
P=[2,1,0]+c*[1,1,-1]
s=1*(c+2)+1*(c+1)-1(-c)==0
s1=solve(s,c)
p=[2,1,0]+s1*[1,1,-1]
```

Projection matrices and least squares:

Output

P =

[3*c + 1, 4*c, c + 1]

s =

26*c + 4 == 1

s1 =

-3/26

p =

[17/26, -6/13, 23/26]

Projection matrices and least squares:

Find the point on a plane 3x+4y+z=1 that is closest to (1,0,1)

Code:

```
syms   c
P=[1,0,1]+c*[3,4,1]
s=3*(1+3*c)+4*(4*c)+(1+c)==1
s1=solve(s,c)
p=[1,0,1]+s1*[3,4,1]
```

Projection matrices and least squares:

Output:

P =

[c + 2, c + 1, -c]

s =

3*c + 3 == 0

s1 =

-1

p =

[1, 0, 1]

Projection matrices and least squares:

Let $u = \begin{bmatrix} 1 \\ 7 \end{bmatrix}$ onto $v = \begin{bmatrix} -4 \\ 2 \end{bmatrix}$ and find P, the matrix that will project any matrix onto the vector v. Use the result to find proj_v u.

Code:

```
u=[1;7]
u =
     1
     7
v=[-4;2]
v =   -4
      2
```

Projection matrices and least squares:

P=(v*transpose(v))/(transpose(v)*v)

P =

```
    0.8000   -0.4000
   -0.4000    0.2000
```

P*u

ans =

```
   -2
    1
```

Projection matrices and least squares:

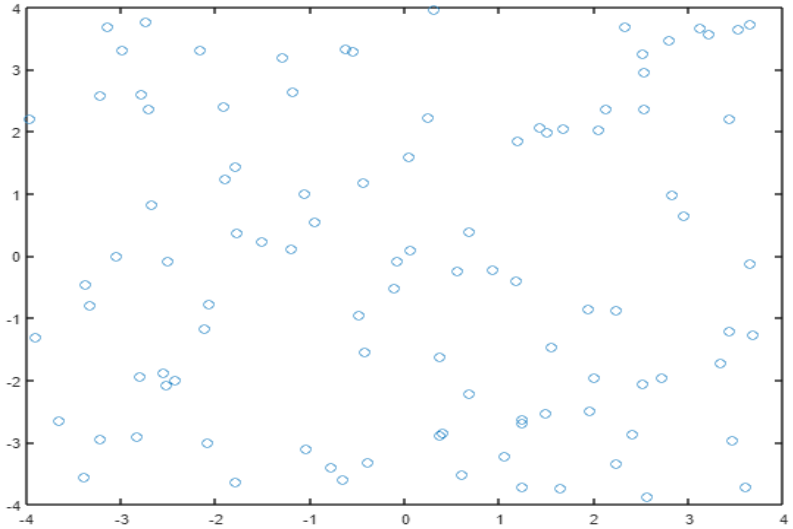
Projecting a lot of vector on a single vector:

Code:

```
u=8*rand(2,100)-4;
x=u(1,: )
y=u(2,: )
plot(x,y,'o')
```

In the below figure I have generated a 100 random vectors.

Projection matrices and least squares:



In this figure each circle represents the tip of a vector whose tail begins at the origin.

Next , I will take the projection matrix P to project each of the 100 2 by 1 vectors in matrix U onto the vector v,

Projection matrices and least squares:

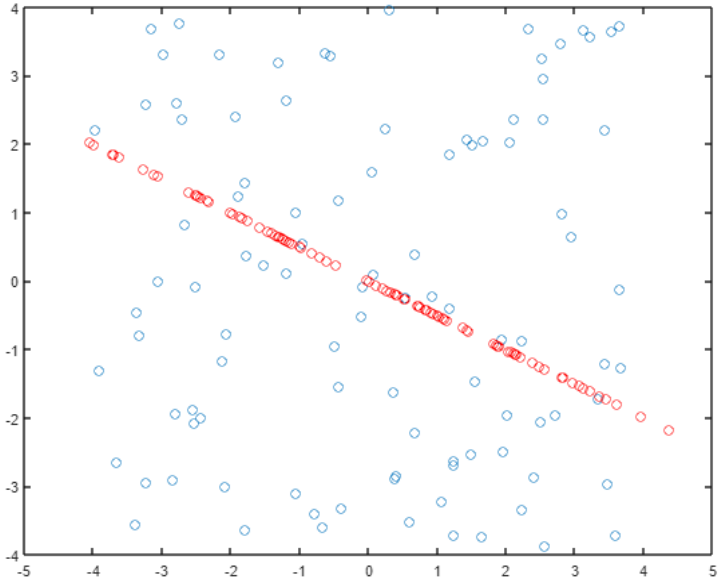
Code:

```
>> P=[0.8,-0.4;-0.4,0.2]

P =
        0.8000         -0.4000
        -0.4000         0.2000

>> Pu=P*u;
x=Pu(1,:)
y=Pu(2,:)
hold on
plot(x,y,'ro‘)
```

Projection matrices and least squares:



Here each vector in the matrix u is projected onto a line in the direction of the vector v=[-1;2].

Projection matrices and least squares:

Example Problems:

1. **Find the least square fit for this system**

$$x + 2y = 3$$
$$3x + 2y = 5$$
$$x + y = 2.09$$

2. **Find the point on a plane 13x+4y+z=1 that is closest to (1,-1,1)**

Projection matrices and least squares:

Example Problems:

3. Find the least square fit for this system

$$x + 2y + z = 3$$
$$3x + 2y - 2z = 5$$
$$x + y + 7z = 21.09$$

THANK YOU

PES UNIVERSITY ONLINE

ENGINEERING MATHEMATICS-I
MATLAB

Department of Science and Humanities

Grams- Schmidt in 9 Lines of MATLAB.

The Gram-Schmidt algorithm starts with n independent vectors a1,….,an (the columns of A). It produces n orthonormal vectors q1,…,qn (the columns of Q). To find qj, start with aj and subtract off its projections onto the previous q's and then divide by the length of that vector v to produce a unit vector.

The inner products (qi)T aj =0 when i is larger than j (later q's are orthogonal to earlier a's, that is the point of the algorithm).

Grams- Schmidt in 9 Lines of MATLAB.

Here is a 9-line MATLAB code to build Q and R from A. Start with [m,n]=size(A); Q=zeros(m,n); R=zeros(n,n);to get the shapes correct.

```
>> for j=1:n                    % Grams-Schmidt orthogonalization
>> v=A(: , j);                  % v begins as column j of A
>> for i=1:j-1
>> R(i,j)=Q(:,i)'*A(:,j);       % modify A(:,j) to v for more accuracy
>> v=v-R(i,j)*Q(:,i);           % subtract the projectior (q_i^T a_j)q_i = (q_i^T v)q_i
>> end                          % v is now perpendicular to all of q1,...qj-1
>> R(j,j)=norm(v);
>> Q(:,j)=v/R(j,j);             % normalize v to be the next unit vector qj
>> end
```

Grams- Schmidt Orthogonalization process continued..

Example:

Apply the Gram-Schmidt process to the vectors (1,0,1), (1,0,0) and (2,1,0) to produce a set of Orthonormal vectors.

```
>> A=[1,1,2;0,0,1;1,0,0]
>> Q=zeros(3)
>> R=zeros(3)
>> for j=1:3
>> v=A(: , j)
>> for i=1:j-1
>> R(i,j)=Q(:,i)'*A(:,j)
>> v=v-R(i,j)*Q(:,i)
>> end
>> R(j,j)=norm(v)
>> Q(:,j)=v/R(j,j)
>> end
```

Grams- Schmidt Orthogonalization process continued..

Output:

```
V =

   -0.0000
    1.0000
    0.0000


R =

    1.4142    0.7071    1.4142
         0    0.7071    1.4142
         0         0    1.0000
```

Grams- Schmidt Orthogonalization process continued..

.Output:

```
Q =

    0.7071    0.7071   -0.0000
         0         0    1.0000
    0.7071   -0.7071    0.0000
```

Grams- Schmidt Orthogonalization process continued..

2. Apply the Gram-Schmidt process to the vectors a=(0,1,1,1), b=(1,1,-1,0) and c=(1,0,2,-1).

```
>> A=[0,1,1;1,1,0;1,-1,2;1,0,-1]
>> Q=zeros(4,3)
>> R=zeros(3)
>> for j=1:3
>> v=A(: , j);
>> For i=1:j-1
>> R(i,j)=Q(:,i)'*A(:,j)
>> v=v-R(i,j)*Q(:,i)
>> end
>> R(j,j)=norm(v)
>> Q(:,j)=v/R(j,j)
>> end
```

Grams- Schmidt Orthogonalization process continued..

Output:

```
v =

    1.3333
         0
    1.3333
   -1.3333


R =

    1.7321         0    0.5774
         0    1.7321   -0.5774
         0         0    2.3094
```

Grams- Schmidt Orthogonalization process continued..

```
Q =

         0    0.5774    0.5774
    0.5774    0.5774         0
    0.5774   -0.5774    0.5774
    0.5774         0   -0.5774
```

PES
UNIVERSITY
ONLINE

THANK YOU

ENGINEERING MATHEMATICS-IV
LINEAR ALGEBRA
MATLAB

PES UNIVERSITY ONLINE

Department of Science and Humanities

QR Decomposition with Gram-Schmidt

The QR decomposition (also called the QR factorization) of a matrix is a decomposition of the matrix into an orthogonal matrix and a triangular matrix. A QR decomposition of a real square matrix A is a decomposition of A as

A = QR,

where Q is an orthogonal matrix (i.e. $Q^T Q = I$) and R is an upper triangular matrix. If A is nonsingular, then this factorization is unique. There are several methods for actually computing the QR decomposition. One of such method is the Gram-Schmidt process.

QR Decomposition with Gram-Schmidt

Gram-Schmidt process:

Consider the GramSchmidt procedure, with the vectors to be considered in the process as columns of the matrix A. That is,

$$A = \left[\mathbf{a}_1 \mid \mathbf{a}_2 \mid \cdots \mid \mathbf{a}_n \right].$$

Then,

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{a}_1, \qquad \mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \\
\mathbf{u}_2 &= \mathbf{a}_2 - (\mathbf{a}_2 \cdot \mathbf{e}_1)\mathbf{e}_1, \qquad \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}. \\
\mathbf{u}_{k+1} &= \mathbf{a}_{k+1} - (\mathbf{a}_{k+1} \cdot \mathbf{e}_1)\mathbf{e}_1 - \cdots - (\mathbf{a}_{k+1} \cdot \mathbf{e}_k)\mathbf{e}_k, \qquad \mathbf{e}_{k+1} = \frac{\mathbf{u}_{k+1}}{\|\mathbf{u}_{k+1}\|}.
\end{aligned}$$

Note that $\|\cdot\|$ is the L_2 norm.

QR Decomposition with Gram-Schmidt

QR factorization:

The resulting QR factorization is

$$A = \left[\begin{array}{c|c|c|c} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{array}\right] = \left[\begin{array}{c|c|c|c} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \end{array}\right] \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{e}_1 & \mathbf{a}_2 \cdot \mathbf{e}_1 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_1 \\ 0 & \mathbf{a}_2 \cdot \mathbf{e}_2 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_n \end{bmatrix} = QR.$$

Note that once we find $\mathbf{e}_1, \ldots, \mathbf{e}_n$, it is not hard to write the QR factorization.

QR Decomposition with Gram-Schmidt

Find QR factorization of the matrix

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

Code:

```
A=[1,1,0;1,0,1;0,1,1]
[Q,R]=qr(A)

Or

[Q,R]=qr([1,1,0;1,0,1;0,1,1])
```

QR Decomposition with Gram-Schmidt

Output:

Q =

-0.7071	0.4082	-0.5774
-0.7071	-0.4082	0.5774
0	0.8165	0.5774

R =

-1.4142	-0.7071	-0.7071
0	1.2247	0.4082
0	0	1.1547

QR Decomposition with Gram-Schmidt

QR Factorization of Pascal Matrix

```
>>A = sym(pascal(3))
```

Output

```
A =
        [ 1, 1, 1]
        [ 1, 2, 3]
        [ 1, 3, 6]
```

```
[Q,R] = qr(A)
```

QR Decomposition with Gram-Schmidt

Output

Q =

[3^(1/2)/3, -2^(1/2)/2, 6^(1/2)/6]
[3^(1/2)/3, 0, -6^(1/2)/3]
[3^(1/2)/3, 2^(1/2)/2, 6^(1/2)/6]

R =

[3^(1/2), 2*3^(1/2), (10*3^(1/2))/3]
[0, 2^(1/2), (5*2^(1/2))/2]
[0, 0, 6^(1/2)/6]

QR Decomposition with Gram-Schmidt

isAlways(A == Q*R)

Output

ans =

3×3 logical array

1	1	1
1	1	1
1	1	1

QR Decomposition with Gram-Schmidt

QR Decomposition to Solve Matrix Equation of the form Ax=b

```
A = sym(invhilb(5))
b = sym([1:5]‘)
```

Output:

```
A =
        [ 25, -300, 1050, -1400, 630]
        [ -300, 4800, -18900, 26880, -12600]
        [ 1050, -18900, 79380, -117600, 56700]
        [ -1400, 26880, -117600, 179200, -88200]
        [ 630, -12600, 56700, -88200, 44100]
b =
        1; 2; 3; 4 ;5
```

QR Decomposition with Gram-Schmidt

```
[C,R] = qr(A,b);
X = R\C
```

Output:

```
X =
          5
         71/20
         197/70
        657/280
         1271/630
```

QR Decomposition with Gram-Schmidt

isAlways(A*X == b)

Output:

```
ans =
          5×1 logical array
          1
          1
          1
          1
          1
```

QR Decomposition with Gram-Schmidt

Example problems:

Find QR Decomposition (Gram Schmidt Method) ...

Using MATLAB command

Example 1

$$\begin{bmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{bmatrix}$$

QR Decomposition with Gram-Schmidt

Example problems:

Example 2:

$$\begin{bmatrix} 3 & 2 & 4 \\ 2 & 0 & 2 \\ 4 & 2 & 3 \end{bmatrix}$$

QR Decomposition with Gram-Schmidt

Example problems:

Example 3:

$$\begin{bmatrix} 1 & 2 & 4 \\ 3 & 8 & 9 \\ 5 & 7 & 3 \end{bmatrix}$$

THANK YOU

PES UNIVERSITY ONLINE

DATA STRUCTURES AND ITS APPLICATIONS UE21MA241B

Dr. Roopa Ravish
Department of Computer Science & Engineering

LINEAR ALGEBRA

Singular Value Decomposition (SVD)

Dr. Roopa Ravish
Department of Computer Science & Engineering

Singular Value Decomposition (SVD)

Compute SVD for a matrix

Case 1: Matrix A is a short matrix (eg: 2x3)

Step 1: Find AA^T (2x2)

Step 2: Find eigenvalues of AA^T : λ_1 and λ_2

Step 3: Find corresponding eigenvectors: x_1 and x_2

Step 4: Normalise them to get u_1 and u_2 such that $U=[u_1, u_2]$ (2x2)

Step 5: Find singular values $\sigma_1 = \sqrt{\lambda_1}$ and $\sigma_2 = \sqrt{\lambda_2}$ such that $(\lambda_1 > \lambda_2)$

$$\sum = \begin{bmatrix} \sqrt{\lambda_1} & 0 & 0 \\ 0 & \sqrt{\lambda_2} & 0 \end{bmatrix}$$

Singular Value Decomposition (SVD)

Step 6: No need to find A^TA . Eigenvalues of A^TA are λ_1 , λ_2 and 0

Step 7: Use formula $v_i = \frac{A^T u_i}{\sigma_i}$ or $v_i^T = \frac{u_i^T A}{\sigma_i}$ and find v_1 and v_2

Step 8: Find v_3 using orthogonality (v_3 is orthogonal to v_1 and v_2)

$$\begin{bmatrix} - & v_1^T & - \\ - & v_2^T & - \end{bmatrix} \begin{bmatrix} | \\ x \\ | \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Step 9: Matrix $V = [v_1 \quad v_2 \quad v_3]_{(3x3)}$

Step 10: Write $A = U \sum V^T$

Singular Value Decomposition (SVD)

Case 2: Matrix A is a tall matrix (eg: 3x2)

Step 1: Find $A^TA(2x2)$

Step 2: Find eigenvalues of A^TA : λ_1 and λ_2

Step 3: Find corresponding eigenvectors: x_1 and x_2

Step 4: Normalise them to get v_1 and v_2 such that $V=[v_1, v_2]_{(2x2)}$

Step 5: Find singular values $\sigma_1 = \sqrt{\lambda_1}$ and $\sigma_2 = \sqrt{\lambda_2}$ such that $(\lambda_1 > \lambda_2)$

$$\Sigma = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \\ 0 & 0 \end{bmatrix}$$

Singular Value Decomposition (SVD)

Step 6: No need to find AA^T . Eigenvalues of AA^T are λ_1 , λ_2 and 0

Step 7: Use formula $u_i = \frac{Av_i}{\sigma_i}$ and find u_1 and u_2

Step 8: Find u_3 using orthogonality (u_3 is orthogonal to u_1 and u_2)

Step 9: Matrix U = $[u_1 \quad u_2 \quad u_3]_{(3x3)}$

Step 10: Write $A= U\sum V^T$

PES UNIVERSITY
CELEBRATING 50 YEARS

Case 3: Matrix A is a square matrix (eg: 2x2)

Step 1: Find if $A=A^T$ (symmetric) and then check if A is positive definite

Step 2: If A is positive definite, SVD is same as diagonalization

$A = U\sum V^T = Q\sum Q^T$ (Q is eigenvector matrix (orthonormal))

Step 3: If A is not positive definite, find eigenvectors of AA^T (columns of U) and A^TA (columns of V).

Step 4: Follow the steps to find U, $\sum$ and V from case 1 and case 2

LINEAR ALGEBRA

Singular Value Decomposition (SVD)

Use in-built function in Matlab:

```
A=[2 3 4; 4 5 6];
[U, S, V] = svd(A)
```

Output:

```
U =
-0.5221 -0.8529
-0.8529  0.5221

S =
10.2846       0       0
     0   0.4763       0

V =
-0.4332  0.8035  0.4082
-0.5669  0.1092 -0.8165
-0.7006 -0.5852  0.4082
```

THANK YOU

Dr. Roopa Ravish

Department of Computer Science & Engineering

rooparavish@pes.edu