UNIT-IV

# APPLICATIONS

## 5.1 LINEAR ALGEBRA

### 5.1.1 Solving a linear system

Solving set of linear algebraic equations is easy in MATLAB.

We will solve a set of given linear algebraic equations. To solve these equations, no prior knowledge of matrix methods is required. The first two steps outlined below are really basic for most people who know a little bit of linear algebra. We will consider the following of equations for our example.

$$5x = 3y - 2z + 10$$
$$8y + 4z = 3x + 20$$
$$2x + 4y - 9z = 9$$

**Step 1** Rearrange Equations:

Write each equation with all unknown quantities on the left-hand side and all known quantities on the right hand side. Thus, for the preceding equations, rearrange them such that all terms involving x, y and z are on the left side of equal sign.

$$\left.\begin{aligned} 5x - 3y + 2z &= 10 \\ -3x + 8y + 4z &= 20 \\ 2x + 4y - 9z &= 9 \end{aligned}\right\} \quad \text{---(1)}$$

**Step 2** Write the equations in Matrix form:

To write the equation in the matrix form $[A]\{x\} = \{b\}$ where $\{x\}$ is the unknown vectors, you have to arrange the unknowns in vector X, the co-efficients of the unknown matrix A, and the constants on the right-hand

of the equations in vector b. In this particular example, the unknown vector is

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

the co-efficient matrix is

$$A = \begin{bmatrix} 5 & -3 & 2 \\ -3 & 8 & 4 \\ 2 & 4 & -9 \end{bmatrix}$$

and the known constant vector is $b = \begin{bmatrix} 10 \\ 20 \\ 9 \end{bmatrix}$.

Step 3 solve the matrix equation in MATLAB

Enter the matrix A and the vector b and solve for vector x with $x = A \backslash b$

```
>>A = [5 -3 2; -3 8 4; 2 4 -9];
                    % Enter matrix A
>>b = [10; 20; 9]; % Enter column vector b
>>x = A\b           % Solve for x
x =
    3.4442
    3.1982
    1.1868
```

The backslash (\) or the left division is used to solve a linear system of equations $[A]\{x\} = \{b\}$. For more information, type: help slash

```
>>C = A*x           % Check the solution
C =
   10.0000
   20.0000
    9.0000
```

You can also use a poweful function, linsolve, to solve linear system if your matrix A has

exploitable structure.
[ex; triangular, positive definite].

5.1.2 Gaussian Elimination:

In introductory linear algebra, we learn to solve a system of linear algebraic equations by Gaussian Elimination. This technique requires forming a rectangular matrix that contains both the co-efficient matrix A and the known vector b in an argument matrix.

Gauss-Jordan reduction procedure is then used to transform the argumented matrix to the so-called row-reduced echelon form. MATLAB has a built in function rref that does precisely this reduction. ie) transforms the matrix to its row reduced echelon form. For example, consider equ(1) with the co-efficient matrix A and the vector b created previously. To solve the equations with rref, type the following commands

```
C = [A b]; % form the augmented matrix
Cr = rref(C); % row reduce the augmented matrix.
```

The last column of Cr is the solution x.

5.1.3 Finding Eigenvalues and Eigen Vectors

The omnipresent eigenvalue problem in scientific computation show up as

$$Av = \lambda v \quad \text{---(2)}$$

$\lambda$ is scalar. The problem is to find $\lambda$ and v for a given A so that equ(2) is satisfied.

```
>> lam 2 = D(2,2); % extract the second
                    eigenvalue from D
>> A * V2 = lam 2 * V2 % check the difference
                    between A*v and lambda*v
A*V(:,2) - D(2,2) * V(:,2) the last
command may not result into zero vector,
as you except but it will have very small number.
```

## 5.2 Curve Fitting and Interpolation

### 5.2.1 Polynomial Curve fitting on the fly

Curve fitting is a technique of finding an algebraic relationship that "best" fits a given set of data.

You have to have an idea of what kind of relationship might exist between the input data (xi) and the output data (yi).

However, if you not have a firm idea but you have data that your trust, MATLAB can help you explore the best possible fit.

**Example:** Straight-line (linear) fit

Let us say that we have the following data for x and y and we want to get the best linear (Straight-line) fit through this data.

| x | 5 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|
| y | 15 | 33 | 53 | 140 | 301 |

Here is all it takes to get the best linear fit along with the equation of the fitted line.

Example 2 Comparing different fits

Let us take another example where we try two different fits, quadratic and cubic for the same data and do a comparison to figure out which one is better. We first create some x and y data

```
x = 0 : pi/30 : pi/3;          % x-data
y = sin(x) + rand(size(x))/100 % y-data (corrupted sine)
```

Step 1 Plot raw data

We already have the x and y data. So, go ahead and plot the raw data with plot(x,y,'o'). Step 2 Use Basic Fitt to do a quadratic & a cubic fit

Go to your figure window, click on Tools, and select Basic Fitting from the pull-down menu. In the Basic Fitting window, check quadratic and cubic boxes. Check the boxes for show equations, plot residuals and show norm of residuals.

5.2.2 Do it yourself : Curve fitting with polynomial functions

In MATLAB, it is fairly easy to do polynomial curve file using built in polynomial functions and get the desired co-efficients.

Forexample:-

The simplest relationship between two variables say x and y, is a linear relationship $y = mx + c$. For a given set of data points $(x_i, y_i)$ the problem is to find m and c $\Rightarrow y_i = mx_i + c$ best fits the data.

## Example Straight-line (linear) fit

The following data is obtained from an experiment aimed at measuring the spring constant of a given spring. Different masses m are hung from the spring and the corresponding deflections $\delta$ of the spring from its unstreched configuration are measured. From physics, we know that $F = k\delta$ and here $F = mg$. Thus we can find k from the relationship $k = mg/\delta$. Here however, we are going to find k by plotting the experimental data, fitting the best straight line (we know that the relationship between $\delta$ and F is linear) through the data, and then measuring the slope of the best-fit line.

| m(g) | 5.00 | 10.00 | 20.00 | 50.00 | 100.00 |
|---|---|---|---|---|---|
| $\delta$(mm) | 15.5 | 33.07 | 53.39 | 140.24 | 301.93 |

Fitting a straight line through the data means we want to find the polynomial co-efficients $a_1$ and $a_0$ (a first order polynomial) such that $a_1 x_i + a_0$ gives the best estimate of $y_i$. In steps, we need to do the following.

**Step 1** Find the co-efficient $a_k$'s:

```
a = polyfit (x, y, 1)
```

**Step 2** Evaluate y at finer (more closely spaced) $x_j$'s using the fitted polynomial.

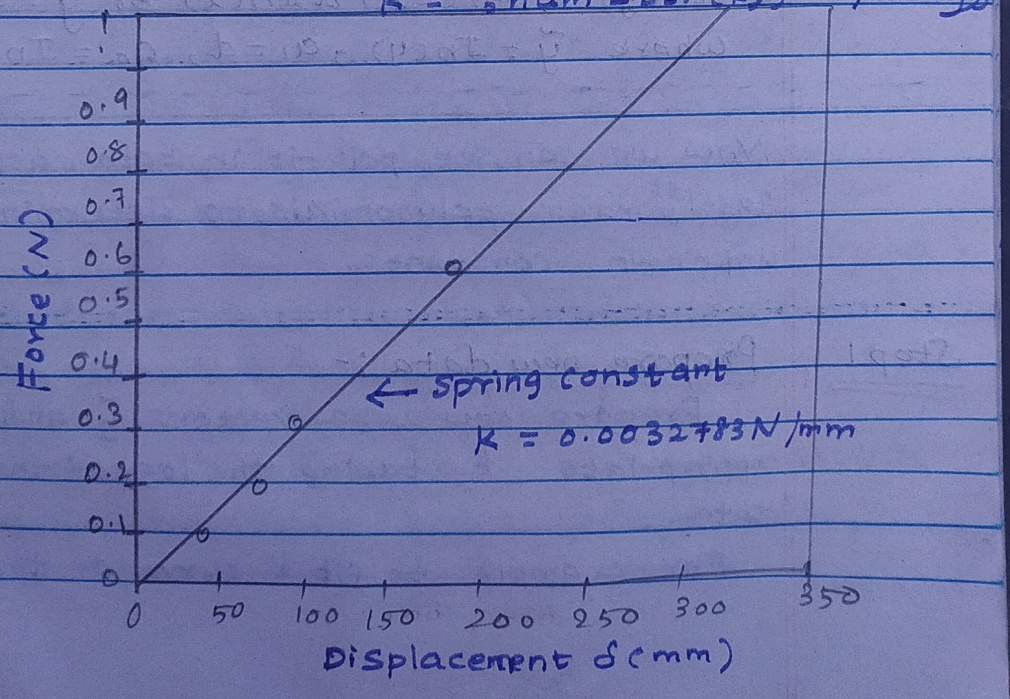
```
y_fitted = polyval (a, x_fine)
```

**Step 3** plot and see. plot the given data as points and fitted data as a line:

```
plot (x, y, 'o', x_fine, y_fitted);
```

The following script file shows all the steps involved in making a straigh-line fit through the given data for the spring experiment and finding the spring constant.

```
m = [5 10 20 50 100];      % mass data (g)
d = [15.5 33.07 53.39, 140.24, 301.03]
                           % displacement data (mm)
g = 9.81                   % g = 9.81 m/S^2
F = m/1000 * g;            % compute spring-force (N)
a = polyfit (d, F, 1);     % fit a line (1st order polynomial)
d-fit = 0 : 10 : 300;      % make a finer grid on d
F-fit = polyval (a, d-fit) % evaluate the
                           polynomial at new points
plot (d, F, 'o', d-fit, F-fit)  % plot data and
                                the fitter curve
xlabel ('Displacement | delta (mm)'), ylabel
    ('Force (N)') % Find the spring constant
k = a(1);
text (100, 32, ['leftarrow Spring constant
                k = ', num2str(k), 'N/mm'])
```



5.2.3 Least Squares Curve fitting

The technique of least squares curve fit can easily be implemented in MATLAB, because the technique results in a set of linear equations that need to be solved.

Most of the curve fits do are either polynomial curve fits or exponential curve fits (includes power laws, e.g $y = ax^b$). If we want to fit a polynomial of order n through our data we can use MATLAB's built in function polyfit, which already does a least squares curve fit.

Two most commonly used functions are:

1. $y = ae^{bx}$
2. $y = cx^d$

We can convert these exponential curve fits into polynomial curve fits by taking the log of both sides of the equations. i.e)

1. $\ln(y) = \ln(a) + bx$ or $\tilde{y} = a_0 + a_1 x$ where $\tilde{y} = \ln(y)$, $a_1 = b$ and $a_0 = \ln(a)$
2. $\ln(y) = \ln(c) + d\ln(x)$ or $\tilde{y} = a_0 + a_1 \tilde{x}$ where $\tilde{y} = \ln(y)$, $a_1 = d$, $a_0 = \ln(c)$ & $\tilde{x} = \ln$

Now we can see polyfit in both cases with just 1st order polynomials to determine the unknown constants.

Step 1 Prepare new data :-

Prepare new data vectors $\tilde{y}$ and $\tilde{x}$, as appropriate, by taking the log of the original data.

For example, to fit a curve of the type

rectangular matrix and the matrix equation to be solved is overdetermined. This, however, poses no problem, because the backslash operator in MATLAB solves the equation in a least squares sense whenever the matrix is rectangular.

5.2.4 General Nonlinear fits

In curve fitting, at times, we need to fit nonlinear equations in which the unknown co-efficients appear inside non-linear functions. $y(x) = C_1 e^{\lambda_1 x} + C_2 e^{\lambda_2 x}$. Here $C_1$ and $C_2$ are linear co-efficients but $\lambda_1$ and $\lambda_2$ are nonlinear co-efficients.

Type fitdemo on the command prompt to see a demo of how you can do such curve fits. However, you are not limitted to fitting nonlinear functions of only this form.

5.2.5 Interpolation:

Interpolation is the technique of finding a functional relationship between variables such that a given set of discrete values of the variables satisfy that relationship.

In MATLAB, you can interpolate your data using Splines or Hermite interpolants on a fly. All you have to do is plot the raw data and use Spline interpolant or hermite interpolant from the Basic Fitting options that you can invoke from the Tools menu of your figure window.

## Interp1

One-dimensional data-interpolation

i.e) given $y_i$ at $x_i$, finds $y_j$ at desired $x_j$ from $y_j = f(x_j)$. Here $f$ is a continuous function that is found from interpolation.

It is called one-dimensional interpolation because $y$ depends on a single variable $x$.

The calling syntax is

ynew = interp1(x, y, xnew, method)

where method is an optional argument discussed after the descriptions of interp2 and interp3

## Interp2

Two-dimensional data-interpolation

i.e) given $z_i$ at $(x_i, y_i)$ finds $z_j$ at desired $(x_j, y_j)$ from $z = f(x, y)$. The function $f$ is found from interpolation. It is called two-dimensional interpolation because z depends on two variables, x and y

Znew = interp2(x, y, z, xnew, ynew, method)

## Interp3

Three dimensional analogue of interp1. ie) given $v_i$ at $(x_i, y_i, z_i)$, finds $v_j$ at desired $(x_j, y_j, z_j)$.

Vnew = interp3(x, y, z, v, xnew, ynew, znew, method).

In addition, there is an n-dimensional analogue interpn, if you ever need it.

To specify cubic interpolation instead of linear. for example in interp1,

use the syntax

ynew = interp1 (x, y, xnew, 'cubic')

Spline

One-dimensional interpolation that uses cubic splines to find $y_j$ at desired $x_j$ given $y_i$ at $x_i$. Cubic splines fit seperate cubic polynomials between successive data points by matching the slopes as well as the curvature of each segment at the given data points. The calling syntax is

ynew = spline (x, y, xnew, method)

interpft:

Fast Fourier transform (FFT) - based 1-D data interpolation. This is similar to interp1 except that the data is interpolated first by taking the Fourier transform of the given data and then calculating the inverse transform using more data points. This interpolation is especially useful for periodic functions.

Example: There are two simple steps involved in interpolation - providing a list (a vector) of points at which you wish to get interpolated data and executing the appropriate function (eg. interp1) with the desired choice for the method of interpolation. We illustrate these steps through an example on the x and y data given in the following table.

| | | | | | | |
|---|---|---|---|---|---|---|
| x | 0 | 0.785 | 1.570 | 2.356 | 3.141 | 3.927 |
| y | 0 | 0.707 | 1.000 | 0.767 | 0.000 | -0.707 |

| | | |
|---|---|---|
| 4.712 | 5.497 | 6.283 |
| -1.000 | -0.707 | -0.000 |

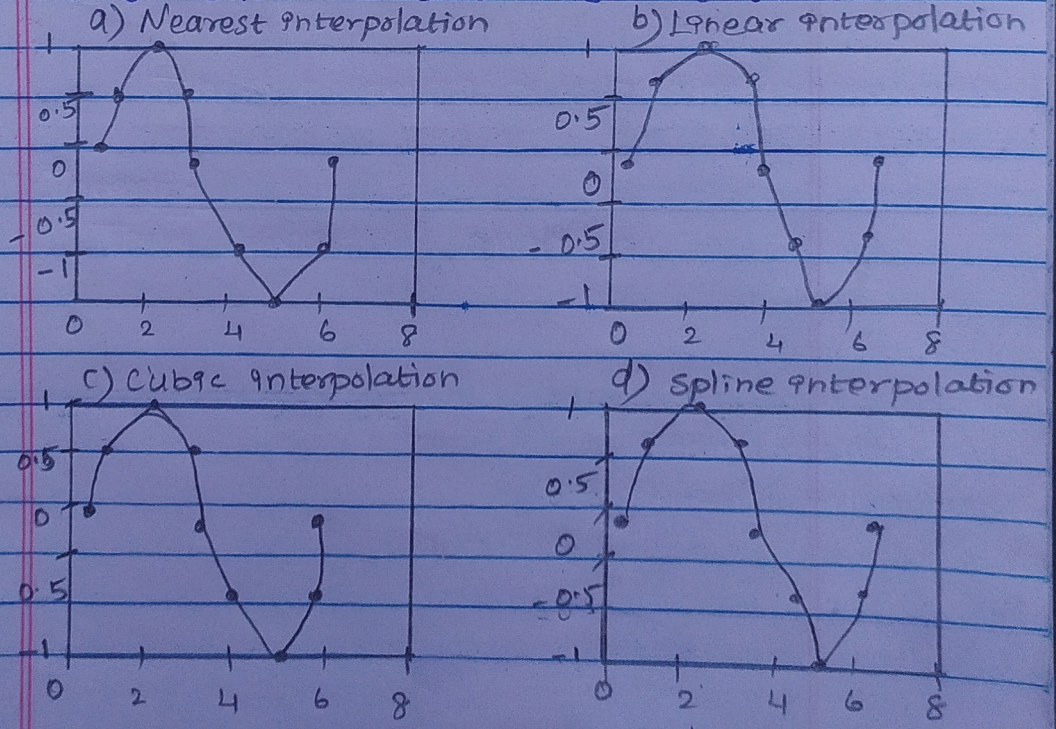
Step 1 Generate a vector xq containing desired points for interpolation.

```
% take equally spaced fifty points
xi = linspace (0, 2*pi, 50);
```

Step 2 Generate data yi at xi

```
% generate yi at xi with cubic interpolation
yi = interp1 (x, y, xi, 'cubic');
```

Here, 'cubic' is the choice for interpolation scheme. The other schemes we could use are nearest, linear, and spline. The corresponding curves show the smoothness obtained.




1-D interpolation of data with different schemes