**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

**BITS C464 – MACHINE LEARNING**

**I Semester 2014-2015**

**WORKSHEET #1**

**Introduction to Matlab and Polyomial Curve fitting**

**OBJECTIVE:-**

* Basics of Matlab
* Polynomial Curve Fitting
* Analyzing different errors (LSE and RMSE)
* Regularization

**BASICS OF MATLAB:**

MATLAB is a programming language developed by MathWorks. It started out as a matrix programming language where linear algebra programming was simple. It can be run both under interactive sessions and as a batch job.

**Following are the basic features of MATLAB:**

* It is a high-level language for numerical computation, visualization and application development.
* It also provides an interactive environment for iterative exploration, design and problem solving.
* It provides vast library of mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, numerical integration and solving ordinary differential equations.
* It provides built-in graphics for visualizing data and tools for creating custom plots.
* MATLAB's programming interface gives development tools for improving code quality and maintainability and maximizing performance.
* It provides tools for building applications with custom graphical interfaces.

**VECTORS:**

A vector is a one-dimensional array of numbers. MATLAB allows creating two types of vectors:

* Row vectors
* Column vectors

**Transpose Of Vectors:**

The transpose operation changes a column vector into a row vector and vice versa. The transpose operation is represented by a single quote(').

**Example**

Create a script file with the following code:

r = [1 2 3 4];

tr = r';

v = [1; 2; 3; 4];

tv = v';

disp(tr);disp(tv);

When you run the file, it displays the following result:

1

2

3

4

1 2 3 4

**Vector Dot Product:**

Dot product of two vectors a = (a1, a2, …, an) and b = (b1, b2, …, bn) is given by:

a.b = ∑(ai.bi)

Dot product of two vectors a and b is calculated using the **dot** function.

dot(a, b);

## Example

Create a script file with the following code:

v1 = [2 3 4];

v2 = [1 2 3];

dp= dot(v1, v2);

disp('Dot Product:');disp(dp);

When you run the file, it displays the following result:

Dot Product:

20

**Vector with Uniformly Spaced Elements**

MATLAB allows you to create a vector with uniformly spaced elements.

To create a vector v with the first element f, last element l, and the difference between elements is any real number n, we write:

v = [f : n : l]

**MATRIX**

A matrix is a two-dimensional array of numbers.

In MATLAB, you create a matrix by entering elements in each row as comma or space delimited numbers and using semicolons to mark the end of each row.

For example, let us create a 4-by-5 matrix *a*:

a = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8]

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

a =

1 2 3 4 5

2 3 4 5 6

3 4 5 6 7

4 5 6 7 8

**Referencing Elements in a Matrix:**

1. To reference an element in the mth row and nth column, of a matrix *mx*, we write:

mx(m, n);

1. To reference all the elements in the mth column we type A(:,m).
2. You can also select the elements in the mth through nth columns, for this we write:

a(:,m:n)

1. In the same way, you can create a sub-matrix taking a sub-part of a matrix.

For example, let us create a sub-matrix *sa* taking the inner subpart of a:

3 4 5

4 5 6

To do this, write:

a = [1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

sa= a(2:3, 2:4)

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

sa =

3 4 5

4 5 6

**Transpose of a matrix:**

The transpose operation switches the rows and columns in a matrix. It is represented by a single quote(').

## Example

Create a script file with the following code:

a = [10 12 23; 14 8 6; 27 8 9]

b = a'

When you run the file, it displays the following result:

a =

10 12 23

14 8 6

27 8 9

b =

10 14 27

12 8 8

23 6 9

**Multiplication of matrix**

In MATLAB, matrix multiplication is performed by using the \* operator.

## Example

Create a script file with the following code:

a = [1 2 3; 2 3 4; 1 2 5]

b = [2 1 3; 5 0 -2; 2 3 -1]

prod = a \* b

When you run the file, it displays the following result:

a =

1 2 3

2 3 4

1 2 5

b =

2 1 3

5 0 -2

2 3 -1

prod =

18 10 -4

27 14 -4

22 16 -6

**Inverse of a matrix:**

The inverse of a matrix A is denoted by A−1 such that the following relationship holds:

AA−1= A−1A=1

Create a script file and type the following code:

a =[1 2 3; 2 3 4; 1 2 5]

inv(a)

When you run the file, it displays the following result:

a =

1 2 3

2 3 4

1 2 5

ans =

-3.5000 2.0000 0.5000

3.0000 -1.0000 -1.0000

-0.5000 0 0.5000

**SPECIAL ARRAYS IN MATLAB**

The **zeros()** function creates an array of all zeros:

**For example:**

zeros(5)

The **ones()** function creates an array of all ones:

**For example:**

ones(4,3)

The **eye()** function creates an identity matrix.

**For example:**

eye(4)

**PLOTTING THE GRAPH:**

To plot the graph of a function, you need to take the following steps:

1. Define **x**, by specifying the **range of values** for the variable **x**, for which the function is to be plotted
2. Define the function, **y = f(x)**
3. Call the **plot** command, as **plot(x, y)**

**Example**

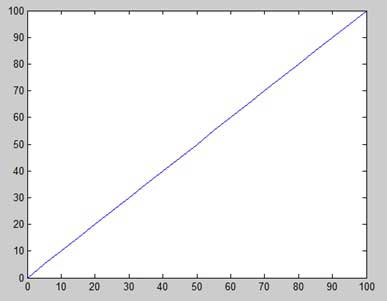
Create a script file and type the following code:

x =[0:5:100];

y = x;

plot(x, y)

When you run the file, MATLAB displays the following plot:



**Adding grid lines, label and title to a graph:**

**Example**

Create a script file and type the following code:

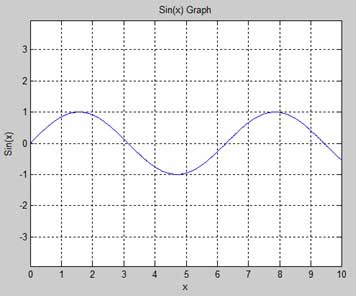
x =[0:0.01:10];

y = sin(x);

plot(x, y),xlabel('x'),ylabel('Sin(x)'), title('Sin(x) Graph'),

grid on, axis equal

MATLAB generates the following graph:



**Drawing Multiple Functions on the Same Graph:**

You can draw multiple graphs on the same plot. The following example demonstrates the concept:

**Example**

Create a script file and type the following code:

x =[0:0.01:10];

y = sin(x);

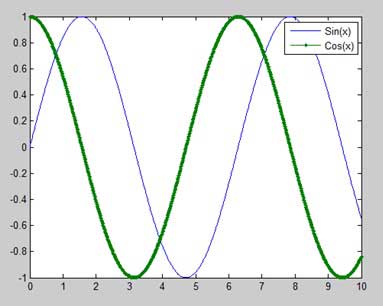
hold on;

g = cos(x);

hold off;

plot(x, y, x, g,'.-'), legend('Sin(x)','Cos(x)')

MATLAB generates the following graph:



**Generating Sub-Plots:**

When you create an array of plots in the same figure, each of these plots is called a subplot. The **subplot** command is for creating subplots.

Syntax for the command is:

subplot(m, n, p)

where, *m* and *n* are the number of rows and columns of the plot array and *p* specifies where to put a particular plot.

Each plot created with the subplot command can have its own characteristics.

**Example**

Let us generate two plots:

y = e−1.5xsin(10x)

y = e−2xsin(10x)

Create a script file and type the following code:

x =[0:0.01:5];

y =exp(-1.5\*x).\*sin(10\*x);

subplot(1,2,1)

plot(x,y),xlabel('x'),ylabel('exp(–1.5x)\*sin(10x)'),axis([0 5 -1 1])

y =exp(-2\*x).\*sin(10\*x);

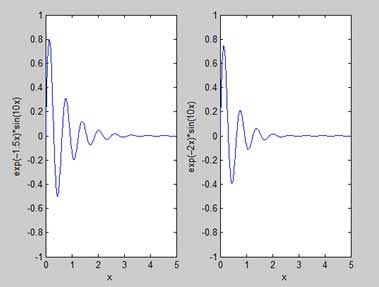
hold on;

subplot(1,2,2)

plot(x,y),xlabel('x'),ylabel('exp(–2x)\*sin(10x)'),axis([0 5 -1 1])

hold off;

When you run the file, MATLAB generates the following graph:



**Plot sin2pix without errors**

*x=[0:0.1:10];*

*y= sin(x);*

*plot(x,y)*

**Plot sin2pix with Gaussian errors**

*x=[0:0.1:10];*

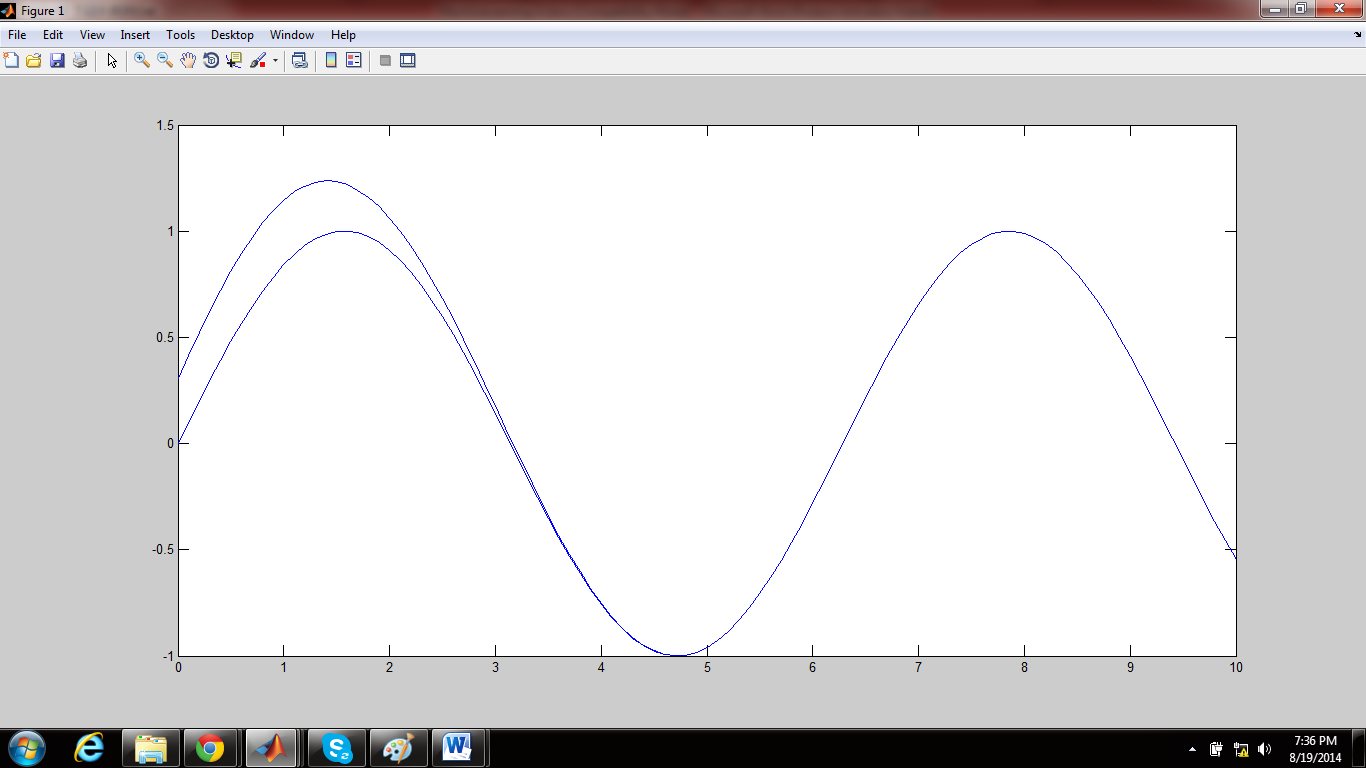
*y1= sin(x)+normpdf(x,0.5,1.2);*

*plot(x,y);*

*hold on;*

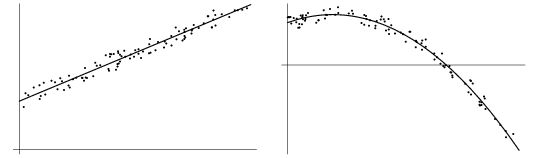
*plot(x,y1);*

*hold off;*



**POLYNOMIAL CURVE FITTING**

Mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets ("the residuals") of the points from the curve. The sum of the *squares* of the offsets is used instead of the offset absolute values because this allows the residuals to be treated as a continuous differentiable quantity. However, because squares of the offsets are used, outlying points can have a disproportionate effect on the fit, a property which may or may not be desirable depending on the problem at hand.

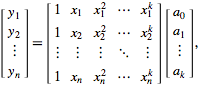


*Curves fitted on the dataset given*

 least squares fitting proceeds by finding the sum of the *squares* of the *vertical* deviations R^2 of a set of n data points

|  |
| --- |
| R^2=sum[y_i-f(x_i,a_1,a_2,...,a_n)]^2 |

*Now using differential calculus, optimal solution is calculated.*



*This is equation where (a0,a1….) is the solution vector.*

*The solution is*

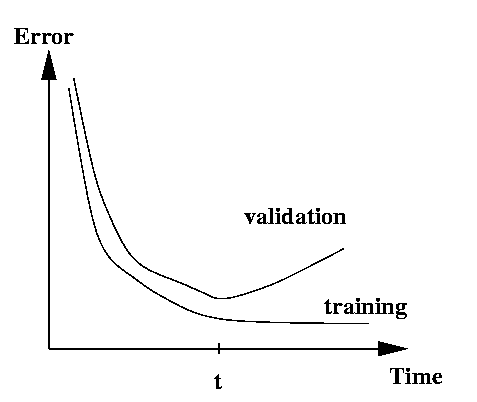
 a=(X^(T)X)^(-1)X^(T)y. 

\operatorname{RMSD}=\sqrt{\frac{\sum_{t=1}^n (\hat y_t - y_t)^2}{n}}.

**Overfitting:**

*When our model performs well on training data set but fails to produce good results on test data set then this problem is referred to as overfitting.*

*Due to overfitting our main aim of supervised learning i.e generalization is not achieved. This usually occurs when complexity of the model is increased.*



*This graph captures the notion of overfitting. During training as the complexity of the model is increased the error decreases but it fails to generalize as error for validation is increased.*

**REGULARIZATION:**

*To avoid overfitting, objective function (least squared error) is modified to add penalty for increased polynomial coefficient. Which means as parameters are increased a penalty is increased for it to balance the effect of overfitting.*

\begin{eqnarray*}
J(\theta) & = & \frac{{1}}{2m}\left[\sum_{i=1}^{m}(h_{\theta}(...
...
& & \mbox{where }\lambda\mbox{ is the regularization parameter}\end{eqnarray*}

*This modified objective function when solved, the solution will ensure that the model will not overfit and will generalize well.*

***Exercise***

1. *From the given data points draw 10 points uniformly.*
2. *Fit a polynomial on the selected data points*
3. *Calculate the error( LSE and RMSE) for both training and cross validation data set.*
4. *Plot the errors and analyze the result*
5. *Repeat the same for 20, 50 and 70 data points*
6. *Observe the coefficients for overfitting.*
7. *Repeat the same process with regularized cost function*
8. *Record the resutls*