MM 220 Autumn 2020
Tutorial 2: Data Analysis using MATLAB

Anirban Patra, Amrita Bhattacharya
Department of Metallurgical Engineering and Materials Science, IIT Bombay

We will learn how to perform data analysis using MATLAB in this tutorial. We will first start with fitting data to polynomial curves. Then we will learn how to fit data to a generic user-defined function. We will also learn how to read/import data into MATLAB.

Lesson 1: *Data fitting using polynomial curves*

Let us create two vectors `x` and `y` with the following data:

```
>> x = [11 31 64 112 176 259 362];
>> y1 = [2 3 4 5 6 7 8];
```

We can plot this data using:

```
>> plot(x,y1,'*','LineWidth',12);
>> hold on;
```

We plot the data using the symbol `*` here.

Now we will use the function `polyfit` to calculate the coefficients of a linear fit (using least square regression) for the above data.

```
>> coeffs1 = polyfit(x,y1,1);
```

Here the parameter `1` indicates that the data is to be fit to a 1st order polynomial. A nth order polynomial in MATLAB has the following form: $p(x) = p_1 x^n + p_2 x^{n-1} + ... + p_n x + p_{n+1}$. We therefore have 2 coefficients for a 1st order polynomial. These coefficients are stored in `coeffs1`.

Now we will use the function `polyval` to create another vector `y2` which uses the values in `coeffs1` to find the values of `y2` corresponding to `x` using the above polynomial.

```
>> y2 = polyval(coeffs1,x);
>> plot(x,y2,'Color','r');
```

This superimposes the plot of `y2` vs. `x` on the plot that we had earlier. As can be seen, the linear fit does not do a very good job of representing the above data. We use a 2nd order polynomial to fit the above data now:

```
>> coeffs2 = polyfit(x,y1,2);
>> y3 = polyval(coeffs2,x);
>> plot(x,y3,’Color’,’g’);
>> xlabel(‘x’);
>> ylabel(‘y1, y2, y3’);
>> legend('Data','1st Order Fit','2nd Order Fit');
>> hold off;
```

We see that the 2nd order polynomial does a much better job of fitting to the given data! Please save this plot to the file `plot1.png` using:

```
>> print(‘plot1’,’-dpng’);
```

Lesson 2: *Data fitting using user-defined functions*

It is likely that in many cases the generic polynomial expression in MATLAB may not yield a satisfactory fit to the data. For example, the data might give a better fit to trigonometric or exponential functions. We need to define our own functions in such situations.

We will create a script and use another data set. All commands in this lessons will be written in the script `example_fit.m`:

```
Function [a,p] = example_fit ()
% Name:
% Roll no.:
x = [0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0];
y = [0.02 0.15 0.3 0.5 0.75 1.1 1.5 1.9 2.4 2.9];
```

`L2.1:` Let us assume that this data represents a physical process which can be represented by the equation: $y = a_1x + a_2x^2 + a_3exp(x)x$. We need to find the parameters a_1, a_2, and a_3 based on the fit of `y` to `x`. This is essentially a system of linear equations of the form: [`data`][`a`] = [`y`], where `data` is a 10x3 matrix with the elements `data(i,1)` = `x(i)`, `data(i,2)` = `x(i)^2`, `data(i,3)` = `x(i)*exp(x(i))`. `a` is a 3x1 matrix of the coefficients of the above equation and `y` is the 10x1 matrix declared above.

```
data(:,1) = x(:);
data(:,2) = x(:).^2;
data(:,3) = x(:).*exp(x(:));
```

Note the `:` operator which can be used to specify operations over all columns. We can solve the set the of linear equations simply by using:

```
a = data\y';
```

Here `\` is the operator used to solve a system of linear equations. It is the same as `mldivide`. `a = data\y'` solves a system of linear equations: `data*a = y'`.

Now we will create another vector `y1` which has the values of the above expression using the fitting coefficients `a`. We plot the original data using symbols and the fitted curve using a line.

```
y1 = a(1)*x + a(2)*(x.^2) + a(3)*x.*exp(x);
plot(x,y,'*','LineWidth',12);
hold on;
plot(x,y1,'Color','r');
xlabel('x');
ylabel('y');
```

Note that this particular technique worked only because we could reduce our model to a system of linear equations. This might not work if we have a more complicated model form!

`L2.2:` Now we will use the same data to fit to another non-linear equation of the form: $y = p_1 (x - p_2)^{p_3}$. We do a least square regression fit to find the parameters p_1, p_2, and p_3. We first define a function: $f = \sum_i (y_i - p_1 (x_i - p_2)^{p_3})^2$. We want to minimize the value of this function over all elements denoted by the index, i, using the function `fminsearch`. First we create a function `f` with the symbolic handle `p`. We provide the initial guess values of p_1, p_2, and p_3 in the vector `pguess`. The function call to `fminsearch` also returns the value of the residual, which is stored in `fminres`. Once we obtain the values of `p`, we create another vector `y2` generated using the above equation and then plot it. We save the plot as `plot2.png`.

```
f = @(p) sum((y - p(1)*((x - p(2)).^p(3))).^2);
pguess = [0.1 0.1 2];
[p,fminres] = fminsearch(f,pguess);
y2 = p(1)*((x - p(2)).^p(3));
```

```
plot(x,y2,'Color','g');
legend(‘Data’,’Fit1’,’Fit2’);
hold off;
print(‘plot2’,’-dpng’);
```

Note that the regression fit is very sensitive to the initial guess values and they must be chosen properly!

Lesson 3: *Reading data files*

We often deal with large amount of data written in text files. We need to read these data files and import the data into MATLAB in order to work with them. MATLAB provides functions to easily read data files. Here we will only learn about one of them: `importdata`. We will repeat the same exercise as above. Instead of assigning the values to `x` and `y` within the script, we will read them in from the file `testfile.txt`. We create a second script file `example_fit2.m`.

```
function example_fit2 ()
% Name:
% Roll no.:

A = importdata(‘testfile.txt’);

x = A(:,1)’;
y = A(:,2)’;

data(:,1) = x(:);
data(:,2) = x(:).^2;
data(:,3) = x(:).*exp(x(:));

a = data\y’;

y1 = a(1)*x + a(2)*(x.^2) + a(3)*x.*exp(x);
plot(x,y,’*’,’LineWidth’,12);
hold on;
plot(x,y1,’Color’,’r’);
xlabel(‘x’);
ylabel(‘y’);

f = @(p) sum((y - p(1)*((x - p(2)).^p(3))).^2);
```

```
pguess = [0.1 0.1 2];
[p,fminres] = fminsearch(f,pguess);
y2 = p(1)*((x - p(2)).^p(3));
plot(x,y2,'Color','g');
legend('Data','Fit1','Fit2');
hold off;
print('plot3','-dpng');
```

The data is read in at the beginning of the script into the matrix `A`. We then assign `x` as the 1st column of `A` and `y` as the 2nd column of `A`. Everything else remains the same as earlier.

Assignment

Create a MATLAB function called `MM220A1` with the file name `MM220A1.m`.

You have been given the experimental data for strain (ε) vs. stress (σ) (in MPa) from a uniaxial tension test for an alloy in the file `exp_stress_strain.txt`.

(a) Import this data into MATLAB.

(b) For this material the Young's modulus (E) is 330 GPa. Calculate the elastic strain ($\varepsilon^e = \sigma/E$) corresponding to each stress value using the imported data.

(c) The plastic strain is given as difference of the total strain with the elastic strain, i.e., $\varepsilon^p = \varepsilon - \varepsilon^e$. Using the imported data, calculate the plastic strain.

(d) Now fit the stress to the plastic strain using polynomials of degree 10 and 20. Plot the total strain vs. stress for both of them on the same graph along with the experimental data. Use `red` line for polynomial of degree 10, `green` line for polynomial of degree 20, and symbols for the experimental data. Label your axes properly along with appropriate legends. Save the plot as `MM220A1.png`.

(e) The error associated with your fit can be calculated as:

$$Error = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\sigma^i_{fit}-\sigma^i_{exp}}{\sigma^i_{exp}}\right)^2}$$

where, σ^i_{fit} represents the fitted stress, and σ^i_{exp} represents the experimental stress for the i^{th} data point, and N is the total number of data points. For each of the fits in (d), calculate the associated error and print these values on the MATLAB terminal. Which fit gives you the minimum error?

The function returns the following variables. Please use the following names carefully for the variables to be returned by the function. This is needed to pass the tests.

- `coeffs10` for coefficients obtained for degree 10 polynomial.
- `coeffs20` for coefficients obtained for degree 20 polynomial.
- `stress10` for stress values obtained using degree 10 polynomial.
- `stress20` for stress values obtained using degree 20 polynomial.
- `error10` for error in fitting of degree 10 polynomial
- `error20` for error in fitting of degree 20 polynomial