

MATLAB Lecture 3. Polynomials and Curve Fitting

Almost all basic data structures in MATLAB are matrices (two or one dimensional). Polynomials are no exception. As MATLAB is primarily a numerical software system, the coefficients are by default floating point numbers. With curve fitting we will see how to reduce errors on approximate data.

Algebraically, we define polynomials by their coefficient vectors. Geometrically, polynomials are defined by their roots. In scientific applications we use polynomials to model data. These often noisy data are then regarded as sampled function values. What we then want to know is whether we can replace the sampled data by a polynomial of a certain degree.

3.1 Polynomials: Coefficient Vectors and Roots

A polynomial in MATLAB is represented by its coefficient vector. For example, to define the cubic polynomial

$$3x^3 - 2.23x^2 - 5.1x + 9.8,$$

we type

```
>> c = [ 3 -2.23 -5.1 9.8 ]
```

Observe that the leading coefficient of the polynomial comes first, to select this coefficient we type `c(1)`.

To evaluate that polynomial at a point x , we apply the command **polyval** as follows

```
>> y = polyval(c,x)    % evaluate at x
```

If x is a vector, then y will be the vector with corresponding function values at the points in x . This last feature is useful to make a plot of the polynomial. Suppose we are interested in the range $-1 \dots 1$ for x and we want to use in the plot evenly spaced data points, each at a distance of 0.1 apart. The sequence of commands

```
>> x = -1:0.1:1;      % define range for plotting
>> y = polyval(c,x);  % compute samples
>> plot(x,y)          % make the plot
```

will generate a smooth curve through those points. If we wish to see only the data points and use the '+' to mark each data point, we execute **plot(x,y,'+')**.

Besides evaluating and plotting a polynomial, we can solve polynomial equations in MATLAB via the command **roots**:

```
>> r = roots(c)       % compute roots of polynomial with coefficients in c
```

For the choice of the coefficient vector c as above we will see three roots: one root is real, and two are complex conjugated. To test the accuracy of the roots, we can invoke `polyval` again at the roots, but we can also make a polynomial from the roots with the command **poly**:

```
>> cr = poly(r)       % defines (x-r(1))*(x-r(2))*...
>> norm(c(1)*cr-c)    % how far from original polynomial?
```

Since the leading coefficient of the polynomial returned by `poly` is one, we multiply with the original leading coefficient in the comparison.

For large degree polynomials, it is interesting to look at the distribution of the roots. Let us make a plot of the roots of the polynomial, marking each root with a blue cross:

```
>> plot(r,'bx')           % see distribution of roots
```

Note that as the roots are complex numbers, MATLAB automatically switches to the complex plane, where the horizontal axis is real and the vertical axis is imaginary.

MATLAB also supports the symbolic differentiation of polynomials via the command **polyder**:

```
>> dc = polyder(c)       % compute derivative
>> hold on                % keep previous plot
>> plot(roots(dc),'rx') % make plot of roots
```

In the last command we compare the location of the roots of the original polynomial (marked with blue crosses) with those of the derivative.

3.2 Curve Fitting

Curve fitting to measured data is a frequently occurring problem. The input to this problem are two vectors x and y of equal size. We ask whether there is a dependency of y on x as a polynomial function. For instance, does y depend linearly on x ? Or are higher degree polynomials needed to represent the dependency? The problem is thus to fit the data with a polynomial. MATLAB uses the method of least squares to solve the overconstrained linear system which gives the coefficients of the fitting polynomial.

For a given data set (x, y) – where x and y are vectors of the same size – we can ask MATLAB to compute a polynomial that best fits (in the least squares sense) these data with the command **polyfit**. We must supply to **polyfit** the degree d of the fitted polynomial. For example,

```
>> c = polyfit(x,y,d)     % fit (x,y) with degree d polynomial
```

returns in c the coefficients of a polynomial of degree d that fits the data (x, y) .

To simulate a situation where curve fitting occurs we will perform the following experiment:

1. Generate samples from a cubic polynomial in the interval $[-1, 1]$, call this sample vector y .
2. Use **randn** to generate a random vector of length one, call it $noise$ and add this to the same vector y . We assign the result of this addition to ey . The vector ey represents data obtained from a numerical experiment, and thus known with limited precision.
3. Invoke **polyfit** to find a cubic polynomial through the data (x, ey) .
4. Compare the results of this fitting visually with a plot and numerically by taking the difference of coefficient vectors.

The commands to conduct this experiment are below:

```
>> c3 = [0.74 0.97 1.1 0.86]; % data source
>> x = -1:0.1:1;           % sample range for x
>> y = polyval(c3,x);      % sample cubic at x
>> noise = randn(1,size(y,2)); % random noise of same size as y
>> noise = noise/norm(noise); % normalize the noise
>> ey = y + noise;         % make noisy data
>> ec = polyfit(x,ey,3);   % fit noisy data with cubic
```

Observe the operations in creating the vector $noise$. If you omit the semicolons above, you probably feel drowned in the flood of numbers. The plot is there to clear it up:

```

>> hold on                % keep all plots in same window
>> plot(x,y,'b--')       % exact polynomial
>> plot(x,ey,'r+')       % noisy data
>> fy = polyval(ec,x)    % sample fitted polynomial
>> plot(x,fy,'g')        % reconstructed data source

```

From the plot we observe that the fitted polynomial has the same shape as the original one. Moreover, the noisy data seems to be further off from the original polynomial than the fitted polynomial. To investigate this numerically, we can take the norm of the difference between y and fy . We can say that the curve fitting has “smoothened” the data.

In the experiment above, we assumed the dependency of y on x was like a cubic polynomial function. In practice – see our original question at the top of the page – this assumption should not be taken for granted. Discovering the degree of the relationship between the variables is a much harder problem. Moreover, if you think about $x^2 + y^2 = 1$, the relation between x and y cannot be adequately described by a function.

3.3 Assignments

1. Denote by p_a and p_b the polynomials with coefficients respectively in a and b . The coefficient of the product $p_c = p_a p_b$ are computed as $c = \mathbf{conv}(a, b)$. With **deconv** we compute the coefficients of the quotient p_q and remainder p_r (satisfying $p_a = p_b p_q + p_r$), as $[q, r] = \mathbf{deconv}(a, b)$.

Divide $p_a = x^2 - 3x + 2$ by $p_b = x - 1$ and verify that the product of p_b with the quotient equals p_a .

2. Generate a random polynomial of degree 100, using the command **randn** to generate its coefficient vector. Plot the distribution of the roots. Repeat your experiment several times. Do you see a pattern emerging?
3. Use **poly** to define the coefficient vector of the polynomial $(x - 1)^{100}$ (*hint*: type **help ones**). With this coefficient vector, compute the roots and make a plot. What do you observe? Is this what you expected to see? Can you explain the difference between what you observed and expected to see?
4. To fit given data (x_i, y_i) , for $i = 1, 2, \dots, n$ with a quadratic polynomial $y = c_2 x^2 + c_1 x + c_0$, the unknown coefficients c_0 , c_1 , and c_2 should satisfy a linear system:

$$\begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \text{or shortened: } Xc = y.$$

Do

```

>> x = [1 2 3 4] % points to sample
>> y = x.^3      % samples of x^3

```

and set up the matrix X and right hand side vector Y to find the solution c .

- (a) Solve the linear system for c and compare with the output of **polyfit**.
- (b) Find a way to define X so that it works for any number n of points used in the fitting. Illustrate with a random vector of size n as x .

5. Define a quadric polynomial by its coefficient vector $c = [-2.23 \quad -5.1 \quad 9.8]$. As in the curve fitting experiment, generate a random noise vector of length one and add this noise vector to samples taken from this quadric polynomial, at points in the interval $[-1, 1]$, with space 0.1 between them. Suppose now (unlike in the curve fitting experiment) that we do not know the data came from a quadric. To discover the degree, invoke **polyfit** three times, for degrees one, two, and three. What degree does best fit the data? Could you conclude that degree two is best? Justify the criterion you use by additional numerical data, using the norm of differences.