SM-2302: Software for Mathematicians

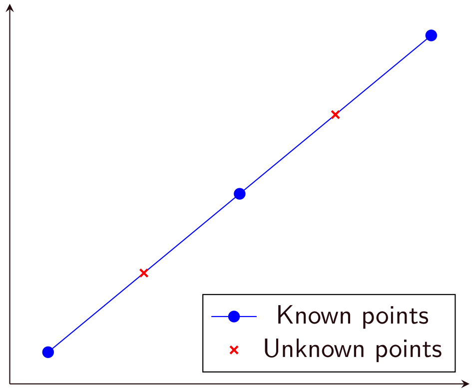
Matlab3: Numerical Techniques

Dr. Huda Ramli
Mathematical Sciences, Faculty of Science, UBD
huda.ramli@ubd.edu.bn

Semester I, 2023/24

Interpolation

Interpolation is used to estimate data points between two known points. The most common interpolation technique is linear interpolation.



- In MATLAB, we can use the `interp1()` function
- There are many options for interpolation methods, for e.g.
  - `linear` (default)
  - `nearest`,
  - `spline`,
  - `cubic`, etc.
- Refer to `help interp1` for more details

Example 1

Given the following data points:

x	0	1	2	3	4	5
y	15	10	9	6	2	0

Find the interpolated value for $x = 3.5$.

Using the default `linear` interpolation, we get the answer 4.

```
x = 0:5;
y = [15, 10, 9, 6, 2, 0];
plot(x,y ,'-o')

% Find interpolated value for x=3.5
y_lin = interp1(x,y,3.5)
```

Using the `spline` interpolation on the same data results in the plotted interpolated points.

```
x = 0:5;
y = [15, 10, 9, 6, 2, 0];
new_x = 0:2.5:5;

% Find interpolated values using spline:
y_spline = interp1(x,y,new_x,'spline')

% Original points and interpolated points in same graph:
plot(x,y, new_x,y_spline, '-o')
```

Curve Fitting

- It is often more convenient to model empirical data as a mathematical function: $y = f(x)$. Then, we can easily calculate any desired data based on this mathematical model.
- MATLAB's built-in curve fitting functions:
  1. `>> p = polyfit(x,y,n)` finds coefficients of a polynomial $p(x)$of degree n
  2. `>> polyval(p,x)` returns the value of a polynomial $p(x)$ evaluated at x

  These techniques use a polynomial of degree n that fits data y best in a least-squares sense.
- A **polynomial** is expressed as:

$$p(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}$$

where $p_1, p_2, \ldots, p_{n+1}$ are the coefficients of the polynomial.
MATLAB represents $p(x)$ as the vector:

$$p = [p_1; \;\; p_2; \;\; \cdots \;\; p_n; \;\; p_{n+1}]$$

Regression Models

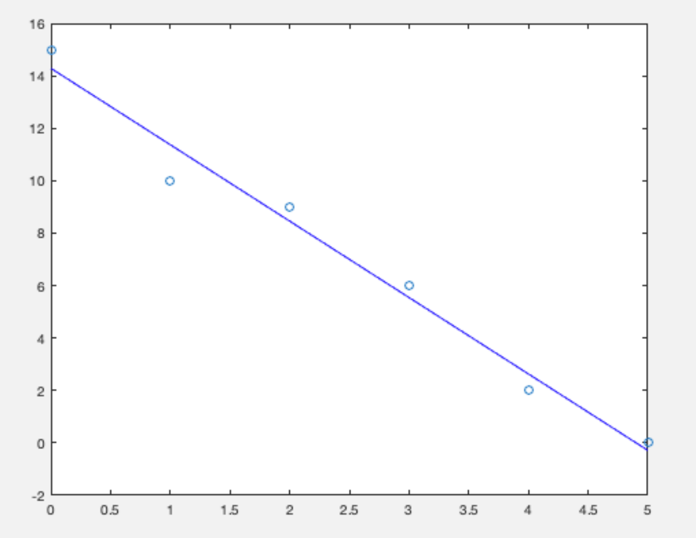
- **Polynomial Regression**: $y(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n$
- **Linear Regression**: $y(x) = ax + b$ is a polynomial of the first order.

Example 2 (Linear Regression)

Find the linear regression model of the data given in Example 1.

```
x = 0:5;
y = [15, 10, 9, 6, 2, 0];
n = 1; % first order polynomial
p = polyfit(x,y,n)
a = p(1); b = p(2);
ymodel = a*x + b;
plot(x,y,'o', x,ymodel,'b-')
```

The answer is `ans = -2.9143    14.2857`
which gives the linear model: $y = -2.9143c + 14.2857$ as shown in the resulting graph.



Example 3 (Polynomial Regression)

Use `polyfit` and `polyval` functions to compare different orders of the polynomial regression models for the same data given in Example 1:

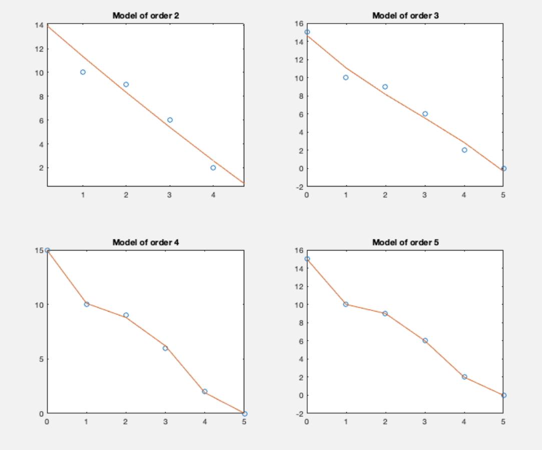
x	0	1	2	3	4	5
y	15	10	9	6	2	0

Since we only have 6 data points, we will study models up to the 5th order polynomial.

```
x=[0, 1, 2, 3, 4 ,5];
y=[15, 10, 9, 6, 2 ,0];
for n=2:5 % From orders 2 to 5
    p=polyfit(x,y,n)
    ymodel=polyval(p,x);
    subplot(2,2,n-1)
    plot(x,y,'o',x,ymodel)
    title(sprintf('Model of order %d', n));
end
```

The resulting polynomial regression models are

$$y_2(x) \approx 0.05x^2 - 3.2x + 14.5$$
$$y_3(x) \approx -0.065x^3 + 0.5x^2 - 4x + 14.7$$
$$y_4(x) \approx 0.2x^4 - 1.9x^3 + 6.3x^2 - 9.4x + 15$$
$$y_5(x) \approx -0.04x^5 + 0.7x^4 - 4.2x^3 + 10.3x^2 - 11.8x + 15$$



Example 4 (Model fitting)

Given the following data:

Height, h(ft)	0	1.7	1.95	2.60	2.92	4.04	5.24
Flow, f(ft^3/s)	0	2.6	3.6	4.03	6.45	11.22	30.61

Create polynomial regression models of the following order:

(i) linear

(ii) quadratic

(iii) cubic

Which gives the best model? Plot the result in the same plot and compare them. Add `xlabel`, `ylabel`, `title` and a `legend` to the plot and use different line styles so the user can easily see the comparison.

```
%% Script to plot polynomial regression models
clear, clc

% Real Data:
height = [0, 1.7, 1.95, 2.60, 2.92, 4.04, 5.24];
flow = [0, 2.6, 3.6, 4.03, 6.45, 11.22, 30.61];
new_height = 0:0.5:6; % generate new height values used to test the model

polyorder = 1; % linear
p1 = polyfit(height, flow, polyorder) % first order model
new_flow1 = polyval(p1,new_height); % use the model to find new flow values

polyorder = 2; % quadratic
p2 = polyfit(height, flow, polyorder) % second order model
new_flow2 = polyval(p2,new_height);

polyorder = 3; % cubic
p3 = polyfit(height, flow, polyorder) % third order model
new_flow3 = polyval(p3,new_height);
```

```
% plot the original data together with the model found for comparison
plot(height, flow, 'bo', new_height, new_flow1, new_height, new_flow2, ...
    new_height, new_flow3)
title('Model fitting')
xlabel('height')
ylabel('flow')
legend('real data', 'linear model', 'quadratic model', 'cubic model')
```

Numerical differentiation

The derivative of a function $y = f(x)$ is a measure of how y changes with x:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

A numerical approach to the derivative of a function $y = f(x)$ is

$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

MATLAB offers functions for numerical differentiation:

- `diff(x)`: Difference and approximate derivative for a vector **x**
- `polyder(p)`: Differentiate polynomial, returns the derivative of the polynomial whose coeffiicents are the elements of vector **p**

Example 5

Use numerical differentiation to find $\frac{dy}{dx}$ on the function $y = x^2$, based on the data points:

x	-2	-1	0	1	2
y	4	1	0	1	4

We know that the exact solution is: $\frac{dy}{dx} = 2x$.

We will compare the results from the numerical differentiation with the exact solution.

```
x = -2:2; y = x.^2;
dydx_num = diff(y)./diff(x);
dydx_exact = 2*x;
dydx = [[dydx_num, NaN]', dydx_exact']  % NaN is added to the vector to get ...
    same length of vectors
plot(x,[dydx_num, NaN],'b', x,dydx_exact,'g') % Plot the derivatives
```

This yields the following results:

```
dydx =
    -3     -4
    -1     -2
     1      0
     3      2
   NaN      4
```

How can we get a better approximation result?

Example 6

Consider the equation $y = x^3 + 2x^2 - x + 3$.

(a) Find $\dfrac{dy}{dx}$ analytically.

(b) Taking $x = [-5, 5]$, use `diff` function to approximate the derivative $\Delta y / \Delta x$.

(c) Compare the data in 2D array and compare the graphs of the exact value of $\dfrac{dy}{dx}$ and the numerical approximation in the same plot.

(d) Increase the number of data points and compare.

The exact solution is

$$\frac{dy}{dx} = 3x^2 + 4x - 1.$$

```matlab
x = -5:1:5;
y = x.^3 + 2*x.^2 - x + 3; % Define the function y(x)

% Plot the function y(x)
plot(x,y)
title('y')

% Find nummerical solution to dy/dx
dydx_num = diff(y)./diff(x);
dydx_exact = 3*x.^2 + 4.*x -1;
dydx = [[dydx_num, NaN]', dydx_exact']

% Plot nummerical vs analytical solution to dy/dx
figure(2)
plot(x,[dydx_num, NaN], x, dydx_exact)
title('dy/dx')
legend('numerical solution', 'analytical solution')
```

Differentiation on Polynomials

A polynomial can be expressed in the general form:

$$y(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n$$

and its derivative is $\quad y'(x) = a_0 n x^{n-1} + a_1 (n-1) x^{n-2} + \ldots + a_{n-1}.$

In Example 6,

- The equation $y = x^3 + 2x^2 - x + 3$ can be defined as a polynomial using the function:
  $>> \mathrm{p} = [1\ 2\ \text{-}1\ 3]$ where $n = 3$.
- Then

```
polyder(p)
ans =

          3    4    -1
```

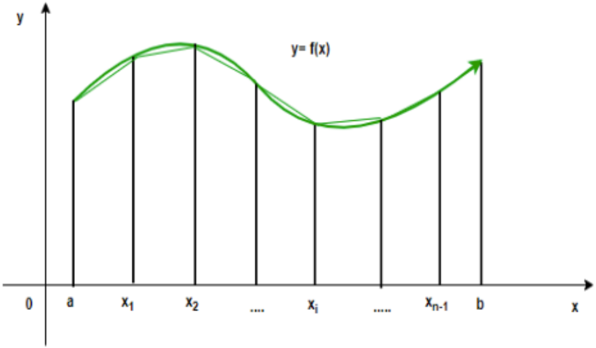
which agrees with our analytical derivative $\dfrac{dy}{dx} = 3x^2 + 4x - 1.$

Numerical integration

- The integral of a function $f(x)$ is denoted as: $\displaystyle\int_a^b f(x)\,dx$.
- An integral can be seen as the area under a curve. Given $y = f(x)$ the area A under the curve can be approximated by dividing the area between the limits into trapezoids:

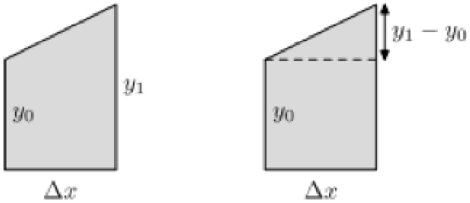
$$A = \sum_{i=1}^{n} \frac{f(x_i) + f(x_{i-1})}{2} \Delta x \quad \text{where} \quad \Delta x = x_i - x_{i-1}$$

This is known as the **trapezoid rule**.



The area of a trapezoid is obtained by adding the area of arectangle and a triangle:

$$A_1 = y_0\Delta x + \frac{1}{2}(y_1 - y_0)\Delta x = \frac{(y_0 + y_1)\Delta x}{2}$$



Numerical integral functions

Function	Description	Example
`diff`	Difference and approximate derivative `diff(x)`: $[x_2 - x_1, x_3 - x_2, \cdots x_n - x_{n-1}]$	$\mathrm{dydx} = \mathrm{diff(y)./diff(x)}$
`trapz`	Computes the approximate integral using the trapezoidal method	trapz(x,y)
`quad`	Numerically evaluate integral, adaptive Simpson quadrature method	quad(func,a,b)
`quadl`	Same as quad but uses adaptive Labatto quadrature	quadl(func,a,b)
`polyint`	Integrate polynomial analytically	polyint(p)
`integral`	Numerically integrates function func from xmin to xmax	integral(func,xmin,xmax)

`func` is a scalar-valued function from a to b.

Example 7

Use the trapezoid rule and `diff` function to solve the numerical integral of x^2 from 0 to 1.

```
x = 0:0.1:1;
func = @(x) x.^2;
y = func(x);
plot(x,y)

% Calculate the Integral (Trapezoid method):
avg_y = y(1:length(x)-1) + diff(y)/2;
A = sum(diff(x).*avg_y)

% Use built-in trapz function:
A_trapz = trapz(x,y)

% Calculate the Integral (Simpson method):
A_simp = quad('x.^2', 0,1)

% Calculate the Integral (Lobatto method):
A_lab = quadl(@(x) func(x), 0,1)
```