Numerical Techniques
Lecture 12
After studying this chapter you should be able to...

- interpolate between data points, using either linear or cubic spline models
- model a set of data points as a polynomial
- use the basic fitting tool
- use the curve-fitting toolbox
- perform numerical differentiations
- perform numerical integrations
- solve differential equations numerically
Section 13.1
Interpolation

• When you take data, how do you predict what other data points might be?

• Two techniques are:
  • Linear Interpolation
  • Cubic Spline Interpolation
What is the corresponding value of y for this x?
Linear Interpolation

- Assume the function between two points is a straight line
How do you find a point in between?

$X=2$, $Y=?$
Linear Interpolation – Connect the points with a straight line to find $y$
MATLAB Code

- interp1 is the MATLAB function for linear interpolation.
- First define an array of x and y.
- Now define a new x array, that includes the x values for which you want to find y values.
- new_y=interp1(x,y,x_new)
```matlab
>> x = 0:5;
>> y = [15, 10, 9, 6, 2, 0];
>> interp1(x,y,3.5)
ans =
    4
```

**Measured Data**

- **x-axis**: 0, 1, 2, 3, 4, 5, 6
- **y-axis**: 0, 2, 4, 6, 8, 10, 12, 14, 16

 interpolated value at 3.5 is 4
>> x = 0:5;
>> y = [15, 10, 9, 6, 2, 0];
>> interp1(x,y,3.5)
ans =
    4
>> new_x = 0:0.2:5;
>> new_y = interp1(x, y, new_x)
new_y =
Columns 1 through 4
     15.0000    14.0000    13.0000    12.0000
Columns 5 through 8
     11.0000    10.0000    9.8000    9.6000
Columns 9 through 12
Columns 13 through 16
     7.8000    7.2000    6.6000    6.0000
Columns 17 through 20
     5.2000    4.4000    3.6000    2.8000
Columns 21 through 24
     2.0000    1.6000    1.2000    0.8000
Columns 25 through 26
     dy = diff(y)
    f = matlabFunction(dy)
    f(2)
clear,clc
    x = 0:5;
    y = [15, 10, 9, 6, 2, 0];
    new_x = 0:0.2:5;
    new_y = interp1(x, y, new_x)
Both measured data points and interpolated data were plotted on the same graph. The original points were modified in the interactive plotting function to make them solid circles.
Cubic Spline

- A cubic spline creates a smooth curve, using a third degree polynomial
We can get an improved estimate by using the spline interpolation technique.
```matlab
>> x = 0:5;
>> y = [15, 10, 9, 6, 2, 0];
>> interp1(x,y,3.5,'spline')
ans =
    3.9417
>> new_x = 0:0.2:5;
>> new_y_spline = interp1(x,y,new_x,'spline');
>> plot(x,y, new_x,new_y_spline, '-o')
```
Cubic Spline Interpolation. The data points on the smooth curve were calculated. The data points on the straight line segments were measured. Note that every measured point also falls on the curved line.
Other Interpolation Techniques

• MATLAB includes other interpolation techniques including
  • Nearest Neighbor
  • Cubic
  • Two dimensional
  • Three dimensional

• Use the help function to find out more if you are interested
Section 13.2
Curve Fitting

- There is scatter in all collected data
- We can estimate the equation that represents the data by “eyeballing” a graph
- There will be points that do not fall on the line we estimate
This line is just an “eyeballed guess”
Least Squares

- Finds the “best fit” straight line
- Minimizes the amount each point is away from the line
- It’s possible none of the points will fall on the line
- Linear Regression
Polynomial Regression

• Linear Regression finds a straight line, which is a first order polynomial

• If the data doesn’t represent a straight line, a polynomial of higher order may be a better fit
• In MATLAB you do both linear and polynomial regression the same way – the only difference is the order
polyfit and polyval

• **polyfit** finds the coefficients of a polynomial representing the data

• **polyval** uses those coefficients to find new values of $y$, that correspond to the known values of $x$
Coefficients of the first order polynomial describing the best fit line:
\[ y = -2.9143 \times x + 14.2857 \]

Evaluate how close a fit you've achieved by taking the difference between the measured and calculated points.
Least Squares Fit

\[ \sum (y - y_{calc})^2 \]

The polyfit function minimizes this number
Second Order Fit

\[ y = 0.0536 \times x^2 - 3.1821 \times x + 14.4643 \]
A fifth order polynomial gives a perfect fit to 6 points
Improve your graph by adding more points.
Section 13.3
Using the Interactive Curve Fitting Tools

- MATLAB 7 includes new interactive plotting tools.
- They allow you to annotate your plots, without using the command window.
- They include
  - basic curve fitting,
  - more complicated curve fitting
  - statistical tools
Use the curve fitting tools...

- Create a graph
- Making sure that the figure window is the active window select
  - Tools-> Basic Fitting
  - The basic fitting window will open on top of the plot
\[
x = 0:5; \\
y = [0,20,60,68,77,110]; \\
\text{plot}(x,y,'o') \ \\
\text{axis}([-1,7,-20,120])
\]
Rate of temperature change, degrees/hour

Rate of Change
time, hour
Plot generated using the Basic Fitting Window

$y = 21x + 3.8$

$y = 1.1x^3 - 9.3x^2 + 41x - 3.1$
Residuals are the difference between the actual and calculated data points.
Basic Fitting Window
You can also access the data statistics window from the figure menu bar.

Select **Tools->Data Statistics** from the figure window.

This window allows you to calculate statistical functions interactively, such as mean and standard deviation, based on the data in the figure, and allows you to save the results to the workspace.
Curve Fitting Toolbox

- An optional toolbox that allows more sophisticated curve fitting
- It must be installed on your computer
- To open type
  - `cftool`
Choose the data you want to plot and analyze in the Curve Fitting and Data Windows
This plot of US Census Data was created using the cftool interactive modeling capability
Section 13.4

Differences and Numerical Differentiation

• We used symbolic differentiation in a previous chapter to find the equation of the slope of a line

• Numerical differentiation is similar, and uses the \texttt{diff} function

• This can be confusing – since the same syntax is used for both
Diff Function

• The **diff** function is easy to understand, even if you haven’t taken Calculus

• It just calculates the difference between the points in an array
The derivative of a data set can be approximated by finding the slope of a straight line connecting each data point.
The slope is an approximation of the derivative – in this case based on data measurements.
The calculated slopes are discontinuous, if they are based on data. The appearance of this graph was adjusted using the interactive plotting tools.
Approximating Derivatives when you know the function

- If we know how $y$ changes with $x$, we could create a set of ordered pairs for any number of $x$ values. The more values of $x$ and $y$, the better the approximation of the slope.
The slope of a function is approximated more accurately, when more points are used to model the function.
Forward, Backward and Central Difference Techniques

- What if you want to approximate the derivative at a point, instead of over a range?
- One approach is to use the slope between adjacent points

\[
\left( \frac{dy}{dx} \right) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}
\]

Implement in MATLAB with
\[dydx = \text{diff}(y)./\text{diff}(x)\]

Forward Difference
Let’s approximate the derivative of \( \sin(x) \)

- We know from basic calculus that the derivative of \( \sin(x) \) is \( \cos(x) \)

\[
y = \sin(x) \\
\frac{dy}{dx} = \cos(x)
\]

- How accurate an estimate can we get by using a difference calculation?
Notice that the size of dydx_analytical and dydx_approx are different.
$$f'$$

NaN is a placeholder
```matlab
>> x=linspace(0,pi/2,10);
>> y=sin(x);
>> dydx_analytical = cos(x);
>> dydx_approx = diff(y)./diff(x);
>> dydx_approx(length(x))=NaN;
>> error = dydx_analytical-dydx_approx;
>> error_percentage = error/dydx_analytical*100;
```

**Forward Difference Approximation of the Derivative of Sin(x)**

<table>
<thead>
<tr>
<th>x</th>
<th>dy/dx</th>
<th>forward approximation</th>
<th>%error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>1.0000</td>
<td>cos(x)</td>
<td></td>
</tr>
<tr>
<td>0.1745</td>
<td>0.9848</td>
<td>0.9647</td>
<td>2.0418</td>
</tr>
<tr>
<td>0.3491</td>
<td>0.9397</td>
<td>0.9052</td>
<td>3.6751</td>
</tr>
<tr>
<td>0.5236</td>
<td>0.8660</td>
<td>0.8181</td>
<td>5.5325</td>
</tr>
<tr>
<td>0.6981</td>
<td>0.7660</td>
<td>0.7062</td>
<td>7.8109</td>
</tr>
<tr>
<td>0.8727</td>
<td>0.6428</td>
<td>0.5728</td>
<td>10.806</td>
</tr>
<tr>
<td>1.0472</td>
<td>0.5000</td>
<td>0.4221</td>
<td>15.5836</td>
</tr>
<tr>
<td>1.2217</td>
<td>0.3420</td>
<td>0.2585</td>
<td>24.4224</td>
</tr>
<tr>
<td>1.3963</td>
<td>0.1736</td>
<td>0.0870</td>
<td>49.8727</td>
</tr>
<tr>
<td>1.5708</td>
<td>0.0000</td>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NaN</td>
<td></td>
</tr>
</tbody>
</table>
Calculation of the Derivative of $\sin(x)$
Using the Forward Difference Technique

- analytical solution
- approximation with 10 points
- approximation with 20 points

Rate of Change
time, hour
Rate of temperature change, degrees/hour

$\frac{dy}{dx}$
angle in radians
Backward Differences

- Instead of approximating the derivative by forecasting forward, we can use the point before the current point in the approximation.

\[
\left( \frac{dy}{dx} \right) = \frac{y_i - y_{i-1}}{x_i - x_{i-1}}
\]
Backward Difference Approximation of the Derivative of Sin(x)

<table>
<thead>
<tr>
<th>x</th>
<th>dy/dx (cos(x))</th>
<th>dy/dx (backward approximation)</th>
<th>%error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>1.0000</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>0.1745</td>
<td>0.9848</td>
<td>0.9949</td>
<td>-1.0279</td>
</tr>
<tr>
<td>0.3491</td>
<td>0.9397</td>
<td>0.9647</td>
<td>-2.6613</td>
</tr>
<tr>
<td>0.5236</td>
<td>0.8660</td>
<td>0.9052</td>
<td>-4.5186</td>
</tr>
<tr>
<td>0.6981</td>
<td>0.7660</td>
<td>0.8181</td>
<td>-6.7970</td>
</tr>
<tr>
<td>0.8727</td>
<td>0.6428</td>
<td>0.7062</td>
<td>-8.9667</td>
</tr>
<tr>
<td>1.0472</td>
<td>0.5000</td>
<td>0.5728</td>
<td>-14.5697</td>
</tr>
<tr>
<td>1.2217</td>
<td>0.3420</td>
<td>0.4221</td>
<td>-23.4085</td>
</tr>
<tr>
<td>1.3963</td>
<td>0.1736</td>
<td>0.2585</td>
<td>-48.8588</td>
</tr>
<tr>
<td>1.5708</td>
<td>0.0000</td>
<td>0.0870</td>
<td>-142155539756746180.0000</td>
</tr>
</tbody>
</table>
Central Difference

- The absolute error associated with the forward difference and backward difference is very similar.

- We can get closer using a central difference – where we use the point before and the point after the point of interest.

\[
\frac{dy}{dx} = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}
\]
One downside of this approach is that it won’t work for either the first point or the last point.

The gradient function uses:

- Forward difference for the first point
- Backward difference for the last point
- Central difference for the intermediate points
Comparison of Calculation Techniques for the Derivative of $\sin(x)$
Numerical Integration

- MATLAB handles numerical integration with two different quadrature functions
  - `quad`
  - `quadl`
An integral is often thought of as the area under a curve.

The area under a curve can be approximated using the trapezoid rule.
Trapazoid rule

These areas are equal
The integral of a function can be estimated using the trapezoid rule.
Quadrature functions

- **quad** uses adaptive Simpson quadrature
- **quadl** uses adaptive Lobatto quadrature
- Both functions require the user to enter a function in the first field.
  - called out explicitly as a character string
  - defined in an M-file
  - anonymous function.
- The last two fields in the function define the limits of integration
Quadrature Functions

>> quad('x.^2', 0,1)
ans =
   0.3333

>> quadl('x.^2', 0,1)
ans =
   0.3333
Section 13.6
Solving Differential Equations Numerically

• MATLAB includes a number of functions that solve ordinary differential equations of the form

$$\frac{dy}{dt} = f(t, y)$$

• Higher order differential equations (and systems of differential equations) must be reformulated into a system of first order expressions.
Family of functions

- Not every differential equation can be solved using the same technique
- All of the differential equation solvers have the same format
- This makes it easy to try different techniques, by just changing the function name
A minimum of three inputs is required

- function handle to a function that describes the first order differential equation or system of differential equations in terms of t and y
- The time span of interest
- An initial condition for each equation in the system
• If you don’t specify the result array, \([t,y]\), the functions create a plot of the results

\[
[t,y] = \text{odesolver}(\text{function\_handle, [initial\_time, final\_time]}, [\text{initial\_cond\_array}])
\]
Function handles

• A function handle is a “nickname” for a function.
• The function handle can refer to
  • a standard MATLAB function, stored as an M-file
  • an anonymous MATLAB function.
• Function handles were described in Chapter 6
Anonymous functions

- Anonymous functions can be defined in an M-file program, or in the command window
This function corresponds to:

\[ \frac{dy}{dt} = 2 \times t \]
You can use an anonymous function as input to an ode solver function

Because we did not assign a name to the ode function output, a graph was created
This figure was generated automatically by the **ode45** function. The title and labels were added in the usual way.
Systems of differential equations

- If you want to specify a system of equations, it is probably easier to define a function m-file. The output of the function must be a column vector of first derivative values.

- For example...

\[
\frac{dy}{dt} = x \\
\frac{dx}{dt} = -y
\]
This function M-file represents a system of equations

\[
\frac{dy}{dt} = x
\]

\[
\frac{dx}{dt} = -y
\]

\[
y_1' = y_2
\]

\[
y_2' = -y_1
\]
Using an M-file function as input to an ODE solver

• The handle for an existing m-file is `@m_file_name`

```matlab
ode45(@two_funs, [-1,1],[1,1])
```
This system of equations was solved using ode45. The title, labels and legend were added in the usual way.
<table>
<thead>
<tr>
<th>Ode solver function</th>
<th>Type of problems likely to be solved with this technique</th>
<th>Numerical Solution Method</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ode45</td>
<td>non-stiff differential equations</td>
<td>Runge-Kutta</td>
<td>Best choice for a first guess technique, if you don’t know much about the function. Uses an explicit Runge-Kutta (4,5) formula, called the Dormand-Prince pair</td>
</tr>
<tr>
<td>ode23</td>
<td>non-stiff differential equations</td>
<td>Runge-Kutta</td>
<td>This technique uses an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. If the function is “mildly stiff” this may be a better approach</td>
</tr>
<tr>
<td>ode113</td>
<td>non-stiff differential equations</td>
<td>Adams</td>
<td>Unlike ode45 and ode23 which are single step solvers, this technique is a multistep solver.</td>
</tr>
<tr>
<td>ode15s</td>
<td>stiff differential equation and differential algebraic equations</td>
<td>NDFs (BDFs)</td>
<td>Uses numerical differentiation formulas or backward differentiation formulas. It is difficult to predict which technique will work best for a stiff differential equation</td>
</tr>
<tr>
<td>ode23s</td>
<td>stiff differential equations</td>
<td>Rosenbrock</td>
<td>modified second order Rosenbock formulation</td>
</tr>
<tr>
<td>ode23t</td>
<td>moderately stiff differential equations and differential algebraic equations</td>
<td>Trapazoid rule</td>
<td>Useful if you need a solution without numerical damping</td>
</tr>
<tr>
<td>ode23tb</td>
<td>stiff differential equations</td>
<td>TR-BDF2</td>
<td>This solver uses an implicit Runge-Kutta formula with the trapazoid rule and a second order backward differentiation formula.</td>
</tr>
<tr>
<td>ode15i</td>
<td>fully implicit differential equations</td>
<td>BDFs</td>
<td>.</td>
</tr>
</tbody>
</table>
Summary

- MATLAB includes an interpolation technique called \texttt{interp1}
- Most commonly used for linear interpolations
- A cubic spline technique is also available
Summary – Curve Fitting

• **polyfit** returns the coefficients of a polynomial model of the data

\[ y = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \ldots + a_{n-1}x + a_n \]

• **polyval** uses those coefficients to find new values of y for specified values of x
Summary – Interactive Curve Fitting

• A basic set of interactive curve fitting tools comes standard with MATLAB
  • Access these tools from the tools menu
  • The Curve Fitting Toolbox offers more extensive tools and is an optional component
Summary

Numerical Derivatives

- The `diff` function finds the difference between adjacent elements in an array
- It can be used to approximate a derivative
Summary - Integration

- MATLAB includes two quadrature functions for use in approximating integrals
  - `quad`
  - `quadl`
Summary – Differential Equations

• MATLAB includes a family of differential equation solvers
• All of the solver functions use a common format
• A good first try is usually the **ode45** solver function, which uses a Runge-Kutta technique.