Computing Solutions of Ordinary Differential Equations

Topics covered in this lecture:

1. Solution of first-order problems
   a. Euler method
   b. Modified Euler method
   c. Runge-Kutta methods
   d. Awareness of other predictor-corrector methods used in practice
2. Solving higher-order differential equations

Motivation

- Analysis of Engineering problems generate lots of differential equations, most of which cannot be easily solved explicitly
- We must learn to solve them numerically on a computer.
- There is a close relationship between solving a finite integral using an approximation method and solving initial value problems as we have here.
- This motivates the development of the techniques outlined below.
First Order Equations

Start with first order equations:

Simplest form, \( y' = \frac{dy}{dx} = f(x) \), where the right hand side is a function of \( x \) only.

Easily solved by a direct method,

Solution \( y(x) = \int_a^x f(x) \, dx + C \),

Constant \( C \) is evaluated from some initial condition, \( y(a) \) say.

More generally \( f(x, y) \) is a function of both \( x \) and \( y \) and we need other methods.

We will look at three methods,

\( \text{Euler, Modified Euler (4th order) Runge-Kutta} \)
Euler Method

Divide the region of interest [a,b] up into discrete values of $x = nh, \ n = 0,1,\cdots,N$, spaced at interval $h=(b-a)/N$. Use the forward difference approximation for the differential coefficient:

$$f(x_n, y_n) \approx \frac{y_{n+1} - y_n}{h}$$

Rearrange to give:

$$y_{n+1} = y_n + hf(x_n, y_n)$$

Using the usual notation $y(nh) = y_n$ etc, and continuing to the next points, step along $x = nh, \ n = 0,1,\cdots,N$ using

$$y_{n+1} = y_n + hf(x_n, y_n)$$

This process is most easy to remember by its graphical construction.
**Example**  Let's solve \( y' = \frac{dy}{dx} = f(x, y) = x + y + 1 \) for \( x = [0, 2] \) with initial condition \( y(0) = 0 \) and compare the Euler Solution with the exact solution \( y = 2e^x - x - 2 \) (check this!) with \( h = 1.0 \). and plot the result:

<table>
<thead>
<tr>
<th>( n )</th>
<th>( x_n )</th>
<th>( y_n )</th>
<th>( f_n )</th>
<th>True ( y_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>1.000</td>
<td>3.000</td>
<td>2.437</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.000</td>
<td>7.000</td>
<td>10.778</td>
<td></td>
</tr>
</tbody>
</table>

The Euler solution rapidly diverges from the true solution

Try \( h = 0.1 \) and the Euler solution is better.

---

**Accuracy of Euler Method**

How accurate is the Euler solution?

Expand in Taylor series: \( y_{n+1} = y_n + h \frac{dy}{dx} + \frac{h^2}{2} y'' + \cdots = y_n + hf_n + \frac{h^2}{2} y'' + \cdots \)

- Thus the error per step is \( O(h^2) \).
- There are \( (L/h=(b-a)/h) \) steps in the interval \( [x_0, x_n] \), so.

*The global error in the Euler method is \( O(h) \).*
Modified Euler Method

Recall the Euler governing equation
\[ y_{n+1} = y_n + hf(x_n, y_n) \]

A better estimation of the slope from \((x_n, y_n)\) to \((x_{n+1}, y_{n+1})\) would be
\[ y_{c_{n+1}} = y_n + \frac{h}{2} \left( f(x_n, y_n) + f(x_{n+1}, y_{c_{n+1}}) \right), \]

BUT we don’t know \(y_{n+1}\).

However, we can estimate it by using the Euler method, to give a two-stage predictor-corrector algorithm that is called the Modified Euler-algorithm:
Modified Euler algorithm

Step 1: Predictor
\[ y_{n+1}^* = y_n + hf(x_n, y_n) \]

Step 2: Corrector
\[ y_{n+1} = y_n + \frac{h}{2} \left( f(x_n, y_n) + f(x_{n+1}, y_{n+1}^*) \right) \]

- Step 1 gives an estimate of \( y' \) at \( x_n \).
- This is used to predict a first guess \( y_{n+1}^* \), which is used to get an estimate of the average \( y' \) over the interval \( h \).
- This is then used to correct the estimate of \( y_{n+1} \) in Step 2.

- Again, use the geometric construction to help remember how it works.
- Equations are in HLT.
- Also called a second-order Runge-Kutta method (why? See later)

Example

Use modified Euler on our example \( y' = \frac{dy}{dx} = f(x, y) = x + y + 1 \) for \( x = [0,2] \) with initial condition \( y(0) = 0 \). For \( h = 1.0 \).

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
n & x_n & y_n & f_n & y_n & f_n & \text{True} y_n \\
\hline
0 & 0.00 & 0.00 & - & 0.000 & 1.00 & 0.00 \\
1 & 1.00 & 1.00 & 3.00 & 2.000 & 4.00 & 2.437 \\
2 & 2.00 & 5.00 & 8.00 & 8.000 & 11.0 & 10.778 \\
\hline
\end{array}
\]

The solution is much more accurate than that for the simple Euler method.
In the \( h = 0.1 \) case plotted above, the difference is only 0.111 after 2 seconds.
**Accuracy of Modified Euler**

Use the normal Taylor Series method:

Define local error $\epsilon$

$$y_{n+1} = y_n + \frac{h}{2} (f_n + f_{n+1}) + \epsilon$$

Now, expanding $f_{n+1}$ as a two-dimensional Taylor Series in terms of $x_n + h \cdot y_n + hf_n$

$$f_{n+1} = f(x_n + h, y_n + hf_n) = f(x_n, y_n) + h \frac{\partial f}{\partial x} + hf_n \frac{\partial f}{\partial y} + O(h^2)$$

Thus

$$y_{n+1} = y_n + \frac{h}{2} (f_n + f_{n+1} + h \frac{\partial f}{\partial x} + hf_n \frac{\partial f}{\partial y} + O(h^2)) + \epsilon$$

$$= y_n + hf_n + \frac{h^2}{2} \frac{\partial f}{\partial x} + \frac{h^2}{2} f_n \frac{\partial f}{\partial y} + O(h^3) + \epsilon$$
Thus

\[ y_{n+1} = y_n + \frac{h}{2} \left( f_n + f_{n+1} + h^2 \frac{\partial f_n}{\partial x} + h f_n \frac{\partial f_n}{\partial y} + O(h^2) \right) + O(e) \]

\[ = y_n + hf_n + \frac{h^2}{2} \frac{\partial f_n}{\partial x} + \frac{h^2}{2} f_n \frac{\partial f_n}{\partial y} + O(h^3) + O(e) \]

\[ = y_n + hf_n + \frac{h^2}{2} \left( \frac{\partial f_n}{\partial x} + f_n \frac{\partial f_n}{\partial y} \right) + O(h^3). \]

- It can be seen from this that the local, per step, error \( e = O(h^3) \).
- Hence the global error for the modified Euler method is \( O(h^2) \).
- Also called a Runge-Kutta method of order 2

**Runge-Kutta and Other Methods**
Runge-Kutta methods

- The 4th order Runge-Kutta method is popular, and uses several predictive steps, not just one. The algorithm is discussed in Kreyzig (pp.1040, 7th Ed.) and is listed in HLT (p.29).

- It can be proved that it is locally $O(h^5)$ and hence globally $O(h^4)$ [Most of us take this proof on trust!].

- It involves calculating four auxiliary quantities $k_1$, $k_2$, $k_3$ and $k_4$.

Fourth order Runge-Kutta algorithm

\[
\begin{align*}
    k_1 &= hf(x_n, y_n) \\
    k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\
    k_3 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2) \\
    k_4 &= hf(x_n + h, y_n + k_3) \\
    y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{align*}
\]

Multi-Step Methods.

- There are other multistep methods, where the computed solutions depend on more than one past step of $y_n$.

- Popular methods include the Adams-Bashforth and the Adams-Moulton algorithms. See the text books for further details (eg Burden and Faires Ch 5.6)
Multi-Step Methods (2)

- Adams-Bashforth technique

\[
y_n^{\text{adv}} = y_n + \frac{h}{24} (55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})
\]

\[
y_n^{\text{adv}} = y_n + \frac{h}{24} (9f_n^{\text{rel}} + 19f_n - 5f_{n-1} + f_{n-2})
\]

- A-B and 4\textsuperscript{th} R-K are of similar accuracy but A-B is more efficient since the \( y_n \rightarrow y(n-2) \), \( f(n-1) \rightarrow f(n-2) \) can be stored from previous steps and only two computations of \( f(x,y) \) are required per step.

- Many computer algorithms start with 4\textsuperscript{th} R-K to get 4 points and then change to A-B for speed.

MATLAB routines for solving O.D.E.’s.

- `ode45` and `ode23` are based on an explicit Runge-Kutta formula.
- `ode113` is a variable order Adams-Bashforth-Moulton solver.
- Look at manuals to see descriptions of these and others.
Second Order and Higher equations

Consider a second-order equation of the form: \( y'' = f(x, y, y') \).

**To solve:** convert to a pair of first order equations by considering \( y' \) as an independent variable:

\[
\frac{dy'}{dx} = f(x, y, y')
\]
\[
\frac{dy}{dx} = y'
\]

This can be written in matrix form:

\[
\frac{d}{dx} \begin{bmatrix} y \\ y' \end{bmatrix} = \begin{bmatrix} y' \\ f(x, y, y') \end{bmatrix}
\]

or

\[
\frac{dY}{dx} = F(x, Y).
\]

The **Euler** algorithm becomes

\[
Y_{n+1} = Y_n + hF_n
\]

The **Modified Euler** becomes

\[
Y_{n+1}^* = Y_n + hF_n
\]
\[
F_{n+1}^* = F(x_{n+1}, Y_{n+1}^*)
\]
\[
Y_{n+1} = Y_n + \frac{h}{2} (F_n + F_{n+1}^*)
\]

Start from initial value \( Y(0) = \begin{bmatrix} y(0) \\ y'(0) \end{bmatrix} \)
Example  Simple harmonic oscillator

\[ y'' = -y \quad \text{with initial conditions} \quad y(0) = 1, \quad y'(0) = 0. \]

In matrix form, this equation can be written as

\[
\begin{bmatrix}
  y' \\
  y''
\end{bmatrix}
= \begin{bmatrix}
  y' \\
  -y
\end{bmatrix}
\quad \text{with initial conditions}
\begin{bmatrix}
  y_0' \\
  y_0''
\end{bmatrix}
= \begin{bmatrix}
  0 \\
  1
\end{bmatrix}.
\]

Write a MATLAB function

```matlab
function shm(h)
    % Euler algorithm to evaluate y'' = -y, y(0) = 1, y'(0) = 0.
    % h is the step size.
    x = [0:h:7]'; % x range
    fn = cos(x); % exact solution
    y = x;
    % get matrix sizes for y and y'
yd = x;
y(1) = 1; % initial conditions
    yd(1) = 0;
    for i=1:(length(x)-1)
        y(i+1) = y(i) + h*yd(i); % Euler algorithm
        yd(i+1) = yd(i) - h*y(i);
    end
    plot(x,fn,x,y, '*')
end
```

run this with a step size \( h = 0.2 \)
Accuracy is poor with $h = 0.2$ it improves with either smaller $h$ or, preferably, by using modified Euler or Runge Kutta.

Summary

Topics covered in this lecture:

1. Solution of first-order problems
   a. Euler method
   b. Modified Euler method
   c. Runge-Kutta methods
   d. Awareness of other predictor-corrector methods used in practice
2. Solving higher-order differential equations