Numerical Solution Techniques in Mechanical and Aerospace Engineering

Chunlei Liang

LECTURE 1
Numerical methods for solving ODEs

1.1. Outline of Lecture
- What is an Ordinary Differential Equation?
- Explicit Runge-Kutta type schemes
- Solve Initial Value Problems (IVPs)
- Solve BVPs using shooting method
- Solve BVPs using finite difference method

1.2. What is an ODE?
- Differential equation: A differential equation is a mathematical equation for an unknown function of one or several variables that relates the values of the function itself and its derivatives of various orders.
- Ordinary differential equation: In mathematics, an ordinary differential equation (or ODE) is a relation that contains functions of only one independent variable, and one or more of their derivatives with respect to that variable.
- Partial differential equation:
  In mathematics, partial differential equations (PDE) are a type of differential equation, i.e., a relation involving an unknown function (or functions) of several independent variables and their partial derivatives with respect to those variables.
You’ve probably all seen an ordinary differential equation (ODE); for example the pendulum equation,

\[
\frac{d^2 \Theta}{dt^2} + \frac{g}{L} \sin \Theta = 0,
\]

describes the angle, \( \Theta \), a pendulum makes with the vertical as a function of time, \( t \). Here \( g \) and \( L \) are constants (the acceleration due to gravity and length of the pendulum respectively), \( t \) is the independent variable and \( \Theta \) is the dependent variable. This is an ODE because there is only one independent variable, here \( t \) which represents time.

### 1.3. Some explicit schemes

Let an initial value problem be specified as follows.

A first ODE is given as \( \frac{dy}{dx} = f(x, y) \) and the initial condition is \( y(x_0) = y_0 \).

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward Euler</td>
<td>( y_{n+1} = y_n + hf_n )</td>
</tr>
<tr>
<td>Modified Euler</td>
<td>( y_{n+1} = y_n + hf \left( x_n + \frac{h}{2}, y_n + \frac{h}{2} f_n \right) )</td>
</tr>
<tr>
<td>Improved Euler</td>
<td>( y_{n+1} = \frac{1}{6} (f_n + 2f_p) )</td>
</tr>
</tbody>
</table>
| Runge-Kutta 3rd order       | \( \begin{align*} 
    k_1 &= f_n, \\
    k_2 &= f \left( x_n + \frac{h}{2}, y_n + \frac{h}{2} k_1 \right), \\
    k_3 &= f \left( x_{n+1}, y_n - hk_1 + 2hk_2 \right), \\
    y_{n+1} &= y_n + \frac{h}{6} (k_1 + 4k_2 + k_3) 
\end{align*} \) |
| Runge-Kutta 4th order       | \( \begin{align*} 
    k_1 &= f_n, \\
    k_2 &= f \left( x_n + \frac{h}{2}, y_n + \frac{h}{2} k_1 \right), \\
    k_3 &= f \left( x_{n+1}, y_n + hk_3 \right), \\
    k_4 &= f \left( x_{n+1}, y_n + h \frac{k_3}{2} \right), \\
    y_{n+1} &= y_n + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) 
\end{align*} \) |

A class of Runge-Kutta type methods can be generalized as follows:

\[
y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i
\]

where \( k_i = f \left( x_n + c_i h, y_n + h \sum_{j=1}^{i} a_{ij} k_j \right) \).

To specify a particular method, one needs to provide the integer \( s \) (the number of stages), and the coefficients \( a_{ij} \) (for \( 1 <= j < i <= s \)).
bi (for i = 1, 2, ..., s) and ci (for i = 2, 3, ..., s). These data are usually arranged in a mnemonic device, known as a Butcher tableau.

\[
\begin{array}{c|cccc}
  c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\
  c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\
  \hline
  b_1 & b_2 & \cdots & b_s
\end{array}
\]

The Runge-Kutta method is consistent if \( \sum_{j=1}^{i-1} a_{ij} = c_i \) for \( i = 2, \ldots, s \).

The simplest Runge-Kutta method is the (forward) Euler method. The formula is given in table 1. The corresponding tableau is

\[
\begin{array}{c|cc}
  0 & \ 0 \\
  1 & \ 1
\end{array}
\]

An example of a second-order method with two stages is the mid-point method (Modified Euler method). The corresponding tableau is:

\[
\begin{array}{c|cc}
  0 & 0 & 0 \\
  1/2 & 1/2 & 0 \\
  \hline
  0 & 1/2 \\
  1/2 & 1/2 
\end{array}
\]

The Improved Euler method can also be tabulated:

\[
\begin{array}{c|cc}
  0 & 0 & 0 \\
  1 & 1 & 0 \\
  \hline
  1/2 & 1/2
\end{array}
\]

We can also drive a third-order Runge-Kutta method which falls in this framework. Its tableau is:

\[
\begin{array}{c|ccc}
  0 & 0 & 0 & 0 \\
  1/2 & 1/2 & 0 & 0 \\
  1 & -1 & 2 & 0 \\
  \hline
  1/6 & 2/3 & 1/6
\end{array}
\]

1.4. Solve Initial Value Problems using the Runge-Kutta methods

In mathematics, in the field of differential equations, an initial value problem is an ordinary differential equation together with specified value, called the initial condition, of the unknown function at a given point in the domain of the solution.

An example is

\[
\begin{cases}
  \frac{du}{dx} = \tan(x)u, & 0 \leq x < \pi/2 \\
  u(0) = 1
\end{cases}
\]
1.4.1. Sample code for solving a first order ODE

A sample code named ivprk4.cpp for solving the above IVP (see equation 1.3) is provided on mae286 course website. Try it out and it will be fun!

1.5. Finite differences and truncation error

Before addressing boundary value problems, we want to develop further the notion of finite difference approximation of derivatives.

A finite difference is a mathematical expression of the form \( f(x + b) - f(x + a) \). If a finite difference is divided by \( b - a \), one gets a difference quotient. The approximation of derivatives by finite differences plays a central role in finite difference methods for the numerical solution of differential equations, especially boundary value problems.

Truncation error is error made by numerical algorithms that arises from taking finite number of steps in computation. It is present even with infinite-precision arithmetic, because it is caused by truncation of the infinite Taylor series to form the algorithm.

Numerically, we often approximate the derivatives via a finite Taylor series expansion. Recall that a Taylor Series provides a value for a function \( f = f(x) \) when the dependent variable \( x \in \mathbb{R} \) is translated by an amount \( \Delta x \), in terms of its derivatives at that point. Thus, \( f(x + \Delta x) \) can be calculated by:

\[
f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{f''(x)(\Delta x)^2}{2!} + \frac{f'''(x)(\Delta x)^3}{3!} + O(\Delta x)^4
\]

Note that \( O(\Delta x) \) represents additional terms that are bounded by a constant term multiplied by \( \Delta x \).

In a similar manner, we can determine \( f \) when the dependent variable \( x \) is translated by an identical amount \( \Delta x \) in the opposite direction:

\[
f(x - \Delta x) = f(x) - f'(x)\Delta x + \frac{f''(x)(\Delta x)^2}{2!} - \frac{f'''(x)(\Delta x)^3}{3!} + O(\Delta x)^4
\]

These two equations when rearranged allow a first order approximation for the derivative \( f'(x) \):

\[
f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} + O(\Delta x)
\]

\[
= \frac{f(x) - f(x - \Delta x)}{\Delta x} + O(\Delta x)
\]

In addition, subtracting the backward Taylor expansion from the forward Taylor expansion provides a second order approximation for the derivative \( f'(x) \):
\[ f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} + O(\Delta x)^2 \]

To obtain a second order approximation for the second derivative of \( f \), \( f''(x) \), we can add the forward and backward Taylor expansions to receive:

\[ f''(x) = \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{(\Delta x)^2} + O(\Delta x)^2 \]

1.6. Boundary value problems (BVPs)

The techniques for initial value problems (IVPs) are, in general, not directly applicable to BVPs. Consider the BVP

\[
\begin{aligned}
\frac{d^2 u}{dx^2} &= f(x, u, \frac{du}{dx}), \quad a \leq x \leq b \\
u(a) &= n_1, \quad u(b) = n_2
\end{aligned}
\]

Equation 1.4 could be nonlinear, depending on \( f \). The methods used for IVPs started at one end \( (x = a) \) and computed the solution step by step for increasing \( x \). For a BVP, not enough information is given at either endpoint to allow a step-by-step solution.

Subdivide the interval \( (a, b) \) into \( n \) equal subintervals (cells) using \( n + 1 \) nodal points. Each subinterval has a size of \( h = (b - a)/n \), so that

\[ x_k = a + kh, \quad k = 0, 1, 2, \ldots, n. \]

Consider first a special case of Equation 1.4 for which the right-hand side depends only on \( x \) and function \( u \) itself:

\[
\begin{aligned}
-\frac{d^2 u}{dx^2} + 2u &= f(x), \quad 0 \leq x \leq 1 \\
y(0) &= 0, \quad y(1) = 0
\end{aligned}
\]

where \( f(x) = x(5 - x)e^x \) and the analytical solution of the above BVP problem is \( u = x(1 - x)e^x \).

In the following, we illustrate how to solve equation 1.6 using the shooting method and the finite difference method.

1.6.1. Solve BVPs using the shooting method

The procedure in the shooting method is similar to the method adopted by a soldier to shoot a distant target. The elevation of the gun is adjusted if the shot misses the target. The name of this method originated from this practice.
The basic idea of the shooting method is to convert BVPs into IVPs and then solve IVPs using the techniques described in table 1. For instance, we can convert the second order ODE in equation 1.4 into two first order ODEs.

The initial conditions at \( x = a \) are

\[
(1.7) \quad u(a) = \eta_1, \quad \frac{du}{dx}(a) = C
\]

The choice of \( C \) shall be made in a way that the solution at the other end \( u(b) \) shall be satisfied, i.e. very close to the value in the original problem \( u(b) = \eta_2 \). The shooting method is based on an iterative procedure from which we can determine \( C \). The procedure as shown in figure 1 involves the following steps:

- Convert the BVP of equation 1.4 into two ODE IVPs. We can then solve them using the explicit Runge-Kutta type schemes under the initial conditions in equation 1.7.
- Evaluate the solution \( u(b) \) at the end point \( x = b \) and compare this with the target value of \( u(b) = \eta_2 \).
- Adjust the value of \( C \) (either bigger or smaller) until a desired level of accuracy is achieved. Once the accuracy is achieved, the numerical solution is found. Sometimes the bisection method can be used to search for \( C \).

1.6.2. Solve BVPs using the finite difference method

We can use a central difference approximation to the second derivative,
The discrete form of equation 1.6 is then

\[ -\frac{1}{h^2} (u_{k+1} - 2u_k + u_{k-1}) + 2u_k = f(x_k), \quad k = 1, 2, \ldots, n - 1. \]

Since this system of equations has \( n - 1 \) equations in \( n + 1 \) unknowns, the two boundary conditions are required to obtain a nonsingular system:

\[ u_0 = 0, \quad u_n = 0 \]

This is clearly a tridiagonal system, I will leave you to tell what are the coefficients of \( c_l, c_m \) and \( c_r \).

1.6.2.1. Solving Tridiagonal Systems. Tridiagonal systems are particularly easy to solve using Gaussian elimination. It is convenient to solve such systems using the following notation:

\[
\begin{align*}
    d_1 \Theta(x_1) + u_1 \Theta(x_2) &= b_1 \\
    l_2 \Theta(x_1) + d_2 \Theta(x_2) + u_2 \Theta(x_3) &= b_2 \\
    l_3 \Theta(x_2) + d_3 \Theta(x_3) + u_3 \Theta(x_4) &= b_3 \\
    & \vdots \\
    l_{n-2} \Theta(x_{n-3}) + d_{n-2} \Theta(x_{n-2}) + u_{n-2} \Theta(x_{n-1}) &= b_{n-2} \\
    l_{n-1} \Theta(x_{n-2}) + d_{n-1} \Theta(x_{n-1}) &= b_{n-1}
\end{align*}
\]

where \( d_i, u_i \) and \( l_i \) are, respectively, the diagonal, upper and lower entries in Row \( i \). The solution algorithm is to first obtain an upper triangular form through a forward Gaussian elimination followed by a direct back-solve process. This algorithm can be summarized as follows:

- For \( k = 1, 2, \ldots, n - 1: \) \([k = \text{pivot row}]\)
  - \( m = -l_{k+1}/d_k \) \([m = \text{multiplier needed to annihilate the following terms}]\)

\[
\begin{align*}
    d_1 \Theta(x_1) + u_1 \Theta(x_2) &= b_1 \\
    l_2 \Theta(x_1) + d_2 \Theta(x_2) + u_2 \Theta(x_3) &= b_2 \\
    \vdots \\
    l_{n-2} \Theta(x_{n-3}) + d_{n-2} \Theta(x_{n-2}) + u_{n-2} \Theta(x_{n-1}) &= b_{n-2} \\
    l_{n-1} \Theta(x_{n-2}) + d_{n-1} \Theta(x_{n-1}) &= b_{n-1}
\end{align*}
\]
8 CHUNLEI LIANG, NUMERICAL METHODS FOR ODES

(b) $d_{k+1} = d_{k+1} + m u_k$ [new diagonal entry in the next row]
(c) $b_{k+1} = b_{k+1} + m b_k$ [new right-hand-side term in the next row]

- $\Theta(x_n) = b_n / d_n$ [start of back solve]
- For $k = n - 1, n - 2, \ldots, 1$: [back-solve loop]
  (a) $\Theta(x_k) = (b_k - u_k \Theta(x_{k+1})) / d_k$

1.6.2.2. **Sample code.** A sample code named `fd1dode.cpp` for solving the above BVP (see equation 1.6) is provided on mae286 course website. Try it out and it will be fun!