REVIEW Lecture 19: Finite Volume Methods

- Review: Basic elements of a FV scheme and steps to step-up a FV scheme
- One Dimensional examples
  - Generic equation: \[ \frac{d}{dt} \left( \Delta x \Phi_j \right) + f_{j+1/2} - f_{j-1/2} = \int_{x_{j-1/2}}^{x_{j+1/2}} s(\xi, t) \, d\xi \]
  - Linear Convection (Sommerfeld eqn): convective fluxes
    - 2\textsuperscript{nd} order in space, then 4\textsuperscript{th} order in space, links to CDS
  - Unsteady Diffusion equation: diffusive fluxes
    - Two approaches for 2\textsuperscript{nd} order in space, links to CDS
- Two approaches for the approximation of surface integrals (and volume integrals)
- Interpolations and differentiations (express symbolic values at surfaces as a function of nodal variables)
  - Upwind interpolation (UDS): \[ \phi_e = \begin{cases} \phi_P & \text{if } (\vec{v} \cdot \vec{n})_e > 0 \\ \phi_E & \text{if } (\vec{v} \cdot \vec{n})_e < 0 \end{cases} \] (first-order and diffusive)
  - Linear Interpolation (CDS): \[ \phi_e = \phi_E \lambda_e + \phi_P (1 - \lambda_e) \] where \[ \lambda_e = \frac{x_e - x_P}{x_E - x_P} \] (2\textsuperscript{nd} order, can be oscillatory)
  - Quadratic Upwind interpolation (QUICK)
    \[
    \phi_e = \begin{cases} \phi_U + g_1 (\phi_D - \phi_U) + g_2 (\phi_U - \phi_U) \\
    6 \phi_U + 3 \phi_D - \frac{1}{8} \phi_{UU} - \frac{3 \Delta x^2 \Phi''}{48} + R_3 \end{cases}
    \]
  - Higher order (interpolation) schemes
TODAY (Lecture 20):
Time-Marching Methods and ODEs – Initial Value Problems

• Time-Marching Methods and Ordinary Differential Equations – Initial Value Problems

  – Euler’s method
  – Taylor Series Methods
    • Error analysis
  – Simple 2nd order methods
    • Heun’s Predictor-Corrector and Midpoint Method
  – Runge-Kutta Methods
  – Multistep/Multipoint Methods: Adams Methods
  – Practical CFD Methods
  – Stiff Differential Equations
  – Error Analysis and Error Modifiers
  – Systems of differential equations
References and Reading Assignments


Methods for Unsteady Problems – Time Marching Methods
ODEs – Initial Value Problems (IVPs)

• Major difference with spatial dimensions: Time advances in a single direction
  – FD schemes: discrete values evolved in time
  – FV schemes: discrete integrals evolved in time

• After discretizing the spatial derivatives (or the integrals for finite volumes), we obtained a (coupled) system of (nonlinear) ODEs, for example:

\[
\frac{d \Phi}{dt} = B \Phi + (bc) \quad \text{or} \quad \frac{d \Phi}{dt} = B(\Phi,t) ; \quad \text{with } \Phi(t_0) = \Phi_0
\]

• Hence, methods used to integrate ODEs can be directly used for the time integration of spatially discretized PDEs
  – We already utilized several time-integration schemes with FD schemes. Others are developed next.
  – For IVPs, methods can be developed with a single eqn.: \( \frac{d \phi}{dt} = f(\phi,t) ; \quad \text{with } \phi(t_0) = \phi_0 \)
  – Note: solving steady (elliptic) problems by iterations is similar to solving time-evolving problems. Both problems thus have analogous solution schemes.
Ordinary Differential Equations
Initial Value Problems

ODE: $x$ often plays the role of time (following Chapra & Canale’s and MATLAB’s notation)

$$\frac{d\phi}{dt} = f(t, \phi) ; \quad \text{with } \phi(t_0) = \phi_0 \quad \Leftrightarrow \quad \frac{dy}{dx} = f(x, y) ; \quad \text{with } y(x_0) = y_0$$

$$y'(x) = f(x, y) , \quad x \in [a, b]$$
$$y(x_0) = y_0$$

i) For Linear Differential Equation:

$$f(x, y) = -p(x)y + q(x)$$

ii) For Non-Linear Differential Equation:

$$f(x, y) \quad \text{non-linear in } y$$

Linear differential equations can often be solved analytically

Non-linear equations almost always require numerical solution
Ordinary Differential Equations
Initial Value Problems: Euler’s Method

Differential Equation
\[ \frac{dy}{dx} = f(x, y), \quad y_0 = p \]

Example
\[ f(x, y) = x \left( y = \frac{x^2}{2} + p \right) \]

Discretization
\[ x_n = nh \]

Finite Difference (forward)
\[ \frac{dy}{dx}\bigg|_{x=x_n} \approx \frac{y_{n+1} - y_n}{h} \]

Recurrence
\[ y_{n+1} = y_n + hf(nh, y_n) \]

Truncation error (in time): \[ O(h^2) \]
Sphere Motion in Fluid Flow

Equation of Motion – 2nd Order Differential Equation

\[ M \frac{d^2 x}{dt^2} = \frac{1}{2} \rho C_d \pi R^2 \left( V - \frac{dx}{dt} \right)^2 \]

Rewrite to 1st Order Differential Equations

\[ \frac{dx}{dt} = u \]
\[ \frac{du}{dt} = \frac{\rho C_d \pi R^2}{2M} (V^2 - 2uV + u^2) \]

Euler’ Method - Difference Equations – First Order scheme

\[ u_{i+1} = u_i + \left( \frac{du}{dt} \right)_i \Delta t, \quad u(0) = 0 \]
\[ x_{i+1} = x_i + u_i \Delta t, \quad x(0) = 0 \]
Sphere Motion in Fluid Flow

MATLAB Solutions

\[ u = \frac{dx}{dt} \]

\[
\begin{align*}
V &\quad x \\
\end{align*}
\]

\[
\begin{align*}
M &\quad R \\
\end{align*}
\]

**dudt.m**

```matlab
function [f] = dudt(t,u)
% u(1) = u
% u(2) = x
% f(2) = dx/dt = u
% f(1) = du/dt=rho*Cd*pi*r/(2m)*(v^2-2uv+u^2)
 rho=1000; Cd=1; m=5; r=0.05; fac=rho*Cd*pi*r^2/(2*m); v=1;
 f(1)=fac*(v^2-2*v*u(1)+u(1)^2);
 f(2)=u(1);
 f=f';
```

**dudt.m**

```matlab
x=[0:0.1:10];
% step size
h=1.0;
% Euler's method, forward finite difference
for n=2:N
  u_e(n)=u_e(n-1)+h*fac*(v^2-2*v*u_e(n-1)+u_e(n-1)^2);
  x_e(n)=x_e(n-1)+h*u_e(n-1);
end % Runge Kutta
% Plot
uu0=[0 0]';[tt,u]=ode45(@dudt,t,uu0);
figure(1)
hold off a=plot(t,u_e,'+b');
hold on a=plot(tt,u(:,1),'.g');
a=plot(tt,abs(u(:,1)-u_e),'+r');
figure(2)
hold off a=plot(t,x_e,'+b');
hold on a=plot(tt,u(:,2),'.g');
a=plot(tt,abs(u(:,2)-x_e),'xr');
```

**sph_drag_2.m**

```matlab
function [f] = dudt(t,u)
% u(1) = u
% u(2) = x
% f(2) = dx/dt = u
% f(1) = du/dt=rho*Cd*pi*r/(2m)*(v^2-2uv+u^2)
 rho=1000; Cd=1; m=5; r=0.05; fac=rho*Cd*pi*r^2/(2*m); v=1;
 f(1)=fac*(v^2-2*u(1)+u(1)^2);
 f(2)=u(1);
 f=f';
```

**sph_drag_2.m**

```matlab
u0=[0 0]';[tt,u]=ode45(@dudt,t,u0);
figure(1)
hold off a=plot(t,u_e,'+b');
hold on a=plot(tt,u(:,1),'.g');
a=plot(tt,abs(u(:,1)-u_e),'+r');
figure(2)
hold off a=plot(t,x_e,'+b');
hold on a=plot(tt,u(:,2),'.g');
a=plot(tt,abs(u(:,2)-x_e),'xr');
```
Sphere Motion in Fluid Flow
Error Propagation

\[ \frac{dv}{dt} = u \]

Velocity Position

Error decreasing with time

Error Increasing with time
Initial Value Problems: **Taylor Series Methods**

“Utilize the known value of the time-derivative (the RHS)”

**Initial Value Problem:**

\[
y' = f(x, y), \quad y(x_0) = y_0
\]

**Taylor Series**

\[
y(x) = y_0 + (x - x_0)y'(x_0) + \frac{(x - x_0)^2}{2} y'' + \ldots
\]

Derivatives can be evaluated using the ODE:

\[
y'(x) = f(x, y) \quad \Rightarrow \quad y'(x_0) = f(x_0, y_0)
\]

\[
y''(x) = \frac{df(x, y)}{dx} = f_x + f_y f' = f_x + f_y f
\]

\[
y'''(x) = \frac{d^2 f(x, y)}{dx^2} = f_{xx} + f_{xy} f + f_{yx} f + f_{yy} f^2 + f_y f_x + f_y f_x + f_y f
\]

\[
= f_{xx} + 2f_x f_y + f_{yy} f^2 + f_x f_y + f_y^2 f
\]

where partial derivatives are denoted by:

\[
\begin{align*}
f_x &= \frac{\partial}{\partial x} \\
 f_y &= \frac{\partial}{\partial y}
\end{align*}
\]

**Truncate series to** \(k\) terms, insert the known derivatives

\[
y_1 = y(x_1) = y_0 + h y'(x_0) + \frac{h^2}{2!} y''(x_0) + \cdots + \frac{h^k}{k!} y^{(k)}(x_0)
\]

\[
y_2 = y(x_2) = y_1 + h y'(x_1) + \frac{h^2}{2!} y''(x_1) + \cdots + \frac{h^k}{k!} y^{(k)}(x_1)
\]

\[
y_n = y(x_n) = y_{n-1} + h y'(x_{n-1}) + \frac{h^2}{2!} y''(x_{n-1}) + \cdots + \frac{h^k}{k!} y^{(k)}(x_{n-1})
\]

with a discretization and step size \(h\),

\[
h = \frac{b - a}{N}
\]

\[
x_n = a + n h, \quad n = 0, 1, \ldots N
\]

**Recursion Algorithm:**

\[
y(x_{n+1}) = y_{n+1} = y_n + hT_k(x_n, y_n) + \frac{h^{k+1}}{(k+1)!} y^{(k+1)}(\xi)
\]

where

\[
T_k(x_n, y_n) = f(x_n, y_n) + \frac{h}{2!} f'(x_n, y_n) + \frac{h^{k-1}}{k!} f^{(k-1)}(x_n, y_n)
\]

**Local Truncation Error:**

\[
E = \frac{h^{k+1} f^{(k)}(\xi, y(\xi))}{(k+1)!} = \frac{h^{k+1} y^{(k+1)}(\xi)}{(k+1)!}, \quad x_n < \xi < x_n + h
\]
Initial Value Problems: Taylor Series Methods

Summary of General Taylor Series Method

\[ x_n = a + nh, \quad n = 0, 1, \ldots N \]

\[ y(x_{n+1}) = y_{n+1} = y_n + hT_k(x_n, y_n) + \frac{h^{k+1}}{(k+1)!}y^{(k+1)}(\xi) \]

where:

\[ T_k(x_n, y_n) = f(x_n, y_n) + \frac{h}{2!}f'(x_n, y_n) + \cdots + \frac{h^{k-1}}{k!}f^{(k-1)}(x_n, y_n) \]

\[ E = \frac{h^{k+1}f^{(k)}(\xi, y(\xi))}{(k+1)!} = \frac{h^{k+1}y^{(k+1)}(\xi)}{(k+1)!}, \quad x_n < \xi < x_n + h \]

Example: \( k = 1 \)

Euler’s method

\[ y_{n+1} = y_n + hf(x_n, y_n) \]

\[ E = \frac{h^2}{2!}y''(\xi) \]

Note: expensive to compute higher-order derivatives of \( f(x,y) \), especially for spatially discretized PDEs => other schemes needed

Numerical Example – Euler’s Method

\( y' = y, \quad y(0) = 1, \quad y = e^x \)

\[ y(0.01) \approx y_1 = y_0 + hf(x_0, y_0) = 1 + 0.01 \cdot 1 = 1.01 \]

\[ y(0.02) \approx y_2 = y_1 + hf(x_1, y_1) = 1.01 + 0.01 \cdot 1.01 = 1.0201 \]

\[ y(0.03) \approx y_3 = y_2 + hf(x_2, y_2) = 1.0201 + 0.01 \cdot 1.021 = 1.03121 \]

\[ y(0.03) = 1.0305 \]

=> Global Error Analysis, i.e.:

As truncation errors are added at each time step and propagated in time, what is the final total/global truncation error obtained?
Initial Value Problems: Taylor Series Methods

Euler’s global/total truncation error bound, obtained recursively

\[ y' = f(x, y), \quad y(x_0) = y_0 \]

Estimate (Euler): \( y_{n+1} = y_n + hf(x_n, y_n), \quad n = 0, 1, \ldots \)

\[ x_n = x_0 + nh \]

Error at step \( n \): \( e_n = y(x_n) - y_n \)

Exact: \( y(x_{n+1}) = y(x_n) + hy'(x_n) + \frac{h^2}{2} y''(\xi_n), \quad x_n < \xi_n < x_{n+1} \)

\[ e_n = \frac{h^2}{2} y''(\xi_n) \]

\[ e_{n+1} = y(x_{n+1}) - y_{n+1} = y(x_n) + hy'(x_n) + \frac{h^2}{2} y''(\xi_n) - y_n - h f(x_n, y_n) \]

\[ e_{n+1} = (g(x_n) - y_n) + h [f(x_n, y(x_n)) - f(x_n, y_n)] + \frac{h^2}{2} y''(\xi_n) \]

Since up to \( O(e_n^2) \):

\[ f(x_n, y(x_n)) - f(x_n, y_n) = \frac{\partial f(x_n, y_n)}{\partial y} (y(x_n) - y_n) = f_y(x_n, y_n) e_n \]

\[ \Rightarrow e_{n+1} = e_n + h f_y(x_n, y_n) e_n + \frac{h^2}{2} y''(\xi_n) \]

\[ |e_{n+1}| \leq |e_n| + h |f_y(x_n, y_n)| e_n + \frac{h^2}{2} |y''(\xi_n)| \]

Assume derivatives are bounded: \( |f_y(x_n, y_n)| \leq L, \quad |y''(\xi_n)| \leq Y \)

\[ |e_{n+1}| \leq (1 + hL) |e_n| + \frac{h^2}{2} Y \]

\[ \eta_{n+1} = (1 + hL) \eta_n + \frac{h^2}{2} Y, \quad \eta_0 = 0 \]

\[ \eta_n = \frac{hY}{2L} [(1 + hL)^n - 1] \]

\[ \Rightarrow \text{Global Error Bound for Euler’s scheme:} \]

\[ |e_n| \leq \eta_n = \frac{hY}{2L} [(1 + hL)^n - 1] \]

\[ \leq \frac{hY}{2L} [(e^{hL})^n - 1] \]

\[ = \frac{hY}{2L} [e^{hLn} - 1] \]

\[ \Rightarrow |e_n| \leq \frac{hY}{2L} [(e^{h(x_n-x_0)L} - 1] \]

\[ = \text{O}(1) \text{ in } h! \]

= Euler’s global or total error bound
Example: $y' = y, \ y(0) = 1, \ x \in [0, 1]$

Exact solution: $y = e^x$

Derivative Bounds: $f_y = 1 \Rightarrow L = 1$

$y''(x) = e^x \Rightarrow Y = e$

$x - x_0 = n \ h = 1 \Rightarrow |e_n| \leq \frac{h e}{2} (e - 1)$

$h = 0.1 \Rightarrow |e_n| \leq 0.24$

Euler’s Method:

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

$y_{11} = 2.5937$

$y(x_{11}) = 2.71828$

$e_{11} = 0.1246 < 0.24$
Improving Euler’s Method

• For one-step (two-time levels) methods, the global error result for Euler can be generalized to any method of \( n^{th} \) order:
  – If the truncation error is of \( O(h^n) \), the global error is of \( O(h^{n-1}) \)

• Euler’s method assumes that the (initial) derivative applies to the whole time interval => 1\textsuperscript{st} order global error

• Two simple methods modify Euler’s method by estimating the derivatives within the time-interval
  – Heun’s method
  – Midpoint rule

• The intermediate estimates of the derivative lead to 2\textsuperscript{nd} order global errors

• Heun’s and Midpoint methods belong to the general class of Runge-Kutta methods
  – introduced now since they are also linked to classic PDE integration schemes
Initial Value Problems: **Heun’s method**
(which is also a “one-step” Predictor-Corrector scheme)

Initial Slope Estimate (Euler)

\[ y'_i = f(x_i, y_i) \]

**Predictor: Euler**

\[ y^0_{i+1} = y_i + f(x_i, y_i) h \]

which allows to estimate the Endpoint Derivative/slope:

\[ y'_{i+1} = f(x_{i+1}, y^0_{i+1}) \]

and so an Average Derivative Estimate:

\[ \bar{y}' = \frac{y'_i + y'_{i+1}}{2} = \frac{f(x_i, y_i) + f(x_{i+1}, y^0_{i+1})}{2} \]

**Corrector**

\[ y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y^0_{i+1})}{2} h \]

Heun can be set implicit, one can iterate => **Iterative Heun:**

\[ y^{k+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y^k_{i+1})}{2} h \]

Notes:

- Heun becomes Trapezoid rule if fully implicit scheme is used
- Heun’s global error is of 2nd order: \( O(h^2) \)
- Convergence of iterative Heun not guaranteed + can be expensive with PDEs

Image by MIT OpenCourseWare.
Initial Value Problems: **Midpoint method**

First: uses Euler to obtain a Midpoint Estimate:

\[ y_{i+1/2} = y_i + f(x_i, y_i) \frac{h}{2} \]

Then: uses this value to obtain a Midpoint Derivative Estimate:

\[ y_{i+1/2}' = f(x_{i+1/2}, y_{i+1/2}) \]

Assuming that this slope is representative of the whole interval => Midpoint Method recurrence:

\[ y_{i+1} = y_i + f(x_{i+1/2}, y_{i+1/2})h \]

Comments:

- Midpoint superior to Euler since it uses a centered FD for the first derivative
- Midpoint’s global error is of 2\textsuperscript{nd} order: \(O(h^2)\)
Initial Value Problems: Heun’s method examples

```matlab
func='4*exp(-0.8*x)-0.5*y';
f=inline(func,'x','y'); y0=2; % step size h=0.5;
% Euler's method, forward finite difference
xt=[0:h:10]; N=length(xt); yt=zeros(N,1); yt(1)=y0; for n=2:N
    yt(n)=yt(n-1)+h*f(xt(n-1),yt(n-1));
end
hold off
a=plot(xt,yt,'r'); set(a,'Linewidth',2)

% Heun's method
xt=[0:h:10]; N=length(xt); yt=zeros(N,1); yt(1)=y0; for n=2:N
    yt_0=yt(n-1)+h*f(xt(n-1),yt(n-1));
    yt(n)=yt(n-1)+h*(f(xt(n-1),yt(n-1))+f(xt(n),yt_0))/2;
end
hold on
a=plot(xt,yt,'g'); set(a,'Linewidth',2)

% Exact (ode45 Runge Kutta)
x=[0:0.1:10]; hold on
[xrk,yrk]=ode45(f,x,y0);
a=plot(xrk,yrk,'b'); set(a,'Linewidth',2)
```

Another example:
\[ y' = -2x^3 + 12x^2 - 20x + 8.5 \]

Image by MIT OpenCourseWare.
Two-level methods for time-integration of (spatially discretized) PDEs

• Four simple schemes to estimate the time integral by approximate quadrature

\[ \frac{d\phi}{dt} = f(t, \phi); \quad \text{with} \quad \phi(t_0) = \phi_0 \quad \Leftrightarrow \quad \int_{t_n}^{t_{n+1}} \frac{d\phi}{dt} \, dt = \phi_{n+1} - \phi_n = \int_{t_n}^{t_{n+1}} f(t, \phi) \, dt \]

- Explicit or Forward Euler:

\[ \phi_{n+1} - \phi_n = f(t_n, \phi^n) \Delta t \]

- Implicit or backward Euler:

\[ \phi_{n+1} - \phi_n = f(t_{n+1}, \phi^n + \Delta t) \Delta t \]

- Midpoint rule (basis for the leapfrog method):

\[ \phi_{n+1} - \phi_n = f(t_{n+1/2}, \phi^{n+1/2}) \Delta t \]

- Trapezoid rule (basis for Crank-Nicholson method):

\[ \phi_{n+1} - \phi_n = \frac{1}{2} [ f(t_n, \phi^n) + f(t_{n+1}, \phi^{n+1}) ] \Delta t \]

Reminder on global error order:

• Euler methods are of order 1
• Midpoint rule and Trapezoid rule are of order 2
• Order n = truncation error cancels if true solution is polynomial of order n

Some comments

- All of these methods are two-level methods (involve two times and are at best 2\textsuperscript{nd} order)
- All excepted forward Euler are implicit methods
- Trapezoid rule often yields solutions that oscillates, but implicit Euler tends to behave well
Runge-Kutta Methods and Multistep/Multipoint Methods

\[ \phi^{n+1} - \phi^n = \int_{t_n}^{t_{n+1}} f(t, \phi) \, dt \]

- To achieve higher accuracy in time, utilize information (known values of the derivative in time, i.e. the RHS) at more points in time. **Two approaches:**
  - **Runge-Kutta Methods:**
    - Additional points are between \( t_n \) and \( t_{n+1} \), and are used strictly for computational convenience
    - Difficulty: \( n^{th} \) order RK requires \( n \) evaluation of the first derivative (RHS of PDE)
      \( \Rightarrow \) more expansive as \( n \) increases
    - But, for a given order, RK methods are more accurate and more stable than multipoint methods of the same order.
  - **Multistep/Multipoint Methods:**
    - Additional points are at past time steps at which data has already been computed
    - Hence for comparable order, less expansive than RK methods
    - Difficulty to start these methods
    - Examples:
      - Adams Methods: fitting a polynomial to the derivatives at a number of past points in time
      - Lagrangian Polynomial, explicit in time (up to \( t_n \)): Adams-Bashforth methods
      - Lagrangian Polynomial, implicit in time (up to \( t_{n+1} \)): Adams-Moulton methods
Runge-Kutta Methods

Summary of General Taylor Series Method

\[ x_n = a + nh, \quad n = 0, 1, \ldots N \]

\[ y(x_{n+1}) = y_{n+1} = y_n + hT_k(x_n, y_n) + \frac{h^{k+1}}{(k+1)!}y^{(k+1)}(\xi) \]

where:

\[ T_k(x_n, y_n) = f(x_n, y_n) + \frac{h}{2!}f'(x_n, y_n) + \cdots + \frac{h^{k-1}}{k!}f^{(k-1)}(x_n, y_n) \]

\[ E = \frac{h^{k+1}f^{(k)}(\xi, y(\xi))}{(k+1)!} = \frac{h^{k+1}y^{(k+1)}(\xi)}{(k+1)!}, \quad x_n < \xi x_n + h \]

Example: \( k = 1 \)

Euler’s method

\[ y_{n+1} = y_n + hf(x_n, y_n) \]

\[ E = \frac{h^2}{2!}y''(\xi) \]

Aim of Runge-Kutta Methods:

- Achieve accuracy of Taylor Series method without requiring evaluation of higher derivatives of \( f(x, y) \)
- Obtain higher derivatives using only the values of the RHS (first time derivative)
- Utilize points between \( t_n \) and \( t_{n+1} \) only

Note: expensive to compute higher-order derivatives of \( f(x, y) \), especially for spatially discretized PDEs => other schemes needed
Initial Value Problems - Time Integrations

Derivation of 2nd order Runge-Kutta Methods

Taylor Series Recursion:

\[ y(x_{n+1}) = y(x_n) + hf(x_n, y_n) + \frac{h^2}{2}(f_x + f_y)n + \frac{h^3}{6}(f_{xx} + 2f_{xy} + f_{yy}f^2 + f_xf_y + f_y^2f)n + O(h^4) \]

Runge-Kutta Recursion:

\[
\begin{align*}
y_{n+1} &= y_n + ak_1 + bk_2 \\
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \alpha h, y_n + \beta k_1)
\end{align*}
\]

Expand \( k_2 \) in a Taylor series:

\[
\frac{k_2}{h} = f(x_n + \alpha h, y_n + \beta k_1)
\]
\[
= f(x_n, y_n) + \alpha hf_x + \beta k_1 f_y
\]
\[
+ \frac{\alpha^2 h^2}{2} f_{xx} + \alpha h \beta k_1 f_{xy} + \beta^2 \frac{k_1^2}{2} f_{yy} + O(h^4)
\]

Set \( a, b, \alpha, \beta \) to match Taylor series as much as possible.

Substitute \( k_1 \) and \( k_2 \) in Runge Kutta

\[
y_{n+1} = y_n + (a + b)hf + bh^2(\alpha f_x + \beta f_y)
\]
\[
+ bh^3(\frac{\alpha^2}{2} f_{xx} + \alpha \beta f_{xy} + \frac{\beta^2}{2} f_{yy}) + O(h^4)
\]

Match 2nd order Taylor series

\[
\begin{align*}
a + b &= 1 \\
b\alpha &= 1/2 \\
b\beta &= 1/2
\end{align*}
\]

We have three equations and 4 unknowns =>

- There is an infinite number of Runge-Kutta methods of 2nd order
- These different 2nd order RK methods give different results if solution is not quadratic
- Usually, number of \( k \)'s (recursion size) gives the order of the RK method.
4\textsuperscript{th} order Runge-Kutta Methods
(Most Popular, there is an \( \infty \) number of them, as for 2\textsuperscript{nd} order)

\[
\begin{align*}
y' &= f(x, y) \\
y(x_0) &= y_0 \\
x_n &= x_0 + nh
\end{align*}
\]

Initial Value Problem:

2\textsuperscript{nd} Order Runge-Kutta (Heun’s version)

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

4\textsuperscript{th} Order Runge-Kutta

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

Second-order RK methods

- \( b = \frac{1}{2}, a = \frac{1}{2} \) : Heun’s method
- \( b = 1, a = 0 \) : Midpoint method
- \( b = \frac{2}{3}, a = \frac{1}{3} \) : Ralston’s Method

The k’s are different estimates of the slope
**4th order Runge-Kutta Example:** \( \frac{dy}{dx} = x, \quad y(0) = 0 \)

Forward Euler’s Method

\[
x_n = nh
\]

Forward Euler’s Recurrence

\[
y_{n+1} = y_n + h f(nh, y_n)
\]

4th Order Runge-Kutta

\[
y_{n+1} = y_n + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)
\]

\[
k_1 = hf(x_n, y_n)
\]

\[
k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2})
\]

\[
k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2})
\]

\[
k_4 = hf(x_n + h, y_n + k_3)
\]

Matlab ode45 has its own convergence estimation

Note: Matlab inefficient for large problems, but can be used for incubation
Multistep/Multipoint Methods

• Additional points are at time steps at which data has already been computed

• Adams Methods: fitting a (Lagrange) polynomial to the derivatives at a number of points in time
  – Explicit in time (up to $t_n$): Adams-Bashforth methods
    \[ \phi^{n+1} - \phi^n = \sum_{k=n-K}^{n} \beta_k f(t_k, \phi^k) \Delta t \]
  – Implicit in time (up to $t_{n+1}$): Adams-Moulton methods
    \[ \phi^{n+1} - \phi^n = \sum_{k=n-K}^{n+1} \beta_k f(t_k, \phi^k) \Delta t \]
  – Coefficients $\beta_k$’s can be estimated by Taylor Tables:
    • Fit Taylor series so as to cancel higher-order terms
2.29 Numerical Fluid Mechanics
Spring 2015

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.