Solving Systems of Ordinary Differential Equations

Larry Caretto
Mechanical Engineering 309

Numerical Analysis of Engineering Systems

April 23, 2014

California State University Northridge

Outline

- Review basics of numerical solutions of ordinary differential equations
- Error control in simple methods discussed previously
- Systems of differential equations
- Reducing higher-order equations into a system of first-order equations
- Writing and using software for solving systems of ordinary differential equations

California State University Northridge 2

Schedule for April and May

Monday	Tuesday	Wednesday	Thursday	Friday
	1	**2**	3	4
Spring Break				
14	15	**16 Quiz 7 & PA5**	17	16
21	22	**23**	24	35
28 PA 6	29	**30 Quiz 8**	*May 1*	2
5 PA 7	6	**7 Prog Exam**	8	9
12 Final				

California State University Northridge 3

Review Numerical Approach

- Solve initial value problem, $dy/dx = f(x,y)$ (known equation) with $y(x_0) = y_0$
  - Use a finite difference grid: $x_{i+1} - x_i = h_i$
  - Replace derivative by finite-difference approximation: $dy/dx \approx (y_{i+1} - y_i) / (x_{i+1} - x_i) = (y_{i+1} - y_i) / h_i$
  - Derive a formula to compute f_{avg} the average value of $f(x,y)$ between x_i and x_{i+1}
  - Replace $dy/dx = f(x,y)$ by $(y_{i+1} - y_i) / h_i = f_{avg}$
  - Repeatedly compute $y_{i+1} = y_i + h_i f_{avg}$

California State University Northridge 4

Review Notation and Order

- x_i is independent variable
- y_i is numerical solution at $x = x_i$
- f_i is derivative found from x_i y_i: $f_i = f(x_i, y_i)$
- $y(x_i)$ is the exact value of y at $x = x_i$
- $f(x_i,y(x_i))$ is the exact derivative
- $e_1 = y(x_1) - y_1$ = local truncation error
- $E_j = y(x_j) - y_j$ is global truncation error
- If e is $O(h^n)$, then E is $O(h^{n-1})$

California State University Northridge 5

Review Simple Methods

- Euler: $y_{i+1} = y_i + h_i f_i = y_i + h_i f(x_i, y_i)$
- Huen's method

$$y_{i+1}^0 = y_i + h_{i+1} f(x_i, y_i) \qquad x_{i+1} = x_i + h_{i+1}$$

$$y_{i+1} = y_i + \frac{h_{i+1}}{2}\left[f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)\right] = \frac{y_i + y_{i+1}^0 + h_{i+1} f(x_{i+1}, y_{i+1}^0)}{2}$$

- Modified Euler method

$$y_{i+1/2} = y_i + \left[\frac{h_{i+1}}{2}\right] f(x_i, y_i) \qquad x_{i+1/2} = x_i + \frac{h_{i+1}}{2}$$

$$y_{i+1} = y_i + h_{i+1} f(x_{i+1/2}, y_{i+1/2})$$

California State University Northridge 6

Review 4th Order Runge-Kutta

- Uses four derivative evaluations per step

$$y_{i+1} = y_i + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \qquad x_{i+1} = x_i + h_{i+1}$$

$$k_1 = h_{i+1} f(x_i, y_i)$$

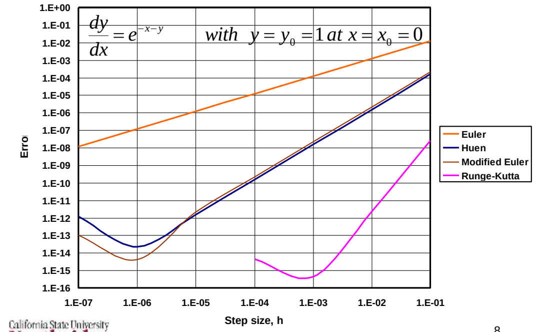
$$k_2 = h_{i+1} f\left(x_i + {h_{i+1}}/{2}, y_i + {k_1}/{2}\right)$$

$$k_3 = h_{i+1} f\left(x_i + {h_{i+1}}/{2}, y_i + {k_2}/{2}\right)$$

$$k_4 = h_{i+1} f(x_i + h_{i+1}, y_i + k_3)$$

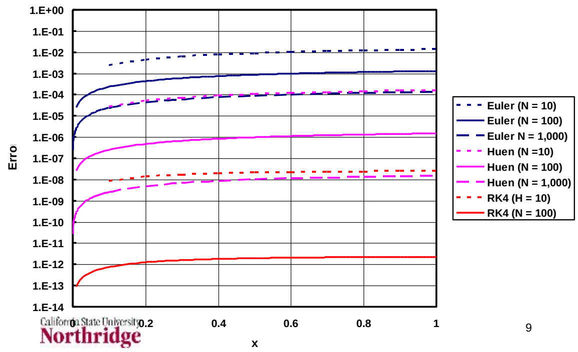
7

Error (at x = 1) versus step size for ODE solvers



8

Error Propagation in Solutions of ODEs



9

Error Control

- How do we choose h to maintain desired accuracy?
- Want to obtain a result with some desired small global error
- Can just repeat calculations with smaller h until two results are sufficiently close
- Can use algorithms that estimate error and adjust step size during the calculation based on the error

10

Runge-Kutta Error Control

- Take step with two formulas of different orders using the same function evaluations in each formula
- Use the difference between the two formulas as an estimate of the error
- Use the error estimate to control the step size based on a user-input desired error
- Some methods in final slides not shown

11

Dormand-Prince Method

- Uses fourth- and fifth-order expressions to find error estimate
- Modifies step size by comparing error estimate with desired error
- Used in MATLAB solver ode45 which is in next programming assignment
- Algorithm equations shown on next two slides

v1.11 of ode45 code at http://users.powernet.co.uk/kienzle/octave/matcompat/scripts/ode_v1.11/ode45.m

12

Dormand-Prince Equations

$$
\begin{aligned}
k_1 &= hf(t_k, y_k),\\
k_2 &= hf(t_k + \tfrac{1}{5}h, y_k + \tfrac{1}{5}k_1),\\
k_3 &= hf(t_k + \tfrac{3}{10}h, y_k + \tfrac{3}{40}k_1 + \tfrac{9}{40}k_2),\\
k_4 &= hf(t_k + \boxed{\tfrac{4}{5}}h, y_k + \boxed{\tfrac{44}{45}k_1 - \tfrac{56}{15}k_2 + \tfrac{32}{9}}k_3),\\
k_5 &= hf(t_k + \tfrac{8}{9}h, y_k + \tfrac{19372}{6561}k_1 - \tfrac{25360}{2187}k_2 + \tfrac{64448}{6561}k_3 - \tfrac{212}{729}k_4),\\
k_6 &= hf(t_k + h, y_k + \tfrac{9017}{3168}k_1 - \tfrac{355}{33}k_2 - \tfrac{46732}{5247}k_3 + \tfrac{49}{176}k_4 - \tfrac{5103}{18656}k_5),\\
k_7 &= hf(t_k + h, y_k + \tfrac{35}{384}k_1 + \tfrac{500}{1113}k_3 + \tfrac{125}{192}k_4 - \tfrac{2187}{6784}k_5 + \tfrac{11}{84}k_6).
\end{aligned}
$$

$$\frac{44}{45} - \frac{56}{15} + \frac{32}{9} = \frac{44 - 168 + 160}{45} = \frac{36}{45} = \frac{4}{5}$$

Toshinori Kimura, "On Dormand-Prince Method
http://depa.fquim.unam.mx/amyd/archivero/DormandPrince_19856.pdf

California State University Northridge

13

Dormand-Prince Method II

$$y_{k+1} = y_k + \frac{35}{384}k_1 + \frac{500}{1113}k_3 + \frac{125}{192}k_4 - \frac{2187}{6784}k_5 + \frac{11}{84}k_6.$$

$$z_{k+1} = y_k + \frac{5179}{57600}k_1 + \frac{7571}{16695}k_3 + \frac{393}{640}k_4 - \frac{92097}{339200}k_5 + \frac{187}{2100}k_6 + \frac{1}{40}k_7.$$

$$|z_{k+1} - y_{k+1}| = |\frac{71}{57600}k_1 - \frac{71}{16695}k_3 + \frac{71}{1920}k_4 - \frac{17253}{339200}k_5 + \frac{22}{525}k_6 - \frac{1}{40}k_7|.$$

$$
\begin{aligned}
s &= \left(\frac{\epsilon h}{2|z_{k+1} - y_{k+1}|}\right)^{\frac{1}{5}},\\
h_{opt} &= sh,
\end{aligned}
$$

ε is desired error per step
h_{opt} is optimum step size

Toshinori Kimura, "On Dormand-Prince Method
http://depa.fquim.unam.mx/amyd/archivero/DormandPrince_19856.pdf

California State University Northridge

14

Next Steps

- We have shown some algorithms for solving the initial value problem, dy/dx = f(x,y) with y = y_0 at x = x_0
- Need to solve higher order equations
- Can show than any n^{th}-order equation can be expressed as a system of n first-order equations
- Then have to consider solving systems of first order equations

California State University Northridge

15

Basic Idea

- Any n^{th} order ODE can be written as a system of n first-order ODEs
- For a system of two or more ODEs with varying orders:
  - Break the k^{th} ODE with order n_k into n_k first-order ODEs; repeat for all equations
  - The result will be a system of first order ODEs
    - The total number of ODEs will be $n_1 + n_2 + \ldots =$ the sum of the orders of each original ODE

California State University Northridge

16

Higher order equations

- Look at spring-mass-damper equation as example: $md^2y/dt^2 + Cdy/dt + ky = F$
  - F is applied force which may be a ***known*** function of t, y and dy/dt (may have F = 0)
  - Define a velocity, v = dy/dt
  - $dv/dt = d^2y/dt^2$
  - Our second order equation becomes mdv/dt + Cv + ky = F giving two ODEs

$$\frac{dv}{dt} = F - \frac{C}{m}v - ky = f_v(t, y, v)$$

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

California State University Northridge

17

Higher Order Equations II

$$\frac{dv}{dt} = f_v(t, y, v) \qquad \frac{dy}{dt} = f_y(t, y, v)$$

General Notation

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}) \qquad \frac{dy_k}{dt} = f_k(t, y_1 \cdots y_N)$$

- The two first order equations we found look like our initial value problem
  - This assumes that we have an initial condition for velocity, v
    - Will cover situations where this is not true (known as boundary value problems) later
  - We can apply all algorithms developed for single first-order equations to any system of simultaneous first-order equations

California State University Northridge

18

Solving Simultaneous ODEs

- Apply same algorithms used for single ODEs
  - Must apply each part of each algorithm step to all equations in system before going on to next step
  - Key is having consistent x and **y** values in determination of $f_k(x,\mathbf{y})$
  - All y_k values in **y** must be available at the same x point when computing the f_k
  - E.g., in Runge-Kutta we must evaluate k_1 for all equations before finding k_2

California State University Northridge 19

Runge-Kutta for ODE System

- $\mathbf{y}_{(n)}$ is vector of dependent variables at $x = x_n$
- $\mathbf{k}_{(1)}$, $\mathbf{k}_{(2)}$, $\mathbf{k}_{(3)}$, and $\mathbf{k}_{(4)}$, are vectors containing intermediate Runge-Kutta results
- **f** is a vector containing the derivatives
- $\mathbf{k}_{(1)} = h\mathbf{f} = h\mathbf{f}(x_n, \mathbf{y}_{(n)})$
- $\mathbf{k}_{(2)} = h\mathbf{f}(x_n + h/2, \mathbf{y}_{(n)} + \mathbf{k}_{(1)}/2)$
- $\mathbf{k}_{(3)} = h\mathbf{f}(x_n + h/2, \mathbf{y}_{(n)} + \mathbf{k}_{(2)}/2)$
- $\mathbf{k}_{(4)} = h\mathbf{f}(x_n + h, \mathbf{y}_{(n)} + \mathbf{k}_{(3)})$
- $\mathbf{y}_{(n+1)} = (\mathbf{k}_{(1)} + 2\mathbf{k}_{(2)} + 2\mathbf{k}_{(3)} + \mathbf{k}_{(4)})/6$

California State University Northridge 20

ODE System by RK4

- $dy/dx = –y + z$ and $dz/dx = y – z$ with $y(0) = 1$ and $z(0) = -1$ with $h = .1$
- Details of first step from y_0 to y_1
- $k_{(1)y} = h[-y + z] = 0.1[-1 + (-1)] = -.2$
- $k_{(1)z} = h[y - z] = 0.1[1 - (-1)] = .2$
- $k_{(2)y} = h[-(y+ k_{(1)y}/2) + z + k_{(1)z}/2] = 0.1[-(1 + -0.2/2) + (-1 + .2/2)] = -.18$
- $k_{(2)z} = h[(y+ k_{(1)y}/2) –(z + k_{(1)z}/2)] = 0.1[(1 + -0.2)/2 - (-1 + .2/2)] = .18$

California State University Northridge 21

ODE System by RK4 II

- $k_{(3)y} = h[-(y+ k_{(2)y}/2) + z + k_{(2)z}/2] = 0.1[-(1 + -0.18/2) + (-1 + .18/2)] = -.182$
- $k_{(3)z} = h[(y+ k_{(2)y}/2) –(z + k_{(2)z}/2)] = 0.1[(1 + -0.18)/2 - (-1 + .18/2)] = .182$
- $k_{(4)y} = h[-(y+ k_{(3)y}) + z + k_{(3)z}] = 0.1[-(1 + -0.182) + (-1 + .182)] = -.1636$
- $k_{(4)z} = h[(y+ k_{(3)y}) –(z + k_{(3)z})] = 0.1[(1 + -0.182) - (-1 + .182)] = .1636$

California State University Northridge 22

ODE System by RK4 III

- $y_{i+1} = y_i + (k_{(1)y} + 2k_{(2)y} + 2k_{(3)y} + k_{(4)y})/6 = 1 + [(–.2) + 2(–.18) + 2(–.182) + (–.1636)]/6 = .8187$ (here $i = 0$)
- $z_{i+1} = z_i + (k_{(1)z} + 2k_{(2)z} + 2k_{(3)z} + k_{(4)z})/6 = –1 + [(.2) + 2(.18) + 2(.182) + (.1636)]/6 = –.8187$
- Continue in this fashion until desired final x value is reached
  - Note all k_m computed before any k_{m+1}
- No x dependence for f in this example

California State University Northridge 23

Numerical Software for ODEs

- Usually written to solve a system of N equations, but will work for N = 1
- User has to code a subroutine or function to compute the **f** array
  - Input variables are x and **y**; **f** is output
  - Some codes have one dimensional parameter array to pass additional information from main program into the function that computes derivatives

California State University Northridge 24

Derivative Subroutine Example

- Visual Basic **sub code** for system of ODEs at right is shown below

$$\frac{dy_1}{dx} = -y_1 + \sqrt{y_2} - y_3 e^{2x} \qquad \frac{dy_2}{dx} = -2y_1^2 \qquad \frac{dy_3}{dx} = -3y_1 y_2$$

```
Sub fsub( x as Double, y() as Double, _
                        f() as Double )
  f(1) = -y(1) + Sqr(y(2)) - y(3)*Exp(2*x)
  f(2) = -2 * y(1)^2
  f(3) = -3 * y(1) * y(2)
End Sub
```

California State University Northridge

25

Derivative Function Example

- Visual Basic **function code** for system of ODEs

$$\frac{dy_1}{dx} = -y_1 + \sqrt{y_2} - y_3 e^{2x} \qquad \frac{dy_2}{dx} = -2y_1^2 \qquad \frac{dy_3}{dx} = -3y_1 y_2$$

```
Function ff( x As Double, y() As Double, _
                  N As Long) As Variant
  Dim fd() As Double; reDim fd(1 to N)
  fd(1) = -y(1) + Sqr(y(2)) - y(3)*Exp(2*x)
  fd(2) = -2 * y(1)^2
  fd(3) = -3 * y(1) * y(2)     : ff = fd
End Function
```

In main program use $f = ff(x,y,N)$ *where f is an array of same size*

California State University Northridge

26

Derivative Subroutine Example

- MATLAB **function** for system of ODEs at right is shown below

$$\frac{dy_1}{dx} = -y_1 + \sqrt{y_2} - y_3 e^{2x} \qquad \frac{dy_2}{dx} = -2y_1^2 \qquad \frac{dy_3}{dx} = -3y_1 y_2$$

```
function f = test( x, y)
  f = zeros(3,1);
  f(1) =-y(1) +Sqrt(y(2)) - y(3)*Exp(2*x);
  f(2) = -2 * y(1)^2;
  f(3) = -3 * y(1) * y(2);
end    %semicolons prevent output
```

California State University Northridge

27

Derivative Subroutine Example

- C++ code for system of ODEs at right is shown below

$$\frac{dy_1}{dx} = -y_1 + \sqrt{y_2} - y_3 e^{2x} \qquad \frac{dy_2}{dx} = -2y_1^2 \qquad \frac{dy_3}{dx} = -3y_1 y_2$$

```
void fsub(double x, double y[], double f[])
{
 f[1] = -y[1] + sqrt(y[2]) - y[3]*exp(2*x);
 f[2] = -2 * y[1] * y[1];
 f[3] = -3 * y[1] * y[2];
}
```

California State University Northridge

28

Derivative Subroutine Example

- Fortran code for system of ODEs at right is shown below

$$\frac{dy_1}{dx} = -y_1 + \sqrt{y_2} - y_3 e^{2x} \qquad \frac{dy_2}{dx} = -2y_1^2 \qquad \frac{dy_3}{dx} = -3y_1 y_2$$

```
subroutine fsub( x, y, f )
  real(KIND=8) x, y(:), f(:)
  f(1) = -y(1) + sqrt(y(2)) -y(3)*exp(2*x)
  f(2) = -2 * y(1)**2
  f(3) = -3 * y(1) * y(2)
end subroutine fsub
```

California State University Northridge

29

Assignment Seven and Last

- Write routine in MATLAB and in VBA to solve general system of N ODEs with following form: $dy_m/dt = f_m(t,\mathbf{y})$, m = 1, N
- Apply to N = 3 system with known exact solutions same as on previous slides

$$\frac{dy_1}{dx} = -y_1 + \sqrt{y_2} - y_3 e^{2x} \qquad y_1 = e^{-t}$$

$$\frac{dy_2}{dx} = -2y_1^2 \qquad \frac{dy_3}{dx} = -3y_1 y_2 \qquad y_2 = e^{-2t} \qquad y_3 = e^{-3t}$$

California State University Northridge

30

Assignment Seven Algorithms

- Code Huen algorithm in MATLAB

$$y^0_{m,i+1} = y_{m,i} + hf_m(t_i, \mathbf{y}_i)$$
$$t_{i+1} = t_i + h$$
$$y_{m,i+1} = \frac{y_{m,i} + y^0_{m,i+1} + hf_m(t_{i+1}, \mathbf{y}^0_{m,i+1})}{2}$$

- Modified Euler algorithm in VBA

$$y_{m,i+1/2} = y_{m,i} + \frac{h}{2} f_m(t_i, \mathbf{y}_i)$$
$$y_{m,i+1} = y_{m,i} + hf_m(t_i + h/2, \mathbf{y}_{i+1/2})$$
$$t_{i+1} = t_i + h$$

- Use MATLAB function ode45 for solving same set of equations and compare work for same accuracy

California State University Northridge 31

Runge-Kutta Error Control

- Following slides not shown in class
- Present other approach to error control in Runge-Kutta algorithms
  - Older method: Compare results for two steps at step size h with one step of 2h
    - Requires 12 derivative function evaluations over both steps
  - Runge-Kutta-Fehlberg: early method to use algorithms of different order with same function evaluations

California State University Northridge 32

Runge-Kutta Error Control

- Control error by doing integration with h and 2h all along integration
  - Integration with 2h step requires 3 additional function evaluations per 2 steps
  - Analyze local truncation error, which is O(h^5), for both steps

$$y(x+2h) = y_h + Ah^5 + Bh^6 + \cdots$$
$$y(x+2h) = y_{2h} + A(2h)^5 + B(2h)^6 + \cdots$$
$$0 = y_{2h} - y_h + (2^5 - 1)Ah^5 + O(h^6)$$

California State University Northridge 33

Runge-Kutta Error Control II

- $y_{2h} - y_h = \Delta$ is measure of truncation error
- User specifies Δ_D, the desired error
  - Many ways to specify this, single value, relative values, relative to increments for y in one step
- Since error scales as h^5, we can adjust step size such that $h_{new} = h_{old}|\Delta_D/\Delta|^{1/5}$
- Typically use safety factor to avoid making h_{new} too large

California State University Northridge 34

Runge-Kutta-Fehlberg

- Uses two equations to compute y_{n+1}, one has O(h^5), the other O(h^6) error
- Requires six derivative evaluations per step (same evaluations used for both equations)
- The error estimate can be used for step size control based on an overall 5th order error
- Cask-Karp version and Runge-Kutta-Verner use same idea

California State University Northridge 35

Runge-Kutta-Fehlberg II

- See page 663 in Rao text for algorithm
- Typical formula components below
- $y_{n+1} = y_n + (16k_1/135 + 6656k_2/12825 \ldots$
- $k_3 = hf(x_n + 3h/8, y_n + 3k_1/32 + 9k_2/32)$
- Error = $k_1/360 - 128k_3/4275 \ldots$
- $h_{new} = h_{old} \| h_{old}E_{Desired}/\text{Error} \|^{1/4}$
- $E_{Desired}$ is set by user
- RKF45 code by Watts and Shampine

California State University Northridge 36

Increments, k, for RKF

$$k_1 = hf(x_j, y_j) \qquad k_2 = hf\left(x_j + \frac{h}{4}, y_j + \frac{k_1}{4}\right)$$

$$k_3 = hf\left(x_j + \frac{3h}{8}, y_j + \frac{3k_1 + 9k_2}{32}\right)$$

$$k_4 = hf\left(x_j + \frac{12h}{13}, y_j + \frac{1932k_1 - 7200k_2 + 7296k_3}{2197}\right)$$

$$k_5 = hf\left(x_j + h, y_j + \frac{439k_1}{216} - 8k_2 + \frac{3680k_3}{513} - \frac{845k_4}{4104}\right)$$

$$k_6 = hf\left(x_j + \frac{h}{2}, y_j - \frac{8k_1}{27} + 2k_2 - \frac{3544k_3}{2565} + \frac{1859k_4}{4104} - \frac{11k_5}{40}\right)$$

California State University Northridge

37

Final RKF Step Results

- Proposed value of y_{i+1} is

$$y_{i+1} = y_i + \frac{16k_1}{135} + \frac{6656k_3}{12825} + \frac{28561k_4}{56430} - \frac{9k_5}{60} + \frac{2k_6}{55}$$

- Estimated error is

$$\varepsilon_{i+1} = \frac{16k_1}{135} + \frac{6656k_3}{12825} + \frac{28561k_4}{56430} - \frac{9k_5}{60} + \frac{2k_6}{55}$$

- If $\varepsilon_{min} < \varepsilon_{i+1} < \varepsilon_{max}$ accept y_{i+1}; keep h
- If $\varepsilon_{i+1} < \varepsilon_{min}$ accept y_{i+1} and get new h
- If $\varepsilon_{i+1} > \varepsilon_{max}$ get new h and redo step
- $h_{new} = h_{old}(\varepsilon^* h_{old}/|\varepsilon_{i+1}|)^{1/4}$ ε* = desired error

California State University Northridge

38