Stiff ODEs and Boundary Value Problems

Larry Caretto
Mechanical Engineering 309

Numerical Analysis of Engineering Systems

April 30, 2014

California State University Northridge

Outline

- Schedule
- Review last lecture
- Theoretical concepts in solving ODEs
- Stability – definition and examples
- Stiff systems of equations
- Boundary-value problems
  - Shoot and try
  - Finite differences

Remaining Course Schedule

- April 30 (today) – Last quiz (on numerical solutions of ODEs) and final lecture on numerical solutions of ODEs
- May 5 – Review for final and programming exams; programming assignment seven due
- May 7 – Programming exam
- May 12 – Final exam, 8 – 10 pm

Review Methods for N ODEs

- Showed coding example of 4th order Runge-Kutta for both single equation and multiple equations
- Have to analyze all ODEs at each step and sub-step of algorithm
- Also looked at MATLAB solvers
  - Use ode45 as first choice
  - All solvers have same format

Review Implicit Trapezoid Method

- Can be expressed in terms of f(x,y) and its partial derivatives

$$y_{n+1} = y_n + \frac{h}{2}(f_n + f_{n+1}) \qquad y_{n+1} = y_n \frac{hf_n + \left.\frac{\partial f}{\partial x}\right|_n \frac{h^2}{2}}{1 - \frac{h}{2}\left.\frac{\partial f}{\partial y}\right|_n}$$

- This was way to get information about point n+1 in solution for that point
- Will examine stability of this method in current lecture
  - Stability means that solution remains finite

Review Multistep Methods

- Oringinal methods (Euler, Runge-Kutta, *etc*.) methods use only information from most recent step (y_n and f_n)
- Multiple function evaluations between x_n and x_{n+1} used to improve accuracy
- Multistep methods use information from previous steps for improved accuracy with less work than single step methods
- Need starting procedure that is a single step method

Review Adams-Bashforth-Moulton

- Predictor-corrector method (y^P and y^C)

$$y^P_{n+1} = y_n + \frac{h}{24}\left(55 f_n - 59 f_{n-1} + 37 f_{n-2} - 9 f_{n-3}\right)$$

$$y^C_{n+1} = y_n + \frac{h}{24}\left[9 f\left(x_{n+1}, y^P_{n+1}\right) + 19 f_n - 5 f_{n-1} + f_{n-2}\right]$$

- Error estimate used for adjusting step size $E_C = \frac{19}{270}\left(y^P_{n+1} - y^C_{n+1}\right)$
  - User sets e_{min} and e_{max}
  - If $e_{min} \le E_C \le e_{max}$, do not change h
  - If $E_C < e_{min}$ double step size, h
  - If $E_C > e_{max}$ half step size, h, and redo step

California State University Northridge 7

Grid halving if error too large

- Normal operation, no step size change

i-3 i-2 i-1 i i+1 (old step)

●---------●---------●---------●---------●---------●

(new) i-3 i-2 i-1 i i+1

- Error too large: Half grid size and repeat step

i-3 i-2 i-1 i i+1 (old step)

●----o----●----o----●----o----●----o----●

(repeated) i-3 i-2 i-1 i i+1

(interpolated points)

California State University Northridge 8

Grid doubling for very small error

- Normal operation, no step size change

i-5 i-4 i-3 i-2 i-1 i i+1 (old step)

o-----o-----o-----o-----o-----o-----o-----o

(new step) i-3 i-2 i-1 i i+1

- Error very small: Double grid size

i-5 i-4 i-3 i-2 i-1 i i+1 (old step)

●-----o-----●-----o-----●-----o-----●----------●

i-3 i-2 i-1 i i+1

(Retained to use for doubling)

California State University Northridge 9

Some Basic Concepts

- A finite difference equation and the original differential equation are
  - ***Consistent*** if both equations give the same result as $h \to 0$
  - ***Convergent*** if the numerical solution approaches the actual solution as $h \to 0$
  - ***Accuracy*** controlled by step size with error estimates used to control step size
  - ***Order*** of the ***global*** truncation error = order of the ***local*** truncation error +1
  - ***Stable*** numerical solutions are bounded

California State University Northridge 10

Stability

- Finite difference equations in numerical algorithms, when iterated, may numerically increase without bound
- Stability usually is obtained by keeping step size h small, sometimes smaller than the h required for accuracy
- For most ODEs stability is not a problem, but it is for stiff systems of ODEs and for partial differential equations

California State University Northridge 11

Kinds of Stability

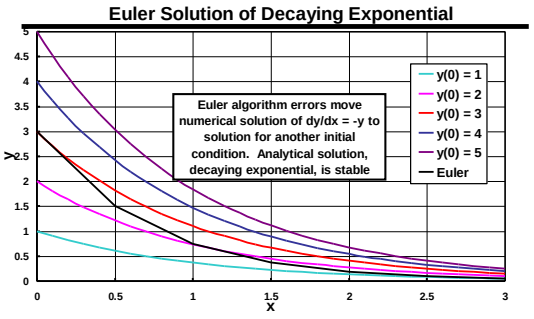
- If a numerical algorithm remains finite for any step size, h, it is said to be **absolutely stable**
- If a numerical algorithm remains finite only if the step size h is less than some value, it is said to be **conditionally stable**
- In some cases requirements for conditional stability may require a smaller h than accuracy requirements

California State University Northridge 12

Stability of Exact Solutions

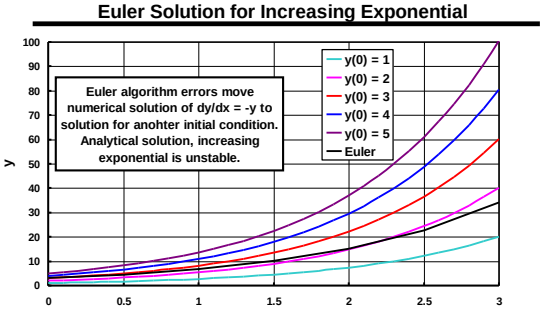
- Exact solutions to differential equations may be unstable
- Solutions of the form Ce^{at} with a > 0 are unstable because they grow without bound as $t \rightarrow \infty$
- Judge stability of a numerical method by test on an exact solution that is stable
  - Test ODE is dy/dt = –ay whose solution is $y = e^{-at}$, where a is a positive constant

California State University Northridge 13



Legend shows initial values for exact solutions; initial value for Euler is 3

California State University Northridge 14



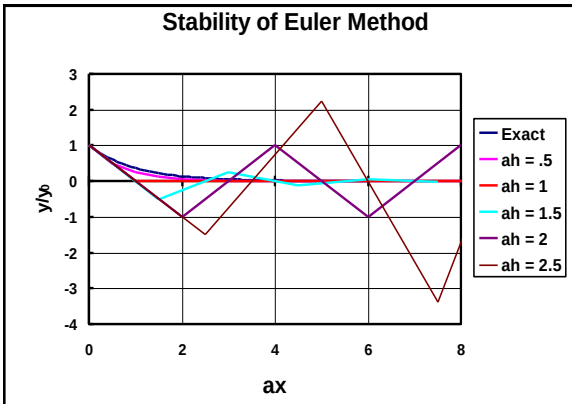
Legend shows initial values for exact solutions; initial value for Euler is 3

California State University Northridge 15

Examine Euler Stability

- Look at test equation with dy/dt = f = -ay
- Exact solution is $y = y_0 e^{-ax}$ so that $y/y_0 = e^{-ax}$ is a function of ax for any a and y_0
- Euler method: $y_{n+1} = y_n + hf_n$
  - For $f_n = -ay_n$ the Euler method equation becomes $y_{n+1} = y_n + h(-ay_n) = y_n(1 - ah)$
  - Write this as $y_{n+1}/y_0 = y_n/y_0(1 - ah)$
- Compare various numerical solutions to exact solution for different values of ah in following plot of y/y_0 *versus* ax

California State University Northridge 16



California State University Northridge 17

Chart Observations

- Used Euler method: $y_{n+1} = y_n + hf_n$ to solve y' = -ay
- For $ah \leq 1$, method looks physically realistic if not accurate
- For $1 < ah \leq 2$, method is not physically realistic but is bounded (stable)
- Method is unstable for ah > 2
- Not shown on chart is that we usually need ah << 2 for accuracy

California State University Northridge 18

Trapezoid Method Stability

- Apply trapezoid method to stability test equation: y' = f(x,y) = –ay so that f_n = –ay_n, $\partial f/\partial x = 0$ and $\partial f/\partial y = -a$

$$y_{n+1} = y_n + \frac{hf_n + \left.\frac{\partial f}{\partial x}\right|_n \frac{h^2}{2}}{1 - \frac{h}{2}\left.\frac{\partial f}{\partial y}\right|_n} = y_n + \frac{h(-ay_n) + (0)\frac{h^2}{2}}{1 - \frac{h}{2}(-a)}$$

$$y_{n+1} = \frac{y_n\left(1 + \frac{ah}{2}\right) + h(-ay_n)}{1 + \frac{ah}{2}} = \frac{y_n + \frac{y_n ah}{2} - hay_n}{1 + \frac{ah}{2}} = \frac{y_n(2 - ah)}{2 + ah}$$

Trapezoid Method Stability

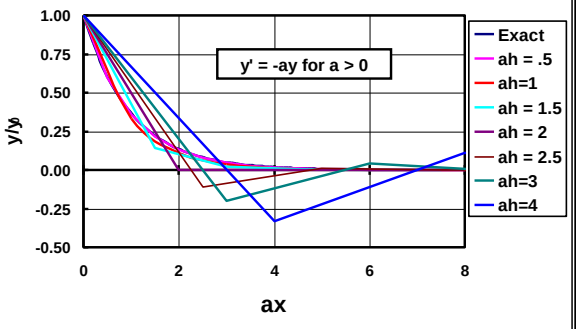


Chart Observations

- Trapezoid method results much closer to exact solution than Euler results
  - Expected because of $O(h^3)$ local error
- For values of ah > 2, solutions undershoot the final value of y = 0
  - Solutions remain stable for any h, but unrealistic, giving oscillations around y = 0
- Stability is not the same as accuracy
- Must have both

Trapezoid method is absolutely stable

Stiff Systems of Equations

- Several definitions
- Basic problem is that there are several length or time constants (eigenvalues) in the system
  - If one is negative and large in magnitude compared to others, this will set stability
  - Such terms quickly drop to zero and do not affect physical solution
  - However, they force small h for stability
  - Gear's Method in lecture appendix

Stiff Systems of Equations II

- Kinds of stiff problems
  - Vibrating systems with different fundamental frequencies
  - Combustion systems with chemical vast differences in chemical reaction rates
  - Closely coupled servomechanisms
- Solution like $Ae^{-t} + Be^{-100t}$ will have Euler stability limits that ah = 100h < 2 even though e^{-100t} is quickly zero

Still ODE MATLAB Solvers

Solver	Problem Type	Order of Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. This should be the first solver you try
ode23	Nonstiff	Low	Problems with crude error tolerances or for solving **mildly stiff problems**
ode113	Nonstiff	Low to high	Problems with stringent error tolerances or computationally intensive problems
ode15s	Stiff	Low to medium	If ode45 is slow because the problem is stiff
ode23s	Stiff	Low	With crude error tolerances to solve stiff systems (mass matrix is constant)
ode23t	Modera-tely Stiff	Low	Moderately stiff problems if you need a solution without numerical damping.
ode23tb	Stiff	Low	If using crude error tolerances to solve stiff systems.

Boundary-Value Problems

- All ODEs solved so far have initial conditions only
  - Conditions for all variables and derivatives set at t = 0 only
- In a boundary-value problem, we have conditions set at two different locations
- A second-order ODE $d^2y/dx^2 = g(x, y, y')$, needs two boundary conditions
  - Simplest are y(0) = a and y(L) = b
  - Can also have ady/dx+by = c at x = 0, L

California State University Northridge 25

MATLAB Boundary-value ODEs

- MATLAB has two solvers bvp4c and bvp5c for solving boundary-value ODEs
  - Bvp5c: finite difference code implements four-stage Lobatto IIIa formula, a collocation formula that provides a C^1-continuous solution that is fifth-order accurate uniformly in [a,b]
  - bvp5c solves algebraic equations directly; bvp4c uses analytical condensation
  - bvp4c handles unknown parameters directly

California State University Northridge 26

Boundary-value Problems II

- Solving boundary-value problems
  - Finite differences (considered later)
  - Shoot-and-try
    - Take an initial guess of derivative boundary conditions at x = 0 and use an initial-value routine to get $y_{(comp)}(L)$ at the other boundary
    - Compare the value of $y_{(comp)}(L)$ found from the previous step to the boundary condition on y(L)
    - Use the difference between $y_{(comp)}(L)$ and y(L) to iterate the initial value of $z = dy/dx|_{x=0}$ and continue until $y_{(comp)}(L) \approx y(L)$

California State University Northridge 27

Shoot and Try Example

- Look at single, second order equation: $y'' = g(x, y, y')$, y(0) = a and y(L) = b
- Define $z = y'$ ($y'' = z'$) to get two first order equations: $z' = g(x, y, z)$ and $y' = z$
- Steps in the shoot and try method
  - Guess initial condition for $z(0) = y'(0)$; typically guess $z^{(0)}(0) = [y(L) - y(0)]/L$
  - Solve equations for $y_{computed}(L)$ and compare to desired result y(L)

California State University Northridge 28

Shoot and Try Notation

- Notation for shoot and try
  - ODE: $d^2y/dx^2 = g(x, y, dy/dx)$
  - System of two first order ODES: $z = dy/dx$ and $dz/dx = g(x, y, z)$
  - Variables for iteration m
    - $z^{(m)}(0)$ guess for initial condition of dy/dx
    - $y^{(m)}(L)$ result at x = L from solving system of two ODEs using $z^{(m)}(0)$
    - y(L) required boundary condition at x = L
    - Error: $E^{(m)} = y^{(m)}(L) - y(L)$

California State University Northridge 29

Shoot and Try Iteration

- Based on value of $E^{(m)} = y^{(m)}(L) - y(L)$ adjust $z^{(m)}(0)$ until |E| < desired error
- After first try with $z^{(0)}(0) = [y(L) - y(0)]/L$ try $z^{(1)}(0) = [y(L) - y^{(0)}(L) - y(0)]/L$
- For subsequent tries use linear interpolation to give zero error

Set this to zero

$$z^{(m+1)}(0) = z^{(m-1)}(0) + \frac{z^{(m)}(0) - z^{(m-1)}(0)}{E^{(m)} - E^{(m-1)}}\left(E^{(m+1)} - E^{(m-1)}\right)$$

$$z^{(m+1)}(0) = z^{(m-1)}(0) - E^{(m-1)}\frac{z^{(m)}(0) - z^{(m-1)}(0)}{E^{(m)} - E^{(m-1)}}$$

California State University Northridge 30

Shoot-and-Try Example

- Solve $d^2y/dx^2 + 16\sin(y) = 0$ with y = 1 at x = 0 and x = L = 1
- Must find pair of first order equations
  - Set dy/dx = z as one ODE
  - Original ODE becomes dz/dx = –16sin(y)
  - We know y(0) = 1, but we need z(0) guess $z^{(0)}(0) = [y(L) – y(0)]/L = (0 – 1)/1 = –1$
    - This $z^{(0)}(0)$ gives y(1) = –0.6824 (RK4, h = .01)
    - Try $z^{(1)}(0) = [y(L) – y^{(0)}(L) – y(0)]/L = [1 – (–0.6824) – 0] = –0.3176$

 $E^{(0)} = y^{(0)}(L) – y(L) = –0.6824 – 0 = -0.6824$ 31

Shoot-and-Try Example II

- RK4 with $z^{(1)}(0) = –0.31756$ gives $y^{(1)}(L) = –0.7773$ so $E^{(1)} = y^{(1)}(L) – y(L) = –0.7773 – 0 = –0.7773$
- Apply general error formula to get $z^{(2)}(0)$

$$z^{(m+1)}(0) = z^{(m-1)}(0) - E^{(m-1)}\frac{z^{(m)}(0) - z^{(m-1)}(0)}{E^{(m)} - E^{(m-1)}}$$

$$z^{(2)}(0) = -1 - -0.6824\frac{-0.31756 - -1}{-0.7774 - -0.6824} = -5.911$$

- Continue to apply Runge-Kutta to get $y^{(m)}(L=1)$, $E^{(m)} = y^{(m)}(L) – y(L)$, and $z^{(m+1)}(0)$ for $E^{(m+1)} \approx 0$
- Repeat calculations with new value of $z^{(m+1)}(0)$ from general error formula until $y^{(m)}(L) \approx y(L)$

Boundary-value Problems III

- Finite-difference approach is alternative to shoot-and-try
  - Construct grid of step size h (variable h possible) between boundaries
    - Similar to grid used for numerical integration
    - $x_0 = x(0)$, $x_N = x(L)$, $h = L / N$, $x_k = x_0 + kh$
  - Replace differential equation at each interior node by finite difference equation
  - Solve resulting set of algebraic equations for interior points using Thomas algorithm

Finite Difference Grid

- Grid may be uniform or non-uniform, but uniform is easier and has higher order truncation error; note: $h = (x_N – x_0)/N$

●-----●--------●----------●~ ~●-------●---●

x_0 x_1 x_2 x_3 x_{N-2} x_{N-1} x_N

- At each node write finite-difference equivalent to differential equation
- Handle boundary conditions at x_0 and x_N (simplest if $y_0 = y(0)$ and $y_N = y(L)$ given, but can have gradient boundaries)

Example Problem

- Solve $d^2T/dx^2 + a^2T = 0$
- Finite difference equation at node i
- $[d^2T/dx^2 + a^2T]_i = (T_{i+1} + T_{i-1} – 2T_i)/h^2 + a^2T_i + $ [ ] $= 0$ *Ignore truncation error*
- Ignore truncation error and get finite-difference equation system
- $T_{i+1} + T_{i-1} – 2T_i + h^2a^2T_i = 0$
- Have N+1 nodes numbered from 0 to N with boundary conditions at 0 and N

General Boundary Conditions

- Must be able to handle three kinds
  - Dirichlet – specify variables at boundary
  - Neumann – specify of boundary gradients
  - Mixed or third kind – specify relationship between value and gradient at boundary
- General format a dT/dx + bT = c (mixed)
  - Fixed T: a = 0, b = 1, boundary T = c
  - Gradient: b = 0, a = 1, boundary dT/dx = c
- Get finite difference equation for boundary gradients if specified

California State University Northridge 37

Example Continued

- Equation is $T_{i-1} + (-2 + h^2a^2)T_i + T_{i+1} = 0$
- Specify boundary values T_A and T_B
  - $T_0 = T(x=0) = T_A$ and $T_N = T(x=L) = T_B$
- With specified boundary values equations at i = 1 and i = N-1 become
  - $(-2 + h^2a^2)T_1 + T_2 = -T_A$
  - $T_{N-2} + (-2 + h^2a^2)T_{N-1} = -T_B$
- Resulting system of equations forms tridiagonal matrix

California State University Northridge 38

Example Matrix Equations

- Finite-difference equations in matrix form with $\alpha = a^2h^2$ have tridiagonal form solved by Thomas Algorithm used with cubic spline (see end slides)

$$\begin{bmatrix} -2+\alpha & 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & -2+\alpha & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -2+\alpha & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \ddots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & 0 & \cdots & -2+\alpha & 1 \\ 0 & 0 & 0 & 0 & \cdots & 1 & -2+\alpha \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ \vdots \\ T_{N-2} \\ T_{N-1} \end{bmatrix} = \begin{bmatrix} -T_A \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ -T_B \end{bmatrix}$$

California State University Northridge 39

Numerical Results

- Look at results for h = 0.1 (N = 10) with $T_A = 0$, $T_B = 1$, a = 2 and L = 1
- Compare to exact solution below
  - Exact gradients also used in comparison

$$T = \frac{T_B - T_A\cos(aL)}{\sin(aL)}\sin(ax) + T_A\cos(ax)$$

$$q_{x=0} = -k\left.\frac{dT}{dx}\right|_{x=0} = -ka\frac{T_B - T_A\cos(aL)}{\sin(aL)}$$

$$q_{x=L} = -k\left.\frac{dT}{dx}\right|_{x=L} = \frac{ka\left[T_A - T_B\cos(aL)\right]}{\sin(aL)}$$

California State University Northridge 40

Results of Finite-Difference Calculations				
i	x_i	T_i	Exact T_i	Error
0	0.0	0	0	0
1	0.1	0.21918	0.21849	0.00070
2	0.2	0.42960	0.42826	0.00134
3	0.3	0.62284	0.62097	0.00187
4	0.4	0.79115	0.78891	0.00224
5	0.5	0.92783	0.92541	0.00242
6	0.6	1.02739	1.02501	0.00238
7	0.7	1.08585	1.08375	0.00211
8	0.8	1.10088	1.09928	0.00160
9	0.9	1.07188	1.07099	0.00089
10	1.0	1	1	0

Error and Error Order

- Get overall measure of error (like norm of a vector)
- Typically use maximum error (in absolute value) or root-mean-squared (RMS) error
- N = 10 has $\varepsilon_{max} = 2.42\text{x}10^{-3}$ and $\varepsilon_{RMS} = 1.83\text{x}10^{-3}$. For N = 100, $\varepsilon_{max} = 2.41\text{x}10^{-5}$ and $\varepsilon_{RMS} = 1.73\text{x}10^{-5}$.
- Second-order error in solution

$$\left|\varepsilon\right|_{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(T_{exact} - T_{numerical})_i^2}$$

California State University Northridge 42

Boundary Gradients

- Use second-order derivative expressions

$$q_0 = -k\left.\frac{dT}{dx}\right|_{x=x_0} = -k\frac{-3T_0 + 4T_1 - T_2}{2h}$$

$$q_N = -k\left.\frac{dT}{dx}\right|_{x=x_N} = -k\frac{3T_N - 4T_{N-1} + T_{N-2}}{2h}$$

x	$-q_{exact}/k$	h	-q/k	Error
0	2.1995	.1	2.2357	.03618
0	2.1995	.01	2.1999	.00036
1	-.9153	.1	-.9332	.01786
1	-.9153	.01	-.9155	.00021

MATLAB Boundary-value ODEs

- MATLAB has two solvers bvp4c and bvp5c for solving boundary-value ODEs
  - bvp5c: finite difference code implements four-stage Lobatto IIIa formula, a collocation formula that provides a C^1-continuous solution that is fifth-order accurate uniformly in [a,b]
  - bvp5c solves algebraic equations directly; bvp4c uses analytical condensation
  - bvp4c handles unknown parameters directly

Reference Slides

- The following slides will not be covered during the lecture unless there are questions
- The first set of slides describes the basics of Gear's method for solving stiff systems of equations
- The second set of slides is a review of the Thomas Algorithm for solving a tridiagonal set of algebraic equations

Gear's Method

- Use implicit algorithms with solution by Newton's method
- General algorithm shown below
  - k is global order of the method
  - Coefficients: See table next slide

$$y_{n+1} = \gamma\beta h f_{n+1} + \sum_{j=0}^{k} \alpha_{-j} y_{n-j}$$

- Solving g(y) = 0 by Newton's method $y^{(m+1)} = y^{(m)} - \frac{g\left(y^{(m)}\right)}{g'\left(y^{(m)}\right)}$

See details on end slides

Gear's Method Coefficients

k	γ	β	α_0	α_{-1}	α_{-2}	α_{-3}	α_{-4}	α_{-5}
1	1	1	1					
2	1/3	2	4	-1				
3	1/11	6	18	-9	2			
4	1/25	12	48	-16	16	-3		
5	1/137	60	300	-300	200	-75	12	
6	1/147	60	360	-450	400	-225	72	-10

Partial Derivative Function/Sub

- Implicit solution procedures for stiff equations sometimes require the calculation of partial derivatives
- This is done in a function or sub that is similar to that used to compute $f_k(t,\mathbf{y})$
  - Users have to write this routine to compute $\partial f_k/\partial t$ for all k and $\partial f_k/\partial y_m$ for all combinations of k and m
  - Sometimes done numerically as $\Delta f_k/\Delta t$ and $\Delta f_k/\Delta y_m$

Gear's Method Newton Iteration

- Iterate to find y such that g(y) = 0, write Taylor series for step from old y value, $y^{(m)}$ to new y value, $y^{(m+1)}$

$$g(y^{(m+1)}) = g(y^{(m)}) + \left.\frac{dg}{dy}\right|^{(m)} \left(y^{(m+1)} - y^{(m)}\right) + O(h^2)$$

- Set $g(y^{(m+1)}) = 0$ and solve for $y^{(m+1)} - y^{(m)}$

$$y^{(m+1)} - y^{(m)} = \frac{-g(y^{(m)})}{\left.\frac{dg}{dy}\right|^{(m)}}$$

  - Use this to iterate since solution is not explicit

Solving $y_{n+1} = \gamma\beta h f_{n+1} + \sum_{j=0}^{k} \alpha_{-j} y_{n-j}$

$$g(y_{n+1}) = y_{n+1} - \gamma\beta h f(x_{n+1}, y_{n+1}) - \sum_{j=0}^{k} \alpha_{-j} y_{n-j} = 0$$

$$y_{n+1}^{(m+1)} - y_{n+1}^{(m)} = \frac{-g(y_{n+1}^{(m)})}{\left.\frac{dg}{dy}\right|_{n+1}^{(m)}}$$

- Iterate to get y_{n+1} (m is iteration index)

$$\left.\frac{dg}{dy}\right|_{n+1}^{(m)} = 1 - \gamma\beta h \left.\frac{\partial f}{\partial y}\right|_{n+1}^{(m)}$$

Multivariable Gear Method

- Implicit equation in $\mathbf{y}_{n+1}$

$$y_{i,n+1} = \gamma\beta h f_i(x_{n+1}, \mathbf{y}_{n+1}) + \sum_{j=0}^{k} \alpha_{-j} y_{i,n-j}$$

- Expand f_{n+1} in Taylor series between iteration m and iteration m+1 for y_{n+1}

$$y_{i,n+1}^{(m+1)} = \gamma\beta h \left[f_i(x_{n+1}, \mathbf{y}_{n+1}^{(m)}) + \sum_{j=1}^{N_{ODE}} \left.\frac{\partial f_i}{\partial y_j}\right|_{n+1}^{(m)} \cdot \left(y_{j,n+1}^{(m+1)} - y_{j,n+1}^{(m)}\right) + O(h^2) \right] + \sum_{j=0}^{k} \alpha_{-j} y_{i,n-j}$$

Multivariable Gear Method II

- We have a set of simultaneous linear equations to solve for $\mathbf{y}_{n+1}$ components at each iteration
  - Partial derivative example on next slide

$$\sum_{j=1}^{N_{ODE}} \left[\delta_{ij} - \gamma\beta h \left.\frac{\partial f_i}{\partial y_j}\right|_{n+1}^{(m)} \right] \left(y_{j,n+1}^{(m+1)} - y_{j,n+1}^{(m)}\right) = -y_{j,n+1}^{(m)} + \gamma\beta h f_i(x_{n+1}, \mathbf{y}_{n+1}^{(m)}) + \sum_{j=0}^{k} \alpha_{-j} y_{i,n-j}$$

Partial Derivatives

- Need partial derivatives of each f_i with respect to x and each y_j

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

$$\frac{\partial f_1}{\partial x} = -2y_3 e^{2x} \quad \frac{\partial f_1}{\partial y_1} = -1 \quad \frac{\partial f_1}{\partial y_2} = \frac{1}{2} y_2^{-1/2} \quad \frac{\partial f_1}{\partial y_3} = -e^{2x}$$

$$\frac{\partial f_2}{\partial x} = 0 \quad \frac{\partial f_2}{\partial y_1} = -4y_1 \quad \frac{\partial f_2}{\partial y_2} = 0 \quad \frac{\partial f_2}{\partial y_3} = 0$$

$$\frac{\partial f_3}{\partial x} = 0 \quad \frac{\partial f_3}{\partial y_1} = -3y_2 \quad \frac{\partial f_3}{\partial y_2} = -3y_1 \quad \frac{\partial f_2}{\partial y_3} = 0$$

Review Thomas Algorithm

- General format for tridiagonal equations

$$\begin{bmatrix} B_0 & C_0 & 0 & 0 & \cdots & 0 & 0 \\ A_1 & B_1 & C_1 & 0 & \cdots & 0 & 0 \\ 0 & A_2 & B_2 & C_2 & \cdots & 0 & 0 \\ 0 & 0 & A_3 & B_3 & & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & B_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & A_N & B_N \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \vdots \\ \vdots \\ D_{N-1} \\ D_N \end{bmatrix}$$

Review Thomas Algorithm II

- Gauss elimination upper triangular form

$$\begin{bmatrix} 1 & -E_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -E_1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & -E_2 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & -E_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ \vdots \\ \vdots \\ F_{N-1} \\ F_N \end{bmatrix}$$

California State University Northridge

Review Thomas Algorithm III

- Forward computations
  - Initial: $E_0 = -C_0 / B_0$ $F_0 = D_0 / B_0$
  - For i = 1,… N-1:

$$E_i = \frac{-C_i}{B_i + A_i E_{i-1}} \qquad F_i = \frac{D_i - A_i F_{i-1}}{B_i + A_i E_{i-1}}$$

$$x_N = F_N = \frac{D_N - A_N F_{N-1}}{B_N + A_N E_{N-1}}$$

- Back substitute: $x_i = F_i + E_i x_{i+1}$

California State University Northridge