Basic Concepts in Numerical Analysis

Larry Caretto
Mechanical Engineering 501AB
Seminar in Engineering Analysis

November 6, 2017

Outline

- Review last class
- Midterm Exam November 15 covers material on differential equations and Laplace transforms (no phase plots)
- Overview of numerical solutions
  - Initial value problems in first-order equations
  - Systems of first order equations and initial value problems in higher order equations
  - Boundary value problems
  - Stiff systems and eigenvalues

Review Last Class

- Phase plots, critical points, and stability
- Look at system of two linear homogenous, autonomous equations
  \[- \frac{dy}{dt} = Ay\] (no function of time)
- Critical points and stability depend on matrix eigenvalues which depend on determinant properties
- Described various critical points: node, center, saddle point and spiral

Numerical Analysis Problems

- Numerical solution of algebraic equations and eigenvalue problems
- Solution of one or more nonlinear algebraic equations \( f(x) = 0 \)
- Linear and nonlinear optimization
- Constructing interpolating polynomials
- Numerical quadrature
- Numerical differentiation
- Numerical differential equations

Interpolation

- Start with \( N \) data pairs \( x_i, y_i \)
- Find a function (polynomial) that can be used for interpolation
- Basic rule: the interpolation polynomial must fit all points exactly
- Denote the polynomial as \( p(x) \)
- The basic rule is that \( p(x_i) = y_i \)
- Many different forms

Newton Polynomials

- \( p(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2) + \ldots + a_{n-1}(x - x_0)(x - x_1)(x - x_2) \ldots (x - x_{n-2}) \)
- \( n - 1 \) data points numbered 0 to \( n - 2 \)
- Terms with factors of \( x - x_i \) are zero when \( x = x_i \)
  - Have \( p(x) = y_1 \) to find \( a_i, i = 0 \) to \( n - 1 \)
  - \( a_0 = y_0, a_1 = (y_1 - y_0) / (x_1 - x_0) \)
  - \( a_2 = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \)
  - Solve for \( a_2 \) using results for \( a_0 \) and \( a_1 \)
Newton Polynomials II

\[ y_2 = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \]

\[ a_2 = \frac{y_2 - a_0 - a_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \]

• Could continue in this fashion to determine coefficients from data
• Use alternative scheme – not derived here – known as divided difference table to compute \( a_k \) from same data

Divided Difference Table

• Enter data on \( x_i \) and \( y_i \) in rows of table skipping one row between entries
• Start with \( y_1 \) data as zeroth divided difference
• First divided difference, \( F_1 = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \)
• Second (or later) divided difference is difference of first (or later) differences
• \( a_i \) coefficients are initial divided differences

Divided Difference Example

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>( y_i )</th>
<th>( F_1 )</th>
<th>( F_2 )</th>
<th>( F_3 )</th>
<th>( F_4 )</th>
<th>( a_0 )</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Start at Any Point in Data Table

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>( y_i )</th>
<th>( F_1 )</th>
<th>( F_2 )</th>
<th>( F_3 )</th>
<th>( a_0 )</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Difference Example from \( x = 10 \)

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>Final data point not shown is ( x = 40, y = 200 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>( \rightarrow a_0 )</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>( \rightarrow a_1 )</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>( \rightarrow a_2 )</td>
</tr>
<tr>
<td>600</td>
<td></td>
<td>( \rightarrow a_3 )</td>
</tr>
</tbody>
</table>

### Divided Difference Calculation II

- Divided difference table gives \( a_0 = 10, a_1 = 3, a_2 = .15 \), and \( a_3 = 1/600 \)
- Polynomial \( p(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2) \)
  
  \[
  = 10 + 3(x - 10) + 0.15(x - 10)(x - 20) + \frac{1}{600}(x - 10)(x - 20)(x - 30)
  \]
- \( \Delta^5 y_k \) is called the nth forward difference
- Can also define backwards and central differences

### Divided Difference Code

```c
for ( i = 0; i < n; i++ )
D[0][i] = y[i];

for ( k = 1; k < n; k++ )
  for ( i = 0; i < n - k; i++ )
    D[k][i] = ( D[k-1][i+1] - D[k-1][i] ) / ( x[i+k] - x[i] );
```

- \( D[k][i] \) is \( k \)th value of \( k \)th divided difference
- Code for \( n \) data points (0 to \( n-1 \))

### Constant Step Size

- Divided differences work for equal or unequal step size in \( x \)
- If \( \Delta x = h \) is a constant we have simpler results
  
  \[
  F_k = \Delta y_k / h = (y_{k+1} - y_k) / h
  \]
  \[
  S_k = \Delta^2 y_k / 2h^2 = (y_{k+2} - 2y_{k+1} + y_k) / 2h^2
  \]
  \[
  T_k = \Delta^3 y_k / 6h^3 = (y_{k+3} - 3y_{k+2} + 3y_{k+1} - y_k) / 6h^3
  \]
- \( \Delta^n y_k \) is called the \( n \)th forward difference
- Can also define backwards and central differences

### Interpolation Approaches

- When we have \( N \) data points how do we interpolate among them?
  - Order \( N-1 \) polynomial not good choice
  - Use piecewise polynomials of lower order (linear or quadratic)
  - Can match first and or higher derivatives where piecewise polynomials join
  - Cubic splines are piecewise cubic polynomials that match first and second derivatives [as well as \((x_k, y_k)\) values]

### Cubic Spline Overview

- Have \( N \) cubic polynomials, \( a_i + b_i x + c_i x^2 + d_i x^3 \), with end point of 1 polynomial the start of next, requires \( N + 1 \) data points
  - Data points numbered 0 to \( N \) with polynomials numbered 1 to \( N \)
- Need \( 4N \) equations to get \( N \) values for polynomial coefficients: \( a_i, b_i, c_i, \) and \( d_i \)
- Each polynomial fits data points at ends: \( p_k(x_{k-1}) = y_{k-1} \) and \( p_k(x_k) = y_k, k = 1, N \)
Cubic Spline Overview II

• Have continuity of first and second derivatives: \( p_{k-1}'(x_k) = p_k'(x_k) \) and \( p_{k-1}''(x_k) = p_k''(x_k) \)
• Matching data points gives 2N equations and derivative continuity gives 2N – 2
• Have 4N – 2 equations for 4N unknown polynomial coefficients
• Different models of end point behavior used to provide additional 2 equations

Splines in MATLAB

• Use spline function in MATLAB to get one or more interpolated points
  – \( xIn \) is array of y data for spline fit
  – \( yIn \) is array of x data for spline fit
  – Apply spline to x which can be a single data point or an array using command below
    \[ \text{>> spline}(xIn, yIn, x) \]
  – Generally uses not-a-knot end slopes
• Also has routine \text{unmkpp} to get details of resulting spline coefficients

Cubic Spline Interpolation

Results show effect of different methods used to treat end points

Newton Interpolating Polynomial

High-order polynomials can give unrealistic fits to data

Polynomial Applications

• Data interpolation
• Approximation functions in numerical quadrature and solution of ODEs
• Basis functions for finite element methods
• Can obtain equations for numerical differentiation
• Statistical curve fitting (not discussed here) usually used in practice

Derivative Expressions

• Obtain from differentiating interpolation polynomials or from Taylor series
• Series expansion for \( f(x) \) about \( x = a \)
  \[ f(x) = f(a) + \frac{df}{dx}_{x=a} (x-a) + \frac{1}{2!} \frac{d^2f}{dx^2}_{x=a} (x-a)^2 + \frac{1}{3!} \frac{d^3f}{dx^3}_{x=a} (x-a)^3 + ... \]
• Note: \( d^0f/dx^0 = f \) and \( 0! = 1 \)
  \[ f(x) = \sum_{n=0}^\infty \frac{1}{n!} \frac{d^n f}{dx^n}_{x=a} (x-a)^n \]
• What is error from truncating series?
### Truncation Error

We truncate series after m terms:

\[
\sum_{n=0}^{\infty} \frac{d^n f}{dx^n} \frac{(x-a)^n}{n!} \approx \sum_{n=0}^{m} \frac{d^n f}{dx^n} \frac{(x-a)^n}{n!} + \varepsilon_m
\]

Terms used: Truncation error, \( \varepsilon_m \)

- Use theorem of mean to write truncation error as single term at unknown location, \( \xi \), between \( x \) and \( a \):

\[
\varepsilon_m = \sum_{n=0}^{\infty} \frac{d^n f}{dx^n} \frac{(x-a)^n}{n!} = \frac{d^{m+1} f}{dx^{m+1}} \frac{(x-a)^{m+1}}{(m+1)!}
\]

### Derivative Expressions

- Look at finite-difference grid with equal spacing: \( h = \Delta x \) so \( x_i = x_0 + ih \)
- Taylor series about \( x = x_i \) gives \( f(x_i + kh) \)
  \[ f[x_0 + (i+k)h] = f_i + kh f'_i + \frac{f''(h)}{2!} h^2 + \frac{f'''(h)}{3!} h^3 + \ldots \]
- Compact derivative notation:
  \[ f'_i = \frac{df}{dx} \bigg|_{x=x_i}, \quad f''_i = \frac{d^2 f}{dx^2} \bigg|_{x=x_i}, \quad \ldots \]
  \[ f^{(n)}_i = \frac{d^n f}{dx^n} \bigg|_{x=x_i} \]

### Derivative Expressions II

- Combine all definitions for compact series notation:
  \[
  f(x_i + kh) = f(x_i) + \frac{df}{dx} \bigg|_{x=x_i} kh + \frac{d^2 f}{dx^2} \bigg|_{x=x_i} \frac{1}{2!} (kh)^2 + \frac{d^3 f}{dx^3} \bigg|_{x=x_i} \frac{1}{3!} (kh)^3 + \ldots
  \]

\[
 f_{i+k} = f_i + f'_i kh + \frac{f''(h)}{2!} h^2 + \frac{f'''(h)}{3!} h^3 + \ldots
\]

- Use this formula to get expansions for various grid locations about \( x = x_i \) and use results to get derivative expressions.

### Order of the Error

- Forward and backward derivative have error term that is proportional to \( h \)
- Central difference error is proportional to \( h^2 \)
- Error proportional to \( h^n \) called \( n^{\text{th}} \) order
- Reducing step size by a factor of \( \alpha \) reduces \( n^{\text{th}} \) order error by \( \alpha^n \): 
  
  \[
  E_2 \approx E_1 \left( \frac{h_1}{h_2} \right)^n
  \]
Order of the Error Notation

- Write the error term for n\textsuperscript{th} error term as \( O(h^n) \)
  - Big oh notation, \( O \), denotes order
  - Recognizes that factor multiplying \( h^n \) may change slightly with \( h \)

First order forward
\[
f'_1 = \frac{f_{i+1} - f_i}{h} + O(h)
\]
First order backward
\[
f'_2 = \frac{f_{i-1} - f_i}{h} + O(h)
\]
Second order central
\[
f'_c = \frac{f_{i+1} - 2f_i + f_{i-1}}{2h} + O(h^2)
\]

Higher Order Derivatives

- Add \( f_{i+1} \) and \( f_{i-1} \) expressions; solve for \( f'' \)
\[
f_{i+1} = f_i + f'_i h + \frac{f''_i h^2}{2!} + \frac{f'''_i h^3}{3!} + \ldots
\]
\[
f_{i-1} = f_i - f'_i h + \frac{f''_i h^2}{2!} - \frac{f'''_i h^3}{3!} + \ldots
\]
\[
f_{i+1} + f_{i-1} = 2f_i + 2f'_i h + \frac{2f''_i h^2}{2!} + \frac{2f'''_i h^3}{3!} + \ldots
\]
\[
f_{i+1} + f_{i-1} = 2f_i + 2f'_i h - \frac{2f''_i h^2}{2!} - \frac{2f'''_i h^3}{3!} + \ldots
\]
\[
f''_i = \frac{f_{i+1} + f_{i-1} - 2f_i}{h^2} + \frac{f''_i h^2}{3!} + \frac{f'''_i h^3}{5!} + \ldots = f''_i + f''_i - 2(f'_i + O(h^3))
\]
- \( f'' \) is second-order, central difference expression for second derivative

Higher Order Directional

- We can get higher truncation error expressions at the expense of more computations
- Get second order forward and backward derivative expressions from previous results and \( f'_{i+2} \) and \( f'_{i-2} \), respectively
- Combine \( f'_{i+2} \) and \( f'_{i-2} \) equations with previous expressions for \( f'_{i+1} \) and \( f'_{i-1} \) to eliminate first order error term

Specific Taylor Series

- General equation
\[
f_{i+4} = f_i + f'_i k h + \frac{f''_i (k h)^2}{2!} + \frac{f'''_i (k h)^3}{3!} + \ldots
\]
- \( k = 2 \)
\[
f_{i+2} = f_i + 2 f'_i h + \frac{2 f''_i h^3}{2!} + \frac{8 f'''_i h^4}{3!} + \ldots
\]
- \( k = -2 \)
\[
f_{i-2} = f_i - 2 f'_i h + \frac{2 f''_i h^3}{2!} - \frac{8 f'''_i h^4}{3!} + \ldots
\]
- \( k = 3 \)
\[
f_{i+3} = f_i + 3 f'_i h + \frac{9 f''_i h^3}{2!} + \frac{27 f'''_i h^4}{3!} + \ldots
\]
- \( k = -3 \)
\[
f_{i-3} = f_i - 3 f'_i h + \frac{9 f''_i h^3}{2!} - \frac{27 f'''_i h^4}{3!} + \ldots
\]

Second Order Forward

- Subtract \( 4f_{i+1} \) from \( f_{i+2} \) to eliminate \( h^2 \) term
\[
f_{i+2} - 4f_{i+1} = \left[f_i + f'_i h + \frac{f''_i h^2}{2} + \frac{f'''_i h^3}{6} + \ldots\right] - 4 \left[f_i + f'_i h + \frac{f''_i h^2}{2} + \frac{f'''_i h^3}{6} + \ldots\right]
\]
\[
-4 \left[f_i + f'_i h + \frac{f''_i h^2}{2} + \frac{f'''_i h^3}{6} + \ldots\right] = -3 f'_i - 2 f''_i h + 4 f'''_i h^3 + \ldots
\]
\[
f'_{i+1} - 4f_{i+1} + 3 f_i = -2f''_i h + 4 f'''_i h^3 + \ldots
\]
\[
f'_i = \frac{-f'_{i+1} + 4f_{i+1} - 3f_i + f''_i h^2}{2h} + \frac{f'''_i h^3}{3}
\]

Second Order Backward

- Add \( 4f_{i-1} \) to \( -f_{i-2} \) to eliminate \( h^2 \) term
\[
- f_{i-2} + 4f_{i-1} = \left[f_i - f'_i h + \frac{f''_i h^2}{2} - \frac{f'''_i h^3}{6} + \ldots\right]
\]
\[
+ 4 \left[f_i - f'_i h + \frac{f''_i h^2}{2} - \frac{f'''_i h^3}{6} + \ldots\right] = 3 f'_i - 2 f''_i h + 4 f'''_i h^3 + \ldots
\]
\[
- f_{i-2} + 4f_{i-1} - 3 f_i = -2f''_i h + 4 f'''_i h^3 + \ldots
\]
\[
f'_i = \frac{-f'_{i-2} + 4f_{i-1} - 3f_i + f''_i h^2}{2h}
\]
Other Derivative Expressions

• Can continue in this fashion
  – Write Taylor series for \( f_{i+1}, f_{i-1}, f_{i+2}, f_{i-2}, f_{i+3}, f_{i-3}, \) etc.
  – Create linear combinations with factors that eliminate desired terms
  – Eliminate \( f_i \) term to obtain central difference
  – Keep only terms in \( f_k \) with \( k \geq i \) for forward difference expressions
  – Keep only terms in \( f_k \) with \( k \leq i \) for forward difference expressions
  – Results in numerical analysis texts/online

Other Derivative Formulas

\[ f'_i = \frac{-11f_i + 18f_{i+1} - 9f_{i+2} + 2f_{i+3} - f_{i+4}h^3}{2h} + \ldots \]
\[ f'_i = \frac{-11f_i - 18f_{i-1} + 9f_{i-2} - 2f_{i-3} + f_{i-4}h^3}{2h} + \ldots \]
\[ f'_i = \frac{-f_{i-2} + 8f_{i-1} - 8f_{i+1} + f_{i+2} + f_i}{2h} + \ldots \]
\[ f'_i = \frac{-f_{i-3} + 16f_{i-2} - 30f_{i-1} + 16f_{i+1} - f_{i+2} + f_i}{12h^2} + \ldots \]
\[ f'_i = \frac{-f_{i+4} + 16f_{i+3} - 30f_{i+2} + 16f_{i+1} - f_i}{60h^4} + \ldots \]

Order of Error Examples

• Table 1 in online notes shows error in first derivative for \( e^x \) around \( x = 1 \)
  – Using first- and second-order forward and second-order central differences
  – Step \( h = 0.4, 0.2, \) and \( 0.1 \)
  – Error ratio for doubling step size
    – 4.01 to 4.02 for central differences
    – 4.01 to 4.02 for first-order forward differences
    – 4.01 to 4.02 for second-order forward

\[ \epsilon_n = \frac{\log(e_i/h_i) - \log(e_j/h_j)}{\log(h_j) - \log(h_i)} \frac{d\log(e)}{d\log(h)} \]

Roundoff Error

• Possible in derivative expressions from subtracting close differences
  – Example \( f(x) = e^x \): \( f'(x) \approx (e^{x+h} - e^{x-h})/(2h) \) and error at \( x = 1 \) is \( (e^{1+h} - e^{1-h})/(2h) - e \)

\[ E = \frac{3.004166 - 2.722815}{2.718282} = 1.07 \times 10^{-4} \]

Richardson Extrapolation

• Uses finite-difference method with two step sizes to get improved accuracy
  • Start with \( E = F(h) + TE = F(h) + O(h^n) \)
    – \( E \) is exact result
    – \( F(h) \) is finite difference approximation with step size \( h \)
    – Truncation error, \( TE, \) is \( O(h^n) \)
    – Actually have an infinite series for error

\[ TE = \frac{h^n}{n!} \left( \frac{d^n f}{dx^n} \right)_{x = 0} = \sum_{i=0}^{\infty} \frac{h^n}{i!} \left( \frac{d^n f}{dx^n} \right)_{x = 0} = \sum_{i=0}^{\infty} \frac{h^n}{i!} \]

Figure 2.1. Effect of Step Size on Error
Richardson Extrapolation II

- Look at evaluating error with two step sizes, h and kh
  - Exact value will not change
  - Create sum to display first error term
    \[ E = F(h) + TE = F(h) + \sum_{n=1}^{\infty} A_n h^n = F(h) + \left(h^k + \sum_{n=1}^{\infty} A_n (kh)^n\right) \]
  - Multiply first equation by \( kn \) and subtract the second equation to eliminate the \( A_n \) term
    \[ k^n E - E = k^n F(h) - F(kh) + k^n A_n h^n \]

Richardson Extrapolation III

- Solve equation from previous slide for E
  \[ E = \frac{k^n F(h) - F(kh)}{k^n - 1} + O(h^{n+1}) \]

- The formula for the Richardson extrapolation, RE, has a higher order of the error
  - Truncation error for RE shown below
    \[ TE = \sum_{i=1}^{n} B_i h^i = \sum_{i=1}^{n} B_i h^i \]

Richardson Extrapolation IV

- What does this mean?
  \[ E = \frac{k^n F(h) - F(kh)}{k^n - 1} + O(h^{n+1}) = RE + O(h^{n+1}) \]
  - E is the exact result, F(h) is a finite difference result with step size h
    - If we have two \( n \)-order finite difference results, with two step sizes h and kh, we can use this formula to get an improved result with an error order of \( n + 1 \) (or higher if the error term has every other power of h)

Richardson Extrapolation V

- Richardson extrapolation for forward dcos(x)/dx at \( x = 1 \) and \( h = 0.1 \) & \( h = 0.2 \)
  - What are k and n? \( k = h_2/h_1 = 2; n = \text{order} = 1 \)
    \[ f'(h) = \frac{f(h_2) - f(h_1)}{h_2 - h_1} + O(h) \]
    \[ RE = \frac{k^n F(h) - F(kh)}{k^n - 1} \]
    \[ f'(h = 0.1) = \cos(1.1) - \cos(1) = -0.8670618, \quad 2^1(-0.86706) = -0.88972 \]
    \[ f'(h = 0.2) = \cos(1.2) - \cos(1) = -0.8897228, \quad 2^2(-0.86706) = -0.8444093 \]
  - Extrapolation closer to correct value of dcos(x)/dx|\( x = 1 \) = \( -\sin(1) = -0.84147098 \)

Richardson Extrapolation VI

- Richardson extrapolation for central dcos(x)/dx at \( x = 1 \) and \( h = 0.1 \) & \( h = 0.2 \)
  - What are k and n? \( k = h_2/h_1 = 2; n = \text{order} = 2 \)
    \[ f''(h) = \frac{f(h_2) - f(h_1)}{2h} + O(h^2) \]
    \[ RE = \frac{k^n F(h) - F(kh)}{k^n - 1} \]
    \[ f''(h = 0.1) = \cos(1.1) - \cos(0.9) = -0.8406922, \quad 2^2(-0.84069) = -0.83587 \]
    \[ f''(h = 0.2) = \cos(1.2) - \cos(0.8) = -0.8358872, \quad 2^2(-0.84069) = -0.841468 \]
  - Extrapolation closer to correct value of dcos(x)/dx|\( x = 1 \) = \( -\sin(1) = -0.84147098 \)