Midterm Review

Larry Caretto
Mechanical Engineering 309

Numerical Analysis of Engineering Systems

March 24, 2014

California State University Northridge

Outline

- VBA and MATLAB coding
  - Variable types
  - Control structures (Looping and Choice)
  - Arrays, Functions and Subs
- Roots of Equations
  - General approach and two examples
- Matrix basics and solution by Gaussian elimination
- Numerical differentiation/Interpolation

Main Variable Data Types

Common VBA Variable types		
Data Type	**Memory Size**	**Range**
Boolean	**2 bytes**	**TRUE (non-zero) or FALSE (0)**
Integer	2 bytes	-32,678 to 32,767
Long (integer)	**4 bytes**	**-32,147,483,648 to 2,147,483,647**
Single	4 bytes	(+/-) 1.401298E-45 to 3.402823E38
Double	**8 bytes**	**(+/-) 4.94065645841247E-324 to 1.79769313486232E308**
String (variable-length)	**10 bytes + length**	**0 to about 2 billion characters**
Date	**8 bytes**	**Date and time of day**
Variant	**16 bytes (more for strings)**	**Holds any data type (Default type)**

MATLAB uses double data type for numerical values and has string data type

All MATLAB variables can be complex arrays; no variable declaration is required

Declaring Variables

- VBA uses the following syntax

```
Dim x As Double
Dim k As Long   'For loop index
Dim s As String, d as Date
Dim y as Double, z As Double
```

- Do ***not*** use the following syntax in which x and y are type variant:

```
Dim x, y, z As Double
```

- Declarations not required in MATLAB

Arithmetic Operators

- Operators for both VBA and MATLAB
  - Exponentiation ^, Unary minus – (*E.g.* –x),
  - Multiply/Divide * /
  - Addition/Subtraction + –
- Both use parentheses to overrule normal rules of precedence
- MATLAB operators * / and ^ apply to arrays, but there are also term-by-term operators: .* ./ and .^
  - MATLAB: A/B => AB^{-1} and A\B = $A^{-1}B$

Symbolic Constants

- Useful way to program items that are constant or not expected to change often
- Syntax:

```
Const PI as Double = 3.14159265358979
```

- A const cannot be assigned a value in any other statement
- Not available in MATLAB

Relational/Logical Operators

- Program logic requires choices based on expressions that are true or false
- Relational operators compare other variables and have true or false results
- MATLAB and VBA use <, <=, =, >, >=
- Not equal is <> in VBA ~= in MATLAB
- VBA: Not, And, Or; MATLAB: ~, &&, ||
  - MATLAB array comparisons return arrays of 1(true) and 0(false) and use ~ & |

If – Else If in VBA

```
If <condition1> Then
   <Statements done if condition1 is true>
ElseIf <condition2> Then
   <Statements done if condition2 is true>
ElseIf <condition3> Then
   <Statements done if condition3 is true>
<May be other conditions>
Else
   <Statements done if all conditions false>
End If
<Execute here after any statements done>
```


If – Else If in MATLAB

```
if <condition1>
   <Statements done if condition1 is true>
elseif <condition2>
   <Statements done if condition2 is true>
elseif <condition3>
   <Statements done if condition3 is true>
<May be other conditions>
else
   <Statements done if all conditions false>
end
<Execute here after any statements done>
```


If – Else If Explained

- If any condition is true, the statements following the If or ElseIf are executed
- Once those statements are executed controls to the first statement after the End If
- Statements for only the first true condition are executed
- The Else block is optional
  - If no conditions are true those statements are executed

```
If <condition1> The
   <Statements don
ElseIf <condition2>
   <Statements don
ElseIf <condition3>
   <Statements don
<May be other cond
Else
   <Statements don
End If
<Execute here after
```


Looping

- Count control loop repeats code a fixed number of times
- Conditional looping repeats while a condition is true or until a condition is false
- Both types of loops may be nested
- May use statements (VBA Exit For or Exit Do; MATLAB break) to exit loop before normal exit

VBA Count Controlled Loop

```
For <counter> = <start> to <end>
      <statements>
Next <counter>
For <counter> = <start> to <end> _
                  Step <increment>
      <statements>
Next <counter>
```

If Step not specified, <increment> = 1

Statements in loop repeated nTimes = Int((<end> – <start>) /<increment>) + 1

Loop not executed if nTimes <= 0

MATLAB Count Controlled Loop

```
for <counter> = <array>
      <statements>
end
```

- Array may be set of values, *e.g.* for T = [300 700 2000 6000]
- Can use array definition like VBA for loop

```
for <counter> = <start>:<increment>:<end>
```

- Statements in loop repeated for each element in array
  - Note ability to specify non-uniform array increment

California State University Northridge 13

VBA Conditional Loop

<cond> is a condition (can be true or false)

<stmts> are statements executed in the loop (which can change the condition)

Do <stmts> if <cond> Then Exit Do <stmts> Loop	Do While <cond> <stmts> Loop	Do Until <cond> <stmts> Loop
	Do <stmts> Loop While <cond>	Do <stmts> Loop Until <cond>

California State University Northridge Note tests before or after loop 14

MATLAB Conditional Loop

- Only one loop, a while loop, with the following structure

```
while <condition>
  <statements>
end
```

- Example (compute machine epsilon)

```
eps = 1
while 1 + eps ~= 1
  eps = eps / 2
end
eps = eps * 2
```

California State University Northridge 15

Arrays

- Arrays can be visualized as data on an experimental variable
  - Could describe pressure data points mathematically as P_1, P_2, *etc*.
  - In programming languages we can represent data points as P(1), P(2), *etc*.
  - We call the numbers (1, 2, *etc*.) indices or subscripts
    - We can use constants or variables for the subscripts: P(4), P(k), where k has a value

California State University Northridge 16

Two-dimensional Arrays

Consider an experiment where you vary the current over six levels, the voltage over four levels and measure the efficiency, e, of an electromechanical device. The data for each combination of current and voltage can be represented as shown below

	I(1)	**I(2)**	**I(3)**	**I(4)**	**I(5)**	**I(6)**
V(1)	e(1,1)	e(1,2)	e(1,3)	e(1,4)	e(1,5)	e(1,6)
V(2)	e(2,1)	e(2,2)	e(2,3)	e(2,4)	e(2,5)	e(2,6)
V(3)	e(3,1)	e(3,2)	e(3,3)	e(3,4)	e(3,5)	e(3,6)
V(4)	e(4,1)	e(4,2)	e(4,3)	e(4,4)	e(4,5)	e(4,6)

California State University Northridge 17

Dimensioning Arrays in VBA

- Can declare arrays as follows

Dim I(1 to 6) as double
Dim V(1 to 4) as double
Dim e(1 to 4, 1 to 6) as double

- Size below depends on Option Base

Dim I(6) as double
Dim V(4) as double
Dim e(4, 6) as double

Array dimensioning not required in MATLAB

California State University Northridge 18

Using Arrays in VBA

- VBA array components are referenced by their subscripts
- This is often done in a for loop

```
PI = 4 * atn(1)
For k = 0 to 100
    x(k) = sin(k * PI / 100)
Next k
```

In MATLAB use:
t = 0:pi/100:pi
x = sin(t)

- x is an array with 101 components giving sin(x) for $0 \le x \le \pi$, with $\Delta x = \pi/100$

Two-Dimensional Arrays in VBA

- Use nested for loops
  - Use example of existing data on current and voltages stored in arrays

```
For k = 1 to 4
   For j = 1 to 6
      Power(k,j) = current(j) * voltage(k)
   Next j
Next k
```


MATLAB Arrays

- Enter arrays as x = [1 2 3; 4 5 6…
  - Elements in one row separated by spaces
  - Semicolon indicates new row
  - Subarrays z = x(r1:r2,c2:c2); transpose x'
  - Array formula operators: +, -, *, /,\,^ give matrix results ($A/B = AB^{-1}$, $A\backslash B = A^{-1}B$)
  - Term-by-term operations with +, -, .*, ./, .^
  - Build larger arrays from smaller arrays using same approach entering initial array

VBA Strings

- Consider only variable length
- Use **Dim str as String** to dimension string
- Use & or + as concatenation operator to join two strings
- Len(str) gives length of string
- Left, Right, and Mid give substrings in same manner as worksheet functions
- InStr function searches for substrings
- String constant s = "string"

MATLAB Strings

- Use single quotes for string constants
- Setting s = 'string' makes s a string variable with the value 'string'
  - No declaration required
  - Concatenate strings by placing them in an array of strings

```
s = ['This string has ' '3' 'characters']
result:   s = 'This string has 3 characters'
```


VBA Functions

- The header has the following form

```
Function <name> ( <arguments> ) As <type>
```

  - <name> is the name of the function
    - Must set name to some value in function code
  - <type> is the data type for the function
  - <arguments> may be blank or have one or more entries of the form <variable> As <type>
    - <variable> is a variable used in the function
    - <type> is the data type for that variable
    - Separate multiple entries in the list by commas
    - Arguments provide input data to function

Writing Your VBA Function

• The function has the following form

```
Function <name> ( <arguments> ) As <type>
    <code to do computations>
    <name> = <value from computations>
End Function
```

• Example function

```
Function vCyl ( R as Double, H as Double ) As _
                                        Double
   vCyl = 4 * atn(1) * R^2 * H
End Function
```


Using Your VBA Function

• Use by cell entries in worksheet

```
=vCyl( B1, B2)
=vCyl( 1, 20)
=vCyl( radiusName, heightName)
```

• Call from other VBA procedures

```
V = vCyl( radius, height)
cylVol = vCyl( 1, 20)
v10cyls = 10 * vCyl( rad, hgt)
```


MATLAB Functions

• The header has the following form

function <return> = <name> (<arguments>)

– <return> is a one variable or row array of variables returned by the function

– <name> is the name of the function

– <arguments> may be blank or have one or more variable names separated by commas

• Arguments provide input data to function

• Variables in the <return> list must be assigned a value in the function

MATLAB Trajectory Function

```
function [x, y] = traj(v0, theta, N)
%Computes frictionless trajectory
%Uses SI units (meters, seconds)
%V0 is initial speed in m/s
%theta is initial angle in degrees
%N is number of points computed
g = 9.80665;    %gravity in m/s^2
tMax = 2 * v0 * sind(theta) / g;
t = 0:tMax/(N-1):tMax;
x = v0*cosd(theta) * t;
y = v0*sind(theta) * t - g * t.^2/2;
plot(x,y);
end
```

Use file names the same as the function names (e.g. traj.m) to save functions

Operator .^ allows use of t as an array

Using the Trajectory Function

• Use only the function name will return value of first argument, x, in default variable ans

```
>> trajectory(10, 45, 101);
```

• Return x array only

```
>>x = trajectory(10, 45, 101)
```

• Return x and y arrays

```
>> [x,y] = trajectory(10, 45, 101)
```

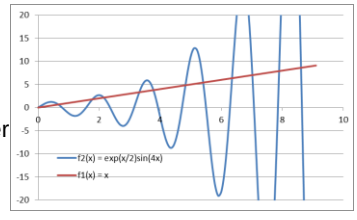

Roots of Equations, f(x) = 0

• Plot of $f_1(x) = x$ and $f_2(x) = e^{x/2}\sin(4x)$

• $f(x) = x - e^{x/2}\sin(4x) = 0$

• Roots about x = 0, x = 0.8, x = 1.6, etc.

Plot shows that this equation has infinite number of roots



Iteration Solutions

- In iteration we make one (or more) initial guesses for x and compute f(x)
  - $x^{(m)}$ or x_m is value of x at iteration m
  - $f^{(m)}$ or $f_m = f(x_m)$ is value of f(x) at $x = x_m$
- Unless we are extremely lucky we will not find f(x) = 0 for initial guesses
- Use recent value(s) of f(x) to estimate a value of x that gets us closer to the solution x* at which f = 0

Convergence Criteria

- $| x_{i+1} - x_i | \le \varepsilon_2 + \varepsilon_3 | x_{i+1} |$ OR $| f_{i+1} | \le \varepsilon_1$ combines tests on x and f
  - Useful when df/dx >> 1 or df/dx << 1 in the region of the solution
- Although we are trying to solve for f = 0, we are really interested in finding the value of x
  - Error tests on x are usually more important
  - Relative error test is more general

Secant Method Operation

- Algorithm:
  - Make initial guesses x_0 and x_1
  - Compute $f_0 = f(x_0)$ and $f_1 = f(x_1)$
  - Repeat
    - $x_{i+1} = x_i - f_i [(x_i - x_{i-1}) / (f_i - f_{i-1})]$
  - Until convergence criterion is met
- Algorithm does not require initial guesses that bracket root, but bad initial guesses may not converge quickly or at all

Secant Method Example

- $f(x) = e^x - x - 2$ Root is 1.14619322062058
- Initial guesses $x_0 = 0$ and $x_1 = 1$
- $f_0 = f(x_0) = f(0) = e^0 - 0 - 2 = -1$
- $f_1 = f(x_1) = f(1) = e^1 - 1 - 2 = -0.281718$

$$x_2 = x_1 - f_1 \frac{x_1 - x_0}{f_1 - f_0} = 1 - (-0.28178) \frac{1 - 0}{-0.28178 - (-1)}$$

- $x_2 = 1.3922$ $f_2 = f(x_2) = e^{1.3922} - 1.3992 - 2 = 0.631526$

$$x_3 = x_2 - f_2 \frac{x_2 - x_1}{f_2 - f_1} = 1.3922 - (0.631526) \frac{1.3922 - 1}{0.6315 - (-0.2818)}$$

$x_3 = 1.12099$; $f_3 = -0.0531$

Newton's Method Operation

- Algorithm:
  - Make initial guess x_0
  - Compute $f_0 = f(x_0)$ and $f'(x_0) = df/dx|_{x=x0}$
  - Repeat
    - $x_{i+1} = x_i - f_i / f'(x_i)$
  - Until convergence criterion is met
- Algorithm requires only one initial guess
- A bad initial guesses may not converge quickly or at all or converge to wrong root

Newton's Method Example

- $f(x) = e^x - x - 2$ $f'(x) = e^x - 1$
- Initial guess $x_0 = 1$ ($x_0 = 0$ gives f' = 0)
- $f_0 = f(x_0) = f(1) = e^1 - 1 - 2 = -0.281718$
- $f'(x) = e^1 - 1 = 1.71828$

$$x_1 = x_0 - \frac{f_0}{f'(x_0)} = 1 - \frac{-0.281718}{1.71828} = 1.16395$$

$$f(x_1) = e^{1.16305} - 1.16395 - 2 = 0.038616$$

$$f'(x_1) = e^{1.16305} - 1 = 2.02569$$

$$x_2 = x_1 - \frac{f_1}{f'(x_1)} = 1.16395 - \frac{0.038616}{2.202569} = 1.464212$$

Find $x_4 =$ 1.14619326, $f(x4) = 8.2 \times 10^{-8}$

Matrix Basics

- Matrix is array with n rows and m columns
  - **A** = **B** if same size and $a_{ij} = b_{ij}$ for all i and j
- Add or subtract matrices
  - **C** = **A** ± **B** only valid if **A**, **B**, and **C** have the same size (rows and columns)
  - Components of **C**, $c_{ij} = a_{ij} \pm b_{ij}$
- Multiplication by a scalar: **C** = x**A**
  - **C** and **A** have the same size (rows and columns)
  - Components of **C**, $c_{ij} = xa_{ij}$
  - For scalar division, **C** = **A**/x, $c_{ij} = a_{ij}/x$

Null (**0**) and Unit (**I**) Matrices

- For any matrix, **A**, **A** + **0** = **0** + **A** = **A**; **IA** = **AI** = **A** and **0** **A** = **A** **0** = **0**
- The unit (or identity) matrix is a square matrix; the null matrix, which need not be square, is sometimes written $\mathbf{0}_{(nxm)}$

$$\mathbf{0} = \begin{bmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & & & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & 0 \end{bmatrix} \qquad \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & 1 \end{bmatrix}$$

Diagonal Matrix and Transpose

- Diagonal matrix, D, has nonzero terms only on diagonal
- Transpose of A, denoted as A^T switches rows and columns

$$\mathbf{D} = \begin{bmatrix} d_1 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & d_2 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & d_3 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & d_n \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 3 & 12 & -6 \\ 14 & -2 & 0 \end{bmatrix} \qquad \mathbf{A}^T = \begin{bmatrix} 3 & 14 \\ 12 & -2 \\ -6 & 0 \end{bmatrix}$$

General Matrix Multiplication

- For matrix multiplication, **C** = **AB**
  - **A** has n rows and p columns
  - **B** has p rows and m columns
  - **C** has n rows and m columns

$$c_{ij} = \sum_{k=1}^{p} b_{ik} a_{kj} \qquad (i = 1, n; j = 1, m)$$

- Example

$$\mathbf{A} = \begin{bmatrix} 3 & 0 & -6 \\ 4 & -2 & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 3 & 4 \\ 1 & 2 \\ 6 & 1 \end{bmatrix}$$

$$\mathbf{AB} = \begin{bmatrix} 3(3)+0(1)-6(6) & 3(4)+0(2)-6(1) \\ 4(3)-2(1)+0(6) & 4(4)-2(2)+0(1) \end{bmatrix} = \begin{bmatrix} -27 & 6 \\ 10 & 12 \end{bmatrix}$$

Inverse of a Matrix

- For a ***square*** matrix, **A**, an inverse matrix, $\mathbf{A}^{-1}$ ***may exist*** such that $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$
- For the algebraic equation ax = b, $x = a^{-1}b$
- For the matrix equation **Ax** = **b**, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$
- Just as $x = a^{-1}b$ is not valid if a = 0, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ is not valid if $\mathbf{A}^{-1}$ does not exist
  - A^{-1} does not exist if Det**A** = 0
- The inverse and the determinant are important concepts in analysis of linear systems, but are not used in computational work

From Equations to **Ax** = **b**

- Usual form for N = 3 equations

$$\begin{aligned} 3x + 7y - 3z &= 8 \\ 2x - 4y + z &= -3 \\ 8x + 6y - 2z &= 14 \end{aligned}$$

$$\underset{\mathbf{A}}{\begin{bmatrix} 3 & 7 & -3 \\ 2 & -4 & 1 \\ 8 & 6 & -2 \end{bmatrix}} \underset{\mathbf{x}}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}} = \underset{\mathbf{Ax}}{\begin{bmatrix} 3x_1 & 7x_2 & -3x_3 \\ 2x_1 & -4x_2 & 1x_3 \\ 8x_1 & 6x_2 & -2x_3 \end{bmatrix}} = \underset{\mathbf{b}}{\begin{bmatrix} 8 \\ -3 \\ 14 \end{bmatrix}}$$

- An equation is a row in the **Ax** = **b** format

Gaussian Elimination

- Solve the set of equations on the right

$$2x_1 - 4x_2 - 26x_3 = -34 \; (i)$$
$$-3x_1 + 2x_2 + 9x_3 = 13 \; (ii)$$
$$7x_1 + 3x_2 + 8x_3 = 14 \; (iii)$$

- Subtract –3/2 times (i) from equation (ii) and 7/2 times (i) from (iii)

$$\left[-3-\left(\frac{-3}{2}\right)2\right]x_1+\left[2-\left(\frac{-3}{2}\right)(-4)\right]x_2+\left[9-\left(\frac{-3}{2}\right)(-26)\right]x_3=\left[13-\left(\frac{-3}{2}\right)(-34)\right]$$
$$\left[7-\left(\frac{7}{2}\right)2\right]x_1+\left[3-\left(\frac{7}{2}\right)(-4)\right]x_2+\left[8-\left(\frac{7}{2}\right)(-26)\right]x_3=\left[14-\left(\frac{7}{2}\right)(-34)\right]$$

Unnecessary computer operations

Gaussian Elimination II

- Result from first set of operations

$$2x_1 - 4x_2 - 26x_3 = -34$$
$$0x_1 - 4x_2 - 30x_3 = -38$$
$$0x_1 + 17x_2 + 99x_3 = 133$$

- Subtract 17/(-4) times (ii) from (iii)

$$x_2\left[17-\left(\frac{17}{-4}\right)(-4)\right]+x_3\left[99-\left(\frac{17}{-4}\right)(-30)\right]=\left[133-\left(\frac{17}{-4}\right)(-38)\right]$$

- Final upper-triangular form

$$2x_1 - 4x_2 - 26x_3 = -34$$
$$0x_1 - 4x_2 - 30x_3 = -38$$
$$0x_1 + 0x_2 - \frac{57}{2}x_3 = -\frac{57}{2}$$

Back Substitution

- Final upper-triangular form

$$2x_1 - 4x_2 - 26x_3 = -34$$
$$-4x_2 - 30x_3 = -38$$
$$-\frac{57}{2}x_3 = -\frac{57}{2}$$

- Solve third equation for x_3

$$x_3 = -\frac{57}{2}\Big/-\frac{57}{2} = 1$$

- Solve second equation for x_2

$$x_2 = \frac{-38+30x_3}{-4} = \frac{-38+30(1)}{-4} = 2$$

- Solve first equation for x_1

$$x_1 = \frac{-34+4x_2+26x_3}{2} = \frac{-34+4(2)+26x_3(1)}{2} = 0$$

General Gaussian

- Loop over all rows from 1 to N – 1 to be used as the pivot row
  - For each pivot row, loop over all rows from pivot + 1 to N
    - For each row loop over all columns from pivot+1 to N + nBcols (work with augmented matrix [**A b**]

$$a_{row,column} \leftarrow a_{row,column} - \frac{a_{row,pivot}}{a_{pivot,pivot}} a_{pivot,column}$$

- Back substitution formula for i = n, n-1, …, 1:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}}$$

Solutions for **Ax** = **b**

- For a set of m equations in n unknowns
  - If Rank(**A**) = Rank([**A b**]) = n, there is a unique solution
  - If Rank(**A**) = Rank([**A b**]) < n, there are an infinite number of solutions
  - If Rank(**A**) ≠ Rank([**A b**]) there are no solutions
- Use Gaussian elimination to find Rank as number of nonzero rows

Three Examples

$x_1 - 4x_2 - 26x_3 = 2$ $2x_2 + 9x_3 = -5$ $7x_1 + 3x_2 + 8x_3 = -13$	$x_1 - 4x_2 - 26x_3 = 2$ $2x_2 + 9x_3 = -5$ $50.5x_3 = 50.5$	$x_1 = 0$ $x_2 = -7$ $x_3 = 1$
$x_1 - 4x_2 - 26x_3 = 2$ $2x_2 + 9x_3 = -5$ $-2x_1 + 10x_2 + 61x_3 = -9$	$x_1 - 4x_2 - 26x_3 = 2$ $2x_2 + 9x_3 = -5$ $0 = 0$	$x_1 = -8 - 8\alpha$ $x_2 = -2.5 - 4.5\alpha$ $x_3 = \alpha$
$x_1 - 4x_2 - 26x_3 = 2$ $2x_2 + 9x_3 = -5$ $-2x_1 + 10x_2 + 61x_3 = -8$	$x_1 - 4x_2 - 26x_3 = 2$ $2x_2 + 9x_3 = -5$ $0 = 1$	No solution

First Example Rank

Original $\mathbf{A}=\begin{bmatrix}1 & -4 & -26\\ 0 & 2 & 9\\ 7 & 3 & 8\end{bmatrix}$ $[\mathbf{A}\ \mathbf{b}]=\begin{bmatrix}1 & -4 & -26 & 2\\ 0 & 2 & 9 & -5\\ 7 & 3 & 8 & -13\end{bmatrix}$

Row-echelon form $\mathbf{A}=\begin{bmatrix}1 & -4 & -26\\ 0 & 2 & 9\\ 0 & 0 & 50.5\end{bmatrix}$ $[\mathbf{A}\ \mathbf{b}]=\begin{bmatrix}1 & -4 & -26 & 2\\ 0 & 2 & 9 & -5\\ 0 & 0 & 50.5 & 50.5\end{bmatrix}$

Here we see that rank(**A**) = rank([**A b**]) = number of unknowns = 3 so we have a unique solution

Second Example Rank

Original $\mathbf{A}=\begin{bmatrix}1 & -4 & -26\\ 0 & 2 & 9\\ -2 & 10 & 61\end{bmatrix}$ $[\mathbf{A}\ \mathbf{b}]=\begin{bmatrix}1 & -4 & -26 & 2\\ 0 & 2 & 9 & -5\\ -2 & 10 & 61 & -9\end{bmatrix}$

Row-echelon form $\mathbf{A}=\begin{bmatrix}1 & -4 & -26\\ 0 & 2 & 9\\ 0 & 0 & 0\end{bmatrix}$ $[\mathbf{A}\ \mathbf{b}]=\begin{bmatrix}1 & -4 & -26 & 2\\ 0 & 2 & 9 & -5\\ 0 & 0 & 0 & 0\end{bmatrix}$

rank(**A**) = rank([**A b**]) = 2 which is less than the number of unknowns (3) so we have an infinite number of solutions

Third Example Rank

Original $\mathbf{A}=\begin{bmatrix}1 & -4 & -26\\ 0 & 2 & 9\\ -2 & 10 & 61\end{bmatrix}$ $[\mathbf{A}\ \mathbf{b}]=\begin{bmatrix}1 & -4 & -26 & 2\\ 0 & 2 & 9 & -5\\ -2 & 10 & 61 & -8\end{bmatrix}$

Row-echelon form $\mathbf{A}=\begin{bmatrix}1 & -4 & -26\\ 0 & 2 & 9\\ 0 & 0 & 0\end{bmatrix}$ $[\mathbf{A}\ \mathbf{b}]=\begin{bmatrix}1 & -4 & -26 & 2\\ 0 & 2 & 9 & -5\\ 0 & 0 & 0 & 1\end{bmatrix}$

Here, rank(**A**) = 2 ≠ rank([**A b**]) = 3; therefore we have no solutions

Numerical Differentiation

- Formulas have following properties
  - Type of derivative (first, second, third, *etc.*)
  - Location of points used in the derivative, relative to the point of the derivative (forward difference, backward difference, central difference)
  - Order of the error: $O(h^n)$ is an n^{th} order error (truncation error proportional to h^n)
- Roundoff error occurs when h is so small that significant figures are lost

Some Derivative Expressions

$$f_i' = \frac{f_{i+1} - f_i}{h} + O(h) \qquad f_i' = \frac{f_i - f_{i-1}}{h} + O(h)$$

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2)$$

$$f_i' = \frac{f_{i-2} - 4f_{i-1} + 3f_i}{2h} + O(h^2)$$

$$f_i' = \frac{-f_{i+2} + 4f_{i+1} - 3f_i}{2h} + O(h^2)$$

$$f_i'' = \frac{f_{i+1} + f_{i-1} - 2f_i}{h^2} + O(h^2)$$

Note order of derivative, order of error, and direction (forward vs. backward)

$O(h^n)$ for information only. Not used in calculations.

More Derivative Expressions

$$f_i'' = \frac{2f_i - 5f_{i-1} + 4f_{i-2} - f_{i-3}}{h^2} + O(h^2)$$

$$f_i' = \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12h} + O(h^4)$$

$$f_i'' = \frac{-f_{i-2} + 16f_{i-1} - 30f_i + 16f_{i+1} + -f_{i+2}}{12h^2} + O(h^4)$$

- What is order of derivative, order of error, and direction (central, forward or backward difference) of expression?
- Is sum of coefficients always zero?

Polynomial Interpolation

- Fit n^{th} order polynomial, p(x), to n + 1 data points, (x_0, y_0) to (x_n, y_n)
  - Can start at any data point in set
  - Select points for interpolation that are closest to the value to be interpolated
  - Basic idea is that polynomial will fit each data point exactly: $p(x_i) = y_i$
  - Example is Newton polynomial $a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots$
  - a_k coefficients from divided-difference table

Newton Polynomials from x_{start}

- $p_0(x) = a_0$ and $p_1(x) = a_0 + a_1(x - x_{start})$
- $p_2(x) = a_0 + a_1(x - x_{start}) + a_2(x - x_{start})(x - x_{start+1})$
- $p_3(x) = a_0 + a_1(x - x_{start}) + a_2(x - x_{start})(x - x_{start+1}) + a_3(x - x_{start})(x - x_{start+1})(x - x_{start+2})$
- $a_0 = y_{start}$, $a_1 = F_{start} = (y_{start+1} - y_{start}) / (x_{start+1} - x_{start})$. $a_2 = S_{start} = (F_{start+1} - \ldots$

$$p_n(x) = \sum_{m=0}^{n-1} a_m \prod_{k=0}^{m-1} (x - x_{start+k}) \qquad p_n(x) = p_{n-1}(x) + a_n \prod_{k=0}^{n-1} (x - x_{start+k})$$

Sample Divided Difference Table

$x_0=0$	$y_0=10$	$\leftarrow a_0$			
		$F_0 = \frac{y_1 - y_0}{x_1 - x_0}$	$\leftarrow a_1$		
10	$y_1=20$		$S_0 = \frac{F_1 - F_0}{x_2 - x_0}$	$\leftarrow a_2$	
		$F_1 = \frac{y_2 - y_1}{x_2 - x_1}$		$T_0 = \frac{S_1 - S_0}{x_3 - x_0}$	$\leftarrow a_3$
25	$y_2=150$		$S_1 = \frac{F_2 - F_1}{x_3 - x_1}$		$R_0 = \frac{T_1 - T_0}{x_4 - x_0}$
		$F_2 = \frac{y_3 - y_2}{x_3 - x_2}$		$T_1 = \frac{S_2 - S_1}{x_4 - x_1}$	$\uparrow a_4$
50	$y_3=320$		$S_2 = \frac{F_3 - F_2}{x_4 - x_2}$		
		$F_3 = \frac{y_4 - y_3}{x_4 - x_3}$			
90	$y_4=790$				

Follow same a_k pattern for other starting points

Divided difference table gives polynomial coefficients (next page)

Divided Difference Table for p(x = 30)

$x_0=0$	$y_0=10$				$F_1 = \frac{150-20}{25-10} = \frac{130}{15} = \frac{26}{3}$
		$F_0 = \frac{y_1 - y_0}{x_1 - x_0}$			
10	$y_1=20$	$\leftarrow a_0$	$S_0 = \frac{F_1 - F_0}{x_2 - x_0}$	$F_2 = \frac{320-150}{50-25} = \frac{170}{25} = \frac{34}{5}$	
		$F_1 = \frac{y_2 - y_1}{x_2 - x_1}$	$\leftarrow a_1$	$T_0 = \frac{S_1 - S_0}{x_3 - x_0}$	
25	$y_2=150$		$S_1 = \frac{F_2 - F_1}{x_3 - x_1}$	$\leftarrow a_2$	$R_0 = \frac{T_1 - T_0}{x_4 - x_0}$
		$F_2 = \frac{y_3 - y_2}{x_3 - x_2}$		$T_1 = \frac{S_2 - S_1}{x_4 - x_1}$	
50	$y_3=320$		$S_2 = \frac{F_3 - F_2}{x_4 - x_2}$		
		$F_3 = \ldots$			
90	$y_4=790$		$S_1 = \frac{\frac{34}{5} - \frac{26}{3}}{50-10} = \frac{\frac{102}{15} - \frac{130}{15}}{50-10} = \frac{-28}{(15)(40)} = -\frac{7}{(150)}$		

Quadratic Polynomial

- Usual form for initial point at $x = x_0$
  - $p_2(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$
- Here we start at $x_{start} = x_1$
  - $p_2(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2)$
  - Using data from previous chart gives

$$p_2(x) = 20 + \frac{26}{3}(x-10) - \frac{7}{150}(x-10)(x-25)$$

  - This gives $p(x_k) = y_k$ for $x_k = 10$, 25, and 50
  - Interpolated value for x = 30 is 189

Midterm Exam

- Closed book (code sheet on exam shows basic VBA and MATLAB structures)
  - Problems like those on quizzes
  - Interpreting VBA or MATLAB code
  - Writing simple VBA or MATLAB code
  - Using a given algorithm to solve f(x) = 0
  - Simple matrix operations (equality, addition, subtraction, multiplication, transpose)
  - Simple Gaussian elimination (≤4 x 4 system)
    - Recognize unique, zero, or infinite solutions
  - Numerical differentiation and interpolation

Sixth Quiz Results

- Number of students: 22
- Maximum Possible Score: 25
- Average score: 18.3
- Median score: 20.5
- Standard deviation: 6.27
- Grade distribution:

```
  5   9 10 10 10 13 15
17 18 18 19 22 22 23 23
 23 23 24 24 25 25 25
```

California State University Northridge 61

Sixth Quiz Comments

- Choose data points in the region of the x value to be interpolated
- Denominators of divided difference expressions are always Δx's
- Choose central-differences derivative expressions if possible; use one-sided differences for boundaries only
- To find the n^{th} derivative of y in a table of y vs. x, use y values in the numerator

California State University Northridge 62

Sixth Quiz Comments II

- $O(h^n)$ is for information about order of error only, not part of calculation
- For derivatives from a data table
  - h = Δ(independent variable)
  - Numerator values are dependent variable
  - Here x is independent, y dependent
    - Asked to find second derivative of y
- Do not have to get entire divided-difference table, only the terms needed
  - Look for type of polynomial required

California State University Northridge 63