Outline

- Programming and Final exams
  - VBA and MATLAB basics
  - Roots of equations
  - Matrix algebra and solution of simultaneous equations
  - Numerical differentiation
  - Interpolation
  - Regression
  - Quadrature
  - Numerical solution of ODEs

Programming Exam

- Can choose to use VBA or MATLAB
- Will have one relatively simple problem with two hours to get solution
  - Open book, notes, online help, but no internet searches for code
- Will have test cases with known solutions
  - Use test cases to verify program correctness
- Done with Excel workbook or commands from MATLAB command window

Programming Exam Rules

- Each student does own work and emails results to instructor
- No instructor help for programming
  - Can ask questions to clarify exam
  - Can get help for grave problems like computer crash
- Try to get as much done as possible
  - Describe future steps if you have not finished

Final Exam Reminder

- Monday, May 12, 8 to 10 pm, this room
- Closed book, no notes, no computer, no consultation, etc.
- Will be given necessary equations
  - If you think that some equation is missing ask and it will be provided
- Final will have same kinds of problems as midterm, with new algorithms mainly

Final Exam Problems

- Write simple VBA and MATLAB code (for general calculations or a given numerical algorithm)
- Given a numerical algorithm, evaluate a few steps with your calculator
- May be some short questions like how many data points does it take to fit a cubic polynomial or short exercises with matrices
Possible Numerical Algorithms

- Roots of equations, f(x) = 0, single equations only
- Simultaneous linear algebraic equations
- Interpolation
- Regression
- Numerical integration
- Numerical solution of Ordinary Differential Equations

Review VBA

- Option Explicit
- Dim and Const statements
- Expressions with operator precedence and replacement statements
  - Arithmetic, relational, logical and string operators
- Type conversion
  - Implicit as in MsgBox " x = " * x
  - Explicit with conversion functions like CDbl

Choice Statements

- The If statement
  - If <condition> Then
    - <statements to be executed if the condition is true>
  - End If
  - <Transfer control here if condition is false; normal transfer at end of if code>
- Alternative version for one statement in If
  - If <condition> Then <statement>

If – Else If Explained

- If any condition is true, the statements following the If or Else If 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
  - End If
  - <Execute here after any statements done>

Looping

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

Count Controlled Loop

For <counter> = <start> to <end>
  <statements>
  If Step not specified, <increment> = 1
Next <counter>
For <counter> = <start> to <end> _
  Step <increment>
  <statements>
Next <counter>
Statements in loop repeated nTimes = (<end> – <start>) /<increment> + 1
Loop not executed if nTimes <= 0
Count Controlled Examples

For k = 1 to 11 step 3
   msgBox k
Next k

Code at left computes $e^x$ for $x = 1$ with relative error of $1 \times 10^{-8}$

Conditional Loop

<cond> is a condition (can be true or false)
<stmts> are statements executed in the loop (which should change the condition)

<table>
<thead>
<tr>
<th>Do While &lt;cond&gt; Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Until &lt;cond&gt; Loop</td>
</tr>
<tr>
<td>Do &lt;stmts&gt; Loop</td>
</tr>
<tr>
<td>Do &lt;stmts&gt; Loop Until &lt;cond&gt;</td>
</tr>
</tbody>
</table>

Note tests before or after loop

Nested For Loops

• For loops used with arrays
• Nested for loops for 2D arrays

For k = n To 1 Step -1
   x(k) = a(n,n+1)
   For j = k+1 to n
      x(k) = x(k) - a(k,j) * x(j)
   Next j
Next k

Arrays

• Arrays can be visualized as data on an experimental variable
  – Could describe pressure data points mathematically as $P_1$, $P_2$, etc.
  – In VBA we can represent these 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

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

<table>
<thead>
<tr>
<th>I(1)</th>
<th>I(2)</th>
<th>I(3)</th>
<th>I(4)</th>
<th>I(5)</th>
<th>I(6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V(1)</td>
<td>e(1,1)</td>
<td>e(1,2)</td>
<td>e(1,3)</td>
<td>e(1,4)</td>
<td>e(1,5)</td>
</tr>
<tr>
<td>V(2)</td>
<td>e(2,1)</td>
<td>e(2,2)</td>
<td>e(2,3)</td>
<td>e(2,4)</td>
<td>e(2,5)</td>
</tr>
<tr>
<td>V(3)</td>
<td>e(3,1)</td>
<td>e(3,2)</td>
<td>e(3,3)</td>
<td>e(3,4)</td>
<td>e(3,5)</td>
</tr>
<tr>
<td>V(4)</td>
<td>e(4,1)</td>
<td>e(4,2)</td>
<td>e(4,3)</td>
<td>e(4,4)</td>
<td>e(4,5)</td>
</tr>
</tbody>
</table>
Using Arrays

• Arrays components are referenced by their subscripts
• This is often done in a For loop
For k = 0 to 100
    x(k) = sin(k * PI / 100)
Next k
• x is an array with 101 components giving sin(x) for 0 ≤ x ≤ π, with Δx = π/100

Two-Dimensional Arrays

• Use nested for loops
  – Use example of current and voltages
For k = 1 to 4
    For j = 1 to 6
        Power(k,j) = I(j) * V(k)
    Next j
Next k

• Recall table:
  - V was in rows
  - I was in columns
  - Power(k,j) is Power(row, column)
  - Are k and j indices correct?

Dynamic Arrays

• What if you do not know array size until program is actually running?
• Use Dim a() to tell compiler that a is an array then use ReDim with actual dimensions
Sub getArray( N as long) as Variant
    Dim x() as Double : ReDim X(1 to N)
• Can go from Dim a() as Double to any size ReDim: ReDim a(1 to 10, 6 to 12)

Passing Arrays to Procedures

• Declare array in argument list with parentheses to indicate array
Sub mine( A() as double)
    ‘No dim statement for A
    A(2,3) =
End Sub
Use this for any size array. Variant arrays do not need ()

• Calling program sets actual dimensions on array and uses only the following
Dim B(1 to 10, 1 to 6) as double
Call mine(B)

Determining Array Bounds

• The UBound and LBound functions determine the upper and lower bounds of unknown array dimensions
• For a two-dimensional array, A(m,k)
  – LBound(A,1) is the lower bound of m
  – UBound(A,1) is the upper bound of m
  – LBound(A,2) is the lower bound of k
  – UBound(A,2) is the upper bound of k

Worksheet Arrays to VBA

• Passed as a range of cells
• First step is to set a type variant variable equal to the input range variable
  – The variant variable is now an two-dimensional array
  – May have single row or single column, but is still a two-dimensional array
  – Lower bound is always one
  – Can use UBound to get sizes
**Worksheet Array Example**

```vba
Function getMean (Ain As Range) _
    As Double
Dim A as Variant
Dim sum As double, cells As Long, k As Long
Dim nRows As Long, nCols As Long, m As Long
A = Ain:
    nRows = UBound(A,1) : sum = 0
nCols = UBound(A, 2) : cells = nRows * nCols
For k = 1 To nRows
    For m = 1 To nCols
        sum = sum + A(k,m)
    Next m
Next k
getMean = sum / cells
End Function
```

**Worksheet Array Example II**

```vba
Dim sum As double, cells As Long, m As Long
Dim nRows As Long, nCols As Long, k As Long
A = Ain:
    nRows = UBound(A,1) : sum = 0
nCols = UBound(A, 2) : cells = nRows * nCols
For k = 1 to nRows
    For m = 1 to nCols
        sum = sum + A(k,m)
    Next m
Next k
getMean = sum / cells
End Function
```

**VBA Array to Worksheet**

- VBA steps to return array to worksheet
  - Declare the function type as Variant
  - In the function or sub declare a working array for calculations
    - Use application.caller for dimensions
    - Write the code for values in working array
    - At end of function set <function name> = <working array name>
- To use the function: select cells; enter function in formula bar; Cont+Shift+Enter

```vba
Function array2wks(<arguments>) As Variant
Dim userRows As Long
Dim userColumns As Long
Dim workArray() as Double
'Statements below determine rows and columns
userRows = Application.Caller.Rows.Count
userColumns = Application.Caller.Columns.Count
ReDim workArray(1 to userRows, 1 to userColumns)

array2wks = workArray
End Function
```

**Strings**

- Consider only variable length
- Use Dim str as 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

**Getting Programs to Work**

- VBA detects syntax errors (one-line)
- Compilation (before execution) detects structure errors (more than one line)
- Programs will halt at many errors (like divide by zero)
- Programs will return errors like #NAME to worksheet instead of results
- Use test cases to make sure that a new program is working correctly
Getting Programs to Work

- Syntax errors: errors in single line
  - Line turns red after “completed” with optional error message and location
    - Select auto-syntax check
- Compilation errors: errors in program structure involving more than one line
  - E.g. If statement without following End If
  - E.g. Next statement without preceding For
  - Could get “incorrect” error message for nested structures (see next slide)

Misleading Error Message

- Error message when Next is omitted, but End If included to match If

Identified Run-time Errors

- Syntax/compilation usually, but not always, easy to remedy
- Some run-time errors will stop and allow debugging (Click “Debug” button)

Run-time Error Highlighted

- After clicking “Debug” on previous dialog
- Highlighted statement caused run-time error
  - Real error cause is values set for k and m are both zero
  - Need to trace back from error statement to error cause

#NAME Error

- This error may be returned by a UDF when the function cannot be found
  - It was never defined
  - It is located in a different workbook
  - There is a module with the same name
  - The name is misspelled in the call
  - Arguments in the function call do not match arguments in the function header
  - Function is not located in a module (located on code for worksheet or ThisWorkbook)
  - Private Function located in a different module

#VALUE Error

- Returned to worksheet when there is an execution error that VBA cannot trap
  - Often linked to attempts to exceed array bounds
    - To find such errors use the debugger repeated times to find the statement causing the error
    - Locate area in code where execution halts for no apparent reason
    - Find exact statement where this error occurs
    - Hover mouse to find “out-of-range” arrays or other possible errors
Incorrect Results

- Programs should always be tested with inputs whose solution is known
- If this solution is not found, use debugger to step through program to find errors
- Use worksheet to compute intermediate results to check against program values
  - Divided-difference table for polynomial interpolation as an example
- Worksheet formulas may have errors too

Debugging

- Debugger allows you to step through a program and see intermediate values
  - Useful to find location of errors
- Items to use in debugger
  - Breakpoints stop execution at certain points
  - Step-by-step execution
  - Intermediate and Watch windows
  - Hover mouse over variable to get its value
  - Change statement to be executed next

Useful Toolbar Icons

- Start execution (will stop at breakpoint)
- Halt execution
- Step to cursor
- Step into (F8)
- Step over
- Step out
- Quick Watch
- Add watch
- Comment and "uncomment" blocks of text
- Hover mouse over variable to get its value
- Change statement to be executed next

Hovering Mouse

- Can hover mouse over scalars to show values of variables
  - Does not work for whole arrays, but works for array components

Watch Window

- Use “Add Watch” to specify variables and Watch Type
- View variables in Watch Window

Help

- Help systems for Excel and VBA
- Search function does not always return what you are looking for
- If you know the keyword, type it, place the cursor in the keyword, and press F1
- Sometimes a Google search for “VBA <subjectYouAreInterestedIn>” works
MATLAB Review I

- Ways of performing commands
  - Command window
    - Answers may be suppressed with semicolon
    - Default answer variable is `ans`
  - Functions and scripts
    - Anonymous functions
  - Data entry commands for arrays
    - `X = [1 2 3; 4 5 6; 7 8 9; 10 11 12]`
    - Spaces between data on same row
    - Semicolons to start new row

MATLAB Review II

- Subarray commands `x(r1:r2, c1:c2)`
  - Use only : for complete row or column
- Other array definition: low:delta:high
  - Can omit delta if delta = 1
- Can get functions of arrays
  - `t = 0:pi/100:2*pi; y = sin(t); plot(t,y)`
- Transpose matrix: `A^T` in MATLAB is `A'`
  - Command `A'` gives

MATLAB Review III

- Matrix operations (+ - * / ^)
- Term-by-term operations (`+ - * ./ .^`)
- Valid operations between matrix, `X`, and scalar `a`: `a + X`, `a - X`, `a * X`, `a ./ X`, `X / a`
- Can create larger matrices from smaller ones if they are compatible
  - `C = [A B]` if `A` and `B` have same rows
  - `C = [A; B]` if `A` and `B` have same columns

Trajectory Function Example

```matlab
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;
tMax = 2 * v0 * sind(theta) / g; % Miscalculation
% using tMax

% Vectorization - no loops

% Generate time vector
% t = 0:tMax/(N-1):tMax;
t = 0:tMax/(N-1):tMax;

% Calculate x and y values
x = v0*cosd(theta) * t;
y = v0*sind(theta) * t - g * t.^2/2;

% Plot the trajectory
plot(x, y);
end
```

MATLAB if Statements

- Use the following format
  - `if <expression1> <statements1>`
  - `elseif <expression2> <statements2>`
  - `<other elseif's possible here>`
  - `else <optional <statements> end`

Same structure as VBA but

- "Then" not used
- All keywords (if, elseif, else) in lower case
- Final statement is end, not endif

Using Your MATLAB Functions

- Used as any MATLAB Function
  - `>>v0 = 10;`
  - `>>theta = 60;`
  - `>>N = 100;`
  - `>>[x, y] = traj(v0, theta, N);`

- Can use only part of return variables
  - `>>x = traj(V0,theta,N)` returns x values in x
  - `>>= traj(v0,theta,N)` returns x values in ans

Cal State Northridge
MATLAB While Statement
• MATLAB has only one conditional looping command with a test before
  while <condition>
  <statements>
end
• The <statements> in the while loop continue to execute while the
  <condition> is true

MATLAB for Statement
• Similar to VBA For statement, but loop limits are a MATLAB array specification
  for <index> = <MATLAB array>
  <statements>
end
• Examples of for statements
  for T = [300, 500, 1000, 5000]
  for x = 0 : 0.01 : 2
  for k = 1 : 25
    (Same as 1:1:25)

Review Roots of Equations
• Write equation in form \( f(x) = 0 \)
• Methods solving \( f(x) = 0 \)
  – Bisection
  – Secant method
  – Newton’s Method
  – False position (regula falsi)
  – Successive substitution
  – May be given algorithm for other method
    and asked to apply it

Methods and Process
• Bisection and False Position require two initial guesses that bracket root
• Newton’s method requires one guess
• Secant method requires two guesses (do not have to bracket root)
• Different convergence conditions
  – Absolute error in \( \Delta x \) or \( f(x) \)
  – Relative error in \( \Delta x \)
  – Combination of above

Matrix Basics
• Define an \( m \) by \( n \) matrix as an array of
  with \( m \) rows and \( n \) columns
  \[
  A = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{21} & a_{22} & a_{23} & a_{24} \\
  a_{31} & a_{32} & a_{33} & a_{34}
  \end{bmatrix}
  \]
  – Row, column, diagonal, unit, null, inverse
    and transpose matrices
• Matrix equality, addition, subtraction require same size matrices

General Matrix Multiplication
• For matrix multiplication, \( C = AB \)
  \[
  c_{ij} = \sum_{k=1}^{p} b_{ik} a_{kj}
  \]
  \( i = 1, n; \ j = 1, m \)
• Example
  \[
  A = \begin{bmatrix} 3 & 0 & -6 \\ 4 & -2 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 3 & 4 \\ 1 & 2 \\ 6 & 1 \end{bmatrix}
  \]
  \[
  AB = \begin{bmatrix} (3(3) + 0(1) - 6(6)) & (3(4) + 0(2) - 6(1)) \\ (4(3) - 2(1) + 0(6)) & (4(4) - 2(2) + 0(1)) \end{bmatrix} = \begin{bmatrix} -27 & 6 \\ 10 & 12 \end{bmatrix}
  \]
From Equations to $\mathbf{Ax} = \mathbf{b}$

- Usual form for $3x + 7y - 3z = 8$
  $\mathbf{N} = 3$
  2 equations
  $8x + 6y - 2z = 14$

- An equation is a row in the $\mathbf{Ax} = \mathbf{b}$ format

Gaussian Elimination

- Solve the set of equations $2x - 4x - 26x = -34$ (i)
  on the right
  $7x + 3x + 8x = 14$ (ii)

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

- Gaussian Elimination II

- Result from first set of operations
  $2x - 4x - 26x = -34$
  $0x - 4x - 30x = -38$
  $0x + 17x + 99x = 133$

- Subtract $17/(-4)$ times
  (ii) from (iii)
  $2x - 4x - 26x = -34$
  $0x - 4x - 30x = -38$
  $0x + 0x - 57/2 x = -57/2$

- Final upper-triangular form

Back Substitution

- Final upper-triangular form
  $2x - 4x - 26x = -34$
  $-4x - 30x = -38$

- Solve third equation for $x_3$
  $x_3 = -57/2$
  $x_3 = 1$

- Solve second equation for $x_2$
  $x_2 = -38 + 30x_3$
  $x_2 = -38 + 30(1)$
  $x_2 = 2$

- Solve first equation for $x_1$
  $x_1 = -34 + 4x_2 + 26x_3$
  $x_1 = -34 + 4(2) + 26(1)$
  $x_1 = 0$

Solutions for $\mathbf{Ax} = \mathbf{b}$

- For a set of $n$ equations in $n$ unknowns
  - If $\text{Rank}(\mathbf{A}) = \text{Rank}([\mathbf{A} \mathbf{b}]) = n$ there is a unique solution $2x + y = 4$; $2x - y = 0$
  - If $\text{Rank}(\mathbf{A}) = \text{Rank}([\mathbf{A} \mathbf{b}]) < n$: an infinite number of solutions $x + y = 1$; $2x + 2y = 2$
  - If $\text{Rank}(\mathbf{A}) \neq \text{Rank}([\mathbf{A} \mathbf{b}])$ there are no solutions $x + y = 1$; $2x + 2y = 3$
  - Use Gaussian elimination to find Rank as number of nonzero rows

Numerical Differentiation

- Formulas have following properties
  - Type of derivative (first, second, third, etc.)
  - Direction of points used in the derivative, relative to the point of the derivative (forward, backward, central)
  - 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
May 5, 2014

Some Derivative Expressions

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

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

More Derivative Expressions

\[ f'' = \frac{2f_i - 5f_{i-1} + 4f_{i-2} - f_{i-3}}{h^2} + O(h^2) \]
\[ f' = \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12h} + O(h^4) \]
\[ f'' = \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i-1} - f_{i-2}}{12h^2} + O(h^4) \]

- What is sum of coefficients in numerator for each expression?
- Is there a reason for this?

Interpolation

- Given a table of data, \((x_i, y_i)\) estimate a value of \(y\) for an \(x\) value not in the table
- Use \(N+1\) table \((x_i, y_i)\) points for \(N\)th-order polynomial
- Pick points that surround the value of \(x\) for which the polynomial is to be evaluated
- Get Newton polynomial from divided difference table

Divided Difference Example

\[
\begin{align*}
0 & \quad 0 & \quad \leftarrow a_0 \\
10 & \quad 10 & \quad \leftarrow a_1 \\
20 & \quad 40 & \quad \leftarrow a_2 \\
30 & \quad 100 & \quad \leftarrow a_3 \\
\end{align*}
\]

Divided Difference Table

\[
\begin{array}{|c|c|c|}
\hline
x_i & y_i & \text{Difference} \\
\hline
0 & 0 & \leftarrow a_0 \\
10 & 10 & \leftarrow a_1 \\
20 & 40 & \leftarrow a_2 \\
30 & 100 & \leftarrow a_3 \\
\hline
\end{array}
\]

Divided Difference Example II

- Divided difference table gives \(a_0 = 0\), \(a_1 = 1\), \(a_2 = 0.1\), and \(a_3 = 1/600\)
- Polynomial \(p(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2)\)
  \(= 0 + 1(x - 0) + 0.1(x - 0)(x - 10) + (1/600)(x - 0)(x - 10)(x - 20) = x + 0.1x(x - 10) + (1/600)x(x - 10)(x - 20)\)
- Check \(p(30) = 30 + 0.1(30)(20) + (1/600)(30)(20)(10) = 30 + 60 + 10 = 100\) (Correct!)
Linear Regression

- Seeks approximate linear relationship among data set \( (x_i, y_i) \)
- Fit equation: \( \hat{y}_i = a + bx_i \)
- Notation \( \hat{y}_i \) indicates approximate value, which may be different from data \( y_i \)
- Equations for \( a \) and \( b \) based on minimizing sum of squares of differences between actual and approximate data
  \[ \sum (y_i - \hat{y}_i)^2 \rightarrow \text{Minimum} \]

Confidence Limits

- \( R^2 \) value gives overall measure of fit
  \(- 0 \leq R^2 \leq 1 \)
  - Confidence limits for the regression parameters \( a \) and \( b \) based on t statistic and user-specified confidence limit \( 1 - \alpha \)
    - Typically choose \( \alpha = .05 \) for 95% confidence
  \[ a \pm t_{\alpha/2, N-k} \left( \frac{1}{N} \sum (x_i - \bar{x})^2 \right)^{1/2} \]
  \[ b \pm t_{\alpha/2, N-k} \left( \frac{1}{N} \sum (x_i - \bar{x})^2 \right)^{1/2} \]

More Equations to be Used

- Compute estimated \( y \) values
  \[ \hat{y}_m = b_0 + \sum_{j=1}^{K} b_j x_{jm} \]
- Compute the \( R^2 \) value
  \[ R^2 = 1 - \left( \frac{\sum_{m=0}^{N-1} (y_m - \hat{y}_m)^2}{\sum_{m=0}^{N-1} y_m^2} \right) - N \left( \frac{\bar{y}}{s^2} \right)^2 \]

Equations for \( a \) and \( b \)

- Substitute equation for \( a \) into equation for \( b \) (both copied below) and solve for \( b \)
  \[ a = \frac{\sum y_i - b \sum x_i}{N} \]
  \[ b = \frac{N \sum x_i y_i - \left( \sum x_i \right) \left( \sum y_i \right)}{N \sum x_i^2 - \left( \sum x_i \right)^2} \]
  \[ = \frac{\sum x_i y_i - N(\bar{x}\bar{y})}{\sum x_i^2 - N(\bar{x})^2} \]
- First solve for \( b \) then solve for \( a \)
  - Can set \( a = 0 \) to force line through origin
- Can use equations with all sums or means

Multivariate Linear Regression

- In general we can have \( K \) predictive variables, \( x_1 \) to \( x_K \)
- General model equation:
  \[ y = b_0 + \sum_{j=1}^{K} b_j x_j \]
- How do we represent the data?
  - Each data set consists of one value of \( y \) and one value for each of the \( x \) variables
  - For data set \( m \), we can call the value of \( y \), \( y_m \), and we can call the value of \( x_i \) for data set \( m \) \( x_{jm} \)
  - Multivariate analysis finds coefficients \( b_0 \) to \( b_K \)
  - Each coefficient has standard error (times t statistic = confidence interval)

Numerical integration formulas

- Trapezoid Rule
  \[ I = \int_a^b f(x) \, dx = T + E = h \left[ f_0 + f_N + \sum_{i=1}^{N-1} f_i \right] + O(h^2) \]
- Simpson’s (1/3) rule (even \( N \) only)
  \[ I = \int_a^b f(x) \, dx = S + E = \frac{h}{3} \left[ f_0 + f_N + 4 \sum_{i=2,4,6} f_i + 2 \sum_{i=3,5,7} f_i \right] + O(h^4) \]
- Basic definitions of step size, \( h \), number of intervals, \( N \),
  \[ x_0 = a, \ x_N = b, \ f_k = f(a + kh) \]
  \[ x_i = a + k \frac{N}{h} \]
Richardson Extrapolation

- Have numerical expression, F, for two different step sizes h and kh
  - Call these F(h) and F(kh)
  - These expressions have a lead error term with order n, O(h^n) (error proportional to h^n)
  - Can get higher order expression by using Richardson Extrapolation formula
    - Typically pick k > 1, but it could be < 1 so long as formula is consistently applied
    \[
    RE = \frac{k^n F(h) - F(kh)}{k^n - 1} \]

Numerical ODE Solution

- Solve initial value problem, dy/dx = f(x,y) (known) with y(x_0) = y_0
  - Use a finite difference grid: x_{i+1} - x_i = h_{i+1}
  - Replace derivative by finite-difference approximation: dy/dx ≈ (y_{i+1} - y_i) / (x_{i+1} - x_i) = (y_{i+1} - y_i) / h_{i+1}
  - 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 \( (y_{i+1} - y_i) / h_{i+1} = f_{avg} \)
  - Repeatedly compute \( y_{i+1} = y_i + h_{i+1} f_{avg} \)

Order of ODE Methods

- Local error is error after one step when the initial conditions are known exactly
- Global error is the error after more than one step
- For an nth-order local error, the global error has an order of n – 1
- The global error is the more important error which is used to describe a method

Review Notation and Order

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

Review Simple Methods

- Euler: \( y_{i+1} = y_i + h f_i \)
  - First order
- Huen’s method (second order)
  \[
  y_{i+1} = y_i + h f_i + \frac{h}{2} \left( f_{i+1} + f_i \right) \]
  \[
  x_{i+2} = x_i + h \]
- Modified Euler method (second order)
  \[
  \begin{align*}
  y_{i+1/2} &= y_i + \frac{h_{i+1}}{2} f(x_i, y_i) \\
  x_{i+1/2} &= x_i + \frac{h_{i+1}}{2} \\
  y_{i+1} &= y_i + h_{i+1} f \left( x_{i+1/2}, y_{i+1/2} \right) \\
  x_{i+1} &= x_i + h_{i+1} 
  \end{align*}
  \]

Romberg Integration

- General forms for initial \( T_{k0} \) and subsequent \( T_{km} \)
  \[
  T_{km} = 4^m T_{m+1} - T_{m} - h_{m+1} \quad m = 0, \ldots, k; \quad all \; k 
  \]
Review 4th Order Runge-Kutta

- Uses four derivative evaluations per step
  \[ y_{i+1} = y_i + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \]
  \[ x_{i+1} = x_i + h i_{i+1} \]
  \[ k_1 = h i f (x_i, y_i) \]
  \[ k_2 = h i f \left( x_i + \frac{h i}{2}, y_i + \frac{k_1}{2} \right) \]
  \[ k_3 = h i f \left( x_i + \frac{h i}{2}, y_i + \frac{k_2}{2} \right) \]
  \[ k_4 = h i f (x_i + h i, y_i + k_3) \]

Example

- Two masses joined by a spring/damper
- Original ODEs for each mass
- Define velocities \( \frac{dx_1}{dt} = v_1 \), \( \frac{dx_2}{dt} = v_2 \)
- Rewrite original ODEs using velocities

Example Continued

- Replace \( x_1, x_2, v_1, v_2 \) in equations below by \( y_1, y_2, y_3, y_4 \)
- Result is standard-form system: \( \frac{dy_1}{dt} = f_1 \)
- Many different ODE Algorithms
- Systems of ODEs
- ODE Systems

ODE Algorithms

- Many different ones, with different types
  - Multistep vs. single-step
  - Implicit vs. explicit
  - Step-size adjustment for better accuracy with fewer operations
  - Prefer higher-order algorithms
  - Special algorithms for stiff systems (wide variation in time constants)
  - Final could give new algorithm and ask you to take 2-3 steps with calculator