

This Week

Monday

- Solving Decision Problems Using VBA (Part I)

Wednesday

- Lab Exercise: Box Packing

Decision Problems in Excel

The characteristics of the decision problem typically dictate the solution approach:

- Direct Computation of a Closed-Form Solution

e.g., EOQ: $\min_{Q \geq 0} \text{Cost}(Q) = (K \times \frac{D}{Q}) + (C \times D) + (H \times \frac{Q}{2}) \longrightarrow Q^* = \sqrt{\frac{2KD}{H}}$

- Enumeration

e.g., Health Insurance Plans: $\min_{Plans} E[Cost] \text{ s.t. } \Pr(Cost > \$1500) \leq 2\%$

- Algorithms Using Directed Search

- **Heuristic Methods**

e.g., Greedy algorithm for Box Packing

- **Exact Methods**

e.g., Simplex method for linear problems, Branch and bound for integer problems, Gradient methods for convex nonlinear problems

VBA is frequently used in implementing Directed Search Methods

Topics

- **Some VBA Fundamentals**
- **Working with VBA Ranges**

A Problem Solving Framework

1. Define the Problem

2. Collect and Organize Data

3. Characterize Uncertainty and Data Relationships

 ***4. Build an Evaluation Model***

 ***5. Formulate a Solution Approach***

 ***6. Evaluate Potential Solutions***

7. Recommend a Course of Action

Some VBA Fundamentals

- **Recording Macros**
- **Objects, Properties, and Methods**
- **Variables and Variable Types**
- **Subroutine Structure**
- **Built-In Functions**
- **Loops and Logic**
- **Avoiding Code Interruption**

Recording Macros

- The easiest way to learn and start using VBA code is to record a macro and then use the Visual Basic Editor (VBE) to edit it: Developer→Code→Record Macro
- Developer→Code→Visual Basic brings up the VBE (as does Alt-F11).
- Once a subroutine is created within a workbook, you can execute it using Developer→Code→Macros (or Alt-F8). You can also create a button using the Forms toolbar and attach a macro to it (Developer→Controls→Insert).

IMPORTANT:

If there is any chance that a macro will be executed from the wrong worksheet, you should be sure to activate the correct worksheet within the subroutine code itself.

Objects, Properties, and Methods

- **Objects (“Nouns”)**

- » Entities within the Excel environment
- » e.g., Application, Workbook, Worksheet, Range, Chart
- » Collections of objects are also objects (e.g., Worksheets)
- » Excel object model is hierarchical
- » For help with Excel objects, open the Object Browser (F2) and select the Excel library

- **Properties (“Adjectives”)**

- » Attributes of objects that you can evaluate and/or change
- » e.g., Range(“A1”).Value, Chart(“Chart 1”).ChartType

- **Methods (“Verbs”)**

- » Actions that you can take on objects
- » e.g., Worksheet(“Sheet 1”).Select, Range(“A1”).Copy

Variables and Variable Types

- **Defining Variables**

- » Declare all variables at the beginning of subroutines using the ***Dim*** keyword:

Dim counter **As Integer**, ratio **As Double**

Dim getrange **As String**

- **Variable Types**

- » String
- » Integer, Long (for values >32,768 in magnitude)
- » Single, Double (for higher precision)
- » Boolean
- » Currency
- » Variant
- » *Object* (e.g., Range, must use **Set** keyword to assign)

Subroutine Structure

Sub *subname*()

' Variable Declarations

Dim *var1* **As Integer**, *var2* **As Double**

Dim *var3* **As Range**

' Execution Statements

var1 = 1

var2 = 2.135

Set *var3* = Range("D4")

. . .

End Sub

Built-In Functions

- **VBA Library**

- » From the VBE Window, open the Object Browser (F2) and select the VBA library. Many classes contain useful built-in functions: DateTime, Math, Strings, Financial, etc.

- **Using Excel Functions in VBA**

- » ***Application.WorksheetFunction*** allows you to “borrow” most Excel functions for use within VBA:

Dim total **As Double**

total = Application.WorksheetFunction.Sum(“A1:C10”)

IMPORTANT:

If a VBA library function exists, then you CANNOT use Excel’s version of the function in VBA. For instance, LN, SQRT, and RAND will not work in VBA (must use log, sqr, rnd instead)

Loops and Logic

- **If/Then Statements**
- **For and Do While Loops**
- **Select/Case Statements**
- **Exit Do, Exit For, Exit Sub Statements**

Loops and Logic Example 1

Sub Factorial()

Dim N **As Integer**, counter **As Integer**, N_fact **As Double**

N = Range("D4").Value

If N < 0 **Then**

MsgBox "Number must be a nonnegative integer"

Exit Sub

End If

N_fact = 1

For counter = 2 **To** N

 N_fact = N_fact * counter

Next counter

Range("D6").Value = N_fact

End Sub

Loops and Logic Example 2

```
Sub Compute_Years_to_Double()
```

```
Dim growth_rate As Double, target As Double
```

```
Dim c_growth As Double, counter As Integer
```

```
    . . .
```

```
    Do While c_growth < target
```

```
        c_growth = c_growth * (1 + growth_rate)
```

```
        counter = counter + 1
```

```
    Loop
```

```
    . . .
```

```
End Sub
```

Loops and Logic Example 3

```
Sub Increment_Stock_Level()  
Dim part_to_increase As String  
    part_to_increase = Range("Part_Type").Value  
    Select Case part_to_increase  
        Case "A"  
            Range("Stock_A").Value = Range("Stock_A").Value + 1  
        Case "B"  
            Range("Stock_B").Value = Range("Stock_B").Value + 1  
        Case "C"  
            Range("Stock_C").Value = Range("Stock_C").Value + 1  
        Case Else  
            MsgBox "Invalid Part Type Entered"  
            Exit Sub  
    End Select  
End Sub
```

Avoiding Code Interruption

- To avoid “screen flicker” while code is running:
Application.ScreenUpdating = False
- To avoid code being halted by a pop-up warning (e.g., when deleting a worksheet):
Application.DisplayAlerts = False
- If calling Solver from a VBA subroutine, to avoid code interruption by the results dialog box when Solver completes, use the optional UserFinish parameter:
SolverSolve UserFinish := True

Working With VBA Ranges

- **Frequently Used Range Methods**
- **Frequently Used Range Properties**
- **VBA Code Examples that:**
 - » Use Range Names within VBA
 - » Execute Range Methods
 - » Modify Range Properties
 - » Define and Use Range Variables

Frequently Used Range Methods

General syntax is: `Range.method [method parameters]`

Brackets are used here to distinguish parameters – do NOT type them

- **Clear** – deletes everything from the range
- **ClearContents** (also **ClearFormats**) – deletes the contents (or formats) of the range
- **Copy** [Destination:= *drange*] – copies the range [to the specified destination range]
- **PasteSpecial** [Paste:= *xlconstant*] – pastes the contents of the clipboard into the range according to the specified criteria
- **Select** – selects the range
- **Sort** [Key1:= *column*, Order1:= *xlconstant*] – sorts the range according to the specified criteria

Frequently Used Range Properties

- **Address** – holds the range address as a string (e.g., “A1:C5”)
- **Cells** – used to reference a particular cell in a range (e.g., Range(“A1:C5”).Cells(1,2) references cell B1)
- **Column** (also **Row**) – holds the number of the first range column
- **Columns** (also **Rows**) – used to reference a particular range column (e.g., Range(“A1:C5”).Columns(1) references cells A1:A5)
- **Font** – holds properties of the range font such as size, bold, italic (e.g., Range(“A1:C5”).Font.Bold = True)
- **Formula** – holds the range formula as a string (e.g., “=SUM(“A1:C5”)”)
- **Interior** – holds the properties of the range interior such as color (e.g., Range(“A1:C5”).Interior.ColorIndex = 5)
- **Name** – holds the range name as a string
- **Offset** – used to reference a cell relative to a range
- **Value** (also **Value2**) – holds the value of the range (default property)

Range Example

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2														
3			Input Values						Output Values					
4		Table	A	B	C	D	E	F	A	B	C	D	E	F
5		1	1	2	3	4	5	6						
6		2	7	8	9	10	11	12						
7		3	13	14	15	16	17	18						
8		4	19	20	21	22	23	24						
9		5	25	26	27	28	29	30						
10														

- » The cell range C5:H9 is named “**Input_Range**”
- » The cell range I5:N9 is named “**Output_Range**”
- » The cell I5 is named “**Output_Range_Start**”
- » We want to create a subroutine that:
 - **Clears** the Output_Range cells
 - **Copies** each cell in Input_Range to its corresponding entry in Output_Range
 - **Formats** the cells in Output_Range to be **bold**, font size **14**, and have a **bright yellow background**

Copy and Format 1

```
Sub CopyAndFormat1()
```

*Executes range
methods to
clear and copy* { Range("Output_Range").**Clear**
Range("Input_Range").**Copy** Destination:=Range("Output_Range")

*Modifies range
properties to
format* { With Range("Output_Range")
.Font.**Bold** = True
.Font.**Size** = 14
.Interior.**ColorIndex** = 6
End With

```
End Sub
```

Copy and Format 2

```
Sub CopyAndFormat2()
```

```
Dim numrows As Long
```

```
Dim counter As Long
```

```
Dim Data As Range, Output As Range
```

*Uses the **Set** keyword to initialize the Range variables*

```
{ Set Data = Range("Input_Range")  
  Set Output = Range("Output_Range")
```

```
Output.Clear
```

```
numrows = Data.Rows.Count
```

Uses the **Rows** property to get the row count of Data and to access the indicated rows of the Data and Output Ranges

*Uses **For** loop to copy the Range rows one by one*

```
{ For counter = 1 To numrows  
    Data.Rows(counter).Copy Destination:=Output.Rows(counter)  
  Next counter
```

```
...
```

```
End Sub
```

Copy and Format 3

```
Sub CopyAndFormat3()
```

```
    Dim cell As Range
```

*Uses the **End** property
to select the
contiguous range to
the right and down*

```
    { Range("Output_Range_Start").Select  
      Range(Selection, Selection.End(xlToRight).End(xlDown)).Clear
```

*Uses **For Each** loop
to copy the Range
cells one
at a time*

```
    { For Each cell In Range("Input_Range")  
      cell.Copy Destination:=cell.Offset(0, 6)  
    Next
```

Uses the **Offset**
property to specify
a cell relative to
another cell

```
    Range("Output_Range_Start").Select
```

```
    With Range(Selection, Selection.End(xlToRight).End(xlDown))
```

```
        .Font.Bold = True
```

```
        .Font.Size = 14
```

```
        .Interior.ColorIndex = 6
```

```
    End With
```

```
End Sub
```

Copy and Format 4

```
Sub CopyAndFormat4()
```

```
Dim numrows As Long
```

```
Dim numcolumns As Long
```

```
Dim counter As Long
```

```
...
```

Uses the *Rows* and *Columns* properties of Input_Range to get the row and column counts.

```
numrows = Range("Input_Range").Rows.Count
```

```
numcolumns = Range("Input_Range").Columns.Count
```

```
With Range("Input_Range").Cells(1, 1)
```

Uses *For* loop to copy the Range rows one by one

```
  { For counter = 0 To (numrows - 1)  
    Range(.Offset(counter, 0), .Offset(counter, (numcolumns - 1))).Copy  
    Destination:=.Offset(counter, numcolumns)  
  } Next counter
```

```
End With
```

```
...
```

```
End Sub
```

Uses the *Offset* property to specify the beginning and the end of the range of cells to copy, as well as the destination cell