A Visual Basic for Applications Package for Simple Simplex Demonstrations in Linear Programming

Ronald D. Notestine
Faculty of Management
Chukyo University

Key word: Linear Programming
VBA
Chvatal Dictionary

Introduction
Linear programming remains the most widely used method in management science applications in business. (We do not consider statistical methods of management science.) This is so despite the rise of computationally intensive, non-linear methods due to the increasing availability of increasingly powerful computers. No matter how powerful computers become, however, they will always be able to solve much larger problems if those problems are in the form of linear models amenable to the simplex algorithm.

Now, even casual users, using easily available software, can use a normal desktop or laptop computer to apply non-linear algorithms, simulation, or even genetic algorithms. This ability to solve problems using other methods on a desktop computer inevitably reduces the amount of time in a management science course available for linear programming and the simplex algorithm. This is not a bad thing, of course, since the gain from being able to handle a much wider class of problems far outweighs drawback of less time for linear programming models. The gain goes beyond just being able to handle more types of problems. For the normal business student, being able to model problems in a more straightforward way, without having to use the non-intuitive tricks often required to make a model suitable for linear programming is a great benefit.
Be that as it may, linear programming is still one of the most important, if not the single most important, type of model in management science courses. This brings us to a discussion of exactly how it should be taught. Or, rather more to the point here, what aspects of it should be taught. Until recently it seems to have been universally accepted that the details of the simplex algorithm should be taught. Students should learn the pivot conditions, how to calculate a tableau pivot, how to interpret the coefficients in the objective row of the pivot tableau, and how to determine optimality (or its lack) in the final tableau.

When the author took his first course in linear programming (as graduate student in mathematics in the mid-1970s), it was a mind-numbing, and not very enlightening experience. The linear programming education zeitgeist seemed to dictate endless hand calculations of tableaux pivots. As the author was one of those seemingly incapable of avoiding errors in hand-calculation, and his beloved HP-35 did not handle fractions, he seemed to spend his life doing little else except multiplying and dividing fractions. There is undoubtedly benefit to one's soul in performing such labors at some time in one's life. However, in doing so, so much time is devoted to the details of calculation that overall concepts are obscured.

In a previous paper, the author discussed a package written in Mathematica for performing the simplex method. While the package certainly works, and one version of it can be used to produces animations, the Mathematica program is widely used in business only in the relatively specialized financial engineering area. The Microsoft Excel spreadsheet program has become a standard in business offices as well as at most universities and personal computers owned by students at home. For this reason, it is desirable to have a version, easily used by beginners, available within the Microsoft Excel environment.

The appendix lists the code for a package titled SimplexExplorer. It allows students to input, pivot, and determine optimality for straightforward linear programming problems of the sort often encountered in basic courses. It is written in the form of an Excel add-in. An attached tool bar contains command buttons for performing all functions of the package.

Chvatal Dictionaries
A feature of the Excel package described here, that was not available in the Mathematica version, is the capability to make a Chvatal dictionary for each tableau. This is an alternative representation of the information of the tableau in which each row is solved for the basic
variable. The right hand side of each equation consists of a constant term, followed by terms for each of the non-basic variables (assuming the coefficient for that variable is non-zero for that row). Each of the non-basic variables is, of course, assumed to be zero for the tableau in question. The question of whether or not the tableau is optimal, and if not which should be the next basis variable, depends on what happens to the objective function when the non-basic variables are made non-zero. While this information is contained in the usual tableau formulation, it is not intuitively clear to most beginners. A Chvatal dictionary, on the other hand, displays this information clearly to anyone familiar with the most basic school algebra.

Summary
The package SimplexExplorer is as yet a basic and bare-bones, but effective, tool for student study of the simplex algorithm. A routine not discussed here, but included in the package, allows the student to insert and use artificial variables. The code as it is written is not nearly as efficient as it might be. It was written as one of the author's first projects when first beginning to use Visual Basic for Applications in Microsoft Excel. Recent modifications have been restricted to relatively minor issues of debugging and cosmetics, rather than a major rewrite.

References
Chvatal, Vasek (1980) Linear Programming; W. H. Freeman and Company

Dantzig, George B. & Thapa, Mukund N. (1997), Linear Programming 1: Introduction (Springer Series in Operations Research); Springer


Public Const SHTBASENAME As String = "Tab"
Public Const XNAMEDEFAULT As String = "X"
Public Const SNAMEDEFAULT As String = "S"
Public Const ANAMEDEFAULT As String = "A"
Public Const MAXMINDEFAULT As String = "MAX"
Public Const NEWSHTNUMBERDEFAULT As Integer = 0
Public Const NBRDECVARS As Integer = 2
Public Const NBRCONSTRAINTS As Integer = 3
Public Const ORIGINCELLADDRESS As String = "C5" 'must be row 4 & col B or greater

Public m As Integer 'number of constraints
Public s As Integer 'number of slack/surplus variables
Public n As Integer 'number of structural variables
Public a As Integer 'number of artificial variables
Public q As Integer 'number of non-structural variables
Public maxMin As String 'is objective to maximize or minimize
Public xName As String 'Name of decision variable (default is "X")
Public sName As String 'Name of slack variable (default is "S")
Public aName As String 'Name of artificial variable (default is "A")
Public NewShsNumber As Integer 'Number given to worksheet of initial tableau
Public slackOff As Integer 'Number of columns to offset first slack column
Public artOff As Integer 'Number of columns to offset first artificial column
Public PivotColumn As Integer 'Column number, within the tableau, of the pivot
Public PivotRow As Integer 'Row number, within the tableau, of the pivot

Private Sub PivotWithDictionary()
    Call PivotTableau
    Call MakeDictionary
End Sub

Private Sub MakeDictionary()
    Application.ScreenUpdating = False
    'begin to debug starting statements
    'first, must get the size of the tableau whence the dictionary is to come
    'variables for this, (n,m,q etc) are now publically declared above
    Dim row As Integer 'used as a loop counter
    Dim col As Integer 'used as a loop counter
    Dim cel As Integer 'used as a loop counter
    Dim dicteol As Integer 'counts cols in the dictionary range
    Dim sig As String 'plus sign or minus sign
    Dim nbint As Integer 'number of binding constraints
A Visual Basic for Applications Package for Simplex Demonstrations in Linear Programming (Notestine)

Dim aij As Variant  'entry in AN-matrix for ith row, jth column
Dim cj As Variant   'jth reduced cost
Dim TAB_ANMatrix As Range
Dim TAB_ReducedCosts As Range
Dim TAB_RHS As Range
Dim TAB_ZValue As Range
Dim TAB_StructuralLabels As Range
Dim TAB_BasisVariables As Range
Dim TAB_AllVariables As Range
Dim TAB_ColRatios As Range
Dim DIC_Entire As Range
Dim DIC_ZRow As Range
Dim DIC_LHS As Range
Dim DIC_EqualsSigns As Range
Dim DIC_RHS As Range
Dim DIC_NonBasics As Range  'the first row is actually the var side of the objective function
Dim foundcell As Range

Set TAB_ANMatrix = ActiveWorkbook.Names("TAB_ANMatrix").RefersToRange
Set TAB_ReducedCosts = ActiveWorkbook.Names("TAB_ReducedCosts").RefersToRange
Set TAB_StructuralLabels = ActiveWorkbook.Names("TAB_StructuralLabels").RefersToRange
Set TAB_RHS = ActiveWorkbook.Names("TAB_RHS").RefersToRange
Set TAB_ZValue = ActiveWorkbook.Names("TAB_ZValue").RefersToRange
Set TAB_BasisVariables = ActiveWorkbook.Names("TAB_BasisVariables").RefersToRange
Set TAB_AllVariables = ActiveWorkbook.Names("TAB_AllVariables").RefersToRange
Set TAB_ColRatios = ActiveWorkbook.Names("TAB_ColRatios").RefersToRange

n = TAB_ANMatrix.Rows.Count  'number of constraints
m = TAB_StructuralLabels.Cells.Count  'number of structural variables
q = TAB_AllVariables.Columns.Count - n  'number of non-structural variables (sl+su+ar)

Set DIC_Entire = TAB_ColRatios.Cells(1).Offset(-1, 2).Resize(n + 1, 3 + 3 * m)
Set DIC_ZRow = TAB_ColRatios.Cells(1).Offset(-1, 2).Resize(1, 3 + 3 * m)
Set DIC_LHS = TAB_ColRatios.Cells(1).Offset(-1, 2).Resize(n + 1, 1)
Set DIC_EqualsSigns = TAB_ColRatios.Cells(1).Offset(-1, 3).Resize(n + 1, 1)
Set DIC_RHS = TAB_ColRatios.Cells(1).Offset(-1, 4).Resize(n + 1, 1)
Set DIC_NonBasics = TAB_ColRatios.Cells(1).Offset(-1, 5).Resize(n + 1, 3 * m)

For cel = 1 To DIC_Entire.Cells.Count
    DIC_Entire.Cells(cel).Value = ""
Next cel

With DIC_Entire
    .ColumnWidth = 3
    .Borders.Color = vbBlack
    .Borders(xlInsideVertical).LineStyle = xlNone
    .Borders(xlInsideHorizontal).LineStyle = xlNone
    .HorizontalAlignment = xlCenter
    .NumberFormat = "General"
End With

With DIC_Entire.Resize(1, 3 + 3 * m)
    .Borders.Color = vbBlack
End With
With DIC_EqualSigns
  .ColumnWidth = 1
  .HorizontalAlignment = xlCenter
  .NumberFormat = "General"
End With
DIC_LHS.Cells(1).Value = "Z"
DIC_EqualSigns.Cells(1).Value = "="
DIC_RHS.Cells(1).Value = TAB_ZValue.Value
For row = 1 To n
  DIC_LHS.Cells(row + 1).Value = TAB_BasisVariables.Cells(row).Value
  DIC_EqualSigns.Cells(row + 1).Value = "="
  DIC_RHS.Cells(row + 1).Value = TAB_RHS.Cells(row).Value
Next row

dictcol = 1
For col = 1 To n + m
  Set foundcell = TAB_BasisVariables.Find(TAB_AllVariables.Cells(col))
  'if the var is basic, skip it (and, if we find it, it is basic)
  If foundcell Is Nothing Then
    'the variable is non-basic: put into rhs
    cj = -TAB_ReducedCosts.Cells(col).Value
    If cj <> 0 Then
      'put into rhs only if it is non-zero
      'About sig: Now am using minus sign automatically displayed for neg numbers
      'For a plus sign, we must put it in ourselves. The results in + being
      'displayed in a different col from the minus sign. We could clean this up
      'by displaying abs(cj) and generating sign in the sign col.
      If cj < 0 Then sig = "-"; Else sig = "+";
      "010711: CHECK THIS... MINUS SIGN???
      With DIC_NonBasis
        .Cells(1, dictcol).Value = sig
        .Cells(1, dictcol).ColumnWidth = 1
        'sign width
        .Cells(1, dictcol + 1).Value = cj
        'if neg, shows with minus sign
        .Cells(1, dictcol + 1).ColumnWidth = 4
        'enuf to show nbr
        .Cells(1, dictcol + 1).HorizontalAlignment = xlRight
        'snug to name
        .Cells(1, dictcol + 2).Value =_
        'var name
        .Cells(row + 1, dictcol + 2).HorizontalAlignment = xlLeft
        'snug to coeff
      End With
    End If
  End If
Next col

dictcol = dictcol + 3
A Visual Basic for Applications Package for Simplex Demonstrations in Linear Programming (Note: line 1)

If aij <> 0 Then 'put into rhs only if it is non-zero
If aij < 0 Then sig = "-" Else sig = "+
With DIC_NonBasics
.Cells(row + 1, dictcol).Value = sig 'sign: +/-
.Cells(row + 1, dictcol).ColumnWidth = 1 'sign width
.Cells(row + 1, dictcol + 1).Value = aij 'abs value
.Cells(row + 1, dictcol + 1).ColumnWidth = 4 'enuf to show nbr
.Cells(row + 1, dictcol + 2).HorizontalAlignment = xlRight 'snug to name
.Cells(row + 1, dictcol + 2).Value = _
    TAB_AllVariables.Cells(col).Value 'var name
.Cells(row + 1, dictcol + 2).HorizontalAlignment = xlLeft 'snug to coeff
End With
End If
Next row
dictcol = dictcol + 3
End If
Next col
Application.ScreenUpdating = True
End Sub

Private Sub CreateTableauWithSlacks()
On Error GoTo errTrap
TSLForm.Show
Dim newSheetName As String 'name the worksheet BEGIN
newSheetName = SHTBASENAME & "." & NewShlNumber
If WorksheetNameQ(newSheetName) Then
    Application.DisplayAlerts = False
    Worksheets(newSheetName).Delete 'DELETE THE EXISTING WORKSHEET
    Application.DisplayAlerts = True
End If
Worksheets.Add
ActiveSheet.Name = newSheetName 'name the worksheet END
'DEFAULT VALUES, CONSTANTS
' Const rowStart As Integer = 3, colStart As Integer = 2 'not used now
Dim originCell As Range
Set originCell = Range(ORIGINCELLADDRESS)
'end DEFAULT VALUES, CONSTANTS
'Dimension the Ranges
Dim TAB_ShowSizes As Range 'gives number of rows and cols in table
Dim TAB_NumRows As Range 'cell 2 of TAB_ShowSizes
Dim TAB_NumCols As Range 'cell 3 of TAB_ShowSizes
'ranges for the labels along top and down left side
Dim TAB_AllVariables As Range 'variable names across top (union of next two ranges)
Dim TAB_StructuralLabels As Range 'structural variable namestop
Dim TAB_SlackLabels As Range 'slack variable names
Dim TAB_BasisVariables As Range 'basic variable names? constraining names?
Dim TAB_ColNumbers As Range 'start with 1 (no z col); for user to pivot on
Dim TAB_RowNumbers As Range  'start with row 1; for user to pivot on
Dim TAB_RowZero As Range    'row 0 for reduced cost row
Dim TAB_ZLabel As Range     'current z value (single cell at upper left corner)
Dim TAB_MaxMinLabel As Range  '"Max" or "Min", put into origin cell (011216)
Dim TAB_Entire As Range    'does this include the header above?
Dim TAB_ReducedCosts As Range    'current z-coeffs
Dim TAB_ReducedStructuralCosts As Range
Dim TAB_ReducedSlackCosts As Range
Dim TAB_ZValue As Range      'current z value
Dim TAB_ANMatrix As Range
Dim TAB_RHS As Range
Dim TAB_ZRowEntire As Range  'includes reduced cost and current z value
Dim TAB_ANandRHSRowsEntire As Range  'coefficient matrix and right hand side (b values)
Dim TAB_ColRatios As Range
Dim TAB_ColRatiosLabel As Range
Dim TAB_RowRatios As Range
Dim TAB_RowRatiosLabel As Range

'set the origin and define the ranges**********DEFINE RANGES **********BEGIN*****
Set TAB_ShowSizes = originCell.Offset(-3, 2).Resize(1, 4)
Set TAB_NumRows = TAB_ShowSizes.Cells(2)
Set TAB_NumCols = TAB_ShowSizes.Cells(3)
Set TAB_AllVariables = originCell.Offset(0, 1).Resize(1, n + m)  'Structural + slack
Set TAB_StructuralLabels = originCell.Offset(0, 1).Resize(1, n) 'Structural
Set TAB_SlackLabels = originCell.Offset(0, n + 1).Resize(1, m) 'Slack
Set TAB_BasisVariables = originCell.Offset(2, 0).Resize(m, 1)
Set TAB_ColNumbers = originCell.Offset(-1, 1).Resize(1, n + m) 'Structural + slack
Set TAB_RowNumbers = originCell.Offset(2, -1).Resize(m, 1) 'does not include reduced cost row
Set TAB_RowZero = originCell.Offset(1, -1).Resize(1, 1) 'reduced cost row (row 0)
Set TAB_ReducedCosts = originCell.Offset(1, 1).Resize(1, n + m)
Set TAB_ReducedStructuralCosts = originCell.Offset(1, 1).Resize(1, n)
Set TAB_ReducedSlackCosts = originCell.Offset(1, n + 1).Resize(1, m)
Set TAB_ZValue = originCell.Offset(1, n + m + 1).Resize(1, 1)
Set TAB_ZLabel = originCell.Offset(1, 0).Resize(1, 1)
Set TAB_MaxMinLabel = originCell.Offset(0, 0).Resize(1, 1)
Set TAB_ANMatrix = originCell.Offset(2, 1).Resize(m, n + m)
Set TAB_RHS = originCell.Offset(2, n + m + 1).Resize(m, 1)
Set TAB_Entire = Union(TAB_ReducedCosts, TAB_ZValue, TAB_ANMatrix, TAB_RHS)
Set TAB_ZRowEntire = Union(TAB_ReducedCosts, TAB_ZValue)
Set TAB_ANandRHSRowsEntire = Union(TAB_ANMatrix, TAB_RHS)
Set TAB_ColRatios = originCell.Offset(2, n + m + 3).Resize(m, 1)
Set TAB_RowRatios = originCell.Offset(m + 3, 1).Resize(1, n + m)
Set TAB_ColRatiosLabel = originCell.Offset(1, n + m + 3).Resize(1, 1)
Set TAB_RowRatiosLabel = originCell.Offset(m + 3, 0).Resize(1, 1)

'******************************************************************************
' Define the WORKBOOK RANGES**********DEFINE WORKBOOK RANGES **********BEGIN****
ActiveWorkbook.Names.Add Name:="TAB_ShowSizes", RefersTo1C1Local:=TAB_ShowSizes
ActiveWorkbook.Names.Add Name:="TAB_NumRows", RefersTo1C1Local:=TAB_NumRows
A Visual Basic for Applications Package for Simple Simplex Demonstrations in Linear Programming (Notestine)

ActiveWorkbook.Names.Add Name:="TAB_NumCols", RefersToRCIC1Local:=TAB_NumCols
ActiveWorkbook.Names.Add Name:="TAB_Entire", RefersToRCIC1Local:=TAB_Entire
ActiveWorkbook.Names.Add Name:="TAB_AllVariables", RefersToRCIC1Local:=TAB_AllVariables
ActiveWorkbook.Names.Add Name:="TAB_ColNumbers", RefersToRCIC1Local:=TAB_ColNumbers
ActiveWorkbook.Names.Add Name:="TAB_StructuralLabels", RefersToRCIC1Local:=TAB_StructuralLabels
ActiveWorkbook.Names.Add Name:="TAB_SlackLabels", RefersToRCIC1Local:=TAB_SlackLabels
ActiveWorkbook.Names.Add Name:="TAB_BasisVariables", RefersToRCIC1Local:=TAB_BasisVariables
ActiveWorkbook.Names.Add Name:="TAB_RowNumbers", RefersToRCIC1Local:=TAB_RowNumbers
ActiveWorkbook.Names.Add Name:="TAB_BasisVariables", RefersToRCIC1Local:=TAB_BasisVariables
ActiveWorkbook.Names.Add Name:="TAB_reducedCosts", RefersToRCIC1Local:=TAB_reducedCosts
ActiveWorkbook.Names.Add Name:="TAB_reducedStructuralCosts", RefersToRCIC1Local:=TAB_reducedStructuralCosts
ActiveWorkbook.Names.Add Name:="TAB_reducedSlackCosts", RefersToRCIC1Local:=TAB_reducedSlackCosts

' put some text in ************ ARRAY DATA **********BEGIN ARRAY *****

Dim i As Integer
TAB_ZValue.Formula = _
   "=IF(ISERR(MMULT(TAB_reducedSlackCosts,TAB_rhs)),""",""MMULT(TAB_reducedSlackCosts,TAB_rhs))""
TAB_MaxMinLabel.Value = maxMin "MAX" or "MIN"
If maxMin = "MAX" Then
   TAB_ZLabel.Value = ";"Z;"
Else
   TAB_ZLabel.Value = ";-Z;"
End If
For i = 1 To n
   TAB_AllVariables.Cells(i).Value = xName & i 'first n cols are structural
   TAB_ColNumbers.Cells(i).Value = i 'numbers 1 to n
Next i
For i = 1 To m
   TAB_AllVariables.Cells(n + i).Value = sName & (i + slackOff) 'next m cols are slack/surplus
   TAB_ColNumbers.Cells(n + i).Value = n + i 'numbers continue up to n+m
Next i
Dim j As Integer
TAB_RowZero.Cells(1).Value = 0 'reduced cost row, which is row 0
For j = 1 To m
   TAB_BasisVariables.Cells(j).Value = sName & (j + slackOff)
   TAB_RowNumbers.Cells(j).Value = j 'rows of coefficient matrix start with row 1
Next j
With TAB_ShowSizes
This cell is also range TAB_NumRows
This cell is also range TAB_NumCols

'end putting text in************* ARRAY DATA *************END ARRAY ****** END
' put in the FORMATS*************FORMATS*************BEGIN FORMATS ******
With TAB_ZLabel
  .Borders.Color = vbBlue
  .HorizontalAlignment = xICenter
End With
With TAB_ZValue
  .Borders.Color = vbBlue
  .HorizontalAlignment = xICenter
End With
With TAB_RHS
  .ColumnWidth = 5
  .Borders.Color = vbRed
  .HorizontalAlignment = xICenter
  .Borders(xlInsideHorizontal).LineStyle = xlDash
  .NumberFormat = "General"
End With
With Union(TAB_AllVariables, TAB_ColNumbers)
  .Borders.Color = vbGreen
  .Borders(xlInsideVertical).LineStyle = xlDash
  .HorizontalAlignment = xICenter
End With
With Union(TAB_RowNumbers, TAB_RowZero)
  .ColumnWidth = 3
  .Borders.Color = vbGreen
  .Borders(xlInsideHorizontal).LineStyle = xlDash
  .HorizontalAlignment = xICenter
End With
With TAB_BasisVariables
  .ColumnWidth = 5
  .Borders.Color = vbGreen
  .Borders(xlInsideHorizontal).LineStyle = xlDash
  .HorizontalAlignment = xICenter
End With
With TAB_ReducedCosts
  .Borders.Color = vbBlue
A Visual Basic for Applications Package for Simplex Demonstrations in Linear Programming (Notestine)

.Borders(xlInsideVertical).LineStyle = xlDash
.Borders(xlInsideHorizontal).LineStyle = xlDash
.HorizontalAlignment = xlRight
.NumberFormat = "General"
End With

With TAB_ANMatrix
 .ColumnWidth = 3
 .Borders.Color = vbBlack
 .Borders(xlInsideHorizontal).LineStyle = xlDash
 .Borders(xlInsideVertical).LineStyle = xlDash
 .HorizontalAlignment = xlRight
 .NumberFormat = "General"
End With

With TAB_ColRatios
 .ColumnWidth = 5
 .Borders.Color = vbGreen
 .Borders(xlInsideHorizontal).LineStyle = xlDash
 .HorizontalAlignment = xlCenter
 .NumberFormat = "General"
End With

With TAB_RowRatios
 .Borders.Color = vbGreen
 .Borders(xlInsideVertical).LineStyle = xlDash
 .HorizontalAlignment = xlCenter
 .NumberFormat = "General"
End With

With Union(TAB_ColRatiosLabel, TAB_RowRatiosLabel)
 .Borders.Color = vbGreen
 .HorizontalAlignment = xlCenter
End With

For i = 1 To originCell.Column - 1
  ActiveSheet.Columns(i).ColumnWidth = 3
Next i

ActiveSheet.Columns(TAB_ColRatios.Column - 1).ColumnWidth = 1
ActiveSheet.Rows(TAB_RowRatios.row - 1).RowHeight = 8
Union(TAB_ShowSizes, TAB_AllVariables, TAB_ANMatrix, TAB_RowRatios).RowHeight = 13

' ***************************************************************
' FORMATS**********************************************************
' ***************************************************************
' shade every few row&col**********************************************************************

Dim tempRng As Range
Const shadeGap As Integer = 3
Const shadeColorIndex As Integer = 36   ' a light tan
Set tempRng = Union(TAB_RowNumbers, TAB_BasisVariables, TAB_ANMatrix, TAB_RHS)
For i = shadeGap To tempRng.Rows.Count Step shadeGap
  tempRng.Rows(i).Interior.ColorIndex = shadeColorIndex
Next i

Set tempRng = Union(TAB_ColNumbers, TAB_AllVariables, TAB_ReducedCosts, TAB_ANMatrix)
For i = shadeGap To tempRng.Columns.Count Step shadeGap
  tempRng.Columns(i).Interior.ColorIndex = shadeColorIndex
Next i
For i = shadeGap To TAB_ColRatios.Rows.Count Step shadeGap
    TAB_ColRatios.Rows(i).Interior.ColorIndex = shadeColorIndex
Next i
For i = shadeGap To TAB_RowRatios.Columns.Count Step shadeGap
    TAB_RowRatios.Columns(i).Interior.ColorIndex = shadeColorIndex
Next i
With TAB_ANMatrix
    .NumberFormat = "#,##0.0e+000"
    With .Borders(xlInsideVertical)
        .LineStyle = xlContinuous
        .Weight = xlHairline
        .ColorIndex = xlAutomatic
    End With
    With .Borders(xlInsideHorizontal)
        .LineStyle = xlContinuous
        .Weight = xlHairline
        .ColorIndex = xlAutomatic
    End With
End With
With Union(TAB_BasisVariables, TAB_RHS)
    With .Borders(xlInsideHorizontal)
        .LineStyle = xlContinuous
        .Weight = xlHairline
        .ColorIndex = xlAutomatic
    End With
End With
With Union(TAB_AllVariables, TAB_ReducedCosts)
    With .Borders(xlInsideVertical)
        .LineStyle = xlContinuous
        .Weight = xlHairline
        .ColorIndex = xlAutomatic
    End With
End With
End With
' ******************************************SHADING***************END SHADING END
TAB_Entire.NumberFormat = "General"
TAB_ColRatios.NumberFormat = "General"
TAB_RowRatios.NumberFormat = "General"
Exit Sub
' ******************************************ERRTRAP**************BEGIN ERRTRAP *****
errTrap:
   'say good bye, and exit the subroutine
   MsgBox Title:="Good Bye", prompt:="Cancelling Operation", Buttons:=48
End Sub' ****************************************** END CreateTableauWithSlacks END

Private Sub ClearSheet() 
    Dim proceed As VbMsgBoxResult
proceed = MsgBox( _
   "Do you really want to delete all data and borders on this worksheet?" & _
   vbCrLf & "(Yes to delete; No to leave the worksheet unchanged.)", _
   vbYesNo, "Clear Worksheet?")

If proceed = vbNo Then End 'else continue: it is okay to clear the worksheet

With Cells
   .Interior.ColorIndex = xlNone
   .Borders.LineStyle = xlNone
   .Clear
End With

End Sub '***************************************************************************** END SheetNoColor END

BEGIN

Private Sub colRatios() '***************************************************************************** ColRatios ***** BEGIN

On Error GoTo ErrorHandler
Dim col As Integer
col = InputBox("Ratios for which column (integer)")
PutColumnRatios (col)
Exit Sub

ErrorHandler:
   Debug.Print "In ColRatios, InputBox returned col = " & col
   MsgBox "Column number must be a positive integer."
End

End Sub '***************************************************************************** ColRatios ***** END

BEGIN

Private Sub PutColumnRatios(col As Integer) '***************************************************************************** PutColumnRatios ***** BEG

Dim row As Integer
Dim colRatiosRange As Range
Dim anMatrixRange As Range
Dim rhsRange As Range
On Error GoTo ErrorHandler

Set colRatiosRange = ActiveWorkbook.Names("TAB_ColRatios").RefersToRange
Set anMatrixRange = ActiveWorkbook.Names("TAB_ANMatrix").RefersToRange
Set rhsRange = ActiveWorkbook.Names("TAB_RHS").RefersToRange

For row = 1 To rhsRange.Rows.Count
   If anMatrixRange.Cells(row, col).Value <> 0 Then
   Else
      colRatiosRange.Cells(row).Value = "."
   End If
Next row
Exit Sub

ErrorHandler:
   Debug.Print "In PutColumnRatios, anMatrixRange.Columns.Count = " & _

"
MsgBox "Column number must be a positive integer between 1 and ", & MatrixRange.Columns.Count & "," End

Private Sub rowRatios() '*************** RowRatios ***** BEGIN
Dim row As Integer
On Error GoTo ErrorHandler
row = InputBox("Ratios for which row (integer)")
PutRowRatios row
Exit Sub
ErrorHandler:
Debug.Print "In rowRatios, InputBox returned row = ", & row
MsgBox "Row number must be a positive integer."
End

Private Sub PutRowRatios(row As Integer) '*************** PutRowRatios ***** BEGIN
Dim col As Integer
Dim basisSearch As Range
'range that holds name of current variable (column)
'our ranges: define the names and connect to ranges in tableau
Dim rowRatiosRange As Range 'we will put the ratios into this range
Dim anMatrixRange As Range 'coefficient matrix, denominators from here (non-zero only)
Dim reducedCostRange As Range 'numerator from here
Dim allVarNames As Range 'use to ensure variable is non-basic
Dim basicVarNames As Range 'use to ensure variable is non-basic
Set rowRatiosRange = ActiveWorkbook.Names("TAB_rowRatios").RefersToRange
Set anMatrixRange = ActiveWorkbook.Names("TAB_ANMatrix").RefersToRange
Set reducedCostRange = ActiveWorkbook.Names("TAB_ReducedCosts").RefersToRange
Set allVarNames = ActiveWorkbook.Names("TAB_AllVariables").RefersToRange
Set basicVarNames = ActiveWorkbook.Names("TAB_BasisVariables").RefersToRange
On Error GoTo ErrorHandler
For col = 1 To reducedCostRange.Columns.Count
 'Basic column? If so, ignore it
 Set basisSearch = basicVarNames.Find what:allVarNames.Cells(col).Value
 If basisSearch Is Nothing Then 'not basic, investigate further
   If anMatrixRange.Cells(row, col).Value <> 0 Then 'calculate ratio (if denominator not zero)
   Else 'denominator is zero, no ratio defined
     rowRatiosRange.Cells(col).Value = "."
   End If
 Else 'variable is basic, no ratio
   rowRatiosRange.Cells(col).Value = "."
 End If
Next col
Exit Sub
ErrorHandler:
Private Sub ShadeBasicColumns(Optional shadeColorIndex As Integer = 36)
    Call ClearShading
    Dim mainRange As Range
    Dim colNumbers As Range
    Dim colLabels As Range
    Dim rowLabels As Range
    Dim reducedCosts As Range
    Dim aMatrix As Range
    Dim rowRatios As Range
    Set colNumbers = ActiveWorkbook.Names("TAB_ColNumbers").ReferToRange
    Set colLabels = ActiveWorkbook.Names("TAB_AllVariables").ReferToRange
    Set rowLabels = ActiveWorkbook.Names("TAB_BasisVariables").ReferToRange
    Set aMatrix = ActiveWorkbook.Names("TAB_ANMatrix").ReferToRange
    Set reducedCosts = ActiveWorkbook.Names("TAB_ReducedCosts").ReferToRange
    ' Combine the main area to have vertical stripe, plus the ratios row (not contiguous)
    Set mainRange = Union(colNumbers, colLabels, reducedCosts, aMatrix)
    Set rowRatios = ActiveWorkbook.Names("TAB_RowRatios").ReferToRange
    ' Now we march thru the column labels and check each against the basic variables
    Dim col As Integer, numCols As Integer
    Dim basisSearch As Range
    numCols = colLabels.Columns.Count
    For col = 1 To numCols
        ' Basic column? If so, shade it
        If Not basisSearch Is Nothing Then
            mainRange.Columns(col).Interior.ColorIndex = shadeColorIndex
            rowRatios.Columns(col).Interior.ColorIndex = shadeColorIndex
        End If
        Next col
    End Sub
    '******************************************** ShadeBasicColumns ****** End
' End ******

Private Sub ClearShading()
    Call SetShadingToColorIndex
End Sub

Private Sub PivotTableau()
    m = ActiveSheet.Names("TAB_RHS").ReferToRange.Rows.Count

CHUKYO KEIEIKENKYU  Vol.12 No.2 / 2003. 2

n = ActiveSheet.Names("TAB_StructuralLabels").RefersToRange.Columns.Count
s = ActiveSheet.Names("TAB_SlackLabels").RefersToRange.Columns.Count
Call PivotTableauRC(PivotRow, PivotColumn)
Call ShadeBasicColumnsLightTan
Exit Sub
errTrap:
  MsgBox Title:="Good Bye", prompt:="Cancelling Operation", Buttons:=48
End Sub

Private Sub PivotTableauRC(pivCol As Integer)
  On Error GoTo errTrap
  Dim pivRow As Integer
  Dim minRatio As Variant
  Dim msganswer As Integer
  Call PutColumnRatios(pivCol) 'puts ratios for column pivCol into range colRatios
  minRatio = MinimumInRange(colRatios)
  If minRatio = 0 Then 'exit if no suitable ratio (no pivot row)
    msganswer = MsgBox(prompt:="Could not find a row to pivot on!! (No positive entry in colRatios range)", 
                     Buttons:=vbOKOnly)
    Exit Sub
  End If
  pivRow = colRatio.Find(minRatio).row
  Call PivotTableauRC(pivRow, pivCol)
  Exit Sub
errTrap:
  MsgBox Title:="Good Bye", prompt:="Cancelling Operation", Buttons:=48
End Sub

Private Sub PivotTableauRC(pivRow As Integer, pivCol As Integer)
Const maxColumns As Long = 500
Application.ScreenUpdating = False
  Dim newSheetName As String 'will be give to current active sheet & pivoted
  Dim oldSheetName As String 'will be given to new sheet showing pre-pivot state
  Dim baseName As String = "Tab" 'Sheets are Tab0, Tab1, Tab2 etc
  oldSheetName = ActiveSheet.Name
  newSheetName = baseName & "." & str(1 + ReverseExtractElement(oldSheetName, 1, "."))
  ActiveSheet.Name = newSheetName
  ActiveSheet.Name = newSheetName 'Range names here
  Sheets.Add before:=ActiveSheet
  ActiveSheet.Name = oldSheetName 'give the newname to the new sheet instead
  Sheets(newSheetName).Activate 'holds pre-pivot tableau & range names
  Call SelectUsedRange
  Dim usedRange As Range 'the next few lines get the column widths
  Dim colWidths(maxColumns) As Variant
  Dim colWidthCtr As Long
  Set usedRange = Selection
  For colWidthCtr = 1 To usedRange.Columns.Count
    ' Publishers: If you prefer a different format, let us know. We've included comments for clarity.
  Next colWidthCtr

A Visual Basic for Applications Package for Simplex Demonstrations in Linear Programming (Note: only)

\[ \text{colWidths(colWidthCtr)} = \text{usedRange.Columns(colWidthCtr).ColumnWidth} \]

Next colWidthCtr ' done getting column widths, will use them below

Selection.Copy

Sheets(oldSheetName).Activate ' sheet to hold old tableau (no range names here)
ActiveSheet.Range("A1").Select ' must past from upper left corner
ActiveSheet.Paste ' old tableau

For colWidthCtr = 1 To usedRange.Columns.Count

ActiveSheet.Columns(colWidthCtr).ColumnWidth = colWidths(colWidthCtr)

Next colWidthCtr ' done setting column widths in sheet to hold old tableau

ActiveSheet.Range("A1").Select
Sheets(newSheetName).Activate ' sheet with range names & tableau still old, but ready to pivot

Dim pivotElement As Variant ' now start the pivot
Dim reducedCosts As Range
Dim zValue As Range
Dim aMatrix As Range
Dim rhsValues As Range

Dim numRows As Integer, numCols As Integer ' sizes of range pivotingRange
Dim allVariables As Range ' column labels, label in pivot column enters basis
Dim basisVariables As Range ' row labels, pivot row becomes label in pivot column
Dim colRatios As Range ' to be cleared of contents
Dim rowRatios As Range ' to be cleared of contents

Set basisVariables = ActiveWorkbook.Names("TAB_BasisVariables").RefersToRange
Set allVariables = ActiveWorkbook.Names("TAB_AllVariables").RefersToRange
Set reducedCosts = ActiveWorkbook.Names("TAB_REducedCosts").RefersToRange
Set zValue = ActiveWorkbook.Names("TAB_ZValue").RefersToRange
Set aMatrix = ActiveWorkbook.Names("TAB_ANMatrix").RefersToRange
Set rhsValues = ActiveWorkbook.Names("TAB_RHS").RefersToRange
Set colRatios = ActiveWorkbook.Names("TAB_colRatios").RefersToRange
Set rowRatios = ActiveWorkbook.Names("TAB_rowRatios").RefersToRange

numRows = aMatrix.Rows.Count ' number of constraints (excludes reduced cost row)
numCols = aMatrix.Columns.Count ' number of variables in all (excludes rhs column)

If aMatrix(pivRow, pivCol) = 0 Then 'ERROR CONTROL: is pivot element = 0? if so, abort

Dim ans As Integer

ans = MsgBox(prompt:="Pivot element is 0.", Buttons:=vbOKOnly)

GoTo errTrap
End If ' END ERROR CONTROL

pivotElement = aMatrix(pivRow, pivCol) ' Assign values to the local variables for the pivot
Dim row As Integer, col As Integer

Dim pivotFactor As Variant

For row = 1 To numRows

If row = pivRow Then GoTo skipPivotRow

rhsValues(row) = rhsValues(row) - (aMatrix(row, pivCol) / pivotElement) * rhsValues(pivRow)

pivotFactor = (aMatrix(row, pivCol) / pivotElement)

For col = 1 To numCols

aMatrix(row, col) = aMatrix(row, col) - pivotFactor * aMatrix(pivRow, col)

Next col

skipPivotRow:
Next row
pivotFactor = (reducedCosts(pivCol) / pivotElement)
For col = 1 To numCols
    reducedCosts(col) = reducedCosts(col) - pivotFactor * aMatrix(pivRow, col)
Next col
zValue = zValue - pivotFactor * rhsValues(pivRow)
For col = 1 To numCols
    aMatrix(pivRow, col) = aMatrix(pivRow, col) / pivotElement
Next col
rhsValues(pivRow) = rhsValues(pivRow) / pivotElement 'new value
basisVariables(pivRow) = allVariables(pivCol) 'Update the basis list and clear out the ratios
colRatios.ClearContents
rowRatios.ClearContents
ActiveSheet.Range("A1").Select
Exit Sub
errTrap:
'say good bye, and exit the subroutine
MsgBox Title:"Good Bye", prompt:"Cancelling Operation", Buttons:=48
End Sub

Private Function WorksheetNameQ(theName As String) As Boolean
    '******BEGIN WorksheetNameQ BEGIN
    Dim i As Integer
    WorksheetNameQ = False
    For i = 1 To ActiveWorkbook.Worksheets.Count
        If ActiveWorkbook.Worksheets(i).Name = theName Then WorksheetNameQ = True
    Next i
    'if not set to True inside the loop, then WorksheetNameQ is still False
    End Function

Private Function MinimumInRange(inRange As Range) As Variant
    'We only look for positive minimums. Ignore zero, negative, non-numeric entries
    Dim curItem As Variant
    Dim numItems As Integer
    Dim isMinQ As Boolean
    Dim i As Integer
    MinimumInRange = 0
    isMinQ = False
    numItems = inRange.Cells.Count
    i = 1
    Do While i <= numItems 'this is to find the first numeric entry
        curItem = inRange.Cells(i)
        If IsNumeric(curItem) Then
            MinimumInRange = curItem 'at least one numeric entry greater than zero
            isMinQ = True 'we have a minimum, this will not change
            Exit Do 'continue with the next do loop, using same i
        End If
    End Do
A Visual Basic for Applications Package for Simple Simplex Demonstrations in Linear Programming (Note: original text is missing)

End If
    i = i + 1  'if we are here, then have not yet found number greater than zero
Loop
i = i + 1  'continue with entry after first numeric one
Do While i <= numItems 'i has value corresponding to space after first numeric entry
curItem = inRange.Cells(i)
If IsNumeric(curItem) Then
    If curItem > 0 Then
        If curItem < MinimumInRange Then
            MinimumInRange = curItem
        End If
    End If
End If
End If
End If
    i = i + 1  'in this loop, we continue checking until the end of the range
Loop

End Function

Private Sub PasteObjFunction()
    Dim rge As Range
    Set rge = ActiveWorkbook.Names("TAB_ZValue").ReferToRange
    On Error GoTo errTrap
    rge.Formula = _
        "=IF(ISERR(MMULT(TAB_ReducedSlackCosts,TAB_RHS)),""","" & _
        "MMULT(TAB_ReducedSlackCosts,TAB_RHS))"
    Exit Sub
errTrap:
    Debug.Print "In PasteObjFunction, TAB_ZValue.address = " & rge.Address
    MsgBox "A problem has occurred in trying to paste the formula to the cell " & _
    rge.Address & ", The operation has been cancelled." , vbOKOnly, "Trouble in ""PasteObjFunction"
End Sub

Private Sub SelectUsedRange()
    Dim lRealLastRow As Long
    Dim lRealLastCol As Long
    Range("A1").Select
    On Error Resume Next
    lRealLastRow = Cells.Find("*", Range("A1"), xlFormulas, , xlByRows, xlPrevious).row
    lRealLastCol = Cells.Find("*", Range("A1"), xlFormulas, , xlByColumns, xlPrevious).Column
    ActiveSheet.Range("A1").Resize(lRealLastRow, lRealLastCol).Select
End Sub