College of Engineering and Computer Science
Mechanical Engineering Department

California State University Northridge

Mechanical Engineering 309
Numerical Analysis of Engineering Systems

Spring 2014 15237 Instructor: Larry Caretto

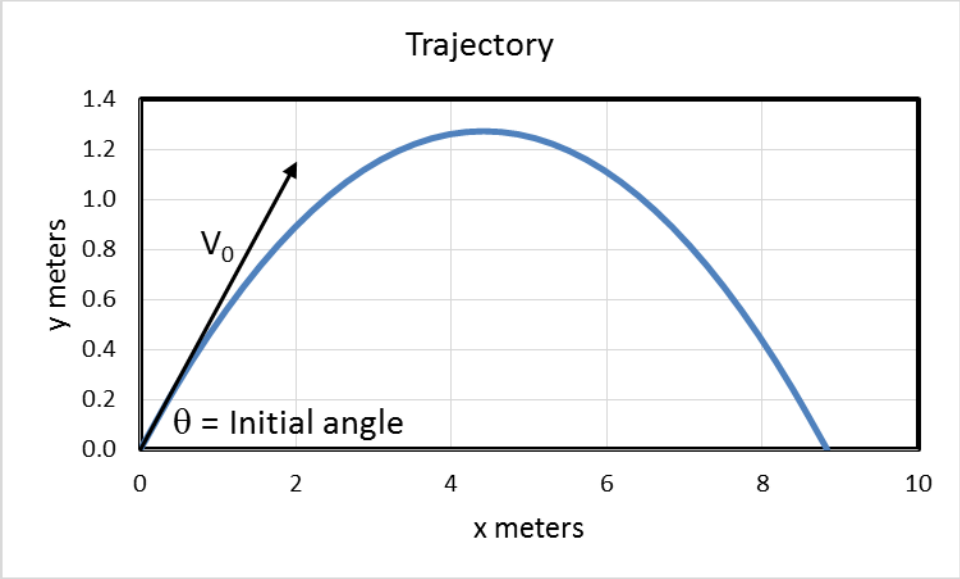
First Programming Assignment – Due Monday, February 3, 11:59 pm[1]

Objective

This assignment is designed to review the various ways to use Excel spreadsheets and the Visual Basic Language. In particular, you should learn the general approach to entering formulas in a spreadsheet and to using the Visual Basic editor to create and run user-defined functions (UDF) and macros. This exercise uses the formula for a trajectory from your introductory physics course.

Background

The trajectory of a frictionless projectile that is released from ground level (y = 0) with an initial speed $|\mathbf{V}_0|$, at an initial angle, θ, to the ground has a trajectory (vertical distance y as a function of horizontal distance x), which is shown in the figure at the right; x has the initial value x = 0 at t = 0. The trajectory can be computed from the following equations:



$$y = (V_0 \sin\theta)t - \frac{gt^2}{2} \qquad \text{and} \qquad x = (V_0 \cos\theta)t \qquad [1]$$

We can solve the y equation to get two time values at which y = 0. The first is the initial condition that y = 0 at t = 0, the initial release time. The second point where y = 0 is the point where the projectile returns to ground level at the maximum time, t_{max}. Setting y = 0 and solving for $t = t_{max}$ (nonzero root) gives

$$t_{max} = \frac{2V_0 \sin\theta}{g} \qquad [2]$$

You will use equations [1] and [2] in the Excel/VBA calculations for this assignment.

Exercises

The following exercises use a variety of approaches to computing the trajectory with Excel spreadsheets and VBA. Each exercise should be done on a separate worksheet in the same workbook. All VBA code

[1] It may be submitted by 11:59 pm, Wednesday, February 5, with a 30% penalty. No later submissions accepted.

Jacaranda Hall 3314 | Mail Code 8348 | Phone: N/A
Email: lcaretto@csun.edu | Fax: 818.677.7062

can be placed in a single module. Make sure that you save the workbook as a macro-enabled (*.xlsm) file.

First exercise. Set up a worksheet that has the following input data in sequential rows. Place a description of the data in column A, the value of the data in column B, and the units in column C. For initial data use V_0 = 10 m/s, θ = 45^o, and 20 time steps.

- Initial velocity, V_0 in m/s.
- Initial angle, θ, in degrees[2]
- Gravitational acceleration, g = 9.80665 m^2/s
- The number of time steps used to generate the table, dimensionless

After you complete the input data, place the calculation results listed below in two rows following the data. Continue the pattern of values in column B, description in column A, and units in column C.

- The maximum time computed from equation [2] in s.
- The time step, Δt, equal to the maximum time divided by the number of time steps.

Create a table with three columns: time in seconds, x in meters and y in meters. Label each column. Start with time = 0 and increment the time by the time step, Δt, until you end with time = t_{max}.in the time column. Use a cell formula to compute the values of x and y for the initial row; then copy the formulas down for all time values. Do you get the correct values of x and y for the maximum time? Do you get the correct maximum y value?

Name the worksheet tab **Cells**.

Second exercise. In this exercise you will modify the first worksheet to use functions (UDFs or user-defined functions). Create a new worksheet from the worksheet for the first exercise and give it the name **simple UDF**.[3] Replace the cell calculation for the maximum time with a user defined function. (The inputs to this function should be the initial velocity and the initial angle in degrees.) Replace the cell formulas for the computation of x and y in each cell with the user defined functions to compute these values. The inputs to these UDF's should be the time, the initial velocity and the initial angle in degrees. Both your UDFs should use a defined constant for g and handle the conversion of the angle from degrees to radians. Use an `Option Explicit` statement at the top of your module to force variable typing.

The initial statements for the VBA code that you have to write are shown below. (This includes the complete coding of the function used to compute the maximum time.) You can use this code directly in your exercise and you can use this function as an example to write the functions to compute x and y.

```
Option Explicit
Const g As Double = 9.80665
Const degreesToRadians As Double = 3.14159265358979 / 180

Function maxTime(ByVal v0 As Double, ByVal theta As Double) as Double
    maxTime = 2 * v0 * Sin(theta * degreesToRadians) / g
End Function
```

Third exercise. Create a new worksheet from the worksheet for the first exercise and give it the name **Cells with Names**.[4] Create range names for each of the cells in column B that were specified by bullet points in the first exercise. The figure below at the left and the narrative below shows how to do this.

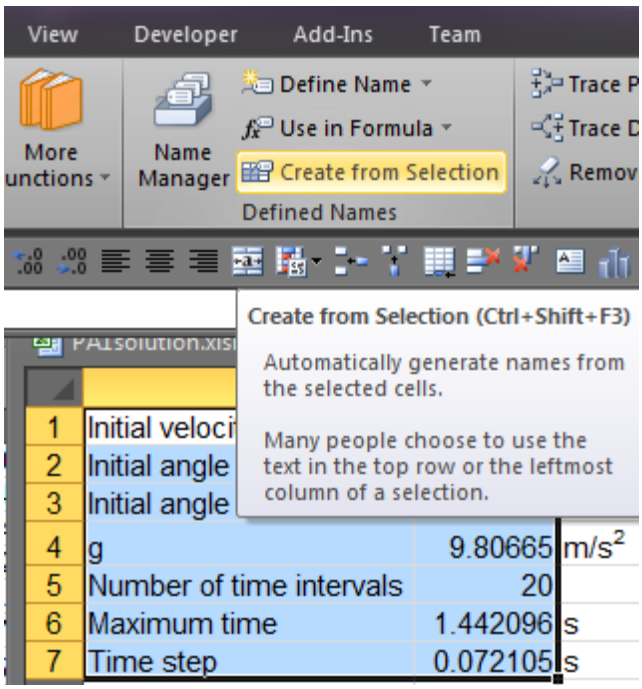
---

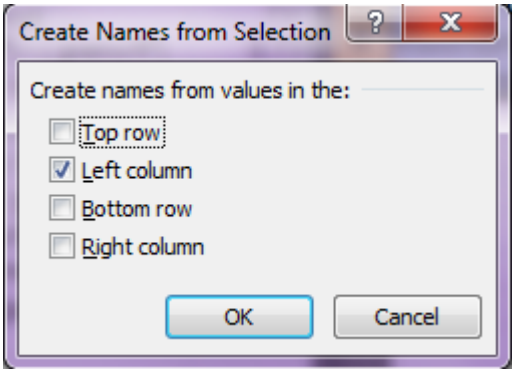
[2] The trig functions in Excel and VBA, like those in almost all programming languages, expect the argument in radians. Recall that an angle in degrees is converted to radians by multiplying it by π/180. For worksheet cell formulas you can use PI()/180 to get this factor. For VBA code you can use a defined constant specified by the following statement: `const degreesToRadians as double = 3.14159265358979/180`. In the Excel worksheet you can also use the RADIANS function. E.g., the statement `=SIN(RADIANS(A1)` computes the sine of an angle, in degrees, stored in cell A1.

[3] To do this, right-click the worksheet from exercise one. On the resulting menu select **Move or Copy**. In the resulting dialog box select **(move to end)**, place a check in the **Create a copy** checkbox, and click **OK**. Double click the tab name created by Excel, Cells (2), to change it to the desired name, simple UDF.

[4] Follow the procedure in the previous footnote.

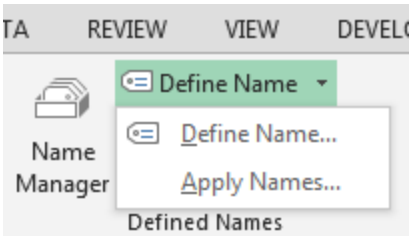
1. Select both the name column and the value column for the six rows with initial data and preliminary results. The figure at the right shows a slightly different calculation where the range A1:B7 is selected.
2. As shown at the right, click on **Create from Selection** from the Defined Names group on the **Formula** tab. You should get the Create from Selection dialog box shown below.
3. In this example the dialog box already has the correct choice of creating the names from the values in the left column. Simply click OK and the desired names will be created.



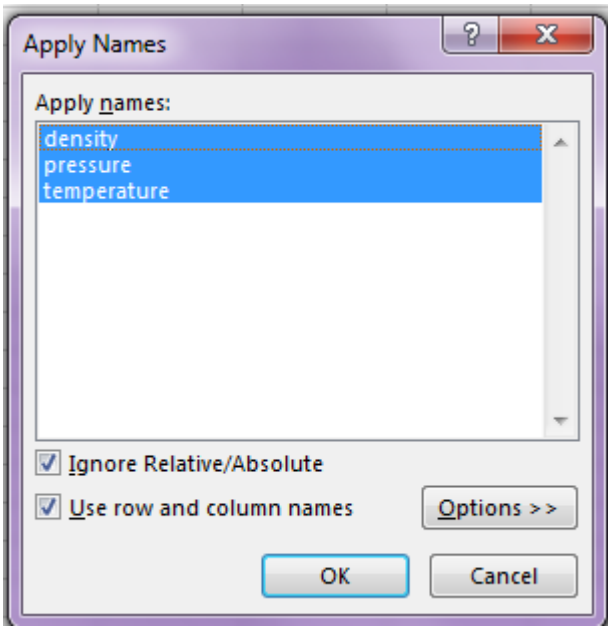


You can confirm that these these names have been defined by selecting the **Name Manager** from the Defined Names group on the **Formula** tab or you can use the pull-down menu in the name box to display the names.

Once you have defined the names, select all the formulas then choose the pulldown arrow from the right of the **Define Name** in the Defined Names group on the Formula tab. From the resulting submenu shown at the right, select **Apply Names**.



This will bring up the **Apply Names** dialog box, an example of which is shown below. In the this dialog box, you should select all the names in the **Apply names** list (if they are not already selected) and click **OK**. Examine the formulas on the worksheet. All the traditional (A1-type) cell references, in all the formulas, except for the time values should have been replaced by cell names. Verify that this is the case. Did you get the same results that you had previously?



An alternative to using the Apply Names procedure is to delete the existing formulas and rewrite them using point and click after you have defined the cell names. When you use point and click on a named cell the cell name, rather than the usual A1-type reference is used.

Fourth exercise. Create a new worksheet from the worksheet for the second or third exercise and give it the name **simple UDF Names**.[5] Edit this copied worksheet so that it has the same range names that

[5] Either worksheet will do as a starting point; the editing depends on the one you start with. It is probably easier to start with the worksheet from exercise three. Copying this worksheet will also copy the range names (as local names) onto the new worksheet.

you used in exercise three (defined locally for this worksheet) and the UDF formulas that you developed in exercise two.

Fifth exercise. This exercise uses the concept of an array formula, a single formula that is entered into multiple cells. In this exercise you will learn how to write and use a UDF that returns an array. When entering an array formula of any kind you have to use the combination Control+Shift+Enter to enter the array formula (instead of the usual enter key). The user directions for the worksheet you will write for this exercise are shown in the text box to the right. Later you will be asked to copy this text box and paste it onto your worksheet. The full directions for this exercise are shown below.

Instructions:
Enter values for initial velocity in cell B1 and initial angle in degrees in cell B2.
Select a range of 3 columns with the desired number of rows to display the trajectory plus the header.
With the full range still selected type one of the following formulas in the formula bar:
=trajectory(B1,B2)
=trajectory(v0,thetaDegrees)
Press control+shift+enter to enter the array formula.

Create a new worksheet from the worksheet for the fourth exercise and give it the name **array UDF**. Delete all the information on the worksheet **except for** the initial velocity and angle.

Copy the text box shown above and paste it onto your new worksheet with the upper left corner in the first row in the middle of column E. This will give the user instructions for using the worksheet that you will create. Edit the second formula in the text box to use the range names that you selected for the initial velocity and initial angle. All the calculations in this exercise will be done using VBA code. The VBA code will automatically detect the size of the area selected by the user to enter the array formula.

The VBA code for the UDF that is called from the worksheet is shown below. You can copy and paste this into your module.

```
Function trajectory(v0 As Double, theta As Double) As Variant

    'Array function to compute complete trajectory based on user inputs for
    '    initial velocity, v0 in m/s and initial angle theta in degrees

    Dim userRows As Long
    Dim userColumns As Long

    'Application.caller is the area in which the array formula is entered;
    'Statements below determine rows and column in this user-selected area

    userRows = Application.Caller.Rows.Count
    userColumns = Application.Caller.Columns.Count
    If userColumns < 3 Then
        MsgBox "You must select three columns to use this array function" _
               , , "ERROR!"
        trajectory = "You must select three columns to use this array function"
    Else
        trajectory = calculateTrajectory(v0, theta, userRows)
    End If

End Function
```

Note the following items in this function.

1. The function type is declared as variant. This allows the function name to be set equal to an array; it also allows the array to have components with different data types (string for the header and double for the values).
2. See the comments in the code and statements about using `application.caller` to find the size of the area selected by the user. These statements are an example of object code that is used to get information directly from the worksheet.

3. This function calls a second function, `calculateTtrajectory`, which is given below. The actual calculations are done in this ssecond function.

The `calculateTrajectory` function is shown below. You can also copy and paste this code into your code module; **however, this code is not complete**. At various places in this code, numbered (1), (2), (3a), and (3b), you are asked to write the statements that will complete the code for this function.

```
Function calculateTrajectory(v0 As Double, theta As Double, nRows As Long) As Variant

    'Array function to return complete trajectory and header row as variant array
    'Return array has columns for time, x, and y with row for each data point
    'Inputs are v0, initial velocity in m/s, theta, initial angle in degrees, and
    'nRows, the number of rows to be returned in the array
    'Note that nRows contains the header row and an initial row for time = 0

    Dim results() As Variant        'Array for results to be written to worksheet
    ReDim results(1 To nRows, 1 To 3)

        ‘(1) Use “Dim” statements to declare variables: time, dTime and k

    results(1, 1) = "Time (s)"      'Place header in first row of array
    results(1, 2) = "x (m)"
    results(1, 3) = "y (m)"

        '(2) Compute time step, dTime, from maximum time and number of rows, nRows.
        '     Remember to account for header row
    dTime =

        '(3a) Complete coding of loop.  Set lower and upper limits for k, the loop index
        'Call UDFs for x and y to determine these values as noted in (3b) below

    For k = <What are my limits?>
        time = (k - 2) * dTime
        results(k, 1) = time
              '(3b) Put statements here to compute x and y as additional
              'columns in the results array.  Use UDFs for x and y.
              'results(k,m) is an array that has results for time = (k-2)*dtime
              '             in the first column and the corresponding x and y
              '             values for that time in columns 2 and 3

    Next k
    calculateTrajectory = results   ‘Places results array as function value

    End Function
```

Hints for completing the calculateTrajectory function.

1. You are creating a two-dimensional array, the results array, which has three columns (for time, x, and y) and nRows rows, where nRows is the number required to have the text header plus all the numerical results from t = 0 to t = t_{max}.
2. To set the for-loop limits look at the following statement in the loop: `time = (k - 2) * dTime`. The first time through the loop you want t = 0 and the last time through the loop you want t = t_{max}. You also want the loop to calculate all the numerical results of the array as described in item one.
3. You have to correctly use nRows in your calculation of dTime. If you are not sure about the loop limits or the dTime calculation, make your best guess and see if your calculations give the correct number of rows with t = t_{max} in the final row. If not, adjust your calculations.

Once you have completed the function, follow the instructions that you just copied onto the worksheet and see if you get a correct trajectory.

Sixth exercise. Create a new worksheet from the worksheet for the fourth exercise and give it the name **Macro**. Delete all worksheet content **except for** the rows with data on initial velocity, initial angle, and number of time intervals.

This exercise uses a macro whose VBA code reads data from a worksheet and writes results to the worksheet. Copy and paste the following code into your VBA code module.

```
Sub getTrajectory()

    'Sub routine (macro) to place trajectory on worksheet
    'Can be called by command button or selecting macro from list (Alt-F8)

    Const startRow As Long = 8  'Start output in row 8
    Const startCol As Long = 2  'Start output in column B
    Dim v0 As Double            'Initial velocity in m/s
    Dim theta As Double         'Initial angle in degrees
    Dim nRows As Long           'Rows specified by user for output

        'Get data from specified cells on active worksheet, clear previous
        '   results, and call calculateTrajectory function to get results
        '   array which will be entered onto worksheet

    v0 = Range("b1").Value
    theta = Range("iniitalAngle").Value  'could also use Range("b2").Value
    nRows = Range("b3").Value + 2
    Cells(startRow, startCol).CurrentRegion.ClearContents
    Range(Cells(startRow, startCol), _
          Cells(startRow + nRows - 1, startCol + 2)).Value = _
                                        calculateTrajectory(v0, theta, nRows)

End Sub
```

Observe the use of the **Cells** and **Range** objects to get data from and write data to a worksheet. The Range command has a variety of options; the simplest one is used here to obtain the values of v0, theta and nRows from known worksheet cells. Note that cell locations like B2 or cell names can be used to define a cell. Here we use an assumed range name, `iniitalAngle`, for the worksheet cell that contains the initial value of the angle, theta. Notice that the cell name can be different from the program name for the same variable.

The cells command refers to a cell with two numerical references, a row number followed by a column number. The statement nRows = Range("b3").Value could be replaced by nRows = Cells(3,2).Value. The cells method is mainly used when the VBA code has to reference a range where the number of rows or columns is not known when the program is written. This is true in this program where the number of rows is an input to the problem; the end of the output range is defined is given by a number of rows to be added. This is done using a form of the Range command in which a range can be defined by two cell locations, the upper-left, and the lower-right. For example Range("B3", G6") is an alternative way of stating Range("B3:G6"). This upper-left-lower-right approach is useful when both the upper-left and lower-right locations are given as references using the Cells object. In the code the following range reference is used to specify a range star starts at cell B8 in the upper left and ends in column D at a row determined by the starting row and the number of rows to be filled.

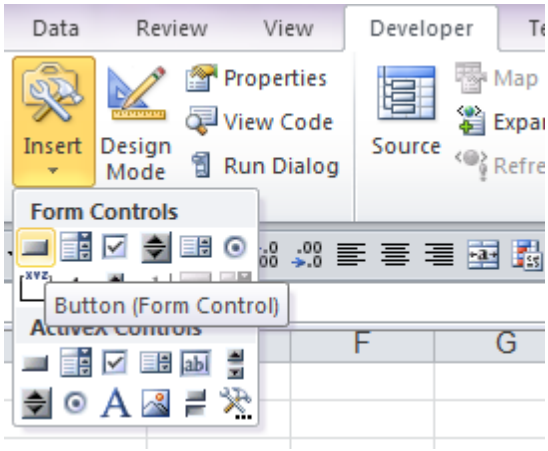
```
Range(Cells(startRow, startCol), Cells(startRow + nRows - 1, startCol + 2))
```

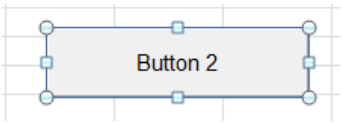
Review this statement to make sure that you understand what it means. Note, that in the code the Value property of this range is set to the result of the `calculateTrajectory` function. This means that each cell in this range is set to a corresponding value in the array that is returned form the `calculateTrajectory` function. (Take another look at the last statement in `calculateTrajectory` and note that function value is set equal to the results array.

After you copy the `getTrajectory` code into your module, return to the worksheet and insert a command button onto the worksheet, following the instructions below. If your Excel installation does not already show the developer tab, use the following instructions to display it: Selecting the **File** Tab | **Options** to bring up the Options dialog. In the resulting dialog box, click on **Customize Ribbon** in the left-side menu. In the tabs box on the right of the dialog, place a check in the checkbox next to **Developer Tab**. Click **OK** to exit.

Use the following steps to place a command button on the worksheet:

1. Select **Insert** from the Controls group in the **Developer** tab. You will see a menu with **Form Controls** as shown at the right.
2. Select the first control from this menu, the **Button (Form Control)**, by left clicking on the button icon.
3. Move the mouse to the worksheet and note that the icon changes to a cross.
4. Click and drag the mouse to give the shape of the button you desire.
5. As soon as you release the mouse you should see the **Assign Macro** dialog box. Select the **getTrajectory** macro and click **OK**.





After you do this your command button should look like the figure shown at the left. Note the circles surrounding the outline of the button. So long as these as in place, the button in in the edit mode. You can rename the button from the default name to a more meaningful one. You can resize it or move it to any other location on the worksheet. When you are ready to use the button, click the mouse anywhere on the worksheet off the button location and the circles will disappear. Now, left-clicking the button will execute the macro you assigned. (If you want to return to the edit mode for the button, simply right-click it.) Note that you will have to click the button every time that you change data; **recalculation is not automatic for macros**. However, that you can now easily change the number of time steps and get the results in a different sized table by clicking the button.

After you convince yourself that the calculations are working, complete the following tasks: (1) create a text box, like the one you copied in the previous exercise that gives instructions on how to use the worksheet, and (2) edit the VBA code to have the table start in cell C7.

Submission requirements

Write a Word document (length between one and two pages) discussing the different approaches used here and the advantages and disadvantages of each. Tell which approach or approaches you prefer and give your reasons for preferring the approaches you choose. You may prefer more than one approach for different reasons. Consider a variety of possible applications ranging from a simple one-time calculation to an application in which a user would want to do calculations repeatedly with different inputs.

Send an email to the instructor with the workbook and word file that you prepared. Use your name and the assignment number as the file names (*e.g.* I would use Caretto_1.xlsm and Caretto_1.docx). This assignment is due by 11:59 pm on Monday, February 3. Late submissions (30% penalty) will be accepted up to 11:59 Wednesday, February 5. No credit for late submissions after this date.