

NEEP 602 -- Engineering Problem Solving II Exercise 5

Using Runge Kutta in Microsoft Excel

Now we'll use Excel macros to solve initial value problems using a Runge Kutta method. As we did in Exercise 0 with Euler's Method, we can write a simple function macro that takes an initial value for a single step of a 4th order Runge Kutta scheme (see Exercise 1 for the algorithm) and returns the end value of the dependent variable. We then call this macro or function repeatedly to generate the full solution. Recalling the ODE from Exercise 0, $dy/dt=2yt$, the macro would contain the following two functions:

```
Function f(t, y)
    f = 2 * y * t
End Function
```

```
Function rk(t, y, dt)
    k1 = dt * f(t, y)
    k2 = dt * f(t + dt / 2, y + k1 / 2)
    k3 = dt * f(t + dt / 2, y + k2 / 2)
    k4 = dt * f(t + dt, y + k3)
    rk = y + (k1 + 2 * (k2 + k3) + k4) / 6
End Function
```

Then in the spreadsheet, we place the initial values for t and y , choose a time step, and make repeated calls to the function (rk) to produce the solution. As before, a typical sheet would look like the following (showing formulas here, rather than the results of the calculation):

	A	B
1	dt	0.1
2		
3	t	y
4	0	1
5	=A4+B\$1	=rk(A4,B4,B\$1)
6	=A5+B\$1	=rk(A5,B5,B\$1)

A Cleaner Approach

A second approach is to use subroutines (Subs), rather than functions. A Sub is much like a function macro, but it does not return a value to the spreadsheet. We will have to write the results directly back to the spreadsheet.

Before we use a Sub to solve differential equations, we'll practice with something similar. We will look at an example that writes a two-column spreadsheet which evaluates and lists a function of some variable X . The Sub below will accomplish this. Look at it first, and then read the explanation of how it works.

```
Function f(t)
    f = t ^ 2 + 3
End Function

Sub writeit()
    NumPoints = 21
    tNot = 0
    dt = 0.1
    ActiveCell.Value = "t"
    ActiveCell.Offset(0, 1).Value = "F(t)"
    t = tNot
    For i = 1 To NumPoints
        fnc = f(t)
        ActiveCell.Offset(i, 0).Value = t
        ActiveCell.Offset(i, 1).Value = fnc
        t = t + dt
    Next
End Sub
```

The first three lines set up the parameters. They indicate that we will plot 21 points starting with $t = 0$ and incrementing by 0.1. The next two lines write labels for t and for the function $F(t)$. Note that `ActiveCell` refers to the cell that is active in the spreadsheet when the macro (Sub) is run. Also, `ActiveCell.Offset(0, 1)` refers to the cell just to the right of the active cell because it offsets one cell from the active cell in the column direction. The Sub then sets the initial t value and loops through all the points to calculate and write results for $F(t)$.

To create and run this macro, go to Tools/Macro, type in the name you want to give it and click "Create." Excel should open a new Module in Visual Basic Editor into which you can type your code. Or, you can go to Tools/Macro/Visual Basic Editor, insert a new module, and type your code. Once you're done, click on a cell in a worksheet, and click Tools/Macro. This will bring up a dialog box that contains the name of your macro. Double-click on that macro name and it will execute.

Sometimes we would rather have the macro always start in the same cell, rather than starting in the active cell. To accomplish this, we select a cell by inserting a command such as `[A12].Select` in the macro. This will make cell A12 the active cell.

One more variation is to add a "user interface" by using buttons in the spreadsheet to run the macro. To do this, you

- Click View/Toolbars and tell the dialog box to show you the Forms Toolbar.
- Activate a spreadsheet (not a macro module).
- On the Forms Toolbar, click on the Button button (second row, second column).
- Go to the spreadsheet and trace out the button. You can double click on the button's label if you want to change it.
- When you define the button you'll be asked to attach a macro to it. This macro will be run when the button is pressed.

Using these techniques to solve ordinary differential equations, we can write a macro that will solve the whole ODE at once. The techniques are the same as above, but we have to adapt the Sub from evaluating a function to stepping through the solution of an ODE. We will have to input the initial value for y , the time at which we wish to stop, and the time step (or number of steps). To accomplish this, we write a subroutine that looks something like the following:

```
Sub writeODE()  
  first set constants  
  then loop through all steps, calling rk(h, y, t) in each step  
  then write the results back to the spreadsheet  
End Sub
```

You'll have to figure out how to do the internals. For the loop, you can use any of a variety of loops. Choose whichever you prefer. I would suggest that you do this for a single first order ODE and then progress to doing systems of ODE's. The problems from Exercises 1 and 2 will be a good place to start; you can compare the solutions as well as the methods.

Getting constants from the spreadsheet

Often such problems require the use of parameters in the function. It is often convenient to set all the constants within the spreadsheet. This can be done by having the macro retrieve the constants from the sheet. To accomplish this:

1. Give a name to the cells you'll want to access in the macro. This is done with **Insert/Name/define**. We'll assume the cell is named *TimeStep*.
2. Now in the macro you can do something like $dt=Range("TimeStep").Value$ in your macro. This sets the value of dt to whatever was in the cell on the spreadsheet.
3. You can do the same with all your variables.

Control Structures in Excel Visual Basic for Applications

There are many control structures that can be used in Excel 5 macros. They include the following:

If <i>condition</i> Then <i>statement</i>	If y=0 Then x = 1
If <i>condition</i> Then <i>statements</i> End If	If radius>1 Then count=count+1 End If
If <i>condition</i> Then <i>statements</i> Else <i>statements</i> End If	If j>1 Then x=3*j Else x=0 End If
Select Case <i>test expression</i> Case <i>first expression</i> <i>first statements</i> Case <i>second expression</i> <i>second statements</i> End Select	Select Case Score Case Is < 50 Grade = "F" Case Is < 70 Grade="C" Case Is <100 Grade="A" End Select
Do While <i>condition</i> <i>statements</i> Loop	Do While i<10 x=x*i i=i+1 Loop
Do <i>statements</i> Loop While <i>condition</i>	Do x=x+i i=i+1 Loop While x<47
Do Until <i>condition</i> <i>statements</i> Loop	Do Until i=50 x=x*i Loop
Do <i>statements</i> Loop Until <i>condition</i>	Do x=x*i Loop Until x=6
For <i>counter=start</i> To <i>end</i> <i>statements</i> Next	For i=1 To 100 sum=sum+i Next
For Each <i>element</i> In <i>group</i> <i>statements</i> Next	For Each cellObject in Selection cellObject.Formula = Application.Proper(cellObject) Next