The Runge-Kutta Method

Let us write our differential equation in the form

$$\frac{dy}{dt} = f(t, y), \tag{2.4}$$

where $f(t, y)$ on the right-hand-side is some function of t and/or y. We can interpret (2.4) as saying that the slope at every point (t, y) on the curve is given by the value of the function f at that point. Euler's method takes the value of f at the beginning of a timestep as the slope with which to move across the timestep. The Runge-Kutta method improves on this by evaluating f at four different places in the timestep and taking a weighted average of them as the slope to use to move across the timestep.

We will illustrate the Runge-Kutta method with the same differential equation as before, namely $dy/dt = t + y$. Refer to Fig. 2.2, which shows one timestep starting at t=1, ending at t=3 and with stepsize h=2. The first evaluation of the function f is at the beginning of the timestep. Call this value f_a. It equals 2. Use this slope of 2 to move halfway across the timestep. Evaluate f here. Call this value f_b. It equals 5. Go back to the beginning of the timestep and use this slope of 5 to again move halfway across the timestep. Evaluate f here. Call this value f_c. It equals 8. Go back to the beginning of the timestep. Use this slope of 8 but this time move all the way across the timestep. Evaluate f here. Call this value f_d. It equals 20. Now take this weighted average:

$$\frac{1}{6}(1 f_a + 2 f_b + 2 f_c + 1 f_d) \tag{2.5}$$

This is the slope that we finally use to move across the timestep. It equals 8. Thus if the timestep begins at $(t_1 = 1,\ y_1 = 1)$ then it ends at $(t_2 = 3,\ y_2 = 17)$. For comparison the analytical solution gives $y(3) = 18.167$.

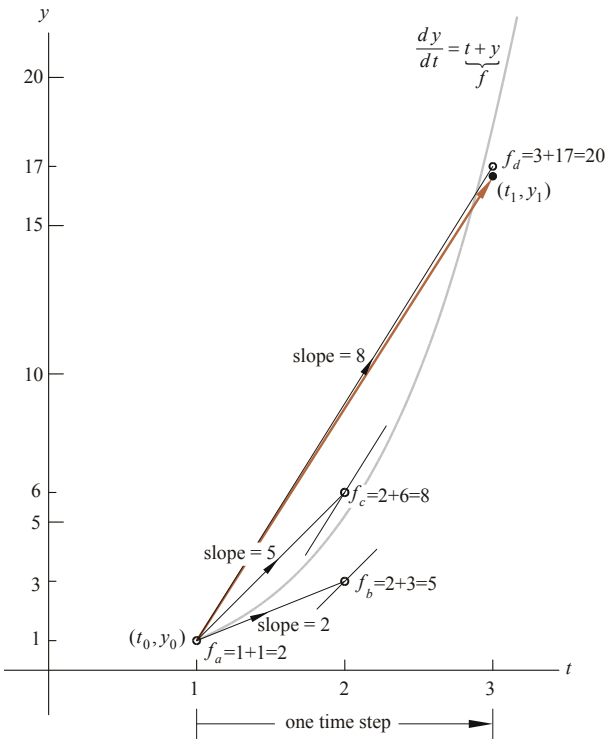

Figure 2.2 The Runge-Kutta method finds four points in sequence: at the beginning, middle and end of the time step, as indicated by the open circles. It evaluates the function *f* on the RHS of Eq. (2.4) at these points. These are the slopes f_a, f_b, f_c, f_d at these locations. It then uses a weighted average of them to move across the timestep from (t_0,y_0) to (t_1,y_1).

On the following page is a function *RungeKutta*, written in Visual Basic, that implements one timestep of the Runge-Kutta method and a subroutine, *Wrapper*, that calls *RungeKutta* repeatedly to step forward through multiple timesteps and print the output to an Excel spreadsheet. There is also a function, *RHS*, which is the right-hand-side of the differential equation to be solved. Do the following:

- Start Excel. Look for the Developer tab. (If it's not there then click on File>Options> Customize ribbon and check the Developer checkbox on the right side of the dialog box.)
- On the Developer tab click the Visual Basic button to open Excel's Visual Basic Editor.
- In the Editor click on Insert>Module to open a module into which the code can be typed.
- Type (or copy) *RHS*, *RungeKutta* and *Wrapper* into the module. Because *Wrapper* is declared "public" it appears in the list of macros that can be run in Excel. Run it.

It prints the following output to a spreadsheet:

index n	time t	y_{approx}	error
0	1.0	1	0
1	1.5	2.445	0.0009
2	2.0	5.152	0.0028
3	2.5	9.938	0.0069
4	3.0	18.152	0.0153

The program is presently set to solve the differential equation $dy/dt = t + y$ subject to the initial condition $y=1$ when $t=1$. Columns 2 and 3 are a table of values for the solution. The right-hand-side of the differential equation, presently $t+y$, is set in the line labelled [1]. The timestep size is set in line [2]. The initial condition is set in line [3]. The number of iterations (timesteps) is set in line [4]. For comparison purposes the program also prints out the error which is the difference between the Runge-Kutta solution and the known exact solution, $y = 3e^{t-1} - t - 1$.

Of course the Runge-Kutta method has truncation and accumulation error too. The following table shows how decreasing the stepsize h reduces the total error in computing forward from $t = 1$ to $t = 3$.

h	steps required	Runge-Kutta approx. to $y(3)$	Error
1	2	18.005	0.16
0.5	4	18.152	0.015
0.25	8	18.166	0.0012

The accuracy of the two methods can be compared using **big *O*** notation. It can be shown that the total error of Euler's method is $O(h)$ (spoken as "*of order h*"), meaning that $\lim_{h\to 0}(Error) \le M\,h$, where M is some constant. The total error of Runge and Kutta's method is $O(h^4)$ (spoken as "*of order h to the fourth*"), meaning that $\lim_{h\to 0}(Error) \le M\,h^4$. What this means is that, assuming h is small, cutting h in half in Euler's method will cut the error in half, but cutting h in half in the Runge-Kutta method will reduce the error by a factor of 16 (since $\left(\frac{1}{2}\right)^4 = \frac{1}{16}$). That is much better!

A Visual Basic Program for the Runge-Kutta Method

```
Function RHS(T, Y)                          'The RHS of the differential equation as defined in Eq. (2.4)
   RHS = T + Y                              '[1]  change this line to the right-hand-side of your DE
End Function

Function RungeKutta(T, Y, H)                'Input:  values of t and y at beginning of timestep, and the stepsize h
                                            'Output: value of y at end of timestep calculated by Runge-Kutta method
   Dim Fa, Fb, Fc, Fd, YTemp                'Note:   This function calls function RHS

   Fa = RHS(T, Y)                           ' fa, fb, fc, fd are as described in the paragraph before Eq. (2.5)

   YTemp = Y + 0.5 * H * Fa                 'use the slope of point a to find height of point b
   Fb = RHS(T + 0.5 * H, YTemp)             'find slope at point b

   YTemp = Y + 0.5 * H * Fb                 'use the slope of point b to find height of point c
   Fc = RHS(T + 0.5 * H, YTemp)             'find slope at point c

   YTemp = Y + H * Fc                       'use the slope of point c to find height of point d
   Fd = RHS(T + H, YTemp)                   'find slope at point d

   RungeKutta = Y + H * (Fa + 2 * Fb + 2 * Fc + Fd) / 6     'use the weighted average defined in (2.5)
End Function                                'and return the y value at the end of the timestep

Public Sub Wrapper( )                       'User calls this macro.  It prints the solution of the IVP as a table of values.
   Const H = 0.5                            '[2]  The step size; smaller h results in better accuracy
   Dim T, Y, YOut, N

   T = 1: Y = 1                                                  '[3]  Set the initial conditions

                                                    'print the heading of the table and first row (the initial conditions)
   Cells(1, 1) = "index n":   Cells(1, 2) = "time t":   Cells(1, 3) = "y approx":   Cells(1, 4) = "error"
   Cells(2, 1) = 0:           Cells(2, 2) = T:          Cells(2, 3) = Y:            Cells(2, 4) = 0

   For N = 1 To 4                                '[4]  Each time through the loop calculate and print one timestep.
      YOut = RungeKutta(T, Y, H)
      Y = YOut                                                   'update y
      T = T + H                                                  'update t
      Cells(N + 2, 1) = N: Cells(N + 2, 2) = T:    Cells(N + 2, 3) = Y     '[5] print the rest of the table
      Cells(N + 2, 4) = (3 * Exp(T - 1) - T - 1) - Y
   Next N
End Sub
```