David Hood

Numerical solution of ordinary differential equations using an MS Excel® spreadsheet

David Hood
RSS Centre for Statistical Education
Nottingham Trent University
david.hood1@ntu.ac.uk

Background

A recent paper in MSOR Connections [1] discussed the representation of the analytic solution of the second order differential equation:

\[ a \frac{d^2y}{dt^2} + b \frac{dy}{dt} + cy = R \sin(\omega t + \varphi) \]

by an MS Excel® spreadsheet. The author referred to alternative packages, such as Matlab™, that make use of numerical approaches, but the cost of buying additional packages is cited as a reason for the MS Excel® approach used. This paper shows how a VBA (Visual Basic for Applications) implementation of a numerical integration technique can be incorporated in an MS Excel® spreadsheet, enabling any number of simultaneous first order differential equations to be solved numerically, without the need for installing other packages.

The standard problem

The set of equations to be solved by the spreadsheet implementation is the standard initial value problem: find the functions \( x_i, i = 1, \ldots, n \) that satisfy the differential equations:

\[ \frac{dx_i}{dt} = f_i(x_1, \ldots, x_n, p_1, \ldots, p_m, t), i = 1, \ldots, n \]

where the \( x_1, x_2, \ldots, x_n \) are known at a particular value of \( t \). The \( p_j \) are known parameters that occur in the equations.

As an illustration, the Kermack & McKendrick model [2, 3] for the spread of an epidemic through an isolated community can be expressed in the above form. In the Kermack-McKendrick model, the population is divided into three classes: \( S \), the number of susceptibles; \( I \), the number of infectives; and \( R \) the number who are immune. The Kermack-McKendrick model is a set of differential equations for \( S, I, \) and \( R \):

\[ \frac{dS}{dt} = -\beta IS, \quad \frac{dI}{dt} = \beta IS - \gamma I, \quad \frac{dR}{dt} = \gamma I \]

Here the \( \beta \) and \( \gamma \) are parameters that depend on the infectivity of the disease, and the period of time for which a sufferer is infective.

These equations, expressed in terms of the general initial value problem, are:

\[ \frac{dx_1}{dt} = -p_1 x_1 x_2, \quad \frac{dx_2}{dt} = p_1 x_1 x_2 - p_2 x_2, \quad \frac{dx_3}{dt} = p_2 x_2, \]

where \( x_1, x_2, x_3 \) represent \( S, I \) and \( R \), and \( p_1 \) and \( p_2 \) represent the parameters \( \beta \) and \( \gamma \).
The VBA function

The VBA function to perform the numerical integration can be incorporated in a standard MS Excel© spreadsheet; the VBA function is then available for use within the spreadsheet in much the same way as the existing SUM and LINEST functions are. In this instance, the numerical integration technique adopted was the standard fourth order Runge-Kutta technique; the VBA function was named RK4. Other integration methods can be implemented by a similar approach.

In designing the VBA function, the following considerations applied:

1. The VBA function has to be supplied with the current values of the dependent and independent variables. Consequently, the dependent and independent variables are arguments of the VBA function.

2. The VBA function has to be supplied with the timestep to be used in the numerical integration. It is unlikely that the user wants to see the results of all timesteps, so the number of timesteps to be completed before return to the spreadsheet also has to be user specified. Consequently, the timestep and number of timesteps are arguments of the VBA function.

3. In the above epidemic modelling application, it is likely that the user is interested in the effect that modifications to the values of $\beta$ and $\gamma$ will have on the solution. The most convenient way of specifying the parameter values is in the spreadsheet itself, so the parameter values are passed to the integration VBA function via the argument list.

4. As the integration program has to be able to solve several first order differential equations, the values of several updated variables have to be returned by the VBA function. Consequently, the integration function is an array valued function.

Taking the above into consideration, the VBA integration function takes the form: RK4(nstep,h,x,p) where nstep and h will contain cell references for the number of integration steps to be carried out and the timestep value respectively. The argument x will contain the range of cells that hold the current values of $t$ and the $x_i$ (with $t$ the first in the range). The argument p will contain the cell range that holds the parameter values. The updated values for $t$ and the $x_i$ will be returned to the cells where this function is activated. The differential equations will be supplied by a further Visual Basic function: RHS.

The code for the VBA functions

The code for the VBA functions can be accessed from the spreadsheet by selecting Tools/Macro/Visual Basic Editor.

The function rhs is where the user specifies the differential equations to be solved.

The function below is appropriate for the Kermack-McKendrick model.

Function rhs(x, t, dxdt, p)
    '=============================================='
    ' Input your right hand sides in the form dxdt(1) = expression
    ' in t, x(1), x(2), etc. The right hand side may also include
    ' several parameter values: p(1), p(2) etc
    '=============================================='
    dxdt(1) = -p(1) * x(2) * x(1)
    dxdt(2) = p(1) * x(2) * x(1) - p(2) * x(2)
    dxdt(3) = p(2) * x(2)
    '=============================================
    ' Function must end with rhs = 0 statement to return a value
    '=============================================
    rhs = 0
End Function

The code for the rk4 function is shown below. It would not normally be modified by the user.

Function rk4(nstep As Integer, h As Double, x As Range, p As Range)
    '==================================================
    'Applies nstep steps of 4th order RK method to the equations
    'specified in module rhs
    'h holds the timestep, x(1) holds the value of t, with
    'x(2)... holding current x vals.
    '==================================================
'p is a range of parameters that can be used in the function rhs.

Dim n, nr, i, step, temp As Integer
n = x.Count 'find how many entries are in array x
nr = x.Rows.Count 'used to check if input range a row or column
neqn = n - 1 'find how many independent variables in equations
Dim k(), xtemp(), ttemp, soln(), dxdt(), xcurr(), xx() As Double
ReDim k(1 To neqn), xtemp(1 To neqn), soln(1 To neqn)
ReDim dxdt(1 To neqn), xcurr(1 To neqn), xx(1 To n)

'Strip off the t value from the x values & store x values in xcurr
't = x(1)
For i = 1 To neqn: xcurr(i) = x(i + 1): Next i

For step = 1 To nstep
    For i = 1 To neqn: xtemp(i) = xcurr(i): Next i
    temp = rhs(xtemp, t, dxdt, p)  'evaluate rhs with current x value
    For i = 1 To neqn
        k(i) = h * dxdt(i)            'set up k1
        soln(i) = xcurr(i) + k(i) / 6 '& add k1 contrbn to new soln pt
        xtemp(i) = xcurr(i) + k(i) / 2 ' set up x for eval of k2
    Next i
    ttemp = t + h / 2                  ' set up t val for eval of k2
    temp = rhs(xtemp, ttemp, dxdt, p) 'evaluate rhs to use to eval k2
    For i = 1 To neqn
        k(i) = h * dxdt(i)             'eval k2
        soln(i) = soln(i) + k(i) / 3   'add k2 contrb to new soln pt
        xtemp(i) = xcurr(i) + k(i) / 2 'set up x for eval of k3
    Next i
    temp = rhs(xtemp, ttemp, dxdt, p) ' evaluate rhs to use to eval k3
    For i = 1 To neqn
        k(i) = h * dxdt(i)            'evaluate k3
        soln(i) = soln(i) + k(i) / 3  'add k3 contrb to new soln pt
        xtemp(i) = xcurr(i) + k(i)    'set up x for eval of k4
    Next i
    t = t + h                         'set up t value for eval of k4
    temp = rhs(xtemp, t, dxdt, p)     'evaluate rhs to use to eval k4
    For i = 1 To neqn
        k(i) = h * dxdt(i)            'eval k4
        xcurr(i) = soln(i) + k(i) / 6 'add k4 contrb to new soln pt
    Next i

'End of R-K process for current step.
'xcurr contains current sol for x

Next step

'End of required number of timesteps.
`Build rk4 to hold t value followed by x values.
'==================================================
xx(1) = t: For i = 1 To neqn: xx(i + 1) = xcurr(i): Next i
If nr > 1 Then 'Test to see if x_in in cols, write soln in cols
    rk4 = WorksheetFunction.Transpose(xx) 'x_in in cols
Else
    rk4 = xx                              'x_in in rows
End If
End Function

Using the VBA function

The use of the VBA integration function will be illustrated
by applying it to the K-M model for the first 5 days of a
short epidemic in an isolated community. A timestep of 0.1
days, was used, with output of the solution each 0.5 days
(achieved by setting nstep = 5). The values of β and γ used
were 0.002 and 0.3 respectively, with the initial values of t, S,
I, and R set to 0, 843, 10, 20 respectively.

In the spreadsheet shown below, the disease parameters
and the integration parameters are specified in cells C7, D7
and F7, G7 respectively. The initial values of t, S, I, and R are
in cells C10 to F10.

Fig 1 – Solution of epidemic model

The formula `" = C10" is entered in cell C13 in order to copy
the initial t value to that cell. Corresponding formulae are
used in cells B13 to D13 so that the initial values for S, I, and
R appear there.

Now to the RK4 function calls. When we make use of the
RK4 function, we need to ensure that the formula entered
can be copied, so that we can repeatedly advance the
solution without having to re-type the formula. Entering
and copying the formula correctly automatically ensures
that at each stage we use the updated variables as input to
the next call of RK4. However, cell reference locking must be
used when referring to cells holding the parameter values
and the integration parameters.

To enter the required formula, the cells C14 to F14 were
highlighted, and the formula:

`= RK4(F$7,G$7,C13:F13,C$7:D$7)`
typed. As this is an array formula, it is entered by pressing and
holding the [Control] & [Shift] keys, and then pressing [Enter].

With the values in the spreadsheet, a call to RK4 advances
the solution by 5 steps of 0.1 days. To use copying to find
the solution at subsequent t values, the cells C14 to F14
were highlighted, and “dragging” used to copy the formula
to the rows 15,16,17 etc., leading to the approximate
solution for the first five days of the epidemic.

Summary

The facilities of Excel may be easily extended, by the use
of VBA functions, to include the numerical solution of
simultaneous first order differential equations. Using a similar
approach, Rosen [4] uses an MS Excel© spreadsheet to display
the solution of a model of the earth's carbon cycle.

Packages such as Matlab™ offer accurate and robust
numerical procedures for numerical integration, and if such
packages are available, it is probably better to encourage
students to use them. However, MS Excel© is not without
its advantages. Students have a great deal of familiarity
with MS Excel©, and generally have ready access to it. Any
subsequent processing and the graphical display of the
solution is aided by this familiarity.

References

1. Mohameden, A., “MS Excel© for higher maths”. MSOR
   Connections Vol 8 No 4: 21-25. Available via:
   http://mathstore.ac.uk/headocs/84Mohameden_A.pdf
   [Accessed 3 March 2009].
   Lond. A 115, 700-721, 1927. Copies of this paper may be
downloaded from the Royal Society of London website.
   Available via: http://rspa.royalsocietypublishing.org/
   content/115/772/700.full.pdf+html
   [Accessed 3 March 2009].
   Differential
   Equations and Mathematical Biology, CRC Press.
   via http://www.meteor.iastate.edu/gccourse/chem/