Transactions on Information and Communications Technologies vol 10, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

Numerical solutions to arbitrarily sized systems of coupled first order ordinary differential equations using computer algebra software

P. Mitic, P.G. Thomas
Faculty of Mathematics and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK

1. Abstract

This paper describes how computer algebra techniques may be applied to finding numerical solutions to systems of differential equations using some simple iterative methods. The advantages and disadvantages of using computer algebra techniques rather than procedural programming languages such as Pascal and C are assessed. We consider how the student may learn from using computer algebra software, using simple iterative methods to emphasise the general approach and the flexibility of the software. Common examples in engineering are considered.

2. Introduction

Mathematica is used to implement the Euler, 4th. order Runge-Kutta and 2nd. order Taylor iterations for approximating a solution to the system of equations

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{x}(t), t), \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0 \qquad \text{.....(1)}$$

where $\boldsymbol{x}(t)$ is a vector of arbitrary dimension. Symbolic computation software makes it possible for the methods used to be independent of the number of differential equations in the system. In the case of the 2nd. order Taylor method, we have developed a method of extending the Euler method which accomplishes the necessary calculus in a particularly efficient way using replacement rules. The use of a computer algebra system is of prime importance here because the differentiation required is done automatically for any input functions within an integrated environment for symbolic computation.

Using computer algebra software, the student can easily vary parameters and input functional forms, and program different iterations. We describe how the software acts as a tool to enable study of the engineering, mathematical and numerical concepts without the need to learn complicated mathematical methods.

Transactions on Information and Communications Technologies vol 10, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

3. A two-equation system using the Euler iteration

Consideration of 2nd. order differential equations is useful in this discussion because of the variety of systems in which they are applicable. They occur in engineering, population dynamics, mechanics, biological systems etc., and all but the simplest have no analytical solution. A simple way of obtaining a numerical solution to such a differential equation is to rewrite it as a pair of two first order differential equations. We consider, as an example, a spring-dashpot system with a non-linear damping term $\varepsilon(x^2-1)\dot{x}$, which gives the Van der Pol equation

$$\ddot{x}+\varepsilon(x^2-1)\dot{x}+x=0\,;\quad \varepsilon>0. \qquad \text{.....(2)}$$

The dependent variable, x , represents the displacement from the equilibrium position. The initial condition $y(0)=0.5$. will be used, and we set $\varepsilon=1$. Minorsky [1] obtained the same form of equation by modelling a spontaneously oscillating valve circuit. Using the substitution $y=\dot{x}$, we obtain the pair of first order differential equations

$$y=\dot{x},\ \dot{y}=\varepsilon(1-x^2)y-x\,. \qquad \text{.....(3)}$$

The Euler method is applicable, and may be summarised by the iteration

$$x_{n+1}=x_n+h;\ \ y_{n+1}=y_n+hf(x_n,y_n);\ \ x_0=a,\ \ y_0=b,\ \ n=0,1,\ldots\ . \qquad \text{.....(4)}$$

This form can be reproduced using Mathematica code due to Maeder [2]. The function `EulerStep` code encompasses both iterations for x_n and y_n. It does not look like the iterations in (4), which is a disadvantage from the student's point of view. It does represent a general case which is independent of the number of equations in the system, and can be easily amended for other numerical techniques. The procedure `Euler` is a repeated application of `EulerStep` and uses the Mathematica function `NestList`, which was designed for just this purpose. There are two versions of `Euler`, one of which takes time, *t*, as an explicit parameter and the other of which does not. Such polymorphic definition is not possible naturally in a procedural language such as C or Pascal, and means that various input forms are possible. The student does not need to learn and use a number of similarly named functions which all do similar things. The code is

```
EulerStep[f_, y_, y0_, dt_]:=
   y0 +  N[ f /. Thread[y -> y0] ] dt

Euler[f_List, y_List, y0_List, {t1_, dt_}]:=
   NestList[ EulerStep[f, y, #, N[dt]]&, N[y0],
   Round[N[t1/dt]] ] /;
         Length[f] == Length[y] == Length[y0]

Euler[f_List, y_List, y0_List, {t_, t0_, t1_, dt_}]:=
Module[{ res },
         res = Euler[Append[f,1], Append[y,t],
     Append[y0,t0], {t1-t0,dt}];
         Drop[#, -1]& /@ res
       ] /;  Length[f] == Length[y] == Length[y0]
```

Transactions on Information and Communications Technologies vol 10, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

Applying `Euler` is done using the following code.

```
equations = {y,-x + (1- x^2) y};
vars = {x,y};
steps = 25;
steplength = .1;
VdP=Euler[equations,vars,{0.5,0},{steps,steplength}];
```

Plotting the result is done using the following code.

```
Y = Map[#[[1]]&,VdP];
t = Range[0,steps, steplength];
tY = Transpose[{t,Y}];
p1 = ListPlot[tY, AxesLabel->{"t","Y"},
              DisplayFunction->Identity]
Show[p1,DisplayFunction->$DisplayFunction]
```

All numerical output is suppressed, and the result is the graph (Figure 1), which gives an excellent overview of the form of the numerical solution.

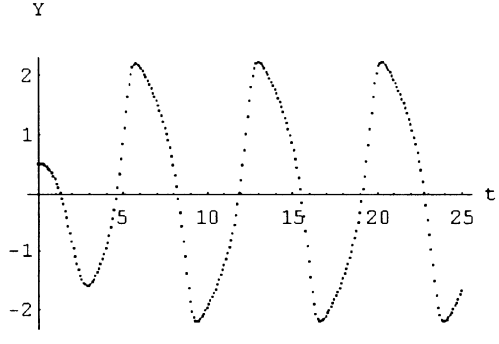

Figure 1: Van del Pol oscillator obtained using the Euler iteration

The power of this computer algebra technique is that the term `equations` can easily be altered in a natural way without the need to write new code and compile it. The student can learn the important features of this and similar systems, and of the Euler method by amending `equations`, `steps` and `steplength`.

4. Extension to a more accurate iteration technique

The principles of programming the Euler method can easily be extended to more sophisticated methods. All that is necessary is to change the function which implements the iterative step. Thus, to implement the Runge-Kutta 4th. order iteration, `EulerStep` is changed to `RKStep` and a new procedure, `RK`, replaces `Euler`. The two are exactly similar except that the former refers to `RKStep` instead of `EulerStep`. The student can do this by knowing nothing more than the form of iteration.

```
RKStep[f_, y_, y0_, dt_]:=
Module[{k1, k2, k3, k4},
       k1 = N[ f /. Thread[y -> y0] ] dt;
       k2 = N[ f /. Thread[y -> y0+k1/2] ] dt;
       k3 = N[ f /. Thread[y -> y0+k2/2] ] dt;
       k4 = N[ f /. Thread[y -> y0+k3] ] dt;
       y0 + (k1 + 2 k2 + 2 k3 + k4)/6
      ]
```

It is instructive to superimpose the Euler and Runge-Kutta profiles for this particular Van der Pol equation. The method for doing this is explained clearly in Shaw and Tigg [3], and the result is shown in Figure 2.

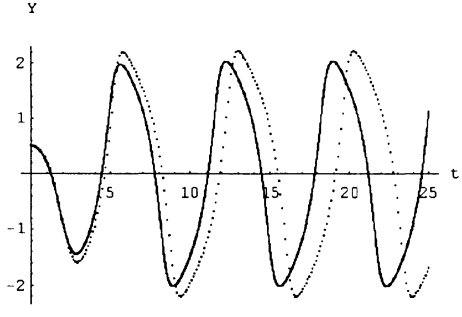

Figure 2: Superimposed Van der Pol profiles:
Euler (dotted), Runge-Kutta (solid)

At this stage, the flexibility of the software makes it possible to study the accuracy and stability of the two iterative methods. Multiple profiles can be superimposed, and it is easy to assess a useful number of steps for the iteration. Of special interest is the least number of steps required to give a given accuracy for a target value of $x(t)$, say $x(25)$. The necessary theoretical considerations are given in, for example, MST204 [4] .

5. Problems arising from programming a Taylor iteration

In principle, replacing a function such as `EulerStep`, and incorporating the result in a new procedure which uses `NestList` is all that is needed to implement an alternative iteration. However, Taylor iterations involve one or more differentiations of the function f of (1). For example, the y-iteration takes the following form for the Taylor 2nd. order method:

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2!} f'(x_n, y_n). \qquad \text{.....(5)}$$

Programming this is tricky and involves replacement rules to effect the replacement of $y'(x)$ terms by a function of x and $y(x)$ once f' has been calculated. The advantage of computer algebra software in this context is that all necessary differention(s) can

be done by means of software. This is not possible using C (or similar), or a spreadsheet. Once it is done, the student can investigate the effect of using a method which converges faster, in exactly the same way as for the Euler and Runge-Kutta methods. In particular, ill-conditioning can be searched for. This is particularly important, and students should be aware that any solution obtained requires intelligent interpretation. It can be tempting for lecturers to set 'do-able' problems (Mitic [5]) in an attempt to make the student's life easy, but this situation is unnecessary given that we have symbolic computation tools.

The new procedures, `Taylor2Step` and `Taylor2`, which implement the iteration are listed below (the time dependent version of `Taylor2` is omitted). The code for the differentiation is the one line

```
df = D[f,t] /. Thread[D[v,t]->f];
```

in `Taylor2`. In order to effect the differentiation step in `Taylor2`, it is necessary to pass numerical values of the derived function to and from `Taylor2Step` as well as numerical values of the function itself. Hence, the first argument in `Taylor2Step` is a pair `{f, df}`, the arguments of which represent the numerical values of the function and the derived function at each stage of the iteration respectively.

```
Taylor2Step[{f_,df_}, var_, y0_, t_, dt_]:=
Module[{},
        y0 +  N[ f /. Thread[var -> y0] ] dt +
       N[ df /. Thread[var -> y0] ] dt^2/2!
     ]

Taylor2[f_,v_,v0_,t_,t1_,dt_]:=
Module[{df},
       df = D[f,t] /. Thread[D[v,t]->f];
       NestList[ Taylor2Step[{f,df},v, #,t,N[dt]]&,
            v0,Round[N[t1/dt]]  ]
      ] /; Length[f] == Length[v] == Length[v0]
```

The need to differentiate makes it necessary to make the dependence on an independent variable explicit. Hence, to use these procedures we must write

```
equations = {y[t],-x[t] + (1- x[t]^2) y[t]};
vars = {x[t],y[t]};   .
```

Thus, the Taylor method has mixed blessings for the user. It is not easy to program and its usage is slightly different to other methods considered here.

6. A one-equation system

In order to illustrate the extreme flexibility of any given iterative method, we now consider a simple first order differential equation which arises from an elementary R-L circuit, and apply the Euler method. The circuit consists of a resistor, R, an inductance, L, and a voltage, $V(t)$, in series. The differential equation for the current, $x(t)$, is $L\dot{x} = V(t) - xR$, $x(0) = x_0$. Only one equation is involved, but it must still be entered as a list containing one element for the sake of maintaining

the generality of the procedure. It is thus marginally harder to use these procedures for one differential equation than to use it for two!

To show, further, how the software can aid the student, we illustrate the case of a voltage function, $V(t)$, which has a piecewise definition, and which leads to a very difficult equation to integrate. Mathematica cannot do it directly, which is due to the piecewise definition, but it would be possible to treat each piece separately. The natural way in which definitions can be stated, and the way in which there is no essential difference between posing the problem for this system and any other, makes computer algebra software ideal for the study of this topic. The required inputs are as follows.

```
R = 100;
V0 = 100;
L = 40;
x0 = {2};
(* V[t] is constant for 0<=t<10 and is a modulated
   sinusoidal function for 10<=t<=20  *)
V[t_/; (t>=0 && t<10)]:= V0/2
V[t_/; (t>=10 && t<20)]:= V0 E^(-t/10) Sin[t]
equations = {(V[t]- R x)/L};
vars = {x};
steps = 20;
steplength = .1;
current=
Euler[equations,vars,x0,{t,0,steps,steplength}];
```

In producing the numerical solution profile, Figure 3, we tried many variations of the voltage function, number of steps and total time. Furthermore, we invite the reader to superimpose the graph of Figure 3 on a graph $V(t)$. This is the sort of exploration we would encourage a student to do, and it can be done with virtually no knowledge of the programming language itself, since this mimics natural mathematical notation.

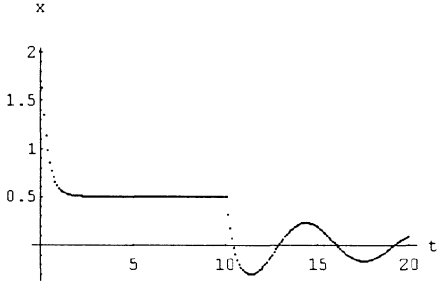

Figure 3: Current, $x(t)$, in an R-L circuit with a piecewise-defined voltage function

Transactions on Information and Communications Technologies vol 10, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

7. The Rössler Attractor: a 3-equation system

In principle, it is easy to perform similar computations with 3 dependent variables, but it is harder to visualise the result in some cases. This is because it is necessary to know the ranges of the space variables, x, y and z. Without this, 3D-plots can appear truncated. Rössler systems are discussed by Peitgen, Jürgens and Saupe [6], and an example is given in the Mathematica input below. A 3-D display is obtained by "joining the dots" with the `Line` function (Figure 4).

```
equations = {-(z+y), x+0.2y, 0.2+z (x-5.7)};
vars = {x,y,z};
steps = 100;
steplength = .02;
rossler = RungeKutta[equations,vars,{-1,0,0},
            {steps, steplength}];
Show[Graphics3D[{Line[rossler]}], Axes->Automatic,
     PlotRange->{0,25}]
```

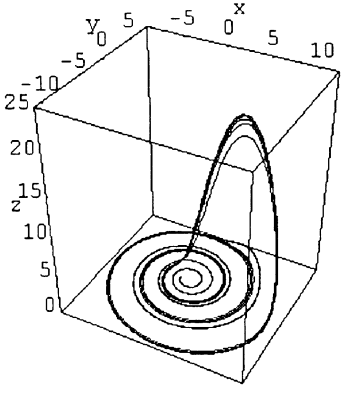

Figure 4: Rössler attractor 3-D plot

Producing illustrations such as Figure 4 requires considerable memory if the curve is to appear smooth, but does not take too long. It is easy to extract the space coordinate separately and plot each against time. Clearly, these techniques can be extended to higher order systems in the same way, extracting coordinates as necessary to plot against each other or against time.

8. Built-in numerical methods

Mathematica has the built-in function `NDSolve`, which can be used as an alternative to the methods described here to solve differential equations numerically. It cannot be used to investigate the properties of individual numerical methods, or to understand how a given method works. We have not used it here for this reason. The form of the output produced by `NDSolve` is not explicit. The result can be cast into a graphic object, but not directly into functional form or a list of points.

Using a user-programmed numerical method in this context can often help to identify an inherent lack of robustness in a mathematical model. The Euler method, although less accurate than other methods discussed here for given step size, should give consistent results for small step sizes. If it does not, ill-conditioning may be indicated. This can be disguised by using a more accurate method exclusively.

9. Conclusion

The major strength of a symbolic computation system for investigating and using numerical techniques is that the functional forms as well as parameters can serve as inputs. Furthermore, there is no need to specify the length of an input list in advance, so that the same procedure can be used with input lists of arbitrary length. Polymorphic procedure definition means that procedures with different numbers of arguments can have the same name. Such a system is of benefit for the student because of its flexibility. Symbolic manipulations, including those involving calculus, can be incorporated such that they are invisible to the user, who can concentrate on interpretation of results and methodology. There is no need to write and recompile code, or to include a parser in the code instead.

Computer algebra packages have their disadvantages. They are much slower than executable Pascal or C code because they interpret code, although it is possible to compile frequently used Mathematica code fragments. Much more memory is also required, and it is not possible to produce a dedicated stand alone Mathematica file.

In particular, scientists and engineers can benefit from using computer algebra software because the supplied functions and procedures are frequently designed with their needs in mind. The way in which these functions and procedures are used constrains the user to follow logical sequences and to perform tedious manipulations, without having to worry about how to do individual sub-tasks or making minor errors.

10. References

1. Minorsky, N. *Nonlinear Oscillations*, Van Nostrand, 1962.
2. Maeder, R.E. *Programming in Mathematica*, Academic Press, 1991.
3. Shaw, W and Tigg, J. *Applied Mathematica, Getting Started, Getting it Done.* Addison-Wesley, 1994
4. MST204, *Applied Methods and Modelling (Unit 19)*, Open University Press, 1982
5. Mitic, P & Thomas, P.G. Pitfalls and Limitations of Computer Algebra, *Computers and Education* **22**, 4. Elsevier Science. 1994
6. Peitgen,H., Jürgens,H. and Saupe,D. Chaos and Fractals: New Frontiers of Science. Springer-Verlag. 1992