Using Spreadsheets to Teach Engineering Problem Solving: Differential and Integral Equations

James P. Blanchard
University of Wisconsin - Madison

ABSTRACT

Spreadsheets offer significant advantages for teaching numerical problem solving because of their intuitive interface. Students have little difficulty implementing algorithms in spreadsheets, allowing them to study the behavior of algorithms without having to spend significant amounts of time writing routines that implement them. This has been found to be particularly true for the solution of differential and integral equations, where Runge-Kutta and finite difference algorithms are often used. This paper describes the use of a particular spreadsheet application (Microsoft Excel 5.0) to solve a variety of equations in an undergraduate problem solving course. The advantages and disadvantages of this approach are discussed.

Introduction

Numerical solutions to engineering problems have historically been carried out using procedural programming languages. This is not efficient from a pedagogical perspective because students typically must put more effort into learning the language itself than they put into solving problems. For example, the numerical solution of a boundary value problem in one dimension using finite difference techniques generally involves the creation of a system of linear equations and the conversion of that system into an equivalent matrix equation that then can be solved. Many students find this process confusing, so, for instance, a simple change such as modifying the boundary conditions often takes substantial effort to incorporate into a working solution. The difficulty here is that students become bogged down in forcing the algorithm to fit a structure required by the procedural language, rather than implementing the change in a more natural way. Modern computational tools can alleviate this difficulty, easing the programming effort required and allowing students to spend more time focusing on the performance of the algorithms and on the behavior of the resulting solutions. Students are able to implement algorithms in a more convenient format, removing some of the steps typically required in reaching a solution and thus allowing more effort to be spent comparing various algorithms and studying the behaviors of the equations themselves.

One example of a tool with which equations are easily solved is the spreadsheet, which is particularly well-suited to the numerical solution of both differential and integral equations. In this paper, Microsoft Excel 5.0 is used to solve a series of problems, including 1-D initial value problems (Runge-Kutta methods), 1-D boundary value

problems (finite difference methods), elliptic partial differential equations (successive overrelaxation), and Volterra and Fredholm integral equations.

Initial Value Problems

Because of Excel's macro language, a simple, 4th order Runge-Kutta algorithm can easily be implemented. For instance, to solve an ordinary differential equation of the form:

$$\frac{dy}{dt} = f(t, y)$$
$$y(0) = y_0$$

one can use a macro like the following:

```
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

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

This macro solves the particular case of *f(t,y)=2ty*. To use this macro, one merely opens a blank worksheet, inputs chosen values for the time step and initial values for *t* and *y*, and repeatedly calls this function to generate values for *y*. A simple spreadsheet using this macro might look like:

	A	B	C	D
1	**dt**		**t**	**y**
2	0.1		0	1
3			0.1	1.01005
4	**yo**		0.2	1.04081
5	1		0.3	1.09417

and the formulas that created this sheet might look like:

	A	B	C	D
1	**dt**		**t**	**y**
2	=0.1		=0	=A5
3			=C2+dt	=rk(C2,D2,dt)
4	**yo**		=C3+dt	=rk(C3,D3,dt)
5	=1		=C4+dt	=rk(C4,D4,dt)

Here I've assumed that cell A2 is given the name "dt," so that the reference to this value can be used by name in the formulas. This makes the formulas more readable and easier to debug. In Excel you can name a cell by clicking on that cell and then going to the menu selection *Insert/Name/Define…*

The advantage here is that students can write this fairly simple macro and then easily make repeated calls to it to generate a solution. Accompanying plots can also be included with minimal difficulty. Because there is no looping required, students can generate solutions such as this with little or no programming background. Later, when students have picked up some programming, they can write an additional macro that calls the `rk` macro repeatedly and writes the solution directly back to the spreadsheet. These principles are easily extrapolated to systems of equations and can be adapted to algorithms that vary the step size to achieve prescribed accuracy. Using this methodology of starting with simple macros and then graduating to more advanced algorithms, students can get involved with problem solving early in the learning process and are not required to first learn advanced programming.

1-D Boundary Value Problems

These problems are typically solved using two techniques: shooting methods and finite difference algorithms. Shooting methods can be used in a spreadsheet by combining the Runge-Kutta techniques described in the previous section with a simple root-finding algorithm or with the built in equation solver ("The Solver") in Excel.

To demonstrate the use of finite difference techniques to solve boundary value problems, I'll use the following model problem:

$$\frac{d^2y}{dx^2} - y = 1$$
$$y(0) = y(1) = 0$$

If we difference the operator in this equation using standard central difference techniques, we find:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} - y_i = 1$$

Solving this for y_i yields:

$$y_i = \frac{y_{i+1} + y_{i-1} - h^2}{2 + h^2}$$

This form allows us to implement an iterative technique to solve the differential equation. In this case, we take advantage of the built-in iterative capability of Excel. We merely place this formula into a series of cells, each depending on its two neighboring cells, and allow Excel to iterate until convergence is reached. A spreadsheet incorporating this scheme might look like the following:

	A	B	C	D
1	**h**		**x**	**y**
2	0.25		0	0
3			0.25	-0.08492
4			0.5	-0.11265
5			0.75	-0.08492
6			1.0	0

and the formulas would be:

	A	B	C	D
1	**h**		**x**	**y**
2	=0.25		=0	=0
3			=C2+h	=(D2+D4-h^2) / (2+h^2)
4			=C3+h	=(D3+D5-h^2) / (2+h^2)
5			=C4+h	=(D4+D6-h^2) / (2+h^2)
6			=C5+h	=0

Note that simply inputting these formulas into Excel will yield an error message of the form: "Can't Resolve Circular References." Circular references occur when two cells refer, either directly or indirectly, to one another. For example, in the sheet above, cell D3 refers to D4 and D4 refers back to D3. Excel can only solve such systems iteratively, but it must be told to do so. To accomplish this, simply click on *Tools/Options*, go to the *Calculation* tab, click the *Iteration* box, and click the *Manual* calculation box. This will set up Excel to iterate. You must press F9 (under Windows) to force calculation.

One advantage of this approach to solving boundary value problems is the ease with which students can alter the boundary conditions. For instance, if we desire a zero-slope boundary condition at x=1 in the above problem, we can simply enforce symmetry by creating an extra mesh point beyond the boundary and forcing it to equal the cell just inside the boundary. This is equivalent to a zero-slope condition. The sheet would thus appear as:

	A	B	C	D
1	**h**		**x**	**y**
2	=0.25		=0	=0
3			=C2+h	=(D2+D4-h^2) / (2+h^2)
4			=C3+h	=(D3+D5-h^2) / (2+h^2)
5			=C4+h	=(D4+D6-h^2) / (2+h^2)
6			=C5+h	=(D5+D7-h^2) / (2+h^2)
7				=D5

Obviously, many variations on this theme can be carried out to solve a wide variety of problems. I've found that students at the undergraduate level find this approach to be intuitive. The primary drawbacks to this method are that the convergence can be slow for large problems and that the mesh spacing is not easily changed. The logical step, then, is to follow this with a treatment in, for instance, Matlab, where the difference equations are converted to a matrix equation and the system is then solved. This is a big leap for undergraduate students and I've found that it takes a lot of practice for them to become proficient at this. It seems that our students lack experience with linear algebra and thus have difficulty dealing with linear systems of equations cast as matrix equations.

An additional advantage of the spreadsheet approach is that it can easily handle nonlinear equations.

Elliptic Partial Differential Equations

The solution of elliptic partial differential equations is also quite simple in a spreadsheet which can handle iteration. If we consider the model problem:

$$\frac{d^2\phi}{dx^2} + \frac{d^2\phi}{dy^2} = 1$$

with homogeneous boundary conditions on a unit square, one can difference this equation and, using equal spacing in both directions, obtain the following difference equation:

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - h\text{^}2}{4}$$

Again we simply set boundary values for the dependent variable and place this formula into all interior cells. Excel can then iterate to convergence. A spreadsheet to carry this out might look like:

	A	B	C	D	E	F
1	**h**	0.25				
2						
3		0	0	0	0	0
4		0	-0.04297	-0.05469	-0.04297	0
5		0	-0.05469	-0.07031	-0.05469	0
6		0	-0.04297	-0.05469	-0.04297	0
7		0	0	0	0	0

and the accompanying formulas would be:

	A	B	C	D	E	F
1	**h**	=0.2				
2						
3		=0	=0	=0	=0	=0
4		=0	=(B4+D4+C3+C5-h^2)/4	=(C4+E4+D3+D5-h^2)/4	similar	=0
5		=0	=(B5+D5+C4+C6-h^2)/4	=(C5+E5+D4+D6-h^2)/4	similar	=0
6		=0	=(B6+D6+C5+C7-h^2)/4	=(C6+E6+D5+D7-h^2)/4	similar	=0
7		=0	=0	=0	=0	=0

The formulas in column **E** have been left out to reduce the table size. They are similar to those in columns **C** and **D**.

This provides a simple solution, but convergence tends to be slow. One can implement a different scheme, successive overrelaxation for instance, simply by making small changes in the formulas. In the example just given, we might rewrite the algorithm as:

$$\phi_{i,j} = \phi_{i,j} + \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j} - h\text{^}2}{4}$$

and then insert an overrelaxation parameter as:

$$\phi_{i,j} = \phi_{i,j} + \omega\left(\frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j} - h\text{^}2}{4}\right)$$

Formulas for implementing this algorithm are:

	A	B	C	D	E
1	**h**	=0.2	**omega**	=1.26	
2					
3	=0	=0	=0	=0	=0
4	=0	=C4+omega*(B4+D4 +C3+C5-4*C4- h^2)/4	=D4+omega*(C4+E4 +D3+D5-4*D4-h^2)/4	=E4+omega*(D4+F4 +E3+E5-4*E4-h^2)/4	=0
5	=0	=C5+omega*(B5+D5 +C4+C6-4*C5- h^2)/4	=D5+omega*(C5+E5 +D4+D6-4*D5-h^2)/4	=E5+omega*(D5+F5 +E4+E6-4*E5-h^2)/4	=0
6	=0	=C6+omega*(B6+D6 +C5+C7-4*C6- h^2)/4	=D6+omega*(C6+E6 +D5+D7-4*D6-h^2)/4	=E6+omega*(D6+F6 +E5+E7-4*E6-h^2)/4	=0
7	=0	=0	=0	=0	=0

The value chosen for the overrelaxation parameter is optimized for equal mesh spacing on a square grid with 5 mesh points in each direction[1].

Integral Equations

Here we solve two types of integral equations: a Volterra integral equation of the second kind and a Fredholm integral equation of the second kind. The Volterra equation can be represented by

$$y(t) = f(t) + \int_0^t K(t,x)y(x)dx$$

where *f(t)* and *K(t,x)* are known functions. Solution of this general equation is cumbersome on a spreadsheet, but if we reduce it to the following form:

$$y(t) = f(t) + \int_0^t K(x)y(x)dx$$

then we can produce a simple spreadsheet that will solve the equation for general cases of *f(t)* and *K(x)*. In order to solve this equation, we can approximate the integral by a simple integration rule, such as the trapezoidal rule, and then use iteration to converge to a solution. A solution for *f(t)=K(x)=1* is given below.

	A	B	C	D	E	F
1	**dt**		**t**	**y(t)**	**K(t)**	**y*K(t)**
2	0.25		0	1	1	1
3			0.1	1.105263	1	1.105263

4			0.2	1.221607	1	1.221607
5			0.3	1.350197	1	1.350197
6			0.4	1.492323	1	1.492323

The formulas that produced these results are:

	A	B	C	D	E	F
1	**dt**		**t**	**y(t)**	**K(t)**	**y*K(t)**
2	=0.25		=0	1	=K(C2)	=D2*E2
3			=C2+dt	=f(C3)+dt*(F$2+F3)/2	=K(C3)	=D3*E3
4			=C3+dt	=f(C4)+dt*((F$2+F4)/2+F$3)	=K(C4)	=D4*E4
5			=C4+dt	=f(C5)+dt*((F$2+F5)/2+ SUM(F$3:F4))	=K(C5)	=D5*E5
6			=C5+dt	=f(C6)+dt*((F$2+F6)/2+ SUM(F$3:F5))	=K(C6)	=D6*E6

Here the cells in column D hold the solution algorithm, which simply adds *f(t)* to an approximation for the integral from *t=0* to the current time. This is an intuitive form and the iteration in Excel allows it to be solved in this way. Using more direct methods requires an additional step of converting the equations into a linear system.

In this sheet the functions *f(t)* and *K(t)* are assumed to be defined in a macro and thus can easily be changed. Note that the cells in column D are written in such a way that the formula in cell D5 can be copied down and the cell references will be updated appropriately. Cells D2, D3, and D4, though, are unique.

The Fredholm integral equation of the second kind can be represented by

$$y(t) = f(t) + \lambda \int_0^1 K(t,x)y(x)dx$$

This paper will only consider degenerate kernels of the form

$$K(t,x) = Kt(t) * Kx(x),$$

because, again, the general case is somewhat cumbersome on a spreadsheet. The spreadsheet to solve these equations is set up with the three functions *f(t), Kt(t),* and *Kx(x)* written as macro functions. This allows one to change these easily and recalculate the problem. The spreadsheet was set up as follows:

	A	B	C	D	E	F	G
1	**dt**						
2	0.25		**t**	**y(t)**	**kx(x)**	**y*kx(x)**	**kt(t)**
3			0	0	0	0	0
4	**lambda**		0.25	0.251572	0.25	0.062893	0.25
5	0.5		0.5	0.503145	0.5	0.251572	0.5
6			0.75	0.754717	0.75	0.566038	0.75
7			1	1.006289	1	1.006289	1

The formulas that produced these values are:

	A	B	C	D	E	F	G
1	**dt**						
2	=0.25		**t**	**y(t)**	**kx(x)**	**y*kx(x)**	**kt(t)**
3			=0	=f(C3)+((F$3+F$7)/2+ SUM(F$4:F$6))*dt*G3* lambda	=kx(C3)	=D3*E3	=kt(C3)
4			=C3+dt	=f(C4)+((F$3+F$7)/2+ SUM(F$4:F$6))*dt*G4* lambda	=kx(C4)	=D4*E4	=kt(C4)
5			=C4+dt	=f(C5)+((F$3+F$7)/2+ SUM(F$4:F$6))*dt*G5* lambda	=kx(C5)	=D5*E5	=kt(C5)
6			=C5+dt	=f(C6)+((F$3+F$7)/2+ SUM(F$4:F$6))*dt*G6* lambda	=kx(C6)	=D6*E6	=kt(C6)
7			=C6+dt	=f(C7)+((F$3+F$7)/2+ SUM(F$4:F$6))*dt*G7* lambda	=kx(C7)	=D7*E7	=kt(C7)

Again I use the trapezoidal rule to approximate the integral and iteration is used to solve the equations.

Conclusions

Modern spreadsheets, with their intuitive interfaces, provide an opportunity for students to study computational algorithms with minimal overhead associated with learning how to program them. This is convenient when teaching numerical methods, as it allows increased focus on the algorithms themselves and decreased focus on a particular implementation.

Availability of Software

Examples of spreadsheets described in this paper can be found at http://elvis.neep.wisc.edu/~jake/asee/paper97.html.

References

[1] William H. Press *et al.*, **Numerical Recipes in C: the art of scientific computing**, 2nd Edition, Cambridge University Press, 1982.

Biographical Information

James P. Blanchard is an Associate Professor of Nuclear Engineering and Engineering Physics at the University of Wisconsin - Madison.