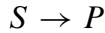


336: Systems and Control Computer Simulation (v1.0)

Contents

- 1 Numerical Solutions 4
 - 1.1 Euler Method 7
 - 1.2 Modified Euler or Heun Method 10
 - 1.3 Runge-Kutta 14
 - 1.4 Variable Step Size Methods 16
 - 1.5 Stiff Models 17
- 2 Matlab Solvers 18
- Further Reading 20
- Exercises 21

In this chapter we will look at ways to solve the differential equations that we generate when we build a model of a system. For example, consider the simplest possible model, the first-order irreversible degradation of a molecular species, S into product P :



The differential equation for this simple reaction is given by the familiar form:

$$\frac{dS}{dt} = -k_1 S \quad (1)$$

Our aim is to solve this equation so that we can describe how S changes in time. There are at least two ways to do this, we can either solve the equation mathematically or use a computer to obtain a numerical solution. Mathematically we can either solve the system using traditional methods from calculus or we can use the Laplace transform method.

To use the traditional method we move the dependent variables onto one side and in the independent variables onto the other, this is called the separation of variables. In the above equation one can easily do this by dividing both sides by S to give:

$$\frac{dS}{dt} \frac{1}{S} = -k_1 \quad (2)$$

In differential calculus, the derivative of $\ln y$ with respect to t is

$$\frac{d \ln y}{dt} = \frac{dy}{dt} \frac{1}{y}$$

This means we can rewrite equation 2 in the following form:

$$\frac{d \ln S}{dt} = -k_1$$

We now integrate both sides with respect to dt , that is:

$$\int \ln S dt = -k_1 \int dt$$
$$\ln S = -k_1 t + C$$

where C is the constant of integration. If we assume that at $t = 0$, $S = S_o$, then $\ln S_o = C$. Substituting this result into the solution gives:

$$\ln S = -k_1 t + \ln S_o$$

$$\ln \left(\frac{S}{S_o} \right) = -k_1 t$$

Taking anti-natural logarithms on both sides and multiply both sides by S_o gives:

$$S = S_o e^{-k_1 t}$$

For simple systems such as this, it is possible to obtain analytical solutions but very rapidly one is confronted with the fact that for 99.99% of problems, no mathematical solution exists. In such cases, we must carry out numerical simulations.

1 Numerical Solutions

In the last section we saw how it was possible to solve a differential equation mathematically. If the system of differential equation is linear there are systematic methods for deriving a solution. Most of the problems we encounter in biology however are non-linear

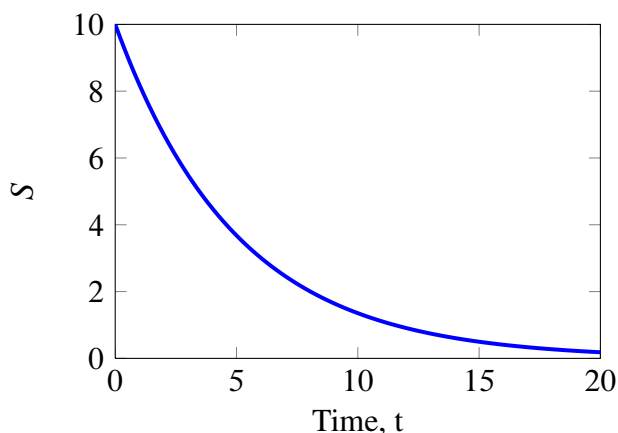


Figure 1: Exponential decay from the equation: $S = S_0 e^{k_1 t}$ where $S_0 = 10, k_1 = 0.2$.

and for such cases mathematical solutions rarely exist. Because of this, computer simulation is often used instead. Since the 1960s, almost all simulations have been carried out using digital computers. Before the advent of digital computers, analog computer were frequently used where an analog of the system was built using either mechanical or more commonly, electrical analogs of concentrations. Here we will focus on methods used on digital computers.

The general approach to obtaining a solution by computer is as follows:

1. Construct the set of ordinary differential equations, with one differential equation for every molecular species in the model.
2. Assign values to all the various kinetic constants and boundary species.
3. Initialize all floating molecular species to their starting concentrations.

4. Apply an integration algorithm to the set of differential equations.
5. If required, compute the fluxes from the computed species concentrations
6. Plot the results of the simulation.

Step four is obviously the key to the procedure and there exist a great variety of integration algorithms. We will describe three common approaches to give a flavor of how they work. Other than for educational purposes (which is significant), it is rare now for a modeler to write their own integration computer code because many libraries and applications now exist that incorporate excellent integration methods. Chapter ?? will discuss more fully the available options for software. Here we will focus on some of the the approaches themselves.

An integration algorithm approximates the behavior of what is, strictly speaking, a continuous system, on a digital computer. Since digital computers can only operate in discrete time, the algorithms convert the continuous system into a discrete time system. This is the reason why digital computers can only generate approximations. In practice a particular discrete step size, h , is chosen, and solution points are generated at the discrete points up to some upper lime limit. As we will discover, the approximation generated by the simplest methods is dependent on the size of the step size and in general the smaller the step size the more accurate the solution. However since computers can only represent numbers to a given precision (usually 15 digits on modern computers), it is not possible to continually reduce the step step in the hope of increasing the accuracy of the solution. For one thing, the algorithm will soon reach the limits of the precision of the computer and secondly, the smaller the step size the long it will take to compute the solution. There is therefore often a tradeoff that is made between accuracy and computation time.

Let us first consider the simplest method, the Euler method, where the tradeoff between accuracy and computer time can be clearly seen.

1.1 Euler Method

The Euler method is the simplest possible way solve a set of ordinary differential equations. The idea is very simple. Consider the following differential equation that describes the degradation rate of a species, S :

$$\frac{dS}{dt} = -k_1 S$$

The Euler method uses the rate of change of S to predict the concentration at some future point in time. Figure 2 describes the method in detail. At time t_1 , the rate of change in S is computed from the differential equation using the known concentration of S at t_1 . The rate of change is used to compute the change in S over a time interval h , using the relation, $h \, dS/dt$. The current time, t_1 is incremented by the time step, h and the procedure repeated again, this time starting at t_2 . The method can be summarized by the following two equations which represent one step in an iteration that repeats until the final time point is reached:

$$\begin{aligned} y(t+h) &= y(t) + h \frac{dy(t)}{dt} \\ t_{n+1} &= t_n + h \end{aligned} \tag{3}$$

Figure 2 also highlights a problem with the Euler method. At every iteration, there will be an error between the change in S we predict and what the change in S should have been. This error is called the **truncation error** and will accumulate at each iteration. If the

step size is too large, this error can make the method numerically unstable resulting in wild swings in the solution.

Figure 2 also suggests that the larger the step size the larger the truncation error. This would seem to suggest that the smaller the step size the more accurate the solution will be. This is indeed the case, up to a point. If the step size becomes too small there is the risk that roundoff error which will propagate at each step into the solution. In addition, if the step size is too small it will require a large number of iterations to simulate even a small time period. The final choice for the step size is therefore a compromise between accuracy and effort. A theoretical analysis of error propagation in the Euler method indicates that the error accumulated over the entire integration period (called the **global error**) is proportional to the step size. Therefore halving the step size will reduce the global error by half. This means that to achieve even modest accuracy, small step sizes are necessary. As a result, the method is rarely used in practice. The advantage of the Euler method is that it is very easy to implement in computer code or even on a spreadsheet.

The Euler method can also be used to solve systems of differential equations. In this case all the rates of change are computed first followed by the application of the Euler equation 3. As in all numerical integration methods, the computation must start with an initial condition for the state variables at time zero. The algorithm is described using pseudo-code in Algorithm 1.

Example 1

Solve the decay differential equation 1 using the Euler method. Assume $k_1 = 0.2$ and the concentration of S_o and P are time = 0 is 10 and 0 respectively. Assume a step size, h , of 0.4. Form a table of four columns, write out the solution to two three decimal places. The 4th column should include the exact solution for comparison.

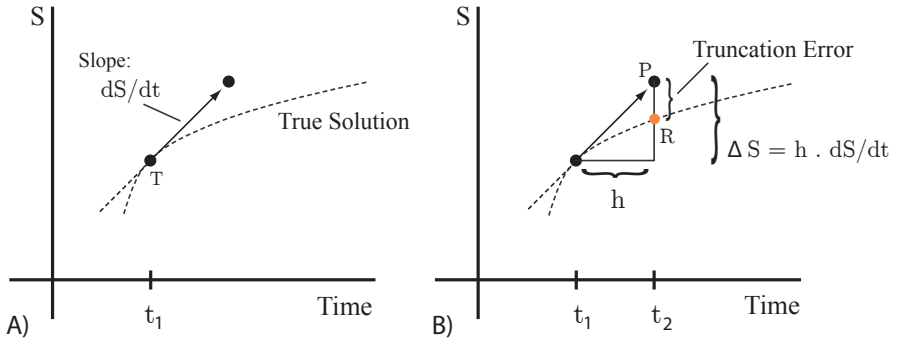


Figure 2: Euler Method. Starting at t_1 , the slope dS/dt at t_1 is computed (Panel A). The slope is used to project forward to the next solution in time step, h , to t_2 (Panel B). The new solution at t_2 is indicated by P . However the solution is given by point R , located on the solution curve at t_2 . Reducing the step size h will reduce the error between the exact and the projected solution, but will simultaneously increase the number of slope projections necessary to compute the solution over a given time period.

Time	Solution (S)	dS/dt	Exact Solution
0	10	2	10
0.4	9.2	1.84	9.23
0.8	8.464	1.6928	8.52
1.2	7.787	0.01	7.87
...			

Table 1: Solution Table

Figure 3 shows the effect of different steps sizes on the Euler method. Four cases are shown, in the worse case the solution is unbounded and the computer will eventually crash with an overflow error. The second case is where the result is bounded but the solution bears no

```

n = Number of state variables
yi = ith variable
Set timeEnd
currentTime = 0
h = stepSize
Initialize all yi at currentTime

while currentTime < timeEnd do
  for i = 1 to n do
    dyi = fi(y)

  for i = 1 to n do
    yi(t + h) = yi(t) + h dyi

  currentTime = currentTime + h
end while

```

Algorithm 1: Euler Integration Method, $f_i(y)$ represents the i^{th} differential equation from the system of ordinary differential equations.

resemblance at all to the actual solution. The third case shows that the solution is beginning to resemble the actual solution but irregularities appear near the beginning of the integration. The final case shows the actual solution generated from a specialized integrator.

1.2 Modified Euler or Heun Method

As indicated in the last section, the Euler method, though simple to implement, tends not to be used in practice because it requires small step sizes to achieve reasonable accuracy. In addition the small step size makes the Euler method computationally slow. A simple modification however can be made to the Euler method to significantly

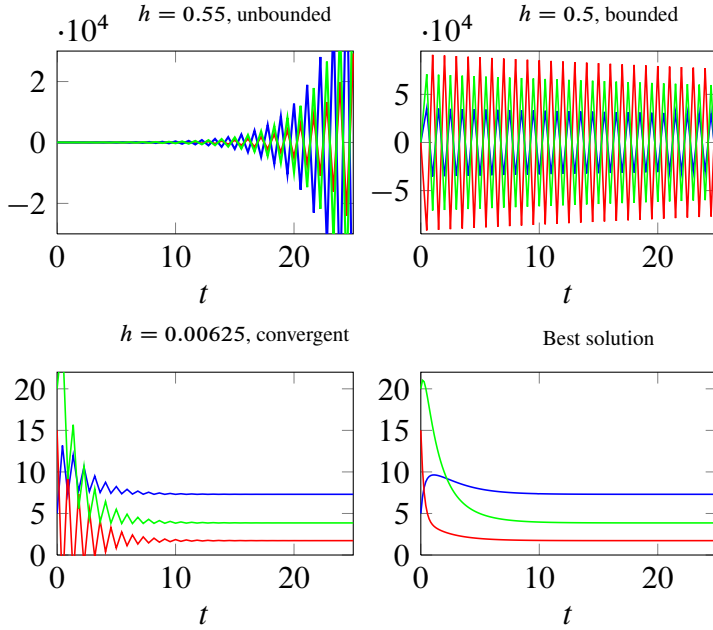


Figure 3: Effect of different step sizes on the Euler method using a simple linear chain of reactions where each reaction follows reversible mass-action kinetics: $X_o \xrightleftharpoons[k_2]{k_1} S_1$, $S_1 \xrightleftharpoons[k_4]{k_3} S_2$, $S_2 \xrightleftharpoons[k_6]{k_5} S_3$, $S_3 \xrightleftharpoons[k_7]{} X_1$ where $k_1 = 0.45, k_2 = 0.23, k_3 = 0.67, k_4 = 1.2, k_5 = 2.3, k_6 = 0.3, k_7 = 0.73, X_o = 10, X_1 = 0, S_1 = 5, S_2 = 15, S_3 = 20$.

improve its performance. This approach can be found under a number of headings, including the modified Euler method, the Heun or the improved Euler method.

The modification involves improving the estimate of the slope by averaging two derivatives, one at the initial point and another at the end point. In order to calculate the derivative at the end point, the first derivative must be used to predict the end point which is then corrected by using the averaged slope (Figure 4). This method is a very simple example of a predictor-corrector method. The method can be summarized by the following equations:

$$\text{Predictor: } y(t+h) = y(t) + h \frac{dy(t)}{dt} \quad (4)$$

$$\text{Corrector: } y(t+h) = y(t) + \frac{h}{2} \left(\frac{dy(t)}{dt} + \frac{dy(t+h)}{dt} \right) \quad (5)$$

$$t_{n+1} = t_n + h \quad (6)$$

Figure 4 describes the Heun method graphically.

A theoretical analysis of error propagation in the Heun method shows that it is a second order method, that is if the step size is reduced by a factor of 2, the global error reduced by a factor of 4. However, to achieve this improvement, two evaluations of the derivatives is required per iteration, compared to only one for the Euler method. Like the Euler method the Heun method is also quite easy to implement.

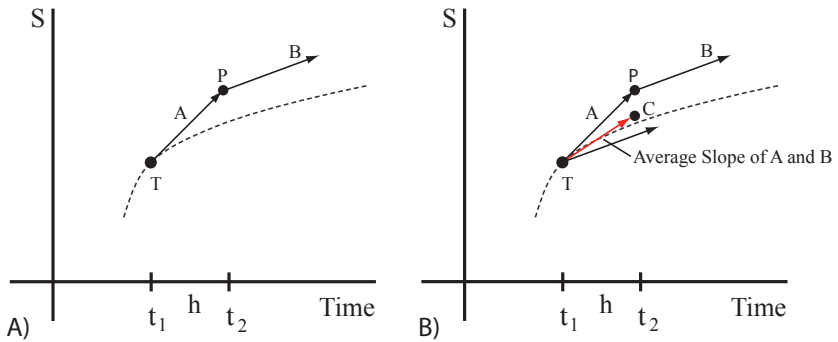


Figure 4: Heun Method. Starting at t_1 , the slope A at T is computed. The slope is used to predict the solution at point P using the Euler method. From point P the new slope, B is computed (Panel A). Slopes A and B are now averaged to form a new slope C (Panel B). The averaged slope is used to compute the final prediction.

```

n = Number of state variables
yi = ith variable
Set timeEnd
currentTime = 0
h = stepSize
Initialize all yi at currentTime

while currentTime < timeEnd do
  for i = 1 to n do
    ai = fi(y)

  for i = 1 to n do
    bi = fi(y + h a)

  for i = 1 to n do
    yi(t + h) = yi(t) +  $\frac{h}{2}$  (ai + bi)

  currentTime = currentTime + h
end while

```

Algorithm 2: Heun Integration Method. $f_i(y)$ is the i^{th} ordinary differential equation

1.3 Runge-Kutta

The Heun method described in the previous section is sometimes called the RK2 method where RK2 stands for 2nd order Runge-Kutta method. The Runge-Kutta methods are a family of methods developed around the 1900s by the German mathematicians, Runge and Kutta. In addition to the 2nd order Heun method there are also 3rd, 4th and even 5th order Runge-Kutta methods. For hand coded numerical methods, the 4th order Runge-Kutta algorithm (often called RK4) is probably the most popular among modelers. The algorithm is a little more complicated in that it involves the evaluation and weighted averaging of four slopes. In terms of global error, however, RK4 is considerably better than Euler or the Heun method and has a global error of the order of four. This means that halving the step size will reduce the global error by a factor of 1/16. Another way of looking at this is that the step size can be increased 16 fold over the Euler method and still have the same global error. The method can be summarized by the following equations which have been simplified by removing the dependence on time:

$$k_1 = h f(y_n)$$

$$k_2 = h f\left(y_n + \frac{k_1}{2}\right)$$

$$k_3 = h f\left(y_n + \frac{k_2}{2}\right)$$

$$k_4 = h f(y_n + k_3)$$

$$y(t+h) = y(t) + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)$$

$$t_{n+1} = t_n + h$$

Figure 5 shows a comparison of the three methods, Euler, Heun and RK4 in solving the Van der Pol equations. The Van der Pol equations are a classic problem set that is often used when comparing numerical methods. The equations model an oscillating system, inspired originally from modeling vacuum tubes but also later formed the basis for developments in modeling action potentials in neurons. The Figure shows that the Heun and RK4 methods are very similar, at least for the Van der Pol equations, though this is not always be the case. For this particular model the solution generated by the RK4 method is very similar to the best possible solution that can be obtained by numerical solution.

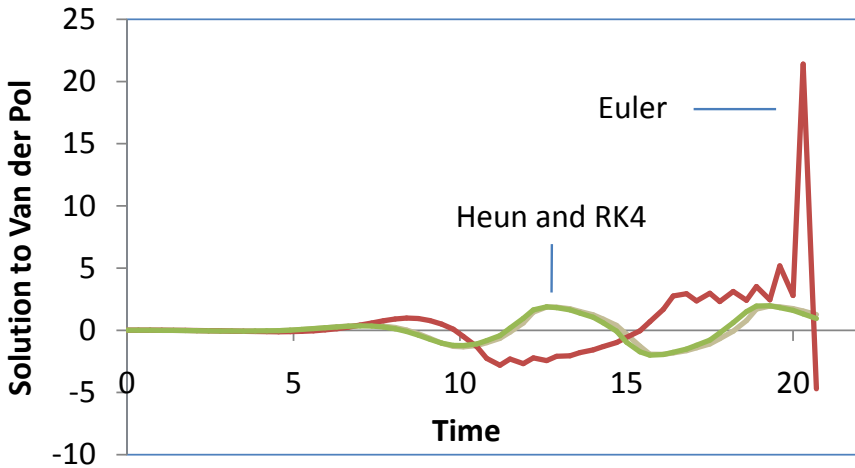


Figure 5: Comparison of Euler, Heun and RK4 numerical methods at integrating the Van der Pol dynamic system: $dy_1/dt = y_2$ and $dy_2/dt = -y_1 + (1 - y_1 y_1)y_2$. The plots show the evolution of y_1 in time. The RK4 solution is almost indistinguishable from solutions generated by much more sophisticated integrators. Step size was set to 0.35.

1.4 Variable Step Size Methods

In the previous discussion of numerical methods for solving differential equations, the step size, h , was assumed to be fixed. This makes implementation quite straight forward but also make the methods inefficient. For example, if the solution is at a point where it changes very little then the method could probably increase the step size without losing accuracy while at the same time achieve a considerable speedup in the time it takes to generate the solution. Likewise if at a certain point in the integration the solution starts to change rapidly, it would be prudent to lower the step size to increase accuracy. Such strategies are implemented in the **variable step size methods**.

The approach used to automatically adjust the steps size can vary from quite simple approaches to very sophisticated methods. The simplest approach is to carry out two integration trials, one at a step size of h and another trial using two steps size but at $h/2$. The software now compares the solution generated by the two trials. If the solutions are significantly different then the step size must be reduced. If the solutions are about the same then it might be possible to increase the step size. These tests are repeatedly carried out, adjusting the step size as necessary as the integration proceeds. This simple variable step size approach can be easily incorporated into some of the simpler algorithms particularly the fourth order Runge-Kutta where it is called the variable step-size Runge-Kutta. Another approach to adjusting the step size is called the Dormand-Prince method. This method carries out two trials based on the fourth and fifth order Runge-Kutta. Any difference between the two trials is used to adjust the step size. Matlab's ode45 implements the Dormand-Prince method. Similar methods to Dormand-Prince include the Fehlberg and more recently the Cash-Karp method.

Many of these simple adjustable step size solvers are quite effective. Sometimes they can be slow however especially for the kinds of problem we find in biochemical models. In particular there is a class of problem called stiff problems which generally plagues the biochemical modeling community. Stiff models require highly specialized solvers which have been developed in the last four decades.

1.5 Stiff Models

Many differential equations we encounter in biochemical models are so-called **stiff** systems. The word stiff apparently comes from earlier studies on spring and mass systems where the springs had large spring constants and therefore difficult to stretch. A stiff system is often associated with widely different time scales in a system, for example when the rate constants are widely different in a biochem-

ical model. Such systems may have molecular species whose decay rates are very fast compared to other components. This means that the step size has to be very small to accommodate the fast processes even though the rest of the system could be accurately solved using a much larger step size. The overall result is the need for very small steps sizes and therefore significant computational cost and the possibility of roundoff error which will tend to be amplified by the large time constants in the fast system. The net result are solutions which bear no resemblance to the true solution.

Most modern simulators will employ specific stiff algorithms for solving stiff differential equations. Of particular importance is the sundials suite and odepack. Sundials includes a number of very useful, well written and documented solvers. In particular the CVODE solver is very well suited to finding solutions to stiff differential equations. As a result sundials is widely used in the biochemical modeling community (for example by Jarnac and roadRunner). Before the advent of sundials, the main workhorse for solving stiff systems was the suite of routines in odepack. Of particular importance was LSODA which in the 1990s was very popular and is still a valuable set of software (currently used in COPASI). The original stiff differential equation solver was developed by Gear in the 1970s and is still used in Matlab in the form of ode15s. We will return to a fuller discussion of software tools in chapter ??.

2 Matlab Solvers

Although this isn't a book about Matlab, given how popular it is in engineering circles a few words will be said about using Matlab to solve differential equations.

Matlab offers a range of solvers to solve ordinary differential equations. Possibly the two most common solvers use are the ode45 and ode15s solvers.

The ode45 solver implements a variable step size Runge-Kutta method. The variable step size is achieved by comparing solutions using a fourth and fifth order Runge-Kutta. If the error between the two methods is too big, then the step size is reduced, otherwise if the step size is below a given threshold then the program will attempt to increase the step size. This method is also called the Dormand-Prince method. The basic syntax for ode45 is:

```
[t,y] = ode45(@myModel, [t0, tend], yo, [], p);
```

where

myModel is the function containing the differential equations

t0, tend is the initial and final values for the independent variable, t .

yo is a vector of initial conditions

p is set of parameters for the model, this can be any size.

The empty vector in the call is where additional options particular to ode45 can be placed.

For example, to solve the set of ODEs:

$$\frac{dy_1}{dt} = v_o - k_1 y_1$$

$$\frac{dy_2}{dt} = k_1 y_1 - k_2 y_2$$

We would write the following .m file and load it into Matlab

```
function dy = myModel(t, y, p)
dy = zeros (2,1);
vo = p(1);
k1 = p(2);
k2 = p(3);
dy(1) = vo - k1 y(1);
dy(2) = k1 y(1) - k2 y(2);
```

We would then call the solver as follows:

```
p = [10, 0.5, 0.35]
y0 = [0, 0]
[t, y] = ode45 (@myModel, [0, 20], y0, [], p)
```

The alternative to `ode45` is `ode15s`. Although many problems can be solved using `ode45` for some models that are stiff, `ode45` will be insufficient and will fail to give the correct solution. In these cases `ode15s` is recommended. `ode15s` is a variable order solver and amongst other things it uses the well know method called Gear's method. Like `ode45`, `ode15s` is also a variable step size method. `ode45` might be faster than `ode15s` on simple problems but with today's fast computers the difference is hardly noticeable. Therefore one might as well use `ode15s` for all problems. `ode15s` is called in the same way as `ode45`.

3 Other Software

Although sometimes it may seem to be the case, Matlab isn't the only software that can be used to solved differential equations. For example Mathematica is an example of another commercial tool that can be used to solve differential equations. For those who require more control or who are unable to purchase a commercial tool there are many free applications and professionally developed open source software libraries. Octave (<http://www.gnu.org/software/octave/>) is an open source tool that is very similar to Matlab, in fact even the syntax is extremely similar if not identical. SciLab (<http://www.scilab.org/>) is another free Matlab like application. If you like programming in Python then Sage (<http://www.sagemath.org/index.html>) is an excellent option. There are therefore many alternative and often free options to using Matlab.

For those who require much more control and higher performance than can be achieved by the tools mentioned above then there is the Sundials C/C++ library developed by the Department of Energy. In particular within Sundials there is the CVODE library that is in fact used by many of the commercial tools. CVODE implements an advanced Gear like algorithm using a variable order and variable step size approach. It is highly suited for stiff systems and is the preferred method for those who need to write their own code. One final library worth mentioning is the GPL (GNU General Public License) licensed GSL library (<http://www.gnu.org/software/gsl/>). Although very comprehensive the GPL license restricts its use to other GPL licensed source code and is therefore its ODEs solvers cannot be used as easily as the CVODE library.

Further Reading

Unfortunately there are very few reasonably priced books on numerical analysis. The two most popular expensive books are by Press and Burden and are included here for reference. Both books can be bought second-hand at reasonable prices and the content has not changed significantly between editions. One could even argue that the code examples in the latest edition of Press are actually worse than the previous editions.

1. Press, Teulolsky, Vetterling and Flannery (2007) Numerical Recipes. Cambridge University Press, 3rd Edition ISBN-10: 0521880688
2. Burden and Faires (2010) Numerical Analysis. Brooks Cole, 9th Edition. ISBN-10: 0538733519

For the budget conscious buyer I can highly recommend the Dover edition:

1. Dahlquist and Björck (2003) Numerical Methods. Dover Publications ISBN-10: 0486428079

Exercises

1. Implement the Euler method in your favorite computer language and use the code to solve the following two problems. Set initial conditions, $S_1 = 10, S_2 = 0$. Set the rate constants to $k_1 = 0.1; k_2 = 0.25$. Investigate the effect of different steps sizes, h , on the simulation results.

a) $dS_1/dt = -k_1 S_1$

b) $dS_1/dt = -k_1 S_1; dS_2/dt = k_1 S_1 - k_2 S_2$

2. The following model shows oscillations at a step of $h = 0.3$ when using the Euler method to solve the differential equations. By using simulation, show that these oscillations are in fact an artifact of the simulation.
3. Solve the following problem using the Euler method. What step size do you need to secure a reliable solution? Run the same problem on a specialist application such as Matlab or SBW. Compare the time take to carry out both simulations.
- 4.

```

n = Number of state variables
yi = ith variable
timeEnd = 10
currentTime = 0
h = stepSize
Initialize all yi at currentTime

while currentTime < timeEnd do
  for i = 1 to n do
    k1i = hf(yi)

  for i = 1 to n do
    k2i = hf(yi + k1i/2)

  for i = 1 to n do
    k3i = hf(yi + k2i/2)

  for i = 1 to n do
    k4i = hf(yi + k3i)

  for i = 1 to n do
    yi(t + h) = yi(t) +  $\frac{h}{6} (k_{1i} + 2 k_{2i} + 2 k_{3i} + k_{4i})$ 

  currentTime = currentTime + h
end while

```

Algorithm 3: 4th Order Runge-Kutta Integration Method