Numerical Methods for the Solution of ODEs

References

Numerical Recipes in Fortran 77
William Press, Saul Teukolsky, William Vetterling, & Brian Flannery
Cambridge University Press, 1992

Computational Methods in Chemical Engineering
Owen Hanna & Orville Sandall
Prentice-Hall, 1995

Problem Solving in Chemical Engineering with Numerical Methods
Michael B. Cutlip & Mordechai Shacham
Prentice Hall, 1999

Applied Numerical Methods
Brice Carnahan, H. A. Luther, & James O. Wilkes
John Wiley & Sons, 1969

Nonlinear Analysis in Chemical Engineering
Bruce A. Findlayson
McGraw-Hill, 1980

Numerical Methods
Germund Dahlquist, Åke Björk, & Ned Anderson
Prentice-Hall, 1974

Computer Methods for Mathematical Computations
George E. Forsythe, Michael A. Malcolm, & Cleve B. Moler
Prentice-Hall, 1977

Numerical Initial Value Problems in Ordinary Differential Equations
C. William Gear
Prentice-Hall, 1971

Introduction

We would like to find a numerical solution to a single non-linear 1st order ODE (ordinary differential equation) of the form:

$$F\left(\frac{dy}{dt}, y, t\right) = 0$$

or:

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

The basic philosophy is to start with a known value of $y(t)$ and then approximate the next value $y(t + h)$ using a finite difference approximation to the derivative dy/dt. For a 1st order finite difference:

$$\frac{dy}{dt} \approx \frac{y^{(n+1)} - y^{(n)}}{h} = f(y^*, t^*) \implies y^{(n+1)} = y^{(n)} + h \cdot f(y^*, t^*)$$

where $y^{(n)}$ refers to the value of y from the numerical scheme at the *n*-th time step. The trick is to find some appropriate values for h and $f(y^*, t^*)$ that lead to high accuracy & stability without requiring an unmanageable number of function evaluations and time steps.

The techniques that we will look at will also be applicable to systems of non-linear 1st order ODEs:

$$\frac{dy_i}{dt} = f_i(y_1, y_2, \ldots, y_N) \text{ for } i = 1, 2, \ldots, N$$

or in vector notation:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y})$$

where $\mathbf{y}$ refers to the set of variables $[y_1, y_2, \ldots, y_N]^T$ and $\mathbf{f}$ refers to the set of defining functions $[f_1, f_2, \ldots, f_N]^T$. Note that a dependency on time can always included by defining a new variable:

$$\frac{dy_{N+1}}{dt} = 1$$

Also, initial value problems of order higher than 1st order can always be recast as a system of 1st order equations by defining new variables to account for the higher order derivatives. For example, the single 3rd order ODE:

$$\frac{d^3 y}{dt^3} + 4\frac{d^2 y}{dt^2} + 5\frac{dy}{dt} + 2y = 2\sin(t) \text{ with } y(0) = y'(0) = y''(0) = 0.$$

Can be recast with the following definitions:

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = y_3$$

$$\frac{dy_4}{dt} = 1$$

so the original ODE becomes:

$$\frac{dy_3}{dt} = 2\sin(y_4) - 4y_3 - 5y_2 - 2y_1$$

with $y_1(0) = y_2(0) = y_3(0) = y_4(0) = 0$. Now the single ODE has become the following system of equations:

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ 2\sin(y_4) - 4y_3 - 5y_2 - 2y_1 \\ 1 \end{bmatrix}$$

As with a single equation, we approximate the $d\mathbf{y}/dt$ terms using 1st order finite differences:

$$\overline{\mathbf{y}}^{(n+1)} = \overline{\mathbf{y}}^{(n)} + h \cdot \overline{\mathbf{f}}\left(\overline{\mathbf{y}}^*\right).$$

Accuracy, Stability, and Step Size

Accuracy is a measure of how closely our numerical solution $y^{(n)}$ matches our "true" solution of $y\left(t^{(n)}\right)$. We will refer to two types of accuracy. The first, **_local accuracy_**, is the accuracy of the numerical method over a single time step. When these numerical methods are used, we will take multiple time steps. The errors from each time step will tend to accumulate. The accuracy of a numerical method over many time steps will be referred to as the **_global accuracy_**. We will sometimes denote this global error as $\varepsilon^{(n)}$ where:

$$y^{(n)} \equiv y\left(t^{(n)}\right) + \varepsilon^{(n)} \quad \Rightarrow \quad \varepsilon^{(n)} = y^{(n)} - y\left(t^{(n)}\right)$$

where $y\left(t^{(n)}\right)$ is the true solution at time $t^{(n)}$.

Stability is a somewhat ambiguous term and may appear with such qualifiers as inherent, partial, relative, weak, strong, absolute, etc. In general, a solution is said to be ***unstable if errors introduced at some stage of the calculations are propagated and magnified without bound throughout the subsequent calculations***. An unstable result will not only be inaccurate but also will be patently absurd. We can show that a criteria for stability is:

$$\left| \frac{\varepsilon^{(n+1)}}{\varepsilon^{(n)}} \right| \leq 1 .$$

The issues of accuracy and stability are important when trying to choose one or more values for a step size for integration. We will generally need a step size much smaller than the range of time in which we are interested, so many time steps must be performed to get our numerical answer. The step size chosen should be ***small enough to achieve our desired accuracy***, but should be as ***large as possible to avoid an excessive number of derivative evaluations***.

In general, accuracy relates to the characteristics of the method when making the step size smaller and smaller, whereas stability relates to the characteristics of the method when making the step size larger and larger.

Forward Euler Method

The simplest method is to evaluate the derivative using the values at the beginning of the time step. The recursion formula is then:

$$y^{(n+1)} = y^{(n)} + h \cdot f\left(y^{(n)}, t^{(n)}\right) = y^{(n)} + h \cdot f^{(n)}$$

where $f^{(n)}$ refers to the derivative function evaluated using the *n*-th time step values. This method is often referred to as the ***forward Euler method***. The method is simple and ***explicit*** (i.e., requires no iteration to get $y^{(n+1)}$). Its disadvantages are that it is not very accurate ($O(h)$ global accuracy) & can have stability problems with time steps that are too large.

Let's look at numerically solving the ODE:

$$\frac{dy}{dt} = 1 - 2y \text{ with } y(0) = 0 .$$

Note that there is an analytical solution:

$$y(t) = \frac{1}{2}\left(1 - e^{-2t}\right).$$

The forward Euler recursion formula will be:

$$y^{(n+1)} = y^{(n)} + h\left(1 - 2y^{(n)}\right) = h + (1 - 2h)y^{(n)}.$$

The following chart shows the numerically approximated solution at various step sizes as compared to the analytical solution. Notice that at a reasonably small step size of $h = 0.1$ the numerical solution is pretty good — the "knee" of the curve is pretty close & we get to the correct ultimate values. However, the larger step sizes give unreasonable values. $h = 0.5$ gets to the ultimate value in only a single step & $h = 0.9$ oscillates. In fact, $h = 1$ oscillates between 1 and 0 while $h > 1$ oscillates in a diverging manner.

Forward Euler Solution to $y' = 1 - 2y$, $y(0) = 0$
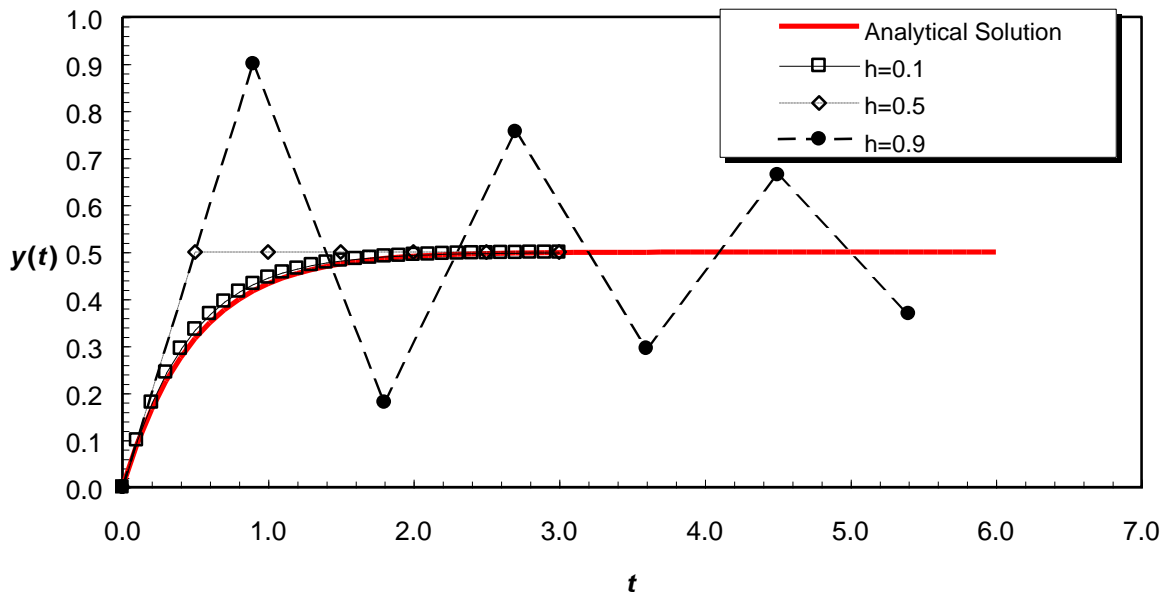
The problems with $h > 1$ show the ___stability___ problems of the forward Euler method. Errors introduced at early time steps, instead of dying out, start to grow and dominate the numerical solution. This shows that the forward Euler method is ___conditionally stable___ for small enough step sizes. For this problem, the stability limit is $h = 1$. However, what if we just wait until the end of the problem to change to a large step size. The following chart shows what happens if we use a step size of $h = 0.1$ until $t = 2$ & then use a step size of

$h = 1.1$. Notice that even though we are almost at the ultimate value, the large step size is still unstable.

Forward Euler Solution to y' = 1 - 2y, y(0) = 0
h = 0.1 for t < 2
h = 1.1 for t > 2
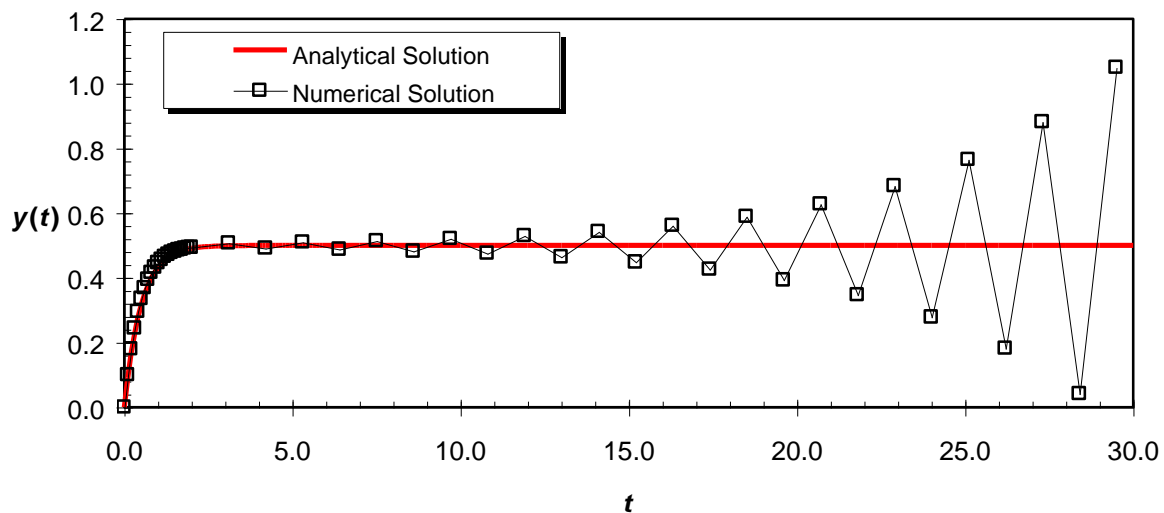
Backward Euler Method

For the forward Euler method, we evaluated the derivative function at the beginning of the time step, $f(y^{(n)}, t^{(n)})$. But we can evaluate f at any appropriate position. If we use the value at the end of the time step we get:

$$y^{(n+1)} = y^{(n)} + h \cdot f\left(y^{(n+1)}, t^{(n+1)}\right) = y^{(n)} + h \cdot f^{(n+1)}.$$

This method is often referred to as the ***backward Euler method***. The method is still simple, but it is ***implicit*** (i.e., it may require iteration to get $y^{(n+1)}$). It only has $O(h)$ global accuracy just like the forward Euler method, but it has much better stability characteristics. For our example, we now have:

$$y^{(n+1)} = y^{(n)} + h\left(1 - 2y^{(n+1)}\right) \implies y^{(n+1)} = \frac{y^{(n)} + h}{1 + 2h}.$$

Note that in this particular case we can solve directly for $y^{(n+1)}$, but this is not always the case. The following chart shows the numerical solution at various step sizes as compared to the analytical solution. Notice that at a reasonably small step size of $h = 0.1$ the numerical

solution is pretty good just like for the backward Euler method — the "knee" of the curve is pretty close & we get to the correct ultimate values. The biggest difference is that even though larger step sizes are not very accurate, the results do not oscillate or diverge. It can be shown that the backward Euler method is unconditionally stable for any step size.

Backward Euler Solution to $y' = 1 - 2y$, $y(0) = 0$
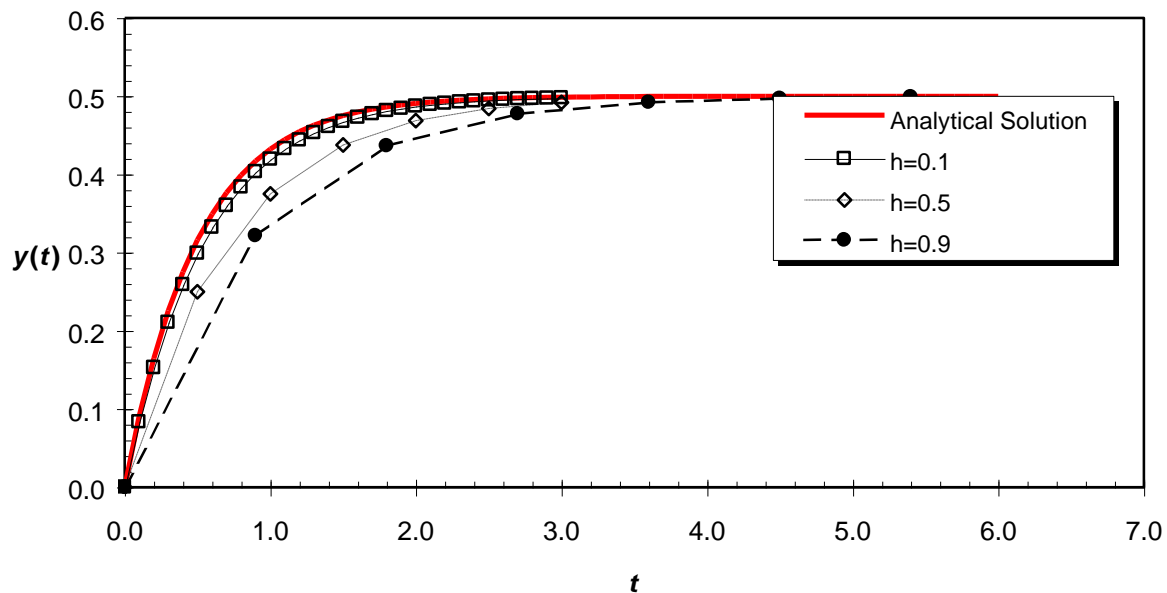
Comparison of the Stability Characteristics of the Euler Methods

A common method used to examine the stability characteristics of a numerical method is to determine what happens when the method is applied to the test equation:

$$\frac{dy}{dt} = -\lambda y \text{ with } y(0) = 1$$

where λ is real & positive. This test equation has the analytical solution

$$y(t) = 1 - e^{-\lambda t} .$$

If we expression the solution as the sum of the exact solution and the error ε, then the error must also satisfy:

$$\frac{d\varepsilon}{dt} = -\lambda\varepsilon.$$

We can now see what happens to the error along a time step. For the forward Euler method:

$$\varepsilon^{(n+1)} = \varepsilon^{(n)} + h\left(-\lambda\varepsilon^{(n)}\right)$$
$$\varepsilon^{(n+1)} = \varepsilon^{(n)}\left(1 - \lambda h\right)$$

So, for stability:

$$\left|\frac{\varepsilon^{(n+1)}}{\varepsilon^{(n)}}\right| = \left|\left(1 - \lambda h\right)\right| \leq 1 \quad \Rightarrow \quad 0 \leq h \leq \frac{2}{\lambda}$$

Only values of $h \leq \lambda/2$ will give stable results. This is why the forward Euler method is _**conditionally stable**_.

Remember our example problem? We found that the numerical solution oscillated up to $h = 1$ and then was unstable for $h > 1$. From this relationship above, the example problem must have had $\lambda = 2$ (remember the term e^{-2t} which corresponds to the term $e^{-\lambda t}$ in the test problem?). Also note:

$$\varepsilon^{(n+1)} = -\left|\left(1 - \lambda h\right)\right|\varepsilon^{(n)} \text{ when } \frac{1}{\lambda} < h \leq \frac{2}{\lambda}.$$

The error will not necessarily grow, but it will change sign, leading to an oscillatory behavior that is not present in the exact solution. For some problems these oscillations may be noticeable and unacceptable.

Now lets do a stability analysis for the backward Euler method:

$$\varepsilon^{(n+1)} = \varepsilon^{(n)} + h\left(-\lambda\varepsilon^{(n+1)}\right)$$
$$\varepsilon^{(n+1)} = \frac{\varepsilon^{(n)}}{\left(1 + \lambda h\right)}$$

So, for stability:

$$\left|\frac{\varepsilon^{(n+1)}}{\varepsilon^{(n)}}\right| = \left|\frac{1}{\left(1 + \lambda h\right)}\right| \leq 1 \quad \Rightarrow \quad h \geq 0$$

Now, all values of h will give stability. This is why the backward Euler method is ***unconditionally stable***. This was shown in our example problem.

Taylor Series Expansions & Order of Accuracy

We can use Taylor series expansions to create new integration schemes and determine their order of accuracy. Remember that the Taylor series expansion of a function around $y(t)$ is:

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2}y''(t) + \frac{h^3}{6}y'''(t) + \cdots$$

We could also have written the expansion around $y(t+h)$:

$$y(t) = y(t+h) - hy'(t+h) + \frac{h^2}{2}y''(t+h) - \frac{h^3}{6}y'''(t+h) + \cdots.$$

Not all of the derivatives of $y(t)$ are explicitly known. We may need to compute the derivatives needed in the Taylor series expansion from the defining equation itself. Since:

$$y'(t) = \frac{dy}{dt} = f(y,t)$$

and:

$$df = \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt \quad \Rightarrow \quad \frac{df}{dt} = \frac{\partial f}{\partial y}\frac{dy}{dt} + \frac{\partial f}{\partial t} = \frac{\partial f}{\partial y}f + \frac{\partial f}{\partial t}$$

then:

$$y''(t) = \frac{d^2 y}{dt^2} = \frac{df}{dt} = \frac{\partial f}{\partial y}f + \frac{\partial f}{\partial t}$$

$$y'''(t) = \frac{d^3 y}{dt^3} = \left(\frac{\partial^2 f}{\partial y^2}\frac{dy}{dt} + \frac{\partial^2 f}{\partial y \partial t}\right)f + \frac{\partial f}{\partial y}\left(\frac{\partial f}{\partial y}\frac{dy}{dt} + \frac{\partial f}{\partial t}\right) + \left(\frac{\partial^2 f}{\partial y \partial t}\frac{dy}{dt} + \frac{\partial^2 f}{\partial t^2}\right)$$

$$= \left(\frac{\partial^2 f}{\partial y^2}f + \frac{\partial^2 f}{\partial y \partial t}\right)f + \frac{\partial f}{\partial y}\left(\frac{\partial f}{\partial y}f + \frac{\partial f}{\partial t}\right) + \left(\frac{\partial^2 f}{\partial y \partial t}f + \frac{\partial^2 f}{\partial t^2}\right)$$

and so on.

Let us take the expansion around $y(t)$. We can ***truncate*** the expression after the linear term and get the forward Euler method:

$$y(t+h) \approx y(t) + hy'(t).$$

Since we have neglected the 2nd order and higher terms, it is said that the approximate function is of order accuracy $O(h^2)$. This represents the local accuracy (i.e., accuracy along a single step). Generally, the error starts to grow when more than one step is made — ***the global accuracy will generally be one order less than the local accuracy***. So, for the forward Euler method, the global accuracy will be $O(h)$.

If we take the expansion around $y(t+h)$ and truncate the expression after the linear term, we get the backward Euler method. Since we have neglected the 2nd order and higher terms, it has local accuracy of order $O(h^2)$ just like the forward Euler method. Also like the forward Euler method, the global accuracy will be $O(h)$.

Just what does the order accuracy mean to us? If a method has order accuracy $O(h^m)$, then reducing the step size by the fraction F will reduce the error in the numerical solution from ε to $F^m \varepsilon$. For example, for order accuracy $O(h)$, cutting the step size in ½ will cut the error also in ½. However, for order accuracy $O(h^2)$, cutting the step size in ½ will cut the error to ¼. This shows a real advantage of methods that have higher orders of accuracy.

How can we get higher orders of accuracy? We can include the effects of the higher order derivatives directly. Or, we can use additional function evaluations to simulate the effects of the higher order derivatives.

Runge-Kutta Methods

It is possible to develop procedures that involve on 1st order derivative evaluations but which produce results equivalent in accuracy to the higher-order Taylor expansion formulas. These algorithms are the ***Runge-Kutta*** methods. Approximations of the 2nd, 3rd, and 4th orders are equivalent to keeping the h^2, h^3, and h^4 terms, respectively. These require function evaluations at 2, 3, and 4 points, respectively, along the interval $t^{(n)} \leq t \leq t^{(n+1)}$. (Method of order $\nu > 4$ require more then ν function evaluations.)

The Runge-Kutta formula of order ν can be written as:

$$y^{(n+1)} = y^{(n)} + \sum_{i=1}^{\nu} w_i k_i.$$

Where the w_i are weighting factors whose sum is 1 (i.e., $w_1 + w_2 + \cdots + w_v = 1$) and:

$$k_i = h \cdot f\left(t^{(n)} + c_i h, y^{(n)} + \sum_{j=1}^{i-1} a_{i,j} k_i\right).$$

and where the c_i and $a_{i,j}$ are constants. The values for the w_i, c_i, and $a_{i,j}$ constants are chosen to simulate keeping higher order derivative terms in the Taylor series expansion & hence give high orders of accuracy.

The Runge-Kutta methods belong to a class of ***predictor-corrector*** methods. The first step is to use an explicit formula to predict the value at the end of the time step. Then, implicit formulas are used to correct this initial estimate. However, instead of iterating to convergence using the implicit formula, the initial estimate is used for only a couple correction steps.

A 2nd order Runge-Kutta method is known as Heun's method or the improved Euler method. It uses a forward Euler prediction step with a backward Euler correction step. The formula is:

$$y^{(n+1)} = y^{(n)} + \frac{1}{2}k_1 + \frac{1}{2}k_2$$
$$k_1 = h \cdot f\left(t^{(n)}, y^{(n)}\right)$$
$$k_2 = h \cdot f\left(t^{(n)} + h, y^{(n)} + k_1\right).$$

The stability equation for this method is:

$$\left|\frac{\varepsilon^{(n+1)}}{\varepsilon^{(n)}}\right| = \left|\frac{1 - \frac{1}{2}\lambda h}{1 + \frac{1}{2}\lambda h}\right| \leq 1 \implies h > 0$$

but will lead to oscillations for $h > 2/\lambda$.

Perhaps the most widely used Runge-Kutta method is the 4th order method with constants developed by Gill. The method is:

$$y^{(n+1)} = y^{(n)} + \frac{1}{6}k_1 + \frac{b}{3}k_2 + \frac{d}{3}k_3 + \frac{1}{6}k_4$$
$$k_1 = h \cdot f\left(t^{(n)}, y^{(n)}\right)$$

$$k_2 = h \cdot f\left(t^{(n)} + \frac{1}{2}h, y^{(n)} + \frac{1}{2}k_1 \right).$$

$$k_3 = h \cdot f\left(t^{(n)} + \frac{1}{2}h, y^{(n)} + ak_1 + bk_2 \right).$$

$$k_4 = h \cdot f\left(t^{(n)} + h, y^{(n)} + ck_2 + dk_3 \right).$$

where the constants are:

$$a = \frac{\sqrt{2}-1}{2}, \ b = \frac{2-\sqrt{2}}{2}, \ c = -\frac{\sqrt{2}}{2}, \ d = \frac{2+\sqrt{2}}{2}.$$

It is reported that the Runge-Kutta-Gill method is stable for $h \le 2.8/\lambda$.

Let's again look at numerically solving our sample ODE:

$$\frac{dy}{dt} = 1 - 2y \text{ with } y(0) = 0.$$

The following chart shows the numerically approximated solution using the 2nd order improved Euler method. We are using the same step sizes as before. Remember that for the larger step sizes in the forward Euler method, the numerical solution was unstable. Here, the larger step sizes are not very accurate, but they are not unstable. Plus, the smaller step size of $h = 0.1$ gives a very good numerical solution — this solution is visually better than either of the forward or backward Euler solutions.

Finally, the following chart shows the numerically approximated solution using the 4th order Runge-Kutta-Gill method. Notice that the numerical solution is excellent, even for the fairly large step size of $h = 0.5$.

Runge-Kutta-Gill Solution to $y' = 1 - 2y$, $y(0) = 0$
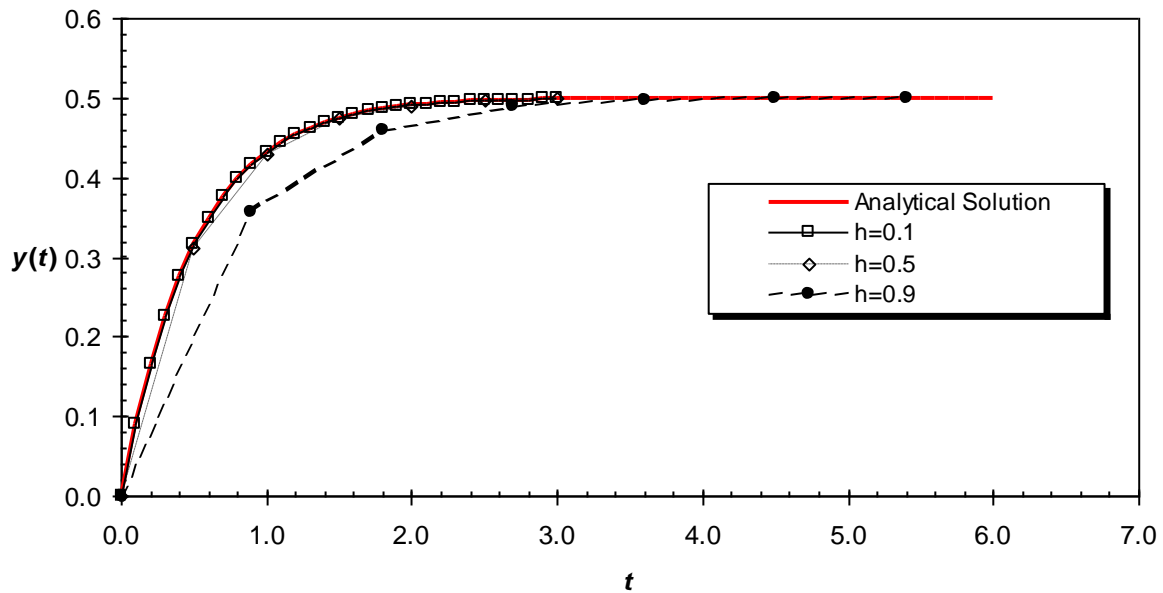
Stiff Equations

An important characteristic of a ___system___ of ordinary differential equations is whether they are ___stiff___ or not. A system is stiff if the characteristic time constants for the problem vary by many orders of magnitude. Mathematically, if

$$\frac{d\overline{\mathbf{y}}}{dt} = \overline{\mathbf{f}}\left(\overline{\mathbf{y}}\right) \approx \overline{\overline{\mathbf{A}}}\,\overline{\mathbf{y}}$$

by linearization, then there are a set of ___eigenvalues___, λ_i, which form the solution to:

$$\det\left(\overline{\overline{\mathbf{A}}} - \lambda_i \overline{\overline{\mathbf{I}}}\right) = 0$$

The solution to the linearized ODE will be of the form:

$$y_i = \sum_{j=1}^{N} C_j e^{-\lambda_j t} = \sum_{j=1}^{N} C_j e^{-t/\tau_j} \quad \text{where } \lambda_i = \frac{1}{\tau_i}.$$

A stiffness ratio S can be defined as:

$$S = \frac{\max\left(\text{Re}(\lambda_i)\right)}{\min\left(\text{Re}(\lambda_i)\right)}.$$

Typically, $S > 1000$ is considered a stiff system & $S > 10^6$ is a very stiff system.

What does this mean to us? If a solution is made up of a sum of exponential decay terms all with different time constants, then we must integrate over a time length that will encompass the term that takes the longest to decay. This will be controlled by the largest time constant (i.e., the smallest eigenvalue). However, for stability considerations, the largest step size that can be used is controlled by the smallest time constant (i.e., the largest eigenvalue). When these are different by several orders of magnitude, we may have a problem in trying to perform the numerical solution in a reasonable number of time steps.

Lets look at a stiff example. The following ODE system:

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} -500.5 & 499.5 \\ 499.5 & -500.5 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \text{ where } \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$
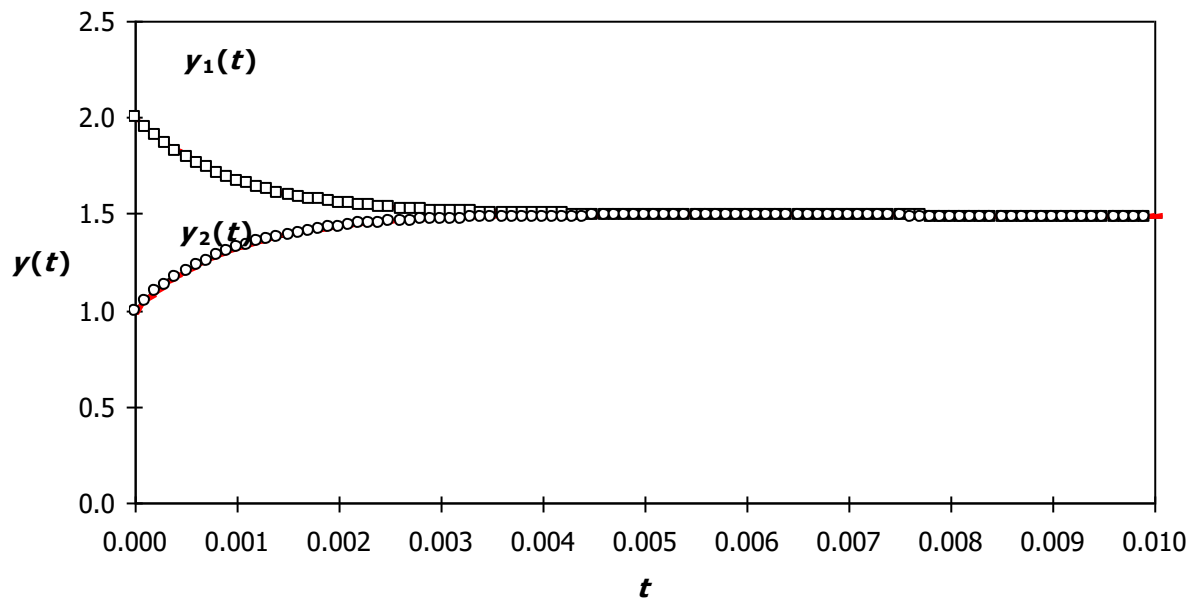
has the solution

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1.5e^{-t} + 0.5e^{-1000t} \\ 1.5e^{-t} - 0.5e^{-1000t} \end{bmatrix}.$$

Here, the eigenvalues are $\bar{\lambda} = \begin{bmatrix} 1,1000 \end{bmatrix}^T$ and the equivalent time constants are $\tau = \begin{bmatrix} 1,0.001 \end{bmatrix}^T$. The stiffness ratio is $S = 1000$, signifying a stiff problem. The exponential term associated with $\lambda = 1000$ will decay 99% of the way to its ultimate value (of zero) by $t = 0.0046$; however, the exponential term associated with $\lambda = 1$ will decay 99% of the way to its ultimate value (of zero) by $t = 4.6$. If we use the forward Euler method, we must keep the step-size as $h \leq 2/\lambda_{\max}$ to keep the numerical solution stable. For this problem, that means $h \leq 0.002$, meaning that we would need at least 2,300 time steps to reach our final time of $t = 4.6$.

The following figure shows the numerical solution at early times. In this chart, the step size used is $h = 0.0001$, much smaller than the stability limit of $h \leq 0.002$. However, using this extremely small step size will require 46,000 steps to reach the ultimate limit of $t = 4.6$. What happens if we raise the step size after $t > 0.01$, where the effects of the larger eigenvalue has already died out?

Forward Euler Solution to Sample Stiff ODE System
$h=0.0001$

The next figure shows the numerical solution after $t > 0.01$ when the step size is increased. We will go just above the stability limit of $h \leq 0.002$ to $h = 0.00201$. Notice that just this little bit above the stability limit starts to show instability — it takes a while, but the oscillations do occur after $t > 1.0$.

Forward Euler Solution to Sample Stiff ODE System
$h=0.0001$ for $t<0.01$
$h=0.00201$ for $t>0.01$
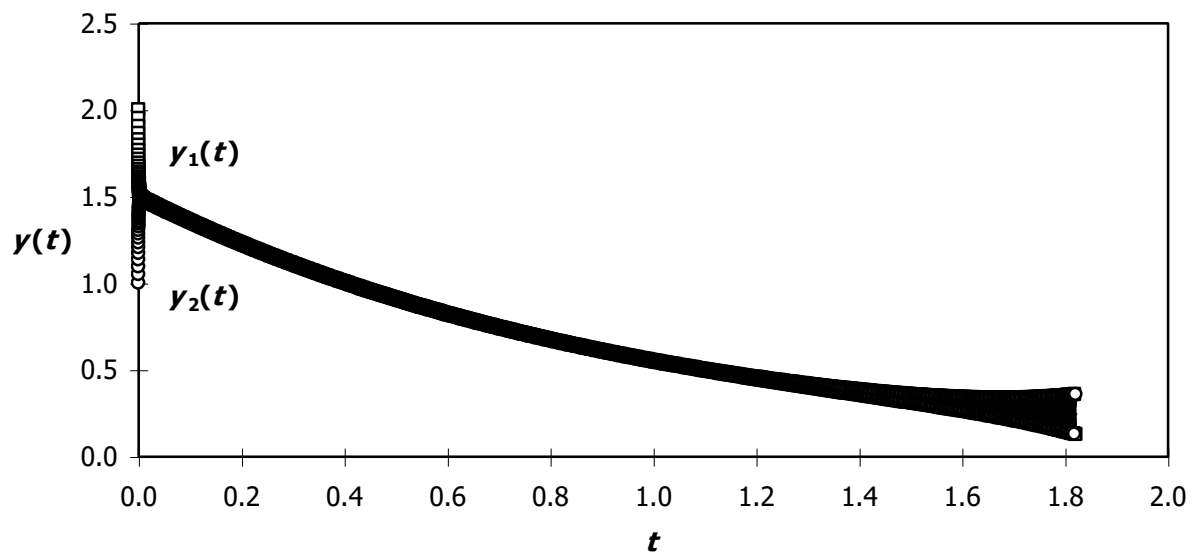
So how do we numerically solve stiff problems? We need to use methods that are of high order but also unconditionally stable. This means that, in general, some type of implicit method will have to be used. Methods have been published by C. W. Gear and are available as the GEARB package of routines. These routines have proven to be very popular for solving stiff problems.

Some of the newer mthods that are successful in solving stiff systems are based upon lineaizing the set of derivatives and effectively doing a single Newton iteration at the end of the step. These are referred to as ***semi-implicit*** methods. The following system of equations is linearized:

$$\overline{\mathbf{y}}^{(n+1)} = \overline{\mathbf{y}}^{(n)} + h\overline{\mathbf{f}}\left(\overline{\mathbf{y}}^{(n+1)}\right) \;\; \Rightarrow \;\; \overline{\mathbf{y}}^{(n+1)} \approx \overline{\mathbf{y}}^{(n)} + h\left[\overline{\mathbf{f}}\left(\overline{\mathbf{y}}^{(n)}\right) + \frac{\partial \overline{\mathbf{f}}}{\partial \overline{\mathbf{y}}}\bigg|_{\overline{\mathbf{y}}^{(n)}}\left(\overline{\mathbf{y}}^{(n+1)} - \overline{\mathbf{y}}^{(n)}\right)\right]$$

and then can be rearranged:

$$\left[\overline{\overline{\mathbf{I}}} - h \cdot \frac{\partial \overline{\mathbf{f}}}{\partial \overline{\mathbf{y}}}\bigg|_{\overline{\mathbf{y}}^{(n)}}\right]\left[\overline{\mathbf{y}}^{(n+1)} - \overline{\mathbf{y}}^{(n)}\right] = h\overline{\mathbf{f}}\left(\overline{\mathbf{y}}^{(n)}\right)$$

and solved as:

$$\overline{\mathbf{y}}^{(n+1)} = \overline{\mathbf{y}}^{(n)} + h\left[\overline{\overline{\mathbf{I}}} - h \cdot \frac{\partial \overline{\mathbf{f}}}{\partial \overline{\mathbf{y}}}\bigg|_{\overline{\mathbf{y}}^{(n)}}\right]^{-1}\overline{\mathbf{f}}\left(\overline{\mathbf{y}}^{(n)}\right)$$

Here, the derivative term is the Jacobian matrix, $\overline{\overline{\mathbf{J}}}$, with elements $J_{ij} = \partial f_i / \partial y_j$. The final solution for $\overline{\mathbf{y}}^{(n+1)}$ requires some type of matrix inversion (usually an effective matrix inversion via an LU decomposition).

Rosenbrock methods are semi-implicit Runge-Kutta methods. One particular set of methods have been proposed by Kaps & Rentrop. Their recursion equations have the following form:

$$\left[\frac{1}{\gamma h}\overline{\overline{\mathbf{I}}} - \overline{\overline{\mathbf{J}}}^{(n)}\right]\overline{\mathbf{g}}_1 = \overline{\mathbf{f}}\left(\overline{\mathbf{y}}^{(n)}\right)$$

$$\left[\frac{1}{\gamma h}\overline{\overline{\mathbf{I}}} - \overline{\overline{\mathbf{J}}}^{(n)}\right]\overline{\mathbf{g}}_2 = \overline{\mathbf{f}}\left(\overline{\mathbf{y}}^{(n)} + a_{21}\overline{\mathbf{g}}_1\right) + \frac{c_{21}}{h}\overline{\mathbf{g}}_1$$

$$\left[\frac{1}{\gamma h}\overline{\overline{\mathbf{I}}} - \overline{\overline{\mathbf{J}}}^{(n)}\right]\overline{\mathbf{g}}_3 = \overline{\mathbf{f}}\left(\overline{\mathbf{y}}^{(n)} + a_{31}\overline{\mathbf{g}}_1 + a_{32}\overline{\mathbf{g}}_2\right) + \frac{1}{h}\left[c_{31}\overline{\mathbf{g}}_1 + c_{32}\overline{\mathbf{g}}_2\right]$$

$$\left[\frac{1}{\gamma h}\bar{\bar{\mathbf{I}}}-\bar{\bar{\mathbf{J}}}^{(n)}\right]\bar{\mathbf{g}}_4=\bar{\mathbf{f}}\left(\bar{\mathbf{y}}^{(n)}+a_{31}\bar{\mathbf{g}}_1+a_{32}\bar{\mathbf{g}}_2\right)+\frac{1}{h}\left[c_{41}\bar{\mathbf{g}}_1+c_{42}\bar{\mathbf{g}}_2+c_{43}\bar{\mathbf{g}}_3\right]$$

$$\bar{\mathbf{y}}^{(n+1)}=\bar{\mathbf{y}}^{(n)}+b_1\bar{\mathbf{g}}_1+b_2\bar{\mathbf{g}}_2+b_3\bar{\mathbf{g}}_3+b_4\bar{\mathbf{g}}_4$$

where $\bar{\bar{\mathbf{J}}}^{(n)}$ is the Jacobian matrix evaluated at the beginning of the step.

The following Table 1 shows the parameters for this method. Note that there are two sets of b_i parameters, one set for a 4th order method and another set for a 3rd order method. These two sets of methods are very useful since they can be used to estimate errors on a particular step as:

$$\bar{\boldsymbol{\varepsilon}}=\left|\bar{\mathbf{y}}^{(n+1)}_{4th}-\bar{\mathbf{y}}^{(n+1)}_{3rd}\right|$$

$$=\left|\left(b_{1,4th}-b_{1,3rd}\right)\bar{\mathbf{g}}_1+\left(b_{2,4th}-b_{2,3rd}\right)\bar{\mathbf{g}}_2+\left(b_{3,4th}-b_{3,3rd}\right)\bar{\mathbf{g}}_3+\left(b_{4,4th}-b_{4,3rd}\right)\bar{\mathbf{g}}_4\right|=\mathrm{O}\left(h^4\right)$$

and, hence, lead to an algorithm for increasing or decreasing the step size to achieve a particular tolerance.

Table 1. Kaps-Rentrop Semi-Implicit Runge-Kutta Parameters

Parameter	Value	Parameter	4th Order Value	3rd Order Value
a_{21}	2	b_1	19/9	97/54
a_{31}	1.92	b_2	1/2	11/36
a_{32}	0.24	b_3	25/108	25/108
c_{21}	-8	b_4	125/108	0
c_{31}	14.88			
c_{32}	2.4			
c_{41}	-0.896			
c_{42}	-0.432			
c_{43}	-0.4			
γ	0.5			

Summary of Formulas

Forward Euler Method:
$$y^{(n+1)} = y^{(n)} + h \cdot f\left(y^{(n)}, t^{(n)}\right) = y^{(n)} + h \cdot f^{(n)}$$

Backward Euler Method:
$$y^{(n+1)} = y^{(n)} + h \cdot f\left(y^{(n+1)}, t^{(n+1)}\right) = y^{(n)} + h \cdot f^{(n+1)}.$$

Improved Euler Method:
$$y^{(n+1)} = y^{(n)} + \frac{1}{2}k_1 + \frac{1}{2}k_2$$

where:
$$k_1 = h \cdot f\left(t^{(n)}, y^{(n)}\right)$$
$$k_2 = h \cdot f\left(t^{(n)} + h, y^{(n)} + k_1\right)$$

Runge-Kutta-Gill Method:
$$y^{(n+1)} = y^{(n)} + \frac{1}{6}k_1 + \frac{b}{3}k_2 + \frac{d}{3}k_3 + \frac{1}{6}k_4$$

where:
$$k_1 = h \cdot f\left(t^{(n)}, y^{(n)}\right)$$
$$k_2 = h \cdot f\left(t^{(n)} + \frac{1}{2}h, y^{(n)} + \frac{1}{2}k_1\right).$$
$$k_3 = h \cdot f\left(t^{(n)} + \frac{1}{2}h, y^{(n)} + ak_1 + bk_2\right).$$
$$k_4 = h \cdot f\left(t^{(n)} + h, y^{(n)} + ck_2 + dk_3\right).$$
$$a = \frac{\sqrt{2}-1}{2}, \ b = \frac{2-\sqrt{2}}{2}, \ c = -\frac{\sqrt{2}}{2}, \ d = \frac{2+\sqrt{2}}{2}.$$