Chapter 4

Systems and Adaptive Step Size Methods

When modeling phenomena where we know the initial state and how it changes with time, we often have either a higher order IVP or a system of IVPs rather than a single first order IVP. In this chapter we first demonstrate how a higher order IVP can be transformed into a system of first order IVPs. Then we extend in a straightforward manner some of the methods from Chapter 3 to systems of equations. We discuss implementation issues and give examples that illustrate the use of systems of IVPs. Then we point out how to extend our stability tests for a single equation to a system.

The final concept we investigate in our study of IVPs are methods which efficiently allow a variable time step to be taken. For these methods we need a means to estimate the next time step. If we can get an estimate for the error made at time t_n then the magnitude of the error can be used to accept or reject the step and, if the step is accepted, to estimate the next time step. Consequently, our goal is to find methods which can be used to estimate the error. One strategy is to obtain two approximations at time t_n and use these to measure the error. Of course obtaining the second approximation must be done efficiently or else the cost is prohibitively large. In addition to variable step, many production codes are also variable order. We will not address these here.

4.1 Higher Order IVPs

Suppose we have the second order IVP

$$\begin{aligned} y''(t) &= 2y'(t) - \sin(\pi y) + 4t \quad 0 < t \leq 2 \\ y(0) &= 1 \\ y'(0) &= 0 \end{aligned}$$

where now the right-hand side is a function of t, y and y'. The methods we have learned only apply to first order IVPs. However, we can easily convert this second order IVP into two coupled first order IVPs. To do this, we let $w_1(t) = y(t)$, $w_2(t) = y'(t)$ and substitute into the equations and initial conditions to get a first order system for w_1, w_2

$$\begin{array}{rcll} w_1'(t) & = & w_2(t) & 0 < t \le 2 \\ w_2'(t) & = & 2w_2(t) - \sin(\pi w_1) + 4t & 0 < t \le 2 \\ w_1(0) & = & 1 \qquad w_2(0) = 0\,. & \end{array}$$

Note that these two differential equations are *coupled*, that is, the differential equation for w_1 depends on w_2 and the equation for w_2 depends on w_1.

In general, if we have the pth order IVP for $y(t)$

$$\begin{array}{rcl} y^{[p]}(t) & = & f(t, y, y', y'', \cdots, y^{[p-1]}) \quad t_0 < t \le T \\ y(t_0) & = & \alpha_1, \quad y'(t_0) = \alpha_2, \quad y''(t_0) = \alpha_3, \quad \cdots \quad y^{[p-1]}(t_0) = \alpha_p \end{array}$$

then we convert it to a system of p first-order IVPs by letting $w_1(t) = y(t)$, $w_2(t) = y'(t), \cdots, w_p(t) = y^{[p-1]}(t)$ which yields the first order coupled system

$$\begin{array}{rcl} w_1'(t) & = & w_2(t) \\ w_2'(t) & = & w_3(t) \\ & \vdots & \\ w_{p-1}'(t) & = & w_p(t) \\ w_p'(t) & = & f(t, w_1, w_2, \ldots, w_p) \end{array} \tag{4.1}$$

along with the initial conditions $w_k = \alpha_k$, $k = 1, 2, \ldots, p$. Thus any higher order IVP that we encounter can be transformed into a coupled system of first order IVPs.

Example 4.1. CONVERTING A HIGH ORDER IVP INTO A SYSTEM

Write the fourth order IVP

$$y^{[4]}(t) + 2y''(t) + 4y(t) = 5 \quad y(0) = 1, y'(0) = -3, y''(0) = 0, y'''(0) = 2$$

as a system of first order equations.

We want four first order differential equations for $w_i(t)$, $i = 1, 2, 3, 4$; to this end let $w_1 = y$, $w_2 = y'$, $w_3 = y''$, and $w_4 = y'''$. Using the first two expressions we have $w_1' = w_2$, and the second and third gives $w_2' = w_3$, the third and fourth gives $w_3' = w_4$ and the original differential equation provides the last first order equation $w_4' + 2w_3 + 4w_1 = 5$. The system of equations is thus

$$\begin{array}{rcl} w_1'(t) - w_2(t) & = & 0 \\ w_2'(t) - w_3(t) & = & 0 \\ w_3'(t) - w_4(t) & = & 0 \\ w_4' + 2w_3 + 4w_1 & = & 5 \end{array}$$

along with the initial conditions

$$w_1(0) = 1, \quad w_2(0) = -3, \quad w_3(0) = 0, \quad \text{and } w_4(0) = 2.$$

Oftentimes our model is already in the form of a system of first order IVPs. Our goal is to apply the methods of the previous chapter to a system of first order IVPs. The notation we use for a general system of N first order IVPs is

$$\begin{aligned} w_1'(t) &= f_1(t, w_1, w_2, \ldots, w_N) \quad t_0 < t \le T \\ w_2'(t) &= f_2(t, w_1, w_2, \ldots, w_N) \quad t_0 < t \le T \\ &\vdots \\ w_N'(t) &= f_N(t, w_1, w_2, \ldots, w_N) \quad t_0 < t \le T \end{aligned} \tag{4.2}$$

along with the initial conditions $w_k(t_0) = \alpha_k$, $k = 1, 2, \ldots, N$. For example, using this notation the pth order IVP written as the system (4.1) has $f_1 = w_2$, $f_2 = w_3$, etc.

Existence, uniqueness and continuous dependence of the solution to the system (4.2) can be established. Analogous to the case of a single IVP each function f_i must satisfy a Lipschitz condition. Details of this analysis can be found in standards texts in ODEs. For the sequel, we will assume that each system has a unique solution which depends continuously on the data.

In the next two sections we demonstrate how one-step and multistep methods from Chapter 3 are easily extended to the system of N equations (4.2).

4.2 Single-step methods for systems

We now want to extend single-step methods to the system (4.2). For simplicity we first extend the forward Euler method for a system and then with the intuition gained from that method we extend a general explicit Runge-Kutta method to a system. Implicit RK methods can be extended in an analogous way. We use the notation W_k^n as the approximation to $w_k(t_n)$ where the subscript of W refers to the unknown number and the superscript to the point t_n.

Suppose we have the first order system (4.2) with the initial conditions $w_k(t_0) = \alpha_k$ for $k = 1, 2, \ldots, N$. The forward Euler method for each equation is

$$W_k^{n+1} = W_k^n + \Delta t f_k\big(t_n, W_1^n, W_2^n, \cdots, W_N^n\big)\,.$$

We write the Euler method as a vector equation so we can solve for all N unknowns simultaneously at each t_n; this is not necessary but results in an efficient implementation of the method. We set $\mathbf{W}^n = \big(W_1^n, W_2^n, \cdots, W_N^n\big)^T$, $\mathbf{W}_0 = (\alpha_1, \alpha_2, \ldots, \alpha_N)^T$, and $\mathbf{F}^n = \big(f_1(t_n, \mathbf{W}^n), f_2(t_n, \mathbf{W}^n), \cdots, f_N(t_n, \mathbf{W}^n)\big)^T$. For $n = 0, 1, 2, \ldots$ we have the following vector equation for the forward Euler method for a system

$$\mathbf{W}^{n+1} = \mathbf{W}^n + \Delta t \mathbf{F}^n\,. \tag{4.3}$$

To implement the scheme at each point t_n we have N function evaluations to form the vector $\mathbf{F}^n$, then we perform the scalar multiplication to get $\Delta t \mathbf{F}^n$ and then a vector addition to obtain the final result $\mathbf{W}^{n+1}$.

Example 4.2. FORWARD EULER FOR A SYSTEM

Consider the system of three IVPs

$$\begin{array}{rcll} w_1'(t) & = & 2w_2(t) - 4t & 0 < t < 10 \\ w_2'(t) & = & -w_1(t) + w_3(t) - e^t + 2 & 0 < t < 10 \\ w_3'(t) & = & w_1(t) - 2w_2(t) + w_3(t) + 4t & 0 < t < 10 \\ w_1(0) & = & -1, \quad w_2(0) = 0, & w_3(0) = 2 \end{array}$$

for the unknown $(w_1, w_2, w_3)^T$. The exact solution is $(-\cos(2t), \sin(2t) + 2t, \cos(2t) + e^t)^T$. We want to compute an approximation at $t = 0.2$ using $\Delta t = 0.1$ and the forward Euler method.

We set $\mathbf{W}^0 = (-1, 0, 2)^T$ and because $\mathbf{F}^n = \left(2W_2^n - 4t_n, -W_1^n + W_3^n - e^{t_n} + 2, W_1^n - 2W_2^n + W_3^n + 4t_n\right)^T$ we have $\mathbf{F}^0 = (0, 4, 1)^T$. With $\Delta t = 0.1$ we form $\mathbf{W}^1$ from

$$\mathbf{W}^1 = \mathbf{W}^0 + \Delta t \mathbf{F}^0 = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} + 0.1 \begin{pmatrix} 0 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} -1.0 \\ 0.4 \\ 2.1 \end{pmatrix}.$$

Now to determine $\mathbf{W}^2$ we need $\mathbf{F}^1$ which is given by

$$\mathbf{F}^1 = \begin{pmatrix} 2(0.4) - 4(.1) \\ 1 + 2.1 - e^{.1} + 2 \\ -1 - 2(.4) + 2.1 + 4(.1) \end{pmatrix} = \begin{pmatrix} 0.400 \\ 3.995 \\ 0.7000 \end{pmatrix}$$

so that

$$\mathbf{W}^2 = \mathbf{W}^1 + \Delta t \mathbf{F}^1 = \begin{pmatrix} -1.0 \\ 0.4 \\ 2.1 \end{pmatrix} + 0.1 \begin{pmatrix} 0.400 \\ 3.995 \\ 0.700 \end{pmatrix} = \begin{pmatrix} -0.9600 \\ 0.7995 \\ 2.1700 \end{pmatrix}.$$

The exact solution at $t = 0.2$ is $(-0.921061, 0.789418, 2.14246)^T$. Unlike the case of a single IVP we now have an error *vector* instead of a single number; at $t = 0.2$ the error vector in our calculations is $(0.038939, .010082, .02754)^T$. To obtain a single number from this vector to use in the calculation of a numerical rate, we must use a vector norm. A common vector norm is the standard Euclidean norm which is often called ℓ_2 norm or the "little l2 norm". For this calculation at $t = 0.2$ the Euclidean norm of the error is $1.98\ 10^{-2}$.

In the table below we tabulate the results using the forward Euler method for this system at $t = 1$ where both the normalized ℓ_2-norm and ℓ_∞-norm (i.e., the maximum norm) of the error normalized by the corresponding norm of the solution is reported. Clearly we have linear convergence as we did in the case of a single equation.

Δt	ℓ_2 Error	rate	ℓ_∞ Error	rate
$1/10$	$6.630\ 10^{-2}$		$6.019\ 10^{-2}$	
$1/20$	$3.336\ 10^{-2}$	0.99	$3.156\ 10^{-2}$	0.93
$1/40$	$1.670\ 10^{-2}$	1.00	$1.631\ 10^{-2}$	0.95
$1/80$	$8.350\ 10^{-3}$	1.00	$8.277\ 10^{-3}$	0.98

Suppose now that we have an s-stage RK method; recall that for a single first order equation we have s function evaluations for each t_n. If we have N first order IVPs, then we need sN function evaluations at each t_n. For example, if we use a 4-stage RK with 10,000 equations then at each time we need 40,000 function evaluations; if we do 100 time steps then we have 4 million function evaluations. If function evaluations are expensive, multistep methods may be more efficient.

In an s-stage RK method for a single equation we must compute each k_i, $i = 1, 2, \ldots, s$ as defined in (3.15). For a system, we have a vector of slopes so each k_i is a vector. Thus for a system an s-stage RK method is written as

$$\begin{aligned}
\mathbf{k}_1 &= \Delta t \mathbf{F}\big(t_n, \mathbf{W}^n\big) \\
\mathbf{k}_2 &= \Delta t \mathbf{F}\big(t_n + c_2\Delta t, \mathbf{W}^n + a_{21}\mathbf{k}_1\big) \\
\mathbf{k}_3 &= \Delta t \mathbf{F}\big(t_n + c_3\Delta t, \mathbf{W}^n + a_{31}\mathbf{k}_1 + a_{32}\mathbf{k}_2\big) \\
&\vdots \\
\mathbf{k}_s &= \Delta t \mathbf{F}\big(t_n + c_s\Delta t, \mathbf{W}^n + a_{s1}\mathbf{k}_1 + a_{s2}\mathbf{k}_2 + \cdots + a_{ss-1}\mathbf{k}_{s-1}\big) \\
\mathbf{W}^{n+1} &= \mathbf{W}^n + \sum_{j=1}^{s} b_j \mathbf{k}_j \,.
\end{aligned}$$

The following example uses the Heun method, a 2-stage RK scheme given in (3.14), to approximate the solution to the IVP in the previous example.

Example 4.3. HEUN METHOD FOR A SYSTEM

We want to approximate the solution to the system given in the previous example using the Heun method. Recall for the Heun method the coefficients are $c_2 = 2/3$, $a_{21} = 2/3$, $b_1 = 1/4$ and $b_2 = 3/4$ so for a system we have

$$\begin{aligned}
\mathbf{k}_1 &= \Delta t \mathbf{F}(t_n, \mathbf{W}^n) \\
\mathbf{k}_2 &= \Delta t \mathbf{F}(t_n + \tfrac{2}{3}\Delta t, \mathbf{W}^n + \tfrac{2}{3}\mathbf{k}_1) \\
\mathbf{W}_{i+1} &= \mathbf{W}^n + \tfrac{1}{4}\mathbf{k}_1 + \tfrac{3}{4}\mathbf{k}_2 \,.
\end{aligned}$$

As in the previous example, $\mathbf{W}^0 = (-1, 0, 2)^T$ and $\mathbf{F}^n = \big(2W_2^n - 4t_n, -W_1^n + W_3^n - e^{t_n} + 2, W_1^n - 2W_2^n + W_3^n + 4t_n\big)^T$. For the first step of length $\Delta t = 0.1$ we have $\mathbf{k}_1 = 0.1(0, 4, 1)^T$ and to determine $\mathbf{k}_2$ we need to evaluate $\mathbf{F}$ at $(\frac{2}{3}(.1), \mathbf{W}_0 + \frac{2}{3}\mathbf{k}_1)$; performing this calculation gives $\mathbf{k}_2 = (.026667, .399773, .08)^T$ so that

$$\mathbf{W}^1 = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} + \frac{1}{4}\begin{pmatrix} 0.0 \\ 0.4 \\ 0.1 \end{pmatrix} + \frac{3}{4}\begin{pmatrix} .026667 \\ .399773 \\ .080000 \end{pmatrix} = \begin{pmatrix} -0.98000 \\ 0.39983 \\ 2.08500 \end{pmatrix} .$$

Similarly for the approximation at 0.2 we have

$$\mathbf{W}^2 = \begin{pmatrix} -0.98000 \\ 0.39983 \\ 2.08500 \end{pmatrix} + \frac{1}{4}\begin{pmatrix} 0.039966 \\ 0.395983 \\ -0.070534 \end{pmatrix} + \frac{3}{4}\begin{pmatrix} 0.066097 \\ 0.390402 \\ 0.051767 \end{pmatrix} = \begin{pmatrix} -0.92040 \\ 0.79163 \\ 2.14150 \end{pmatrix} .$$

The exact solution at $t = 0.2$ is $(-0.921061, 0.789418, 2.14246)^T$ giving an error vector of $(0.000661, .002215, .000096)^T$; calculating the standard Euclidean norm of the error and

normalizing by the Euclidean norm of the solution gives 1.0166×10^{-3} which is considerably smaller than we obtained for the forward Euler. The following table provides results at $t = 1$ with the error measured in both the normalized ℓ_2 and ℓ_∞ norms. The rates of convergence clearly demonstrate quadratic convergence.

Δt	ℓ_2 Error	rate	ℓ_∞ Error	rate
$1/10$	$5.176\ 10^{-3}$		$5.074\ 10^{-3}$	
$1/20$	$1.285\ 10^{-3}$	2.01	$1.242\ 10^{-3}$	2.03
$1/40$	$3.198\ 10^{-4}$	2.01	$3.067\ 10^{-4}$	2.02
$1/80$	$7.975\ 10^{-5}$	2.00	$7.614\ 10^{-5}$	2.01

4.3 Multistep methods for systems

Recall that explicit multistep methods use values from previously calculated times to extrapolate the solution to the new point. The m-step explicit method from § 3.2.2 for a single IVP is

$$\begin{aligned} Y^{n+1} &= a_{m-1}Y^n + a_{m-2}Y^{n-1} + a_{m-3}Y^{n-2} + \cdots + a_0 Y^{n+1-m} \\ &\quad + \Delta t\Big[b_{m-1}f(t_n, Y^n) + b_{m-2}f(t_{n-1}, Y^{n-1}) \\ &\quad + \cdots + b_0 f(t_{n+1-m}, Y^{n+1-m})\Big]. \end{aligned}$$

For a system of N equations the function f is now a vector $\mathbf{F}$ so we must store its value at the previous m steps. In the Adams-Bashforth or Adams Moulton methods $a_0, a_1, \ldots, a_{m-2} = 0$ so only the solution at t_n is used. This saves additional storage because we only have to store m slope vectors and a single vector approximation to the solution. So for the system of N equations using an m-step Adams-Bashforth method we must store $(m+1)$ vectors of length N.

As a concrete example, consider the 2-step Adams-Bashforth method

$$Y^{n+1} = Y^n + \Delta t\Big[\frac{3}{2}f(t_n, Y^n) - \frac{1}{2}f(t_{n-1}, Y^{n-1})\Big]$$

for a single IVP. Using the notation of the previous section we extend the method for the system of N equations as

$$\mathbf{W}^{n+1} = \mathbf{W}^n + \Delta t\Big[\frac{3}{2}\mathbf{F}(t_n, \mathbf{W}^n) - \frac{1}{2}\mathbf{F}(t_{n-1}, \mathbf{W}^{n-1})\Big]. \tag{4.4}$$

At each step we must store three vectors $\mathbf{W}^n$, $\mathbf{F}(t_n, \mathbf{W}^n)$, and $\mathbf{F}(t_{n-1}, \mathbf{W}^{n-1})$. In the next example we apply this 2-step method to the system of the previous examples.

Example 4.4. ADAMS-BASHFORTH METHOD FOR A SYSTEM

To apply the 2-step Adams-Bashforth method (4.4) to the system of the previous examples we need values for $\mathbf{W}^1$ because we set $\mathbf{W}^0$ from the initial conditions. Because this method is second order we can use either a first or second order scheme to generate an approximation to $\mathbf{W}^1$. Here we use the results from the Heun method from the previous example for $\mathbf{W}^1$ and use $\Delta t = 0.1$ as before. Consequently we have

$$\mathbf{W}^0 = \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} \quad \text{and} \quad \mathbf{W}^1 = \begin{pmatrix} -0.98000 \\ 0.39982 \\ 2.08500 \end{pmatrix}.$$

From the previous example we have $\mathbf{F}(0, \mathbf{W}^0) = (0.0, 4.0, 1.0)^T$ and $\mathbf{F}(0.1, \mathbf{W}^1)$=$(.39966, 3.95982, -.704659)^T$. Then $\mathbf{W}^2$ is given by

$$\mathbf{W}^2 = \begin{pmatrix} -0.98000 \\ 0.39982 \\ 2.08500 \end{pmatrix} + 0.1 \left[\frac{3}{2} \begin{pmatrix} 0.39966 \\ 3.95983 \\ -0.70466 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 0.0 \\ 4.0 \\ 1.0 \end{pmatrix} \right] = \begin{pmatrix} -0.92005 \\ 0.79380 \\ 1.92930 \end{pmatrix}.$$

The table below tabulates the errors at $t = 1$. Of course we can only use the starting value $\mathbf{W}_1 = (. - 0.98000, 0.39982, 2.08500)^T$ as starting values for the computations at $\Delta t = 0.1$; for the other step sizes we must generate starting values at the different value of t_1. From the results we see that the rate is two, as expected.

Δt	ℓ_2 Error	rate	ℓ_∞ Error	rate
$1/10$	$1.346\ 10^{-2}$		$1.340\ 10^{-2}$	
$1/20$	$3.392\ 10^{-3}$	1.99	$3.364\ 10^{-3}$	1.99
$1/40$	$8.550\ 10^{-4}$	1.99	$8.456\ 10^{-4}$	1.99
$1/80$	$2.149\ 10^{-5}$	1.99	$2.121\ 10^{-5}$	1.99

4.4 Stability of Systems

Oftentimes we have a system of first order IVPs or we have a higher order IVP which we first write as a system of first order IVPs. We want to extend our definition of absolute stability to a system but we first look at stability for the differential equations themselves. Analogous to the problem $y'(t) = \lambda y$ we consider the linear model problem

$$\mathbf{w}'(t) = A\mathbf{w}$$

for an $N \times N$ system of IVPs where A is an $N \times N$ matrix. Consider first the simple case where A is a diagonal matrix and the equations are uncoupled so basically we have the same situation as a single equation. Thus the stability criteria is that the real part of each diagonal entry must be less than or equal to zero. But the diagonal entries of a diagonal matrix are just its N eigenvalues λ_i[1] counted according to multiplicity. So an equivalent statement of stability when A is diagonal is that $\text{Re}(\lambda_i) < 0$, $i = 1, 2, \ldots, N$; it turns out that this is the stability criteria for a

[1]The eigenvalues of an $N \times N$ matrix A are scalars λ such that $A\mathbf{x} = \lambda\mathbf{x}$; the vector $\mathbf{x}$ is called the eigenvector corresponding to the eigenvalue λ.

general matrix A. Recall that even if the entries of A are real the eigenvalues may be complex. If A is symmetric we are guaranteed that the eigenvalues are real. If we have the general system (4.2) where $f_i(t, \mathbf{w})$ is not linear in $\mathbf{w}$, then the condition becomes one on the eigenvalues of the Jacobian matrix for f where the (i, j) entry of the Jacobian is $\partial f_i/\partial w_j$.

Now if we apply the forward Euler method to the system $\mathbf{w}'(t) = A\mathbf{w}$ where the entries of A are a_{ij} then we have the system

$$\mathbf{W}^{n+1} = \begin{pmatrix} 1+\Delta t a_{11} & \Delta t a_{12} & \Delta t a_{13} & \cdots & \Delta t a_{1N} \\ \Delta t a_{21} & 1+\Delta t a_{22} & \Delta t a_{23} & \cdots & \Delta t a_{2N} \\ & \ddots & \ddots & & \\ & \cdots & \cdots & \Delta t a_{N,N-1} & 1+\Delta t a_{N,N} \end{pmatrix} \mathbf{W}^n$$

The condition on Δt is determined by choosing it so that all the eigenvalues of the matrix have real parts less than zero.

4.5 Adaptive time step methods

If the solution to a differential equation varies rapidly over a portion of the time interval and slowly over the remaining time, then clearly using a fixed time step is inefficient. In this section we want to investigate methods which allow us to estimate an error and then use this error to decide if the time step can be increased or if it should be decreased. We have already encountered Predictor-Corrector methods which can easily provide an estimate for the error by comparing the results of the predictor and corrector steps. Another approach is to use two approximations such as a p-order and a $(p+1)$-order RK method and compare the local truncation error between the two which should be $\mathcal{O}(\Delta t)$. Of course, in order to do this efficiently the methods should be nested in the sense that the function values required for the $(p+1)$-order method include all the function evaluations from the p-order method. The best known of these methods is the Runge-Kutta-Fehlberg method which uses a fourth and fifth order explicit RK method (RKF45). However, one of the simplest adaptive RK method is a combination of the first order forward Euler method and the second order Heun's method. In the exercises you are asked to derive a method which uses a second and third order RK method.

Idea to have time steps

$$\Delta t_0 = t_1 - t_0, \quad \Delta t_1 = t_2 - t_1, \quad \cdots \quad \Delta t_{N-1} = t_N - t_{N-1}$$

4.5.1 Adaptive methods using Richardson extrapolation

We begin this section by first considering a very simple example of an algorithm which produces an automatic step size selector. Recall that Richardson extrapolation was introduced in § 3.3 as a technique to combine lower order approximations to get more accurate approximations. We will explore this idea by taking an approximation at t_{n+1} obtained from t_n using a step size of Δt_n and a second one at t_{n+1}

which was obtained from t_n by taking two steps with step size $\Delta t_n/2$. We want to see how these two approximations can be used to estimate the local truncation error and provide an estimate for the next time step Δt_{n+1}. To describe this approach we use the forward Euler method because its simplicity should make the technique clear.

We assume we have the solution at t_n and have an estimate for the next time step Δt_n and the goal is to estimate the next time step Δt_{n+1}. First we take an Euler step with Δt_n to get the approximation Y_1^{n+1} where the subscript denotes the specific approximation because we will have two. Next we take two Euler steps from t_n using a step size of $\Delta t_n/2$ to get the approximation Y_2^{n+1}.

Recall that the local truncation error for the forward Euler method using a step size of Δt is $C(\Delta t)^2 + \mathcal{O}(\Delta t^3)$. Thus the exact solution satisfies

$$y(t_{n+1}) = y(t_n) + \Delta t f(t_n, y(t_n)) + C(\Delta t)^2 + \mathcal{O}(\Delta t^3)$$

where we have equality because we have included the local truncation error. Consequently, for our solution Y_1^{n+1} we have the local truncation error $C(\Delta t_n)^2 + \mathcal{O}(\Delta t_n)^3$. Now for Y_2^{n+1} we take two steps each with a local truncation error of $C(\Delta t_n)^2/4 + \mathcal{O}(\Delta t_n/2)^3$ so basically at t_{n+1} we have twice this error a which is $C(\Delta t_n)^2/2 + \mathcal{O}(\Delta t_n/2)^3$. Now using Richardson extrapolation we solution $2Y_2^{n+1} - Y_1^{n+1}$ should have a local truncation error of $\mathcal{O}(\Delta t_n)^3$.

Now we want to see how to use these truncation errors to decide whether to accept or reject the improved solution $2Y_2^{n+1} - Y_1^{n+1}$ at t_{n+1} and if we accept it to choose a new time step Δt_{n+1}. Suppose that the user inputs a tolerance for the maximum rate of increase in the error. We have an estimate for this rate, r_n, from our solutions so we require it to be less than the given tolerance σ, i.e.,

$$r_n = \frac{|Y_1^{n+1} - Y_2^{n+1}|}{\Delta t_n} \leq \text{prescribed tolerance} = \sigma\,. \tag{4.5}$$

If this is satisfied then we accept the improved solution $2Y_2^{n+1} - Y_1^{n+1}$; otherwise we reduce Δt_n and repeat the procedure. We estimate the next step size by computing the ratio of the acceptable rate (i.e., the prescribed tolerance) and the actual rate r_n. Then we take this fraction of the current step size to estimate the new one. In practice, one often adds a multiplicative factor less than one for "safety" since we have made certain assumptions. For example, we could compute a step size from

$$\Delta t_{n+1} = 0.9 \left(\frac{\sigma}{r_n}\right) \Delta t_n\,. \tag{4.6}$$

From this expression we see that if the acceptable rate σ is smaller than the actual rate r_n then Δt_n is decreased. If the criteria (4.5) is not satisfied then we must repeat the step with a reduced time step. We can estimate this from (4.6) without the safety factor because in this case $r_n > \sigma$ so the fraction is less than one. The following example illustrates this technique.

Example 4.5.

Example 4.6. Consider the IVP

$$y'(t) = -22ty(t) \quad -1 < t \leq 1 \quad y(-1) = e^{-7}$$

whose exact solution is $y(t) = e^{4-11t^2}$ which is plotted in the figure below. As can be seen from the plot, the solution is small and varies slowly in the domain $[-1, -.5] \cup [.5, 1]$ but peaks to over fifty at the origin so a variable time step algorithm is needed. We use the algorithm described above with Richardson extrapolation to solve this IVP using the time step formula (4.6). We first tabulate the actual error rate computed using (4.5) along with whether the step is accepted or rejected and the new step size at several different times. The starting step size is chosen to be $\Delta t_0 =$ and the tolerance $\sigma =$.

t_n	Δt_n	accept/reject	Δt_n (if rejected)	Δt_{n+1} (if accepted)

4.5.2 Adaptive methods using predictor-corrector schemes

Using predictor-corrector pairs also provides a way to estimate the error and thus determine if the current step size is appropriate. For example, for the third order predictor and corrector pair given in (3.37) one can specifically compute the constant in the local truncation error to get

$$|y(t_{n+1}) - Y_p^{n+1}| = \frac{9}{24} y^{[4]}(\xi)(\Delta t)^4 \qquad |y(t_{n+1}) - Y^{n+1}| = \frac{1}{24} y^{[4]}(\eta)(\Delta t)^4 \,.$$

For small Δt we assume that the fourth derivative is constant over the interval and so

$$|y(t_{n+1}) - Y^{n+1}| \approx \frac{1}{9} |y(t_{n+1}) - Y_p^{n+1}| \,.$$

If the step size Δt is too large, then the assumption that the fourth derivative is constant from t_n to t_{n+1} may not hold and the above relationship is not true. Typically the exact solution $y(t_{n+1})$ is not known so instead we monitor the difference in the predicted and corrected solution $|Y^{n+1} - Y_p^{n+1}|$. If it is larger than our prescribed tolerance, then the step is rejected and Δt is decreased. Otherwise the step is accepted; if the difference is below the minimum prescribed tolerance then the step size is increased in the next step.

4.5.3 Embedded RK methods

To use RK methods for step size control we use two different methods to approximate the solution at t_{n+1} and compare the approximations which should give an approximation to the local truncation error. If the results are close, then we are confident that a correct step size was chosen; if they vary considerably then we assume that too large a step size was chosen and we reduce the time step and repeat the calculation. If they are extremely close then this suggests a larger

step size can be used. Typically, an ad hoc formula is used to estimate the next time step based on these observations. Of course, to efficiently implement this approach we should choose the methods so that they have function evaluations in common to reduce the work.

A commonly used RK pair for error control is a combination of a fourth and fifth order explicit method; it is called the Runge-Kutta-Fehlberg method and was developed by the mathematician Erwin Fehlberg in the late 1960's. It is typically known by the acronym RKF45. For many years it has been considered the "work horse" of IVP solvers. Recall that to get an accuracy of $(\Delta t)^5$ at least six function evaluations are required. The six function evaluations are

$$\begin{aligned}
k_1 &= f\left(t_n, Y^n\right) \\
k_2 &= f\left(t_n + \frac{1}{4}\Delta t, Y^n + \frac{1}{4}k_1\right) \\
k_3 &= f\left(t_n + \frac{3}{8}\Delta t, Y^n + \frac{3}{32}k_1 + \frac{9}{32}k_2\right) \\
k_4 &= f\left(t_n + \frac{12}{13}\Delta t, Y^n + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right) \\
k_5 &= f\left(t_n + \Delta t, Y^n + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right) \\
k_6 &= f\left(t_n + \frac{1}{2}\Delta t, Y^n - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right).
\end{aligned}$$

The fourth order RK method

$$Y^{n+1} = Y^n + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5 \tag{4.7}$$

is used to first approximate $y(t_{n+1})$ and then fifth order RK method

$$Y^{n+1} = Y^n + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6 \tag{4.8}$$

is used for comparison. Note that the fifth order method uses all of the coefficients of the fourth order method so it is efficient to implement because it only requires an additional function evaluation. For this reason, we call RK45 an *embedded method*. Also note that the fourth order method is actually a 5-stage method but remember that no 5-stage method is fifth order. Typically the Butcher tableau for the coefficients c_i and a_{ij} is written for the higher order method and then two lines are appended at the bottom for the coefficients b_i in

each method. For example, for RKF45 the tableau is

$$
\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & & \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & \\
\hline
 & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5} & 0 \\
 & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{2856}{56430} & -\frac{9}{50} & \frac{2}{55}
\end{array}
$$

To implement the RKF45 scheme we find two approximations at t_{n+1}; Y_4^{n+1} using the fourth order scheme (4.7) and Y_5^{n+1} using the fifth order scheme (4.8). We then determine the difference $|Y_5^{n+1} - Y_4^{n+1}|$ which should be $\mathcal{O}(\Delta t)$. This error is used to make the decision whether to accept the step or not; if we accept the step then the decision must be made whether or not to increase the step size for the next calculation or keep it the same. One must choose a priori a minimum and maximum acceptable value for the normalized difference between $|Y_5^{n+1} - Y_4^{n+1}|$ and use these values for deciding whether a step is acceptable or not. To implement the RK45 method the user needs to input a maximum and minimum time step so that we never allow the time step to get too larger or too small. Typically the code has some default values that the user can either accept or modify. In addition, the user One must choose a priori a minimum and maximum acceptable value for the normalized difference between $|Y_5^{n+1} - Y_4^{n+1}|$ and use these values for deciding whether a step is acceptable or not.

4.6 Stiff systems

Some differential equations are more difficult to solve than others. We know that for problems where the solution curve varies a lot, we should take a small step size and where it changes very little a larger step size should be used for efficiency. If the change in the solution curve is relatively small everywhere then a uniform step size is the most efficient approach. This all seems very heuristic. However, there are problems which require a very small step size even when the solution curve is very smooth. There is no universally accepted definition of stiff differential equations but typically the solution curve changes rapidly and then tends towards a slowly-varying solution. Because the stability region for implicit methods is typically much larger than explicit methods, most stiff equations are approximated using an implicit method.

To illustrate the concept of stiffness we look at a single IVP which is considered stiff. The example is from a combustion model and is due to Shampine (2003) who is one of the authors of the Matlab ODE suite. The idea is to model flame propagation as when you light a match. We know that the flame grows

rapidly initially until it reaches a critical size which is dependent on the amount of oxygen. We assume that the flame is a ball and $y(t)$ represents its radius; in addition we assume that the problem is normalized so that the maximum radius is one. We have the IVP

$$y'(t) = y^2(1-y) \quad 0 < t \leq \frac{2}{\delta}; \qquad y(0) = \delta \tag{4.9}$$

where $\delta << 1$ is the small given initial radius. At ignition the solution y increases rapidly to a limiting value of one; this happens quickly on the interval $[0, 1/\delta]$ but on the interval $[1/\delta, 2/\delta]$ the solution is approximately equal to one. Knowing the behavior of the problem suggests that we should take a small step size initially and then on $[1/\delta, 2/\delta]$ where the solution is almost constant we should be able to take a large step size. However, if we use the RKF45 method with an automatic step size selector, then we can capture the solution on $[0, 1/\delta]$ but on $[1/\delta, 2/\delta]$ the step size is reduced by so much that the minimum allowable step size is surpassed and the method often fails if the minimum step size is set too large. Initially the problem is not stiff but it becomes stiff as its approaches the value one, its steady state solution. The term "stiff" was used to described this phenomena because it was felt the steady state solution is so "rigid".

When one has a system of equations like (4.2) the stiffness of the problem depends upon the eigenvalues of the Jacobian matrix. Recall that we said we need all eigenvalues to have real part less than zero for stability. If the Jacobi matrix has eigenvalues which have a very large negative real part and eigenvalues with a very small negative real part, then the system is stiff and special care must be used to solve it. You probably don't know a priori if a system is stiff but if you encounter behavior where the solution curve is not changing much but you find that your step size needs to be smaller and smaller, then your system is probably stiff. In that case, an implicit method is typically used.

EXERCISES

---

1. Convert each IVP into a system of first order IVPs.

   a. $y''(t) + 6y'(t) - \frac{1}{2}y(t) = 4 \qquad y(0) = 1, y'(0) = -3$

   b. $y'''(t) - 2y'(t) + y^2(t) = 0 \qquad y(0) = 0, y'(0) = 1, y''(0) = 4$

2. Determine the amplification factor for the Midpoint method (3.6). Then determine the region of absolute stability.

3. Show that all members of Adams multistep methods (both explicit and implicit) are stable.