Appendix A

Implementation Details

A.1 Solving Nonlinear Systems

There are basically two approaches that are used to solve the nonlinear algebraic equations arising in the implementation of implicit methods. These are *functional iteration* and *Newton iteration*. We discuss both of these below.

Functional iteration. The nonlinear systems encountered in implicit methods usually have the form

$$\mathbf{y} = \tau \mathbf{g}(\mathbf{y}) + \mathbf{c} \tag{A.1}$$

for some function $\mathbf{g}$ and constant vector $\mathbf{c}$. For backward Euler, for example, to determine $\mathbf{y}_{n+1}$ we have such a relation with $\mathbf{g} = \mathbf{f}$ and $\mathbf{c} = \mathbf{y}_n$.

Functional iteration for (A.1) is given by

$$\mathbf{y}^{(\nu+1)} := \tau \mathbf{g}(\mathbf{y}^{(\nu)}) + \mathbf{c}. \tag{A.2}$$

It is clear that for $\tau = 0$ this iteration converges in one step to $\mathbf{c}$. Although the nonlinear problem (A.1) may have multiple solutions, if the Jacobian of (A.1), i.e. $I - \tau\, \partial \mathbf{g}/\partial \mathbf{y}$, is nonsingular as $\tau \to 0$, then there is a unique solution for τ small enough (by the implicit function theorem.)

Convergence of functional iteration requires $\tau \|\partial \mathbf{g}/\partial \mathbf{y}\| < 1$. For stiff problems, $\|\partial \mathbf{g}/\partial \mathbf{y}\|$ is large, so this criterion severely restricts the stepsize in a similar manner as does stability. We have gained nothing. However, for nonstiff problems functional iteration may be acceptable, if one has another reason for using implicit methods.

Newton iteration. Given a nonlinear system $\mathbf{G}(\mathbf{y}) = 0$, Newton iteration is

$$\mathbf{y}^{(\nu+1)} := \mathbf{y}^{(\nu)} - \left(\frac{\partial \mathbf{G}}{\partial \mathbf{y}}(\mathbf{y}^{(\nu)}) \right)^{-1} \mathbf{G}(\mathbf{y}^{(\nu)}). \tag{A.3}$$

To solve the system (A.1), we take

$$\mathbf{G}(\mathbf{y}) = \mathbf{y} - \tau \mathbf{g}(\mathbf{y}) - \mathbf{c}, \qquad \partial \mathbf{G}/\partial \mathbf{y} = I - \tau \partial \mathbf{g}/\partial \mathbf{y}.$$

in (A.3). For τ sufficiently small in magnitude, Newton iteration converges quadratically: if the initial guess $\mathbf{y}^{(0)}$ is close enough to a stationary point $\bar{\mathbf{y}}$ of the iteration (A.3), then there exists a constant C such that

$$\|\mathbf{y}^{(\nu+1)} - \bar{\mathbf{y}}\| \le C \|\mathbf{y}^{(\nu)} - \bar{\mathbf{y}}\|^2.$$

Note that the 'smallness' of τ here is much larger than that for functional iteration just discussed. In particular, for linear $\mathbf{G}$, Newton iteration converges in one iteration, regardless of stiffness.

In practice, the evaluation of the Jacobian and its inverse in the iteration (A.3) is prohibitively expensive. Instead, the Jacobian is usually evaluated once per timestep

$$J = \frac{\partial \mathbf{g}}{\partial \mathbf{y}}(\mathbf{y}^{(0)}),$$

where the initial guess $\mathbf{y}^{(0)}$ is, for example, taken to be the solution from the previous time step $\mathbf{y}^{(0)} = \mathbf{y}_n$. The Jacobian can be factored using Gaussian elimination, $I - \tau J = LU$, and used repeatedly in (A.3). The resulting method is sometime called 'modified' Newton iteration. The computational cost of Gaussian elimination is $\mathcal{O}(d^3)$ whereas the solution of the systems with triangular matrices L and U can be done with a cost of $\mathcal{O}(d^2)$ for each system. So re-using the Jacobian results in a factor d savings in computational effort in succeeding iterations, assuming the convergence is unaffected.

Unfortunately the convergence *is* affected. In fact, the the modified Newton method is simply a functional iteration (A.2), with $\mathbf{g}(\mathbf{y})$ and $\mathbf{c}$ replaced with $(I-\tau J)^{-1}(\mathbf{g}(\mathbf{y}) - J\mathbf{y})$ and $(I-\tau J)^{-1}\mathbf{c}$, respectively. Fortunately the reduction in computational expense by not evaluating and factoring the Jacobian in each step usually outweighs the increase in expense due to slower convergence rate, and furthermore the inverse of the Jacobian tends to cancel the effect of stiffness.

Since the convergence of (modified) Newton iteration depends on τ, it may be faster to take more steps with a smaller stepsize, in each of which Newton converges in a few iterations, than to take a fewer, large steps in which convergence is sluggish. For this reason, practical codes usually perform a small number (say 10) modified Newton iterations, and if the convergence is not sufficient, throw out the whole step and reduce the stepsize. Clearly this adds additional complexity to the error control procedure we defined earlier.

A.2 Error Control

A.2.1 Error control

In this section we follow the approach to extrapolation and error control in Hairer, Nørsett & Wanner (1993).

A constant stepsize τ may be sufficient for academic studies. However, real industrial-strength ODE software makes use of variable steps to keep the error below a user-specified tolerance. In this section we develop one approach to stepsize selection based on local error estimation. We define $\tau_n := t_{n+1} - t_n$.

As mentioned before, the residual (**??**) obtained upon substitution of the Taylor expansion of the exact solution into the difference equation (**??**) is the incremental error introduced in the nth time step. Put another way, this is the error incurred in a single step of Euler's method for the initial value problem

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_n, \tag{A.4}$$

The first term of the residual is a good approximation of the incremental error (improving as $\tau \to 0$).

To control the error we first need a good estimate. The first term of the residual, while accurate, may not be available as computer code, and indeed may be a very complicated expression, since it involves higher derivatives of the function $\mathbf{f}$. Instead we will determine a better approximation of the solution of the local problem (A.4).

Richardson extrapolation

For the problem (A.4), a single step of size τ with a method of order p leads to a residual

$$\mathbf{r}_1 = \Phi^\tau \mathbf{y}_n - \mathbf{y}_{n+1} = \tau^{p+1}\boldsymbol{\kappa} + \mathcal{O}(\tau^{p+2}) \tag{A.5}$$

Consider instead taking two steps of size $\tau/2$. For the first of these we again have

$$\hat{\mathbf{r}}_{1/2} = \Phi^{\tau/2}\,\mathbf{y}_n - \hat{\mathbf{y}}_{n+1/2} = \left(\frac{\tau}{2}\right)^{p+1}\boldsymbol{\kappa} + \mathcal{O}(\tau^{p+2}),$$

where the hatted vectors refer to the second process with stepsize $\tau/2$. For the second step we get

$$\hat{\mathbf{r}}_1 = \Phi^{\tau/2}\,\hat{\mathbf{y}}_{n+1/2} - \hat{\mathbf{y}}_{n+1} = \left(\frac{\tau}{2}\right)^{p+1}\hat{\boldsymbol{\kappa}} + \mathcal{O}(\tau^{p+2}) \tag{A.6}$$

where $\hat{\boldsymbol{\kappa}} = \boldsymbol{\kappa} + \mathcal{O}(\tau)$ is the residual evaluated at $t_{n+1/2}$ instead of t_n. Now the total error made in the two steps is due to the propagation of $\hat{\mathbf{r}}_{1/2}$ under the exact solution through $\hat{\mathbf{y}}_{n+1/2}$ plus $\hat{\mathbf{r}}_1$. It can be shown that

$$\Phi^{\tau}\,\mathbf{y}_n - \Phi^{\tau/2}\,\hat{\mathbf{y}}_{n+1/2} = \left(\frac{\tau}{2}\right)^{p+1}\boldsymbol{\kappa} + \mathcal{O}(\tau^{p+2}) \tag{A.7}$$

Adding (A.6) and (A.7) gives

$$\Phi^{\tau}\,\mathbf{y}_n - \hat{\mathbf{y}}_{n+1} = 2\left(\frac{\tau}{2}\right)^{p+1}\boldsymbol{\kappa} + \mathcal{O}(\tau^{p+2})$$

The constant $\boldsymbol{\kappa}$ can be eliminated between the above relation and (A.5) to give an $\mathcal{O}(\tau^{p+2})$ approximation to $\mathbf{y}(t_{n+1})$:

$$\mathbf{y}(t_{n+1}) = \frac{2^p\,\hat{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1}}{2^p - 1} + \mathcal{O}(\tau^{p+2}).$$

A stepsize selection algorithm

Given the approximate solution $\mathbf{y}_{n+1}$ and a more accurate approximation $\bar{\mathbf{y}}_{n+1}$, obtained using Richardson extrapolation or otherwise, we wish to determine a stepsize for which the error remains within a user-specified tolerance. Specifically we want to check that

$$\|\bar{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1}\| \le tol,$$

and if this is not so, we reject the result, decrease the stepsize and try again. Defining

$$err = \frac{\|\bar{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1}\|}{tol}, \tag{A.8}$$

the criterion becomes $err \le 1$. Since we expect $err \approx C\tau^{p+1}$, the optimal stepsize stepsize is given by $C\tau_{\text{opt}}^{p+1} \approx 1$, we obtain an improved stepsize (by eliminating C)

$$\tau_{\text{opt}} = \tau / err^{1/(p+1)}.$$

Since rejected steps are expensive, we multiply the last expression by a safety factor *fac*, having a value of 0.8 or 0.9 or so. Furthermore it is recommended to place a limit on how fast the stepsize can grow and decay, giving

$$\tau_{\text{new}} = \tau \cdot \min(facmax, \max(facmin, fac \cdot (1/err)^{1/(p+1)})) \tag{A.9}$$

where $facmax \approx 2$ and $facmin \approx 0.5$.

To apply the above formula in a code, after computing a solution at time t_n, the error is estimated using (A.8) and this is compared against 1. If the $err \le 1$, the step is accepted and (A.9) gives an estimate for the stepsize τ_{n+1} of the next step. If $err > 1$, the step is rejected, and (A.9) gives an estimate of τ_n for a new attempt.

Example.
We integrate the Van der Pol equation with parameter $\mu = 5$ this time. Figure A.1 shows the global error as a function of the number of steps. The reduction in the number of steps is approximately a factor of 3. This does not mean that the computation is a factor of 3 faster however. In the first place, the Richardson extrapolation involves a second evaluation of the function $\mathbf{f}$. For large, complex systems, this means the expense of a single step approximately doubles compared to the method without stepsize control. Furthermore, any discarded steps are wasted effort, so a real measure of the benefits of error control must take those into account as well.

Code `vareuler.m` (`http://www.cwi.nl/~jason/numwisk/vareuler.m`)

```
function [Y,T] = vareuler(y0,tau,Tend,fun,param,tol);
%
%  [Y,T] = vareuler(y0,tau,Tend,fun,param,tol);
%
%  Integrates the function 'yprime = fun (y,t,param)'
%  using Forward Euler.
%
%  y0 is the initial condition
%  tau is an initial stepsize guess
%  Tend is the length of the integration interval
%  tol is an absolute tolerance
%
%  The output is stored columnwise in the array Y
%  The array T contains the temporal grid
%

Y = y0;  %% output array
y = y0;  %% solution at current time level
T = 0;   %% output array
t = 0;   %% current time level

facmin = 0.5;
facmax = 2;
fac = 0.8;


while t<Tend,  %% Note:  we do not know in advance how many steps are needed
   remesh = 1; %% Assume remeshing is necessary (1==true, 0==false)

   while remesh

      %% Euler step
      F = fun(y,param);
      y1 = y + tau * F;

      %% Richardson extrapolation
      ya = y + tau/2 * F;
      yb = ya + tau/2 * fun(ya,param);
      y2 = 2*yb-y1;

      %% Error estimate:
```

```
      err = max(abs(y2-y1))/tol;

      if err <= 1
         remesh = 0;  %% remeshing unnecessary, drop out of the loop
         t = t + tau; %% increment time
      end

      %% Estimate next timestep
      tau = min(Tend-t,tau * min(facmax,max(facmin,fac/sqrt(err))));
   end

   y = y1;

   Y(:,end+1) = y;
   T(end+1) = t;

end
```

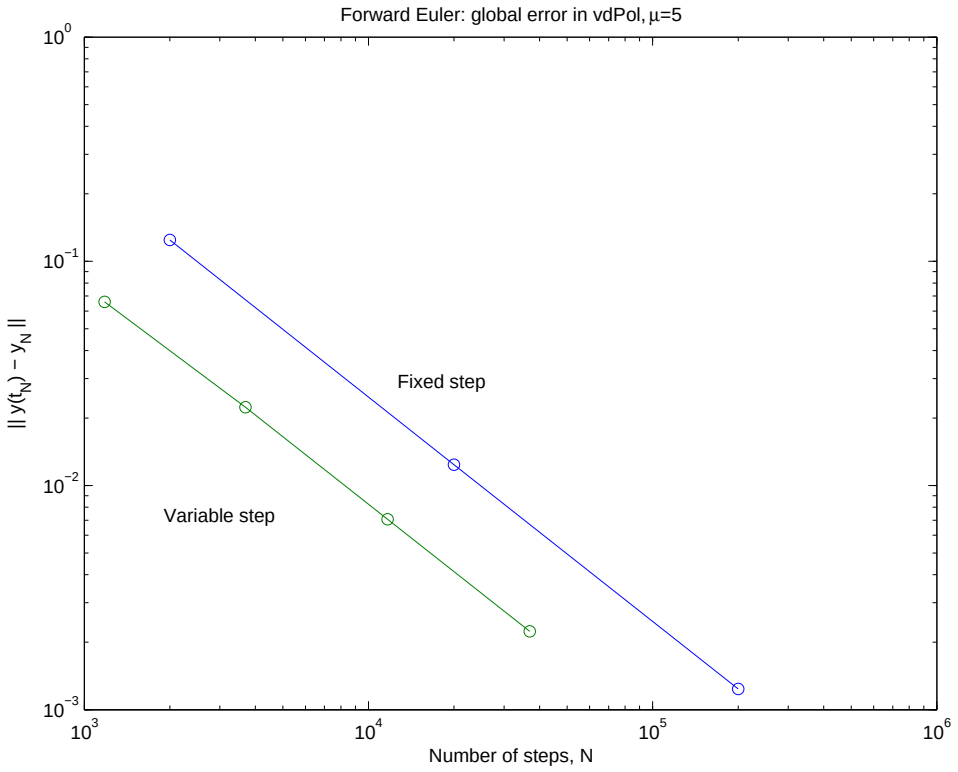

Figure A.1: Solution to Example 1, with $\mu = 5$, using Euler's method with fixed and variable stepsize.