Initial Value Problems: ODEs

M.Sc. in Mathematical Modelling & Scientific Computing,
Practical Numerical Analysis

Michaelmas Term 2020, Lecture 4

The Problem

We wish to find $\mathbf{u}(t)$ such that

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathbf{f}(t, \mathbf{u}) \; ,$$

for $t > 0$ with $\mathbf{u}(0) = \mathbf{u}_0$.

We shall write everything in terms of the scalar problem: find $u(t)$ such that

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f(t, u) \; ,$$

for $t > 0$ with $u(0) = u_0$, but all methods are easily generalised.

Simplest Methods — Euler Methods

Perhaps the simplest numerical methods are the explicit and implicit Euler methods (also known as forward and backward Euler):

$$
\begin{aligned}
\frac{U_{n+1} - U_n}{\Delta t} &= f(t_n, U_n), \quad \text{(explicit/forward Euler)} \\
\frac{U_{n+1} - U_n}{\Delta t} &= f(t_{n+1}, U_{n+1}), \quad \text{(implicit/backward Euler)}
\end{aligned}
$$

for $n = 0, 1, \ldots$ and with $U_0 = u_0$.

Simplest Methods — Euler Methods

Explicit Euler is particularly simple. Given $U_0 = u_0$ and the function f we compute

$$U_{n+1} \quad = \quad U_n + \Delta t f(t_n, U_n)$$

for $n = 0, 1, \ldots$.

Implicit Euler is more complex in the sense that if we are given $U_0 = u_0$ and the function f we compute U_{n+1} as the solution to the *nonlinear* equation

$$U_{n+1} \quad = \quad U_n + \Delta t f(t_{n+1}, U_{n+1})$$

for $n = 0, 1, \ldots$. The solution to this nonlinear equation can be computed by (say) Newton's method. At timestep $n + 1$ a good starting guess for Newton's method is U_n.

Generalisation — θ-Methods

Both the explicit and implicit Euler methods are specific cases of the θ-method which is given by

$$\frac{U_{n+1} - U_n}{\Delta t} = \theta f(t_{n+1}, U_{n+1}) + (1-\theta) f(t_n, U_n)$$

for $n = 0, 1, \ldots$ and with $U_0 = u_0$. Special cases are

- $\theta = 0$ — explicit Euler
- $\theta = 1$ — implicit Euler
- $\theta = 1/2$ — Crank Nicolson method

For all non-zero values of θ, the method is implicit and a nonlinear equation must be solved at each time-step.

Truncation Error

All the methods can be derived by truncating Taylor series and the truncation error measures the error commited by doing this. The truncation error is defined as

$$T_n \;=\; \frac{u_{n+1} - u_n}{\Delta t} - \theta f(t_{n+1}, u_{n+1}) - (1-\theta) f(t_n, u_n)\;,$$

where $u_n = u(t_n)$ is the exact solution at the point t_n.

It can be shown (using Taylor series expansions) that for constant θ

$$T_n \;=\; \begin{cases} \mathcal{O}(\Delta t) & \text{for } \theta \neq 1/2 \\ \mathcal{O}(\Delta t^2) & \text{for } \theta = 1/2 \end{cases}$$

so that the truncation error of the Crank Nicolson scheme converges twice as fast as that of all other θ-methods.

Pointwise Errors

It can be shown that if the right-hand-side function $f(t, u)$ satisfies a Lipschitz condition, with Lipschitz constant L, then

$$|e_n| \leq \left(\frac{1+(1-\theta)L\Delta t}{1-\theta L\Delta t}\right)^n |e_0| + \frac{T}{L}\left[\left(\frac{1+(1-\theta)L\Delta t}{1-\theta L\Delta t}\right)^n - 1\right],$$

for $n = 0, 1, \ldots$ and where T is a bound on the truncation error.

This, along with the fact that

$$\begin{aligned} \frac{1+(1-\theta)L\Delta t}{1-\theta L\Delta t} &= 1 + \frac{L\Delta t}{1-\theta L\Delta t} \\ &\leq \exp\left(\frac{L\Delta t}{1-\theta L\Delta t}\right), \end{aligned}$$

can be used to determine how many steps of an algorithm are required to achieve a desired accuracy.

Modifications of θ-Methods

As we have already stated, unless $\theta = 0$, the θ-method requires us to solve a nonlinear equation at each timestep. However, there exist modified methods to avoid this.

Recall the Crank Nicolson scheme

$$\frac{U_{n+1} - U_n}{\Delta t} = \frac{1}{2}\left(f(t_{n+1}, U_{n+1}) + f(t_n, U_n)\right) .$$

We can approximate U_{n+1} using the explicit Euler scheme. This leads to the improved Euler method

$$\frac{U_{n+1} - U_n}{\Delta t} = \frac{1}{2}\left(f(t_{n+1}, U_n + \Delta t f(t_n, U_n)) + f(t_n, U_n)\right) .$$

This has a truncation error $T_n = \mathcal{O}(\Delta t^2)$.

Runge-Kutta Schemes

The improved Euler method is a specific example of an explicit Runge-Kutta scheme. Such schemes take the general form

$$\frac{U_{n+1} - U_n}{\Delta t} = \sum_{i=1}^{s} b_i k_i$$

where

$$k_1 = f(t_n, U_n)$$

and

$$k_i = f(t_n + c_i \Delta t, U_n + \Delta t \sum_{j=1}^{i-1} a_{i,j} k_j) \ ,$$

for $i = 2, \ldots, s$.

Runge-Kutta Schemes — Butcher Tableaux

The coefficients of explicit Runge-Kutta schemes are chosen to make the methods as high order as possible and are often stored as Butcher tableaux in the form

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{2,1} & & & & \\
c_3 & a_{3,1} & a_{3,2} & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_s & a_{s,1} & a_{s,2} & \dots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \dots & b_{s-1} & b_s
\end{array}
$$

Example: Improved Euler

The improved Euler scheme

$$\frac{U_{n+1} - U_n}{\Delta t} = \frac{1}{2}\left(f(t_{n+1}, U_n + \Delta t f(t_n, U_n)) + f(t_n, U_n)\right) ,$$

can be written in the form

$$\frac{U_{n+1} - U_n}{\Delta t} = \frac{1}{2}(k_1 + k_2) ,$$

where

$$k_1 = f(t_n, U_n)$$

and

$$k_2 = f(t_n + \Delta t, U_n + \Delta t k_1) .$$

Thus the Butcher tableau takes the form

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Example: Modified Euler

Another commonly used 2-stage scheme is the modified Euler scheme, given by

$$\frac{U_{n+1} - U_n}{\Delta t} = f\left(t_n + \frac{1}{2}\Delta t, U_n + \frac{1}{2}\Delta t f(t_n, U_n)\right) .$$

The Butcher tableau for the modified Euler scheme takes the form

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$

Example: RK4

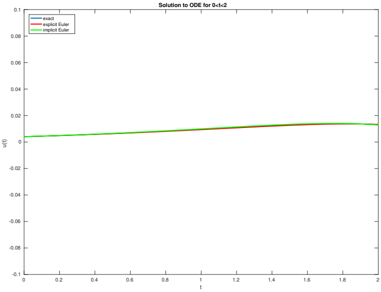
Finally, a very common 4-stage method is the so-called RK4 scheme, defined by the Butcher tableau

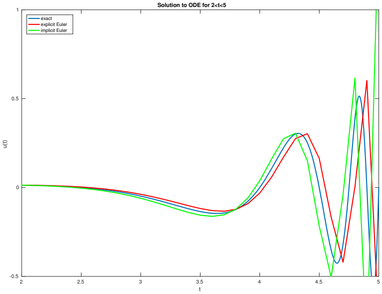
$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

Adaptivity — Motivation

If an IVP has a solution with different timescales (i.e. a region of rapid change and a region of much less rapid change) then using a uniform timestep can be either inaccurate or inefficient.

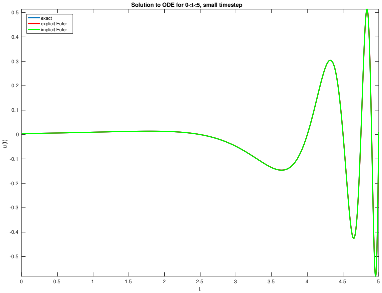
If the timestep is too large it may capture the slowly varying part of the solution but not that which is rapidly varying.





Adaptivity — Motivation

If the timestep is small it may be very inefficient for the slowly changing part of the solution.



Remedy: use a large timestep when the solution does not change rapidly and a small timestep when it does.

Runge Kutta Methods and Adaptivity (1)

One step methods can easily be modified to have adaptivity of the timestep length Δt as the function u varies. As an example, consider the fourth order Runge-Kutta method (RK4).

The background theory is that

$$\begin{aligned} \text{Trunction error} \qquad T_n &\sim K_1(\Delta t)^4 u^{(v)} \\ \text{Local error} \qquad |e_n| &\sim K_2 \Delta t |T_n| \\ \text{so} \qquad |e_n| &\sim K_3(\Delta t)^5 u^{(v)}. \end{aligned}$$

Runge Kutta Methods and Adaptivity (1)

Suppose we are at t_n and want to use a step Δt_n

1. apply RK4 over step Δt_n to get value U_a where

$$U_a = u_{n+1} + \frac{(\Delta t_n)^5 u^{(v)}}{5!} + \mathcal{O}(\Delta t_n^6)$$

2. apply RK4 *twice* over step $\Delta t_n/2$ to get value U_b

$$U_b = u_{n+1} + \frac{2(\frac{\Delta t_n}{2})^5 u^{(v)}}{5!} + \mathcal{O}(\Delta t_n^6)$$

Then

$$U_a - U_b \sim \frac{15}{16}\frac{u^{(v)}}{5!}(\Delta t_n)^5 \sim K_4 e_n.$$

Runge Kutta Methods and Adaptivity (1)

Hence for fixed Δt_n:

- if $|U_a - U_b|$ is large then so too is the error $|U_a - u_{n+1}|$ so the step length should be decreased
- if $|U_a - U_b|$ is small, so is the error so we could take a larger step.

Of course we have to do more work each step since we effectively apply RK4 *three* times per timestep. The hope is that being able to use larger timesteps compensates for this.

Runge Kutta Methods and Adaptivity (1)

Since we have

$$U_a - U_b \;\sim\; \frac{15}{16}\frac{u^{(v)}}{5!}(\Delta t_n)^5 \;\sim\; K_4 e_n,$$

we may write

$$|U_a - U_b| \;=\; c(\Delta t_n)^5.$$

Hence, if we require

$$|U_a - U_b| \;\leq\; \mathtt{TOL}$$

then we should choose the new timestep, Δt, to satisfy

$$c(\Delta t)^5 \;=\; \frac{|U_a - U_b|}{(\Delta t_n)^5}(\Delta t)^5 \;\leq\; \mathtt{TOL}$$

or equivalently

$$\Delta t \;\leq\; \left(\frac{\mathtt{TOL}}{|U_a - U_b|}\right)^{1/5} \Delta t_n.$$

Runge Kutta Methods and Adaptivity (1)

Algorithm: User provides start time, end time, tolerance.

1. Set TOL=tolerance
2. Set t_0 =start time
3. while $t_n \leq$ end time
   - 3.1 apply RK4 to determine U_a and U_b with current Δt_n
   - 3.2 if $|U_a - U_b| >$TOL, step fails, set

$$\Delta t_n = \left(\frac{\texttt{TOL}}{|U_a - U_b|}\right)^{1/5} \Delta t_n$$

and go back to 3.1. (This reduces the step and repeats.)
else $|U_a - U_b| \leq$TOL, set

$$\begin{aligned} \Delta t_{n+1} &= \left(\frac{\texttt{TOL}}{|U_a - U_b|}\right)^{1/5} \Delta t_n \qquad (1) \\ U_{n+1} &= U_b \\ t_{n+1} &= t_n + \Delta t_n \\ n &= n + 1 \end{aligned}$$

(this increases the step length for *next* step).

end while

Runge Kutta Methods and Adaptivity (1)

As we have been dealing with local error the lengthening of step can be misleading, the global error has order $(\Delta t)^4$ so in (1) above can use

$$\Delta t_{n+1} \;=\; \left(\frac{\mathtt{TOL}}{|U_a - U_b|}\right)^{1/4} \Delta t_n.$$

This is more robust in practice.

Runge Kutta Methods and Adaptivity (2)

An alternative to the method described above is to use two Runge Kutta methods, one of order p and one of order $\tilde{p} \geq p+1$.

Let U_{n+1} be the numerical approximation to $u(t_{n+1})$ using the pth order method and let $\tilde{U}_{n+1}$ be the numerical approximation to $u(t_{n+1})$ using the $\tilde{p}$th order method.

Then (making error free assumption, i.e. all earlier iterates are exact)

$$U_{n+1} = u(t_{n+1}) + c\Delta t^{p+1} + \mathcal{O}(\Delta t^{p+2}) , \qquad (2)$$

$$\tilde{U}_{n+1} = u(t_{n+1}) + \mathcal{O}(\Delta t^{p+2}) , \qquad (3)$$

as $\Delta t \to 0$. Here c depends on the derivative of u. Subtracting gives

$$U_{n+1} - \tilde{U}_{n+1} \approx c\Delta t^{p+1} ,$$

and substituting this in (2) gives

$$U_{n+1} - u(t_{n+1}) \approx U_{n+1} - \tilde{U}_{n+1} .$$

Runge Kutta Methods and Adaptivity (2)

We can use this final equation

$$U_{n+1} - u(t_{n+1}) \;\approx\; U_{n+1} - \tilde{U}_{n+1}\,,$$

to determine when to refine the mesh in an adaptive algorithm.

Now the idea is to choose the ERK schemes so that the pth order method has nodes and a matrix which are a subset of those in the $\tilde{p}$th order method so that the values can be re-used. This approach is called an *embedded RK pair*.

Runge Kutta Methods and Adaptivity (2)

Example 1: Choose the pair consisting of the Butcher tableaux

$$\begin{array}{c|cc} 0 & & \\ \frac{2}{3} & \frac{2}{3} & \\ \hline & \frac{1}{4} & \frac{3}{4} \end{array}$$

and

$$\begin{array}{c|ccc} 0 & & & \\ \frac{2}{3} & \frac{2}{3} & & \\ \frac{2}{3} & 0 & \frac{2}{3} & \\ \hline & \frac{1}{4} & \frac{3}{8} & \frac{3}{8} \end{array}$$

Runge Kutta Methods and Adaptivity (2)

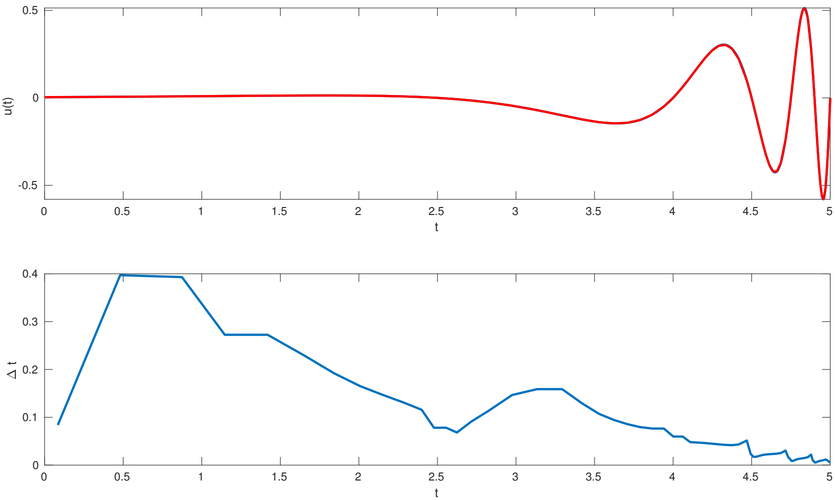
Example 2: Choose the pair consisting of the Butcher tableaux

$$
\begin{array}{c|ccccc}
0 & & & & & \\
1/4 & 1/4 & & & & \\
3/8 & 3/32 & 9/32 & & & \\
12/13 & 1932/2197 & -7200/2197 & 7296/2197 & & \\
1 & 439/216 & -8 & 3680/513 & -845/4104 & \\
\hline
 & 25/216 & 0 & 1408/2565 & 2197/4104 & -1/5
\end{array}
$$

and

$$
\begin{array}{c|cccccc}
0 & & & & & & \\
1/4 & 1/4 & & & & & \\
3/8 & 3/32 & 9/32 & & & & \\
12/13 & 1932/2197 & -7200/2197 & 7296/2197 & & & \\
1 & 439/216 & -8 & 3680/513 & -845/4104 & & \\
1/2 & -8/27 & 2 & -3544/2565 & 1859/4104 & -11/40 & \\
\hline
 & 16/135 & 0 & 6656/12825 & 28561/56430 & -9/50 & 2/55
\end{array}
$$

Results



0.5
0
-0.5
u(t)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
t
0.4
0.3
0.2
0.1
0
Δ t
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
t