Initial Value Problems: ODEs

Runge-Kutta Schemes

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

Michaelmas Term 2026, Lecture 5

The Problem

We continue to study the scalar first order initial value problem: find $u(t)$ such that

$$\begin{aligned} u'(t) &= f(t,u), \quad t>0 \\ u(0) &= u_0. \end{aligned} \tag{1}$$

Last time, in order to solve (1) numerically over the time interval $[0,T]$, we defined a set of time points at which we wish to approximate the solution. We set $t_n = n\Delta t$ for $n = 0,1,\ldots,N$ where $\Delta t = T/N$. We then used the θ-method to approximate $u(t_n)$ by U_n satisfying

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

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

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 $T_n = \mathcal{O}(\Delta t^2)$.

Explicit 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

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

for $i = 2, \ldots, s$. The k_i are known as the stages of the method and the method is often referred to as an s-stage method.

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

0		
$\frac{1}{2}$	$\frac{1}{2}$	
	0	1

Example: RK4

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

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

Truncation Error for Explicit Runge-Kutta Schemes

The truncation error for an explicit Runge-Kutta scheme is given by

$$T_n = \frac{u(t_{n+1}) - u(t_n)}{\Delta t} - \sum_{i=1}^{s} b_i \tilde{k}_i,$$

where

$$\tilde{k}_1 = f(t_n, u(t_n))$$

and

$$\tilde{k}_i = f\left(t_n + c_i \Delta t, u(t_n) + \Delta t \sum_{j=1}^{i-1} a_{i,j} \tilde{k}_j\right),$$

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

Truncation Error Example

Recall the improved Euler scheme 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

$$T_n = \frac{u(t_{n+1}) - u(t_n)}{\Delta t} - \frac{1}{2}(\tilde{k}_1 + \tilde{k}_2),$$

where

$$\begin{aligned} \tilde{k}_1 &= f(t_n, u(t_n)), \\ \tilde{k}_2 &= f(t_n + \Delta t, u(t_n) + \Delta t \tilde{k}_1). \end{aligned}$$

Truncation Error Example

Performing a Taylor series expansion of $u(t_{n+1})$ about the point t_n gives

$$u(t_{n+1}) = u(t_n) + \Delta t u'(t_n) + \frac{1}{2}\Delta t^2 u''(t_n) + \frac{1}{6}\Delta t^3 u'''(t_n) + \mathcal{O}(\Delta t^4).$$

Also performing a Taylor series expansion of $\tilde{k_2}$ about the point $(t_n, u(t_n))$ to get

$$\begin{aligned}
\tilde{k_2} &= f(t_n + \Delta t, u(t_n) + \Delta t \tilde{k_1}) \\
&= f(t_n + \Delta t, u(t_n) + \Delta t f(t_n, u(t_n))) \\
&= f(t_n, u(t_n)) + \Delta t f_t(t_n, u(t_n)) + \Delta t f(t_n, u(t_n)) f_u(t_n, u(t_n)) \\
&\quad + \frac{1}{2}\Delta t^2 f_{tt}(t_n, u(t_n)) + \Delta t^2 f(t_n, u(t_n)) f_{tu}(t_n, u(t_n)) \\
&\quad + \frac{1}{2}\Delta t^2 (f(t_n, u(t_n)))^2 f_{uu}(t_n, u(t_n)) + \mathcal{O}(\Delta t^3).
\end{aligned}$$

Truncation Error Example

Now substitute these expansions into the expression for the truncation error

$$\begin{aligned} T_n &= \frac{u(t_{n+1}) - u(t_n)}{\Delta t} - \frac{1}{2}\left[f(t_n, u(t_n)) \right. \\ &\quad \left. + f\left(t_n + \Delta t, u(t_n) + \Delta t f(t_n, u(t_n))\right)\right], \end{aligned}$$

simplify and reorder to get

$$\begin{aligned} T_n &= \left(u' - f\right) + \frac{\Delta t}{2}\left[u'' - \left(f_t + f\ f_u\right)\right] \\ &\quad + \Delta t^2\left[\frac{1}{6}u''' - \frac{1}{4}\left(f_{tt} + 2f\ f_{tu} + f^2 f_{uu}\right)\right] + \mathcal{O}(\Delta t^3), \end{aligned}$$

where we have suppressed the arguments so that u represents $u(t_n)$ and f represents $f(t_n, u(t_n))$ (and similarly for all (partial) derivatives).

Truncation Error Example

Now recall that

$$u'(t) = f(t, u(t)),$$

so that the chain rule gives

$$\begin{aligned} u''(t) &= f_t + f\ f_u, \\ u'''(t) &= f_{tt} + 2f\ f_{tu} + f_t f_u + f\ f_u^2 + f^2 f_{uu}. \end{aligned}$$

Then we have

$$T_n = \frac{\Delta t^2}{12}\left(2f_t f_u + 2f\ f_u^2 - \left(f_{tt} + 2f\ f_{tu} + f^2 f_{uu}\right)\right) + \mathcal{O}(\Delta t^3).$$

So $T_n = \mathcal{O}(\Delta t^2)$ and the improved Euler scheme is second order.

Order of Runge-Kutta Schemes

Theorem

The order p of an explicit s-stage Runge-Kutta method is bounded by $p \leq s$.

It is possible to construct Runge-Kutta methods that achieve this maximal order.

Example 1

We revisit Example 1 from last time, namely

$$\begin{aligned} u'(t) &= \lambda u, \quad t > 0, \\ u(0) &= 1. \end{aligned}$$

Numerical schemes for this are:

- Explicit Euler:

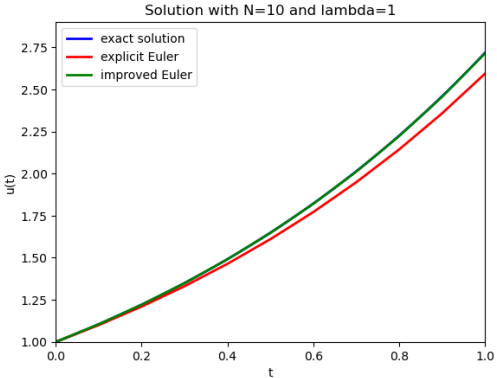
$$U_{n+1} = U_n + \lambda \Delta t U_n = (1 + \lambda \Delta t) U_n.$$

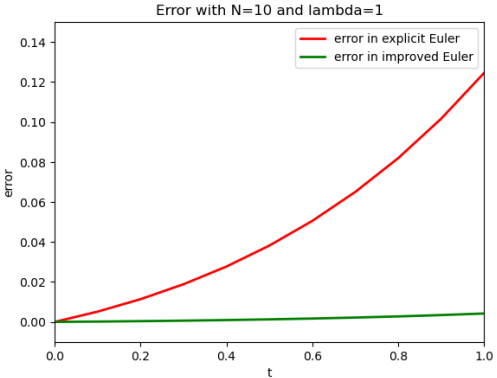
- Improved Euler:

$$U_{n+1} = \left(1 + \lambda \Delta t + \frac{1}{2}(\lambda \Delta t)^2\right) U_n.$$

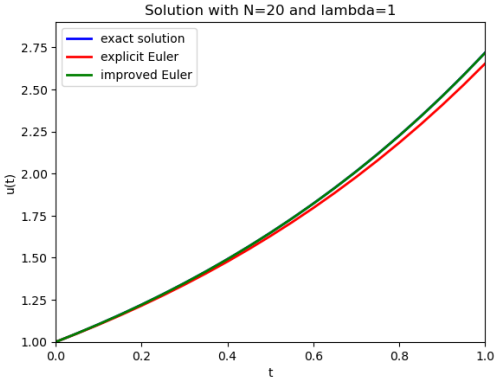
for $n = 0, 1, \ldots, N-1$ and with $U_0 = 1$.

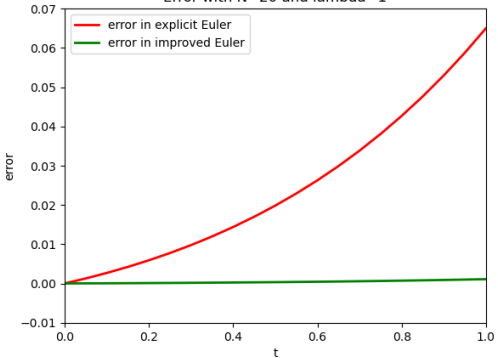
Example 1 Results



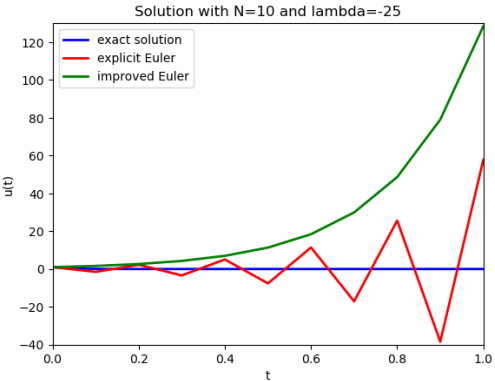


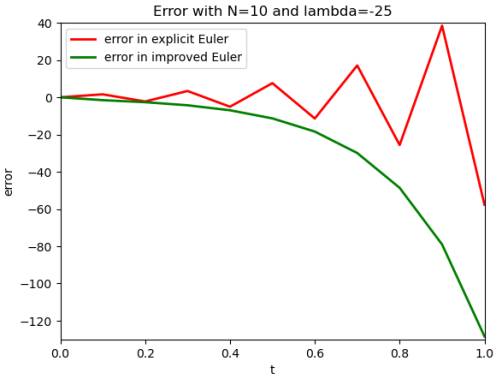
Example 1 Results



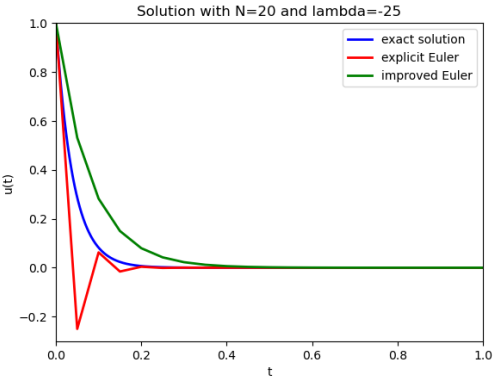


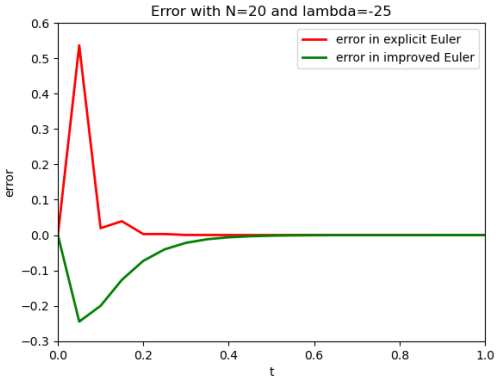
Example 1 Results





Example 1 Results





Explanation — Stability

We are solving

$$\begin{aligned} u'(t) &= \lambda u, \quad t > 0, \\ u(0) &= 1, \end{aligned}$$

which has exact solution $u(t) = \mathrm{e}^{\lambda t}$.

If $\lambda < 0$ the exact solution satisfies

$$\lim_{t \to \infty} u(t) = 0.$$

We would like our numerical scheme to mimic this behaviour.

Explanation — Stability

The explicit Euler scheme for this problem is

$$U_{n+1} = (1 + \lambda \Delta t) U_n,$$

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

Thus the explicit Euler solution is

$$U_n = (1 + \lambda \Delta t)^n.$$

For $\lambda < 0$ we require

$$\lim_{n \to \infty} U_n = 0.$$

This holds if

$$-1 < 1 + \lambda \Delta t < 1,$$

or equivalently if

$$\Delta t < \frac{2}{|\lambda|}.$$

Explanation — Stability

We can perform a similar analysis for the improved Euler scheme.

Improved Euler:

$$U_{n+1} = \left(1 + \lambda\Delta t + \frac{1}{2}(\lambda\Delta t)^2\right) U_n,$$

for $n = 0, 1, \ldots, N-1$ and with $U_0 = 1$.Then

$$U_n = \left(1 + \lambda\Delta t + \frac{1}{2}(\lambda\Delta t)^2\right)^n$$

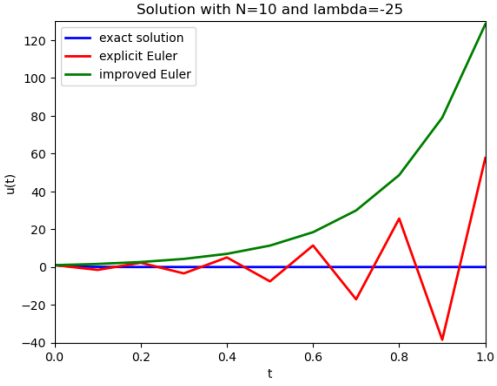
so that $\lim_{n\to\infty} U_n = 0$ if

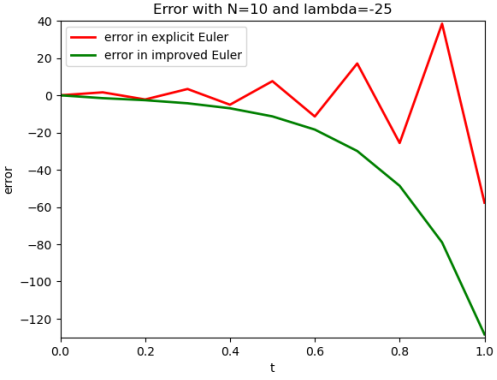
$$-1 < 1 + \lambda\Delta t + \frac{1}{2}(\lambda\Delta t)^2 < 1,$$

or equivalently if

$$\Delta t < \frac{2}{|\lambda|}.$$

Example 1 Results Revisited



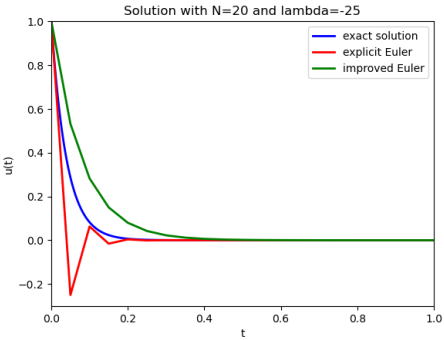


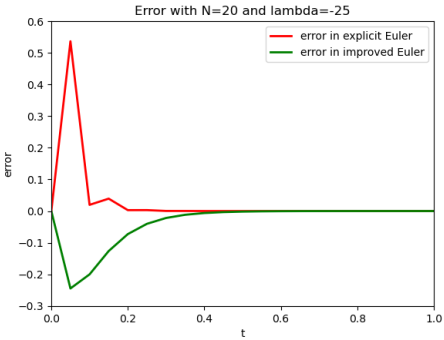
Here $\lambda = -25$ and $\Delta t = 0.1$ so $\Delta t > 2/|\lambda|$.

For explicit Euler we have $U_n = (1 + \lambda \Delta t)^n = (-1.5)^n$.

For improved Euler we have $U_n = (1 + \lambda \Delta t + (\lambda \Delta t)^2/2)^n = 1.625^n$.

Example 1 Results Revisited





Here $\lambda = -25$ and $\Delta t = 0.05$ so $\Delta t < 2/|\lambda|$.

For explicit Euler we have $U_n = (1 + \lambda \Delta t)^n = (-0.25)^n$.

For improved Euler we have $U_n = (1 + \lambda \Delta t + (\lambda \Delta t)^2/2)^n = 0.53125^n$.

The exact solution is $u(t_n) = \mathrm{e}^{\lambda t_n} = \mathrm{e}^{\lambda n \Delta t} = \left(\mathrm{e}^{\lambda \Delta t}\right)^n \approx 0.2865^n$.

Interval of Absolute Stability

The interval of absolute stability of a numerical scheme is the interval of values of $\lambda \Delta t$ such that, when applied to the problem

$$\begin{aligned} u'(t) &= \lambda u, \quad t > 0, \\ u(0) &= 1, \end{aligned}$$

the numerical solutions satisfy

$$\lim_{n \to \infty} U_n = 0.$$

Thus the interval of absolute stability of both the explicit Euler and improved Euler schemes is $\lambda \Delta t \in (-2, 0)$.

Interval of Absolute Stability

It can be shown that for the RK4 scheme applied to $u'(t) = \lambda u$, the numerical solution satisfies

$$U_{n+1} = \left(1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}\right) U_n,$$

where $z = \lambda \Delta t$.

The Python command `polyroots([1,1,1/2,1/6,1/24])` (from the numpy.polynomial.polynomial module) reveals only complex roots so

$$1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24} > 0.$$

To find when

$$1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24} < 1,$$

we can use `polyroots([0,1,1/2,1/6,1/24])` which gives real roots at $z = 0$ and $z = -2.7853$ (as well as two complex roots). Thus the RK4 scheme is stable for $\lambda \Delta t \in (-2.7853, 0)$.

Aside: Interval of Absolute Stability for θ-Method

We have been talking about explicit Runge-Kutta schemes but the idea of stability also holds for the θ-method discussed last time.

Recall that the θ-method is

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

so that for the problem

$$\begin{aligned} u'(t) &= \lambda u, \quad t > 0, \\ u(0) &= 1, \end{aligned}$$

we have

$$\frac{U_{n+1} - U_n}{\Delta t} = \theta \lambda U_{n+1} + (1-\theta)\lambda U_n.$$

Aside: Interval of Absolute Stability for θ-Method

Re-arranging gives

$$U_{n+1} = \frac{1+(1-\theta)\lambda\Delta t}{1-\theta\lambda\Delta t}U_n.$$

Thus the interval of absolute stability is the range of values of $\lambda\Delta t$ for which

$$-1 < \frac{1+(1-\theta)\lambda\Delta t}{1-\theta\lambda\Delta t} < 1.$$

After some algebra we find that this is satisfied if

$$2+(1-2\theta)\lambda\Delta t > 0.$$

Two cases:

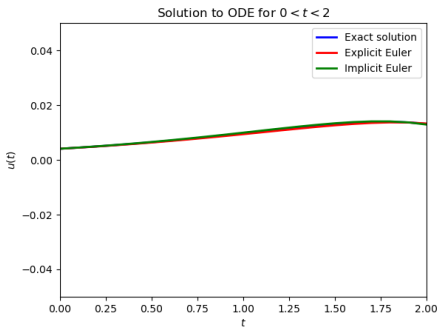
- If $\theta \in [1/2, 1]$, this holds for any negative value of $\lambda\Delta t$.
- If $\theta \in [0, 1/2)$, then we require

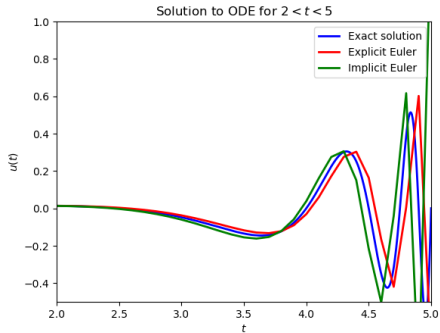
$$\lambda\Delta t \in \left(-\frac{2}{1-2\theta}, 0\right).$$

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.

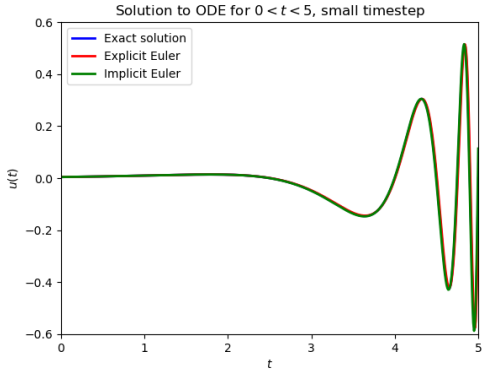
If the timestep is too large it may capture the slowly varying part of the solution but not that which is rapidly varying.





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

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

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

$$U_a = u_{n+1} + K_3(\Delta t_n)^5 u^{(v)} + \mathcal{O}(\Delta t_n^6)$$

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

$$U_b = u_{n+1} + 2K_3 \left(\frac{\Delta t_n}{2}\right)^5 u^{(v)} + \mathcal{O}(\Delta t_n^6)$$

Then

$$U_a - U_b \sim \frac{15}{16} K_3 u^{(v)} (\Delta t_n)^5 \sim K_4 e_n.$$

Runge Kutta Methods and Adaptivity (1)

Hence for fixed Δt_n:

- if $|U_a - U_b|$ is large then so 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

$$U_a - U_b \;\sim\; \frac{15}{16} K_3 u^{(v)} (\Delta t_n)^5 \;\sim\; K_4 e_n,$$

we may write

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

Hence, if we require

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

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

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

or equivalently

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

Runge Kutta Methods and Adaptivity (1) — Pseudocode

Input: t_0, T (final time), TOL, Δt_0 (initial timestep)

Output: t_n and U_n, $0 \leq n \leq N$

$n \leftarrow 0$

while $t_n \leq T$ **do**

  error $\leftarrow$ 2*TOL

  while error>TOL **do**

    Compute U_a: one step of RK4 and timestep Δt_n

    Compute U_b: two steps of RK4 and timestep $\Delta t_n/2$

    error $\leftarrow |U_a - U_b|$

    if error>TOL **then** ▷ step fails

      $\Delta t_n \leftarrow \left(\frac{\texttt{TOL}}{|U_a - U_b|}\right)^{1/5} \Delta t_n$ ▷ reduce Δt, repeat

    end if

  end while

  $\Delta t_{n+1} \leftarrow \left(\frac{\texttt{TOL}}{|U_a - U_b|}\right)^{1/5} \Delta t_n$ ▷ increase Δt for *next* step

  $U_{n+1} \leftarrow U_b$, $t_{n+1} \leftarrow t_n + \Delta t_n$, $n \leftarrow n+1$ ▷ accept solution

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 step 12: above can use

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

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)

$$U_{n+1} = u(t_{n+1}) + c\Delta t^{p+1} + \mathcal{O}(\Delta t^{p+2}), \quad (3)$$

$$\tilde{U}_{n+1} = u(t_{n+1}) + \mathcal{O}(\Delta t^{p+2}), \quad (4)$$

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 (3) 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

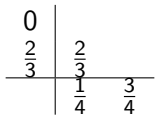
$$U_{n+1} - u(t_{n+1}) \quad \approx \quad U_{n+1} - \tilde{U}_{n+1},$$

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

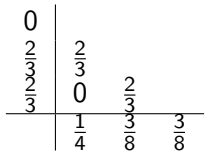
Now the idea is to choose the explicit Runge Kutta schemes so that the pth order method has nodes and a matrix which are a subset of those in the $\tilde{p}$th order method so that the values can be re-used. This approach is called an *embedded RK pair*.

Runge Kutta Methods and Adaptivity (2)

Example 1: Choose the pair consisting of the Butcher tableaux



and



Runge Kutta Methods and Adaptivity (2)

Example 2: Choose the pair consisting of the Butcher tableaux

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	
	25/216	0	1408/2565	2197/4104	−1/5

and

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	
	16/135	0	6656/12825	28561/56430	−9/50	2/55

Results

