Math 6630: Numerical Solutions of Partial Differential Equations
Solvers for initial value problems, Part III

See Ascher and Petzold 1998, Chapters 1-5

Akil Narayan[1]

[1] Department of Mathematics, and Scientific Computing and Imaging (SCI) Institute
University of Utah

February 1, 2023

Initial value problems

$$\boldsymbol{u}'(t) = \boldsymbol{f}(t; \boldsymbol{u}), \qquad \boldsymbol{u}(0) = \boldsymbol{u}_0.$$

$$\boldsymbol{u}_n \approx \boldsymbol{u}(t_n)$$

$$\boldsymbol{u}_{n+1} \approx \boldsymbol{u}_n + \int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t$$

We have previously discussed

- Simple schemes: forward/backward Euler, Crank-Nicolson
- Consistency and LTE
- 0-stability and scheme convergence
- absolute/A-stability and consequences

Now we'll delve into more advanced schemes, in particular multi-stage schemes.

Higher-order schemes

The schemes we've seen previously are relatively low order: first order for Euler-type, and second order for Crank-Nicolson.

Recall that our schemes result from discretization (approximation) of an integral:

$$\begin{aligned} \boldsymbol{u}(t_{n+1}) &= \boldsymbol{u}(t_n) + \int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t \\ \boldsymbol{u}_{n+1} &\approx \boldsymbol{u}_n + \int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t. \end{aligned}$$

Our choices so far were to

- Use a one-point approximation using the left-hand value (forward Euler)
- Use a one-point approximation using the right-hand value (backward Euler)
- Use a two-point Trapezoidal approximation (Crank-Nicolson)

Higher-order schemes

The schemes we've seen previously are relatively low order: first order for Euler-type, and second order for Crank-Nicolson.

Recall that our schemes result from discretization (approximation) of an integral:

$$\boldsymbol{u}(t_{n+1}) = \boldsymbol{u}(t_n) + \int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t$$
$$\boldsymbol{u}_{n+1} \approx \boldsymbol{u}_n + \int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t.$$

In moving foward, we could consider the approximation

$$\int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t \approx \sum_{j=1}^{s} k b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{u}(t_{n,j})), \qquad t_{n,j} = t_n + k c_j,$$

for some constants b_j and c_j and number of points s.
For example, we could determine these constants by enforcing high-degree polynomial interpolation conditions.

The major problem with this approach is that it's unclear what approximation should be used for u at the intermediate time points $t_{n,j}$.

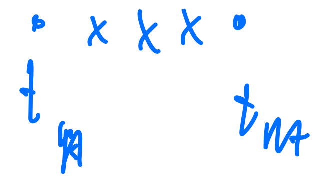
Higher-order schemes

The schemes we've seen previously are relatively low order: first order for Euler-type, and second order for Crank-Nicolson.

Recall that our schemes result from discretization (approximation) of an integral:

$$\boldsymbol{u}(t_{n+1}) = \boldsymbol{u}(t_n) + \int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t$$

$$\boldsymbol{u}_{n+1} \approx \boldsymbol{u}_n + \int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t.$$



In moving foward, we could consider the approximation

$$\int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t \approx \sum_{j=1}^{s} k b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{u}(t_{n,j})), \qquad t_{n,j} = t_n + k c_j,$$

for some constants b_j and c_j and number of points s.
For example, we could determine these constants by enforcing high-degree polynomial interpolation conditions.

The major problem with this approach is that it's unclear what approximation should be used for $\boldsymbol{u}$ at the intermediate time points $t_{n,j}$.

A simple method

To illustrate what we must accomplish, let us consider a simple case.

We'll again use a one-point method to approximate the integral, but collocate the point at the midpoint of the interval:

$$\int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t))\mathrm{d}t \approx k b_1 \boldsymbol{f}(t_{n,1}, \boldsymbol{u}(t_{n,1})), \qquad t_{n,1} = t_n + \frac{k}{2}.$$

I.e., we have chosen $c_1 = 1/2$, and b_j must be determined. ($s = 1$)

Note, however, that consistency of the approximation requires $b_1 = 1$.

Therefore, the (only) major question we have to answer is how we compute $\boldsymbol{u}(t_{n,1})$ from $\boldsymbol{u}_n$.

A straightforward idea is to approximate $\boldsymbol{u}(t_{n,1})$ with, say, Euler's method:

$$\boldsymbol{u}(t_n + k/2) \approx \boldsymbol{U}_1 := \boldsymbol{u}_n + \frac{k}{2}\boldsymbol{f}(t_n, \boldsymbol{u}_n)$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + k\boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1).$$

A simple method

To illustrate what we must accomplish, let us consider a simple case.

We'll again use a one-point method to approximate the integral, but collocate the point at the midpoint of the interval:

$$\int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t))\mathrm{d}t \approx k b_1 \boldsymbol{f}(t_{n,1}, \boldsymbol{u}(t_{n,1})), \qquad\qquad t_{n,1} = t_n + \frac{k}{2}.$$

I.e., we have chosen $c_1 = 1/2$, and b_j must be determined.

Note, however, that consistency of the approximation requires $b_1 = 1$.

Therefore, the (only) major question we have to answer is how we compute $\boldsymbol{u}(t_{n,1})$ from $\boldsymbol{u}_n$.

A straightforward idea is to approximate $\boldsymbol{u}(t_{n,1})$ with, say, Euler's method:

$$\boldsymbol{u}(t_n + k/2) \approx \boldsymbol{U}_1 := \boldsymbol{u}_n + \frac{k}{2}\boldsymbol{f}(t_n, \boldsymbol{u}_n)$$
$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + k\boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1).$$

A simple method

To illustrate what we must accomplish, let us consider a simple case.

We'll again use a one-point method to approximate the integral, but collocate the point at the midpoint of the interval:

$$\int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t))\mathrm{d}t \approx k b_1 \boldsymbol{f}(t_{n,1}, \boldsymbol{u}(t_{n,1})), \qquad\qquad t_{n,1} = t_n + \frac{k}{2}.$$

I.e., we have chosen $c_1 = 1/2$, and b_j must be determined.

Note, however, that consistency of the approximation requires $b_1 = 1$.

Therefore, the (only) major question we have to answer is how we compute $\boldsymbol{u}(t_{n,1})$ from $\boldsymbol{u}_n$.

A straightforward idea is to approximate $\boldsymbol{u}(t_{n,1})$ with, say, Euler's method:

$$\begin{aligned} \boldsymbol{u}(t_n + k/2) \approx \boldsymbol{U}_1 &:= \boldsymbol{u}_n + \frac{k}{2}\boldsymbol{f}(t_n, \boldsymbol{u}_n) \\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + k\boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1). \end{aligned}$$

Order of consistency, I

$$\boldsymbol{u}(t_n + k/2) \approx \boldsymbol{U}_1 := \boldsymbol{u}_n + \frac{k}{2}\boldsymbol{f}(t_n, \boldsymbol{u}_n)$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + k\boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1).$$

This idea seems fruitful, but there is a conceptual problem: Note that,

$$D^+ \boldsymbol{u}_n = \boldsymbol{f}(t_n + k/2, \boldsymbol{u}(t_n + k/2)) + \mathcal{O}(k^2)$$

$u(t_n)$

leading to an order-2 scheme.

$$D^+ u(t_n) = \frac{1}{k}\int_{t_n}^{t_{n+1}} f(t, u(t))\, dt$$

$$f(t, u(t)) \simeq f(t_n + k/2, u(t_n + k/2)) + (t - (t_n + k/2)) f' + \frac{1}{2}(t - (t_n + k/2))^2 f'' + \cdots$$

$$= C k^2 + O(k^3)$$

Order of consistency, II

$\frac{u(t_{n+1}) - u(t_n)}{k} = f(t_n, u(t_n)) + O(k) \rightarrow u(t_{n+1}) = \underbrace{u(t_n) + k f(t_n, u(t_n))}_{u_{n+1}} + O(k^2)$

$$\boldsymbol{u}(t_n + k/2) \approx \boldsymbol{U}_1 := \boldsymbol{u}_n + \frac{k}{2}\boldsymbol{f}(t_n, \boldsymbol{u}_n)$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + k\boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1).$$

$$D^+\boldsymbol{u}_n = \boldsymbol{f}(t_n + k/2, \boldsymbol{u}(t_n + k/2)) + \mathcal{O}(k^2)$$

The problem is that we are approximating with $\boldsymbol{U}_1$, which is only first-order accurate. Neverheless, one can show that this approximation is sufficient to retain an overall second-order LTE:

$$\begin{aligned}
\boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1) &\approx \boldsymbol{f}(t_n + k/2, \boldsymbol{u}(t_n + k/2)) \\
&\quad + (\boldsymbol{U}_1 - \boldsymbol{u}(t_n + 1/2))\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}(t_n + k/2, \boldsymbol{u}(t_n + k/2)) \\
\boldsymbol{f}(t_n + k/2, \boldsymbol{u}(t_n + k/2)) &= \boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1) \\
&\quad + (\boldsymbol{u}(t_n + 1/2) - \boldsymbol{U}_1)\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}(t_n + k/2, \boldsymbol{u}(t_n + k/2)) \\
&= \boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1) + \mathcal{O}(k^2).
\end{aligned}$$

The midpoint method

$$\boldsymbol{u}(t_n + k/2) \approx \boldsymbol{U}_1 := \boldsymbol{u}_n + \frac{k}{2}\boldsymbol{f}(t_n, \boldsymbol{u}_n)$$
$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + k\boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1).$$

Thus, the procedure above is actually second-order accurate, and is our first example of an explicit second-order method.

This scheme is called the (explicit) midpoint method.

The above shows how we might hope to generate higher-order schemes using higher-order quadrature.

Some happy coincidences occurred above, in particular making computations somewhat simple. In general, things are more technical.

The midpoint method

$$\begin{aligned}\boldsymbol{u}(t_n + k/2) \approx \boldsymbol{U}_1 &:= \boldsymbol{u}_n + \frac{k}{2}\boldsymbol{f}(t_n, \boldsymbol{u}_n) \\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + k\boldsymbol{f}(t_n + k/2, \boldsymbol{U}_1).\end{aligned}$$

Thus, the procedure above is actually second-order accurate, and is our first example of an explicit second-order method.

This scheme is called the (explicit) midpoint method.

The above shows how we might hope to generate higher-order schemes using higher-order quadrature.

Some happy coincidences occurred above, in particular making computations somewhat simple. In general, things are more technical.

Multi-stage methods

A generalization of our previous approach is the quadrature approximation:

$$\int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t))\mathrm{d}t \approx \sum_{j=1}^{s} k b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{u}(t_{n,j})), \qquad t_{n,j} = t_n + k c_j,$$

This leads to the following scheme:

$$\begin{aligned} \boldsymbol{u}(t_{n,j}) \approx \boldsymbol{U}_j &= \boldsymbol{u}_n + k \sum_{\ell=1}^{s} a_{j,\ell} \boldsymbol{f}(t_{n,\ell}, \boldsymbol{U}_\ell) \qquad t_{n,j} = t_n + k c_j, \\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + k \sum_{j=1}^{s} b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{U}_j), \end{aligned}$$

where the $a_{j,\ell}$, b_j, and c_j coefficients must be identified.

Multi-stage methods

A generalization of our previous approach is the quadrature approximation:

$$\int_{t_n}^{t_{n+1}} \boldsymbol{f}(t, \boldsymbol{u}(t)) \mathrm{d}t \approx \sum_{j=1}^{s} k b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{u}(t_{n,j})), \qquad t_{n,j} = t_n + k c_j,$$

This leads to the following scheme:

$$\boldsymbol{u}(t_{n,j}) \approx \boldsymbol{U}_j = \boldsymbol{u}_n + k \sum_{\ell=1}^{s} a_{j,\ell} \boldsymbol{f}(t_{n,\ell}, \boldsymbol{U}_\ell) \qquad t_{n,j} = t_n + k c_j,$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + k \sum_{j=1}^{s} b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{U}_j),$$

$$\int_0^1 f(x)\,dx \approx \frac{1}{\varsigma} \sum_{\varsigma=0}^{\varsigma-1} f(j/\varsigma)$$

where the $a_{j,\ell}$, b_j, and c_j coefficients must be identified.

The above is the general form for a multi-stage scheme with s intermediate stages. It is more commonly known as a Runge-Kutta method.

- If $a_{j,\ell} \neq 0$ for any $\ell \geq j$, then the procedure above is implicit. Otherwise it is explicit.
- If the overall scheme has order p LTE, it is typically not necessary that $\boldsymbol{U}_j$ correspond to an order p LTE.
- For $s \geqslant 3$, deriving and matching appropriate conditions can be quite cumbersome.

Consistency for order conditions

To see why things get hairy, first note that,

$f = f(t, u(t))$

$$
\begin{aligned}
\boldsymbol{u}' &= \boldsymbol{f}(t_n, \boldsymbol{u}(t_n)) = \boldsymbol{f} =: \boldsymbol{f}^{(0)} \\
\boldsymbol{u}'' &= \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{f} = \boldsymbol{f}_t + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\boldsymbol{u}' =: \boldsymbol{f}^{(1)} \\
\boldsymbol{u}''' &= \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{f}^{(1)} = \boldsymbol{f}_t^{(1)} + \frac{\partial \boldsymbol{f}^{(1)}}{\partial \boldsymbol{u}}\boldsymbol{u}' =: \boldsymbol{f}^{(2)} \\
&\vdots
\end{aligned}
$$

And by direct Taylor expansion, we have

$$
\begin{aligned}
D^+\boldsymbol{u}(t_n) &= \boldsymbol{u}' + \frac{k}{2}\boldsymbol{u}'' + \cdots . \\
&= \boldsymbol{f}^{(0)} + \frac{k}{2}\boldsymbol{f}^{(1)} + \cdots .
\end{aligned}
$$

Consistency for order conditions

To see why things get hairy, first note that,

$$\begin{aligned}
\boldsymbol{u}' &= \boldsymbol{f}(t_n, \boldsymbol{u}(t_n)) = \boldsymbol{f} =: \boldsymbol{f}^{(0)} \\
\boldsymbol{u}'' &= \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{f} = \boldsymbol{f}_t + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\boldsymbol{u}' =: \boldsymbol{f}^{(1)} \\
\boldsymbol{u}''' &= \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{f}^{(1)} = \boldsymbol{f}_t^{(1)} + \frac{\partial \boldsymbol{f}^{(1)}}{\partial \boldsymbol{u}}\boldsymbol{u}' =: \boldsymbol{f}^{(2)} \\
&\vdots
\end{aligned}$$

And by direct Taylor expansion, we have

$$\begin{aligned}
D^+\boldsymbol{u}(t_n) &= \boldsymbol{u}' + \frac{k}{2}\boldsymbol{u}'' + \cdots . \\
&= \boldsymbol{f}^{(0)} + \frac{k}{2}\boldsymbol{f}^{(1)} + \cdots .
\end{aligned}$$

Therefore, attaining an order p LTE amounts to enforcing,

$$\sum_{j=1}^{s} b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{U}_j) = \boldsymbol{f}^{(0)} + \frac{k}{2}\boldsymbol{f}^{(1)} + \cdots + \frac{k^{p-1}}{p!}\boldsymbol{f}^{(p-1)} + \mathcal{O}(k^p).$$

This then involves Taylor expansions for $\boldsymbol{f}(t_{n,j}, \boldsymbol{U}_j)$. ☹

Order conditions

We can count the number of required matching conditions (e.g., different types of derivatives) necessary to achieve order p:

p	1	2	3	4	5	6	7	8
# of conditions	1	2	4	8	17	37	115	200

And we can compare this to the number of free parameters for an s-stage method:

s	1	2	3	4	5	6	7	8
# of parameters	1	3	6	10	15	21	28	36

Order conditions

We can count the number of required matching conditions (e.g., different types of derivatives) necessary to achieve order p:

p	1	2	3	4	5	6	7	8
# of conditions	1	2	4	8	17	37	115	200

And we can compare this to the number of free parameters for an s-stage method:

s	1	2	3	4	5	6	7	8
# of parameters	1	3	6	10	15	21	28	36

This suggests that there is an order barrier, i.e., an order at which we must invest a superlinear number of stages relative to the order p. In fact, this is a theorem:

Theorem

There is no pth order Runge-Kutta method with $s = p$ stages if $p \geqslant 5$.

Order conditions

We can count the number of required matching conditions (e.g., different types of derivatives) necessary to achieve order p:

p	1	2	3	4	5	6	7	8
# of conditions	1	2	4	8	17	37	115	200

And we can compare this to the number of free parameters for an s-stage method:

s	1	2	3	4	5	6	7	8
# of parameters	1	3	6	10	15	21	28	36

This suggests that there is an order barrier, i.e., an order at which we must invest a superlinear number of stages relative to the order p. In fact, this is a theorem:

Theorem

There is no pth order Runge-Kutta method with $s = p$ stages if $p \geqslant 5$.

However, the situation is not so dire as the tables above suggest:

Stages s	1	2	3	4	5	6	7	8	9	10
Achievable RK order p	1	2	3	4	4	5	6	6	7	7

In particular, this suggests that $s = p = 4$ is an optimal tradeoff point.

Butcher tableaus

$$t_{n,j} = t_n + k c_j, \qquad c_j < 1$$

$$\boldsymbol{u}(t_{n,j}) \approx \boldsymbol{U}_j = \boldsymbol{u}_n + k \sum_{\ell=1}^{s} a_{j,\ell} \boldsymbol{f}(t_{n,\ell}, \boldsymbol{U}_\ell)$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + k \sum_{j=1}^{s} b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{U}_j),$$

In order to compactly communicate RK schemes, the Butcher tableau is the standard tool: the parameters $a_{j,\ell}$, b_j, and c_j are collected and arranged as follows:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

Some familiar schemes

Using tableau notation we can rehash some schemes we've previously seen:

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

Forward Euler   Backward Euler   Crank-Nicolson

FE: $s=1$,

$$U_1 = u_n + \sum_{j=1}^{s} k \, \cancel{b_j}^{\,a_{1,j}} f(t_n + k c_j, U_j)$$

$$= u_n + k \, \cancel{b_1}^{\,a_{1,1}} f(t_n + k c_1, U_1)$$

$$= u_n$$

$$u_{n+1} = u_n + k \sum_{j=1}^{s} b_j f(t_n + k c_j, U_j) = u_n + k f(t_n, u_n)$$

More examples

There is a one-parameter family of two-stage second-order methods:

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

(RK2)

for $c \in (0, 1]$:

- $c = 1$: explicit trapezoid method
- $c = 1/2$: explicit midpoint method

More examples

There is a one-parameter family of two-stage second-order methods:

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

for $c \in (0, 1]$:

- $c = 1$: explicit trapezoid method
- $c = 1/2$: explicit midpoint method

$a_{j,\ell} = 0 \quad \ell \geq j$

$\Downarrow$

explicit

And here is the classical fourth-order RK scheme:

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

Stability, convergence

Multi-stage (RK) methods are 0-stable, hence we obtain convergence commensurate with the LTE.
(Recall that this does not imply practical utility of error estimates)

A more interesting investigation involves the region of stability for these methods.

Note that this investigation makes sense since for A-stability we consider a scalar problem with,

$$f(t,u) = \lambda u,$$

and so intermediate stages have the form,

$$U_j = u_n + k \sum_{\ell=1}^{s} a_{j,\ell} \boldsymbol{f}(t_{n,\ell}, U_\ell) = u_n + z \sum_{\ell=1}^{s} a_{j,\ell} U_\ell,$$

where $z = \lambda k$. Therefore, the update is,

$$u_{n+1} = u_n + k \sum_{j=1}^{s} b_j f(t_n + kc_j, U_j) = u_n + z \sum_{j=1}^{s} b_j U_j,$$

which is a polynomial in z if the method is explicit.

Regions of stability

For some "standard" explicit RK methods of orders $1 - 4$, stability regions are as follows:

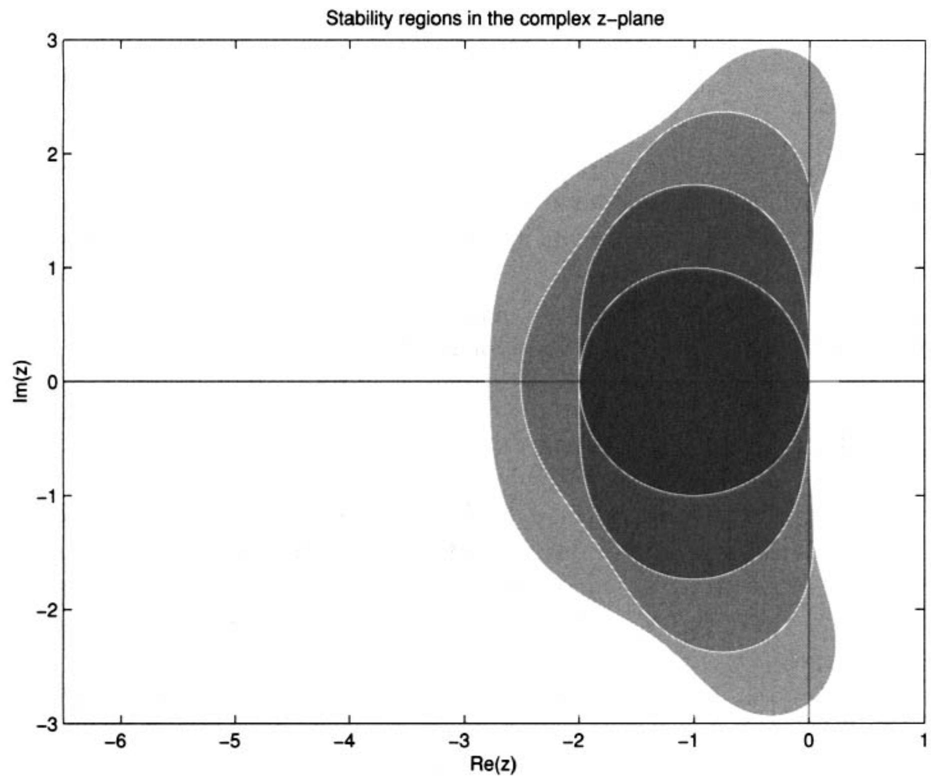


Figure: ROS for RK methods of order 1, 2, 3, 4. Darkest region for $p = 1$, lightest for $p = 4$. Ascher and Petzold 1998, Figure 4.4

Note that, by this measure of stability, higher order methods are more stable than lower order ones.

Practical RK methods: error estimation

In "production"-level simulations, a single time-stepping method is rarely used in isolation: methods are used in combination to empirically measure error.

The basic idea behind error estimation is to compute two approximations:

- $\boldsymbol{u}_n$: a less accurate approximation (typically $\Rightarrow$ lower order)
- $\widetilde{\boldsymbol{u}}_n$: a more accurate approximation (typically $\Rightarrow$ higher order)

Practical RK methods: error estimation

In "production"-level simulations, a single time-stepping method is rarely used in isolation: methods are used in combination to empirically measure error.

The basic idea behind error estimation is to compute two approximations:

- $\boldsymbol{u}_n$: a less accurate approximation (typically $\Rightarrow$ lower order)
- $\widetilde{\boldsymbol{u}}_n$: a more accurate approximation (typically $\Rightarrow$ higher order)

If $\widetilde{\boldsymbol{u}}_n$ is (much) more accurate than $\boldsymbol{u}_n$, then,

$$\|\boldsymbol{e}_n\| = \|\boldsymbol{u}_n - \boldsymbol{u}(t_n)\| \approx \|\boldsymbol{u}_n - \widetilde{\boldsymbol{u}}_n\|,$$

and the latter is computable.

A simplistic idea: use two multi-stage methods, say $\boldsymbol{u}_n$ is RK3 and $\widetilde{\boldsymbol{u}}_n$ is RK4.

The downside: this essentially requires (a little more than) twice the work.

Embedded multi-stage methods

Embedded methods allow us to construct more efficient error estimation procedures.

Consider a multi-stage method,

$$
\begin{aligned}
t_{n,j} &= t_n + k c_j, \\
\boldsymbol{U}_j &= \boldsymbol{u}_n + k \sum_{\ell=1}^{s} a_{j,\ell} \boldsymbol{f}(t_{n,\ell}, \boldsymbol{U}_\ell) \\
\boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + k \sum_{j=1}^{s} b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{U}_j),
\end{aligned}
$$

with local truncation error $\mathrm{LTE}_n \sim k^p$.

Embedded multi-stage methods

Embedded methods allow us to construct more efficient error estimation procedures.

Consider a multi-stage method,

$$
\begin{aligned}
t_{n,j} &= t_n + k c_j, \\
\boldsymbol{U}_j &= \boldsymbol{u}_n + k \sum_{\ell=1}^{s} a_{j,\ell} \boldsymbol{f}(t_{n,\ell}, \boldsymbol{U}_\ell) \\
\boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + k \sum_{j=1}^{s} b_j \boldsymbol{f}(t_{n,j}, \boldsymbol{U}_j),
\end{aligned}
$$

with local truncation error $\mathrm{LTE}_n \sim k^p$.

Suppose, somehow, we can identify other values of b_j for a different approximation:

$$
\widetilde{\boldsymbol{u}}_{n+1} = \boldsymbol{u}_n + k \sum_{j=1}^{s} \widetilde{b}_j \boldsymbol{U}_j, \qquad f(t_{n,j}, U_j)
$$

so that the LTE for $\widetilde{\boldsymbol{u}}_n$ obeys $\mathrm{LTE}_n \sim k^{p+1}$.

Embedded multi-stage methods

Embedded methods allow us to construct more efficient error estimation procedures.

Consider a multi-stage method,

$$t_{n,j} = t_n + kc_j,$$

$$\boldsymbol{U}_j = \boldsymbol{u}_n + k\sum_{\ell=1}^{s} a_{j,\ell}\,\boldsymbol{f}(t_{n,\ell}, \boldsymbol{U}_\ell)$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + k\sum_{j=1}^{s} b_j\,\boldsymbol{f}(t_{n,j}, \boldsymbol{U}_j),$$

with local truncation error $\mathrm{LTE}_n \sim k^p$.

Suppose, somehow, we can identify other values of b_j for a different approximation:

$$\widetilde{\boldsymbol{u}}_{n+1} = \boldsymbol{u}_n + k\sum_{j=1}^{s} \widetilde{b}_j \boldsymbol{U}_j,$$

$f(t_{n,j}, U_j)$

so that the LTE for $\widetilde{\boldsymbol{u}}_n$ obeys $\mathrm{LTE}_n \sim k^{p+1}$. Since $k \ll 1$, we can reasonbly expect that $\widetilde{\boldsymbol{u}}_n$ is much more accurate than $\boldsymbol{u}_n$.
RK methods, with two pairs of b_j coefficients corresponding to different orders, are called embedded methods.

An embedded method example

The following is a particularly well-known embedded method of order 4/5:

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40} \\
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0
\end{array}
$$

This is the Dormand-Prince 4(5) method.

Note that this has more stages (7) than a corresponding non-embedded order-5 RK method (6).
Nevertheless, this extra stage is typically worth the effort.

Embedded methods and adaptive time-stepping

With an embedded method, say of order p, we can attempt to certify error tolerances:

$$\|\boldsymbol{e}_n\| \approx \|\boldsymbol{u}_n - \widetilde{\boldsymbol{u}}_n\| \sim \mathcal{O}(k^p)$$

This implies that to achieve $\|\boldsymbol{e}_n\| \sim \epsilon_{\mathrm{tol}}$, then we should choose a new time step $\widehat{k}$ satisfying,

$$\left(\frac{\widehat{k}}{k}\right)^p \|\boldsymbol{u}_n - \widetilde{\boldsymbol{u}}_n\| \approx \epsilon_{\mathrm{tol}}.$$

This furnishes a *precise, computable* strategy with an embedded method for adaptively choosing $k = \Delta t$.

Embedded methods and adaptive time-stepping

With an embedded method, say of order p, we can attempt to certify error tolerances:

$u' = f(t, u)$

$$\|\boldsymbol{e}_n\| \approx \|\boldsymbol{u}_n - \widetilde{\boldsymbol{u}}_n\| \sim \mathcal{O}(k^p)$$

This implies that to achieve $\|\boldsymbol{e}_n\| \sim \epsilon_{\mathrm{tol}}$, then we should choose a new time step $\widehat{k}$ satisfying,

$$\left(\frac{\widehat{k}}{k}\right)^p \|\boldsymbol{u}_n - \widetilde{\boldsymbol{u}}_n\| \approx \epsilon_{\mathrm{tol}}.$$

This furnishes a *precise, computable* strategy with an embedded method for adaptively choosing $k = \Delta t$.

This strategy is actually what is used in many popular suites.
For example, the following are implementations of a Dormand-Prince 4(5) embedded method with adaptive time-stepping:

- Matlab's `ode45` command
- SciPy's `integrate.ode` command via the `integrate.ode.set_integrator('dopri5')` option
- Julia's `solve(..., DP5())` command from `DifferentialEquations.jl`

Multi-stage odds and ends

$u' = \lambda u, \quad \mathrm{Re}\,\lambda \ll 0$

There are *numerous* concepts in multi-stage methods we haven't discussed:

- dense output
- singly/diagonally implicit RK (S/DIRK), low-storage RK (LSRK), ...
- stiff problems and order reduction
- Gauss/-Radau/-Lobatto implicit RK methods
- error estimation/embedding for stiff problems

References I

Ascher, Uri M. and Linda R. Petzold (1998). *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM. ISBN: 978-1-61197-139-2.

Butcher, J. C. (2006). "General Linear Methods". In: *Acta Numerica* 15, pp. 157–256. ISSN: 1474-0508, 0962-4929. DOI: 10.1017/S0962492906220014.

Dormand, J. R. and P. J. Prince (1980). "A Family of Embedded Runge-Kutta Formulae". In: *Journal of Computational and Applied Mathematics* 6.1, pp. 19–26. ISSN: 0377-0427. DOI: 10.1016/0771-050X(80)90013-3.

LeVeque, Randall J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM. ISBN: 978-0-89871-783-9.