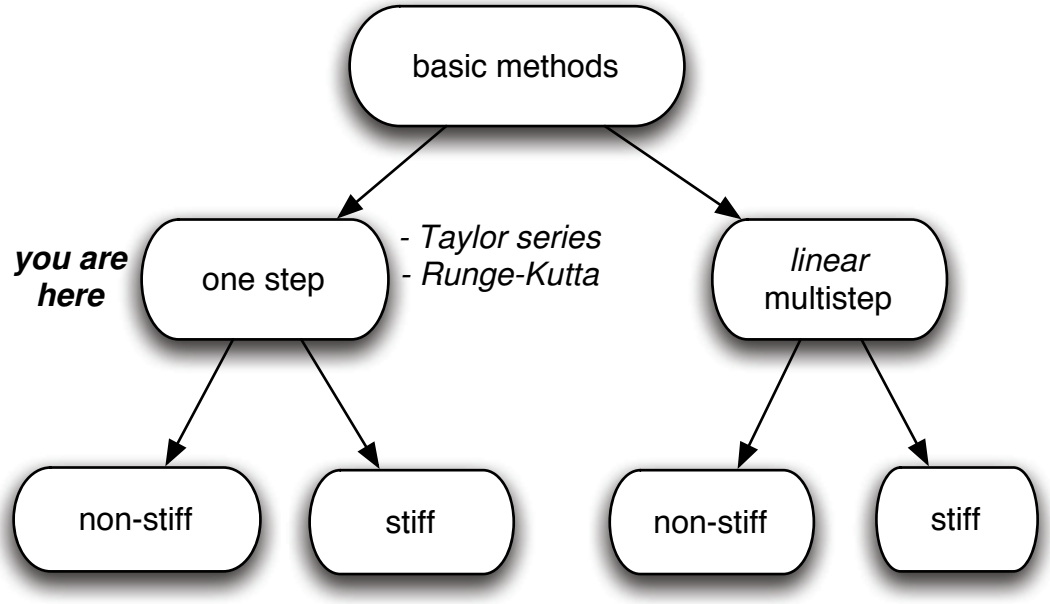
CHAPTER 4: *One-step Methods*

So far, we have looked at low-order methods
$\rightarrow$ OK for low accuracy or "non-smooth" problems.

- Not competitive if you want a high-accuracy solution to a relatively smooth problem (also common).
  $\rightarrow$ It pays to use a high-order method
  (each step more expensive; but many fewer steps !)



- One-step methods use only information on one subinterval $[t_{n-1}, t_n]$ to advance the solution.

  Note: The division non-stiff / stiff

  $$\updownarrow$$

  explicit / implicit

- Taylor series methods
  Taylor's theorem for a function of several variables

$$
\begin{aligned}
F(x,y) \;=\; & F(\hat{x},\hat{y}) \\
& + \left[\frac{\partial F}{\partial x}(\hat{x},\hat{y})(x-\hat{x}) + \frac{\partial F}{\partial y}(\hat{x},\hat{y})(y-\hat{y}) \right] \\
& + \frac{1}{2!} \left[\frac{\partial^2 F}{\partial x^2}(\hat{x},\hat{y})(x-\hat{x})^2 \right. \\
& \qquad + 2\frac{\partial^2 F}{\partial x \partial y}(\hat{x},\hat{y})(x-\hat{x})(y-\hat{y}) \\
& \qquad \left. + \frac{\partial^2 F}{\partial y^2}(\hat{x},\hat{y})(y-\hat{y})^2 \right] \\
& + \cdots \\
& + \frac{1}{n!} \sum_{j=0}^{n} \binom{n}{j} \\
& \qquad \frac{\partial^n F}{\partial x^j \partial y^{n-j}}(\hat{x},\hat{y})(x-\hat{x})^j (y-\hat{y})^{n-j} \\
& + \cdots
\end{aligned}
$$

- Just use Taylor series !

$$y_n = y_{n-1} + \Delta t_n \dot{y}_{n-1} + \frac{(\Delta t_n)^2}{2!}\ddot{y}_{n-1} + \cdots + \frac{(\Delta t_n)^p}{p!} y^{(p)}_{n-1},$$

where

$$\begin{aligned}
\dot{y}_{n-1} &= f(t_{n-1}, y_{n-1}), \\
\ddot{y}_{n-1} &= \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{dy}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f, \\
\dddot{y}_{n-1} &= \frac{\partial^2 f}{\partial t^2} + 2\frac{\partial^2 f}{\partial t \partial y} f + \frac{\partial f}{\partial y}\frac{\partial f}{\partial t} + \frac{\partial^2 f}{\partial y^2} f^2 + \left(\frac{\partial f}{\partial y}\right)^2 f.
\end{aligned}$$

Local truncation error $\frac{(\Delta t_n)^p}{(p+1)!} y^{(p+1)}(t_{n-1}) + \mathcal{O}((\Delta t_n)^{p+1})$.

Messy! (use symbolic or automatic differentiation)

$\rightarrow$ look for methods that only use f.

4.1 Basic Runge-Kutta Methods

Many RK methods are based on quadrature rules

$$\updownarrow$$

numerical integration

Quick review: evaluate $\int_a^b f(t)dt$.

Idea: Replace $f(t)$ with an interpolating polynomial $\phi(t)$ and then integrate $\phi(t)$ exactly.

Suppose we have s interpolation points $c_1, c_2, \cdots, c_s$. Lagrange interpolating polynomial is

$$\phi(t) = \sum_{j=1}^{s} f(c_j) L_j(t),$$

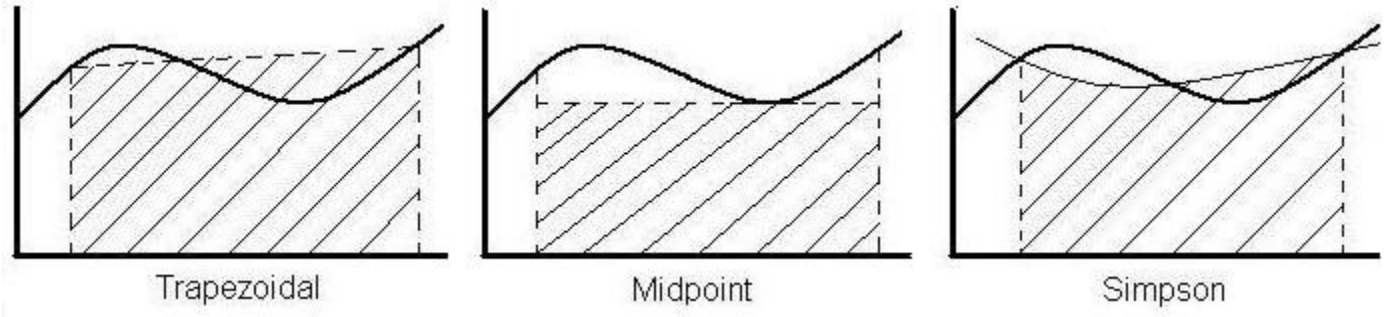
where $L_j(t) = \prod_{\substack{i=1 \\ (i \neq j)}}^{s} \frac{t - c_i}{c_j - c_i}$.

Then

$$\int_a^b f(t)dt \approx \sum_{j=1}^{s} w_j f(c_j),$$

where $w_j = \int_a^b L_j(t)dt.$

Example 1.



Because $y(t_n) = y(t_{n-1}) + \underbrace{\int_{t_{n-1}}^{t_n} \dot{y}(t)dt}_{\text{approximate by quadrature}}$

Left-hand sum: $\int_{t_{n-1}}^{t_n} \dot{y}(t)dt \approx \Delta t_n \dot{y}_{n-1}$
$\rightarrow$ forward Euler

Right-hand sum: $\int_{t_{n-1}}^{t_n} \dot{y}(t)dt \approx \Delta t_n \dot{y}_n$
$\rightarrow$ backward Euler

Trapezoidal rule:

$$\int_{t_{n-1}}^{t_n} \dot{y}(t)dt \approx \frac{\Delta t_n}{2}[\dot{y}_{n-1} + \dot{y}_n] \rightarrow \text{trapezoidal method.}$$

Midpoint rule:

$$\int_{t_{n-1}}^{t_n} \dot{y}(t)dt \approx \Delta t_n \dot{y}_{n-\frac{1}{2}} = \Delta t_n f(t_{n-\frac{1}{2}}, y_{n-\frac{1}{2}})$$

Let's generate an approximation to $y_{n-\frac{1}{2}}$:

$$y_{n-\frac{1}{2}} \approx \frac{1}{2}(y_{n-1} + y_n). \quad \text{(still implicit)}$$

or take half a step of forward Euler:

$$\begin{aligned} y_{n-\frac{1}{2}} &\approx \hat{y}_{n-\frac{1}{2}} = y_{n-1} + \frac{\Delta t_n}{2} f(t_{n-1}, y_{n-1}), \\ y_n &= y_{n-1} + \Delta t_n f(t_{n-\frac{1}{2}}, \hat{y}_{n-\frac{1}{2}}). \end{aligned}$$

- This is an explicit method !
  This is the idea of Runge-Kutta methods:
  high order is obtained by repeated f evaluations.

Note 1. – *This method is not linear in f.*
– *First stage uses forward Euler ($\mathcal{O}(\Delta t_n)$).*
But contribution is multiplied by (powers of) Δt_n,
So y_n can have high order !
– *Local truncation error is*

$$\begin{aligned} d_n &= \frac{y(t_n) - y(t_{n-1})}{\Delta t_n} - f\big(t_{n-\frac{1}{2}}, y(t_{n-1}) + \frac{\Delta t_n}{2} f(t_{n-1}, y(t_{n-1}))\big) \\ &\vdots \\ &= \mathcal{O}((\Delta t_n)^2). \quad \textit{(verify !)} \end{aligned}$$

$\rightarrow$ *Call this method ERK2.*

- Classical fourth-order (explicit) RK method (ERK4):

$$\begin{aligned} k_1 &= f(t_{n-1}, y_{n-1}), \\ k_2 &= f(t_{n-\frac{1}{2}}, y_{n-1} + \frac{1}{2}\Delta t_n\, k_1), \\ k_3 &= f(t_{n-\frac{1}{2}}, y_{n-1} + \frac{1}{2}\Delta t_n\, k_2), \\ k_4 &= f(t_n, y_{n-1} + \Delta t_n\, k_3), \\ y_n &= y_{n-1} + \frac{\Delta t_n}{6}\big(k_1 + 2(k_2 + k_3) + k_4\big). \end{aligned}$$

- **Example 2.** $\dot{y} = -5ty^2 + \frac{5}{t} - \frac{1}{t^2}$,
$$y(1) = 1, \quad t_f = 25.$$
Verify the orders of forward Euler, ERK2, ERK4
Exact solution: $y(t) = \frac{1}{t}$.

Note 2. • *So far, stages* (s) = *order* (p).
This is not necessary !
It is not even possible for $p > 4$.

- *These methods are not unique;*
e.g., other two-stage, second-order ERKs exist.

- *(See text, p. 79.)*

$$\left.\begin{aligned} e_n(\Delta t) &\approx c(\Delta t)^p \\ e_{2n}(\tfrac{\Delta t}{2}) &\approx c\big(\tfrac{\Delta t}{2}\big)^p \end{aligned}\right\} \quad p = \log_2\left(\frac{e_n(\Delta t)}{e_{2n}(\frac{\Delta t}{2})}\right).$$

We can verify order of accuracy p *by halving* Δt *and comparing errors.*

step h	Euler error	rate	RK2 error	rate	RK4 error	rate
0.2	.40e-2		.71e-3		.66e-6	
0.1	.65e-6	12.59	.33e-6	11.08	.22e-7	4.93
0.05	.32e-6	1.00	.54e-7	2.60	.11e-8	4.34
0.02	.13e-6	1.00	.72e-8	2.20	.24e-10	4.16
0.01	.65e-7	1.00	.17e-8	2.08	.14e-11	4.07
0.005	.32e-7	1.00	.42e-9	2.04	.89e-13	3.98
0.002	.13e-7	1.00	.66e-10	2.02	.13e-13	2.13

Table 4.1: Errors and calculated convergence rates for the forward Euler, the explicit midpoint (RK2) and the classical Runge-Kutta (RK4) methods

- *Choice of the method depends on accuracy requirement.*
  Given Δt, if forward Euler has error e_1,
  then with $2\Delta t$, ERK2 is more efficient if its error e_2 satisfies $e_2 < e_1$.
  (Same goes for ERK4.)

- *For $\Delta t = 0.2$, we have instability polluting the error.*
  Notice that p is never exactly an integer.
  This is especially true when e approaches roundoff.
  For predictable results, we want to live where truncation error $\gg$ roundoff error.

4.2 General RK Methods

Given the ODE system $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$, the general s-stage RK method can be written as

$$\mathbf{k}_i = \mathbf{f}\left(t_{n-1} + \Delta t_n c_i, \mathbf{y}_{n-1} + \Delta t_n \sum_{j=1}^{s} a_{ij}\mathbf{k}_j\right), \quad i = 1, 2, \cdots, s,$$

$$\mathbf{y}_n = \mathbf{y}_{n-1} + \Delta t_n \sum_{i=1}^{s} b_i \mathbf{k}_i.$$

Compact notation (Butcher array)

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

Note 3. *We will always choose*

$$c_i = \sum_{j=1}^{s} a_{ij}, \qquad i = 1, 2, \cdots, s.$$

- The RK method is explicit if $\mathbf{A}$ is strictly lower triangular.
  Then each $\mathbf{k}_i$ is expressed in terms of $\mathbf{k}_1, \mathbf{k}_2, \cdots, \mathbf{k}_{i-1}$.

- Historically, the first RK methods were explicit.

- The RK method is implicit if any a_{ij} on or above the diagonal is non-zero.

  Implicit RK methods are useful for stiff IVPs and BVPs.

- **Example 3.** *Some examples of ERKs:*
  - *Forward Euler:*

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

  - *One-parameter family of second-order methods:*

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

$\alpha = 1 \leftrightarrow$ *explicit trapezoidal method (verify!)*
$\alpha = \frac{1}{2} \leftrightarrow$ *explicit midpoint method (verify!)*

There are 3 one-parameter families of three-stage, third-order ERK methods.
Here is one of them:

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

Classical ERK4:

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

(verify !)

- There are s-stage explicit methods of order s, but only for $s = 1, 2, 3, 4$.
  $\rightarrow$ ERKs cannot have $p > s$.
  $\rightarrow$ When $p > 4$, it is necessary that $s > p$.

Note 4. $\mathbf{k}_i = \mathbf{f}\left(t_{n-1} + \Delta t_n c_i, \underbrace{\mathbf{y}_{n-1} + \Delta t_n \sum_{j=1}^{s} a_{ij}\mathbf{k}_j}\right)$

this can be interpreted as an approximation to $\mathbf{y}(t_{n-1} + \Delta t_n c_i)$ *(call it* $\mathbf{Y}_i$*)*

Then the RK method can be written

$$\mathbf{Y}_i = \mathbf{y}_{n-1} + \Delta t_n \sum_{j=1}^{s} a_{ij}\mathbf{f}(t_{n-1} + \Delta t_n c_j, \mathbf{Y}_j), \quad i = 1, 2, \cdots, s,$$

$$\mathbf{y}_n = \mathbf{y}_{n-1} + \Delta t_n \sum_{i=1}^{s} b_i \mathbf{f}(t_{n-1} + \Delta t_n c_i, \mathbf{Y}_i).$$

4.3 Convergence, 0-stability, and Order for RK Methods

- All the schemes we have looked at are consistent (they are at least first order).

- For one-step schemes (under mild restrictions) it can be shown that

$$\text{consistency} \Rightarrow \text{0-stability.}$$

- And we recall

$$\text{consistency} + \text{0-stability} \Rightarrow \text{convergence.}$$

  So, RK methods are convergent.

- What about the order ?
  Use Taylor series (in principle)
  $\rightarrow$ Expand the RK method in a Taylor series.

Use the RK coefficients to match terms with the Taylor series of the exact solution.

This is not trivial for high s or p !

- Let's try it for a 2-stage, 2nd-order ERK:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ a_{21} & a_{21} & 0 \\ \hline & b_1 & b_2 \end{array}$$

We want a relationship between a_{21}, b_1, b_2 for the scheme to be second-order accurate.

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(t_{n-1}, \mathbf{y}_{n-1}), \\ \mathbf{k}_2 &= \mathbf{f}(t_{n-1} + \Delta t_n a_{21}, \mathbf{y}_{n-1} + \Delta t_n a_{21}\mathbf{k}_1), \\ \mathbf{y}_n &= \mathbf{y}_{n-1} + \Delta t_n(b_1\mathbf{k}_1 + b_2\mathbf{k}_2), \\ \mathbf{y}(t_n) &= \mathbf{y}(t_{n-1}) + \Delta t_n \dot{\mathbf{y}}(t_{n-1}) + \frac{(\Delta t_n)^2}{2}\ddot{\mathbf{y}}(t_{n-1}) + \mathcal{O}((\Delta t_n)^3). \end{aligned}$$

↑ ↑

we have to match these terms

Note 5.

$$\begin{aligned}\mathbf{y}_{n-1} + \Delta t_n a_{21}\mathbf{k}_1 &= \mathbf{y}_{n-1} + \Delta t_n a_{21}\mathbf{f}(t_{n-1}, \mathbf{y}_{n-1}) \\ &= \mathbf{y}_{n-1} + \Delta t_n a_{21}\dot{\mathbf{y}}(t_{n-1}).\end{aligned}$$

$$\begin{aligned}\mathbf{k}_2 &= \mathbf{f}(t_{n-1} + \Delta t_n a_{21}, \mathbf{y}_{n-1} + \Delta t_n a_{21}\mathbf{k}_1) \\ &= \mathbf{f}(t_{n-1}, \mathbf{y}_{n-1}) + \Delta t_n a_{21}\frac{\partial \mathbf{f}}{\partial t}(t_{n-1}, \mathbf{y}_{n-1}) \\ &\quad + \Delta t_n a_{21}\underbrace{\mathbf{k}_1}_{\mathbf{f}(t_{n-1}, \mathbf{y}_{n-1}) \text{ or } \dot{\mathbf{y}}(t_{n-1})}\frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_{n-1}, \mathbf{y}_{n-1}) + \mathcal{O}((\Delta t_n)^2).\end{aligned}$$

Now

$$\begin{aligned}
\ddot{\mathbf{y}}(t_{n-1}) &= \dot{\mathbf{f}}(t_{n-1}, \mathbf{y}_{n-1}) = \frac{d\mathbf{f}}{dt}(t_{n-1}, \mathbf{y}_{n-1}) \\
&= \frac{\partial \mathbf{f}}{\partial t}(t_{n-1}, \mathbf{y}_{n-1}) \\
&\quad + \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_{n-1}, \mathbf{y}_{n-1})\dot{\mathbf{y}}(t_{n-1}, \mathbf{y}_{n-1}).
\end{aligned}$$

$$\text{So } \mathbf{k}_2 = \dot{\mathbf{y}}(t_{n-1}) + \Delta t_n a_{21}\ddot{\mathbf{y}}(t_{n-1}) + \mathcal{O}((\Delta t_n)^2).$$

$$\begin{aligned}
\therefore \quad \mathbf{y}(t_n) &= \mathbf{y}(t_{n-1}) + \Delta t_n \Big[b_1 \dot{\mathbf{y}}(t_{n-1}) \\
&\quad + b_2 \big(\dot{\mathbf{y}}(t_{n-1}) + \Delta t_n a_{21} \ddot{\mathbf{y}}(t_{n-1}) + \mathcal{O}((\Delta t_n)^2) \big) \Big] \\
&= \mathbf{y}(t_{n-1}) + \Delta t_n (b_1 + b_2) \dot{\mathbf{y}}(t_{n-1}) \\
&\quad + (\Delta t_n)^2 a_{21} b_2 \ddot{\mathbf{y}}(t_{n-1}) + \mathcal{O}((\Delta t_n)^3).
\end{aligned}$$

$\therefore$

$$\begin{aligned} b_1 + b_2 &= 1 \\ a_{21} b_2 &= \frac{1}{2} \end{aligned}$$

$\rightarrow$ 2 equations, 3 unknowns
$\rightarrow$ A one-parameter family of solutions.

Exercise: Let $\alpha = a_{21}$ be the parameter.
Verify that the Butcher tableau matches that presented earlier.

- For the general case of order p, consider

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f}(\mathbf{y}), \\ \mathbf{y}(t_{n-1}) &= \mathbf{y}_{n-1}. \end{aligned}$$

For the exact solution,

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f} := \mathbf{f}^{(0)}, \\ \ddot{\mathbf{y}} &= \dot{\mathbf{f}} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}\mathbf{f} := \mathbf{f}^{(1)}, \\ \dddot{\mathbf{y}} &= \ddot{\mathbf{f}} = \frac{\partial \dot{\mathbf{f}}}{\partial \mathbf{y}}\mathbf{f} := \mathbf{f}^{(2)}, \\ &\vdots \\ \mathbf{y}^{(k)} &= \mathbf{f}^{(k-1)} = \frac{\partial \mathbf{f}^{(k-2)}}{\partial \mathbf{y}}\mathbf{f} := \mathbf{f}^{(k-1)}. \end{aligned}$$

Taylor expansion at $\mathbf{y}_{n-1}$:

$$\begin{aligned}
\mathbf{y}(t_n) &= \mathbf{y}_{n-1} + \Delta t_n \dot{\mathbf{y}}(t_{n-1}) + \frac{(\Delta t_n)^2}{2!}\ddot{\mathbf{y}}(t_{n-1}) \\
&\quad + \cdots + \frac{(\Delta t_n)^{p+1}}{(p+1)!}\mathbf{y}^{(p+1)}(t_{n-1}) + \cdots \\
&= \mathbf{y}_{n-1} + \Delta t_n \mathbf{f}_{n-1}^{(0)} + \frac{(\Delta t_n)^2}{2!}\mathbf{f}_{n-1}^{(1)} \\
&\quad + \cdots + \frac{(\Delta t_n)^{p+1}}{(p+1)!}\mathbf{f}_{n-1}^{(p)} + \cdots
\end{aligned}$$

For an s-stage RK method to be order p, we need

$$\sum_{i=1}^{s} b_i \mathbf{k}_i = \mathbf{f}_{n-1}^{(0)} + \frac{\Delta t_n}{2}\mathbf{f}_{n-1}^{(1)} + \cdots + \frac{(\Delta t_n)^{p-1}}{p!}\mathbf{f}_{n-1}^{(p-1)} + \mathcal{O}((\Delta t_n)^p).$$

$\rightarrow$ We could determine relationships between the coefficients a_{ij}, b_j, c_j by Taylor series (brute force).

There is an elegant theory based on rooted trees developed by J.C. Butcher (1960s) to give the order conditions of a general s-stage RK method. It is difficult to understand but relatively easy to use. From it we get necessary and sufficient conditions for an s-stage RK method to be of order p.

4.4 Absolute Stability Regions for ERK Methods

Recall, the absolute stability region is the region in the complex z-plane ($z = \lambda \Delta t_n$) where $|y_n| \leq |y_{n-1}|$ when the method is applied to the test equation

$$\dot{y} = \lambda y.$$

Recall

$$\begin{aligned} Y_i &= y_{n-1} + \Delta t_n \sum_{j=1}^{s} a_{ij} f(t_{n-1} + c_j \Delta t_n, Y_j) \\ &= y_{n-1} + \underbrace{\Delta t_n \lambda}_{z} \sum_{j=1}^{s} a_{ij} Y_j. \end{aligned}$$

In matrix form,

$$\mathbf{Y} = \begin{pmatrix} Y_1 \\ \vdots \\ Y_s \end{pmatrix}$$

$$\mathbf{Y} = y_{n-1}\mathbf{1} + z\mathbf{A}\mathbf{Y}, \qquad \mathbf{1} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}, \qquad \text{(verify)}$$

or

$$\mathbf{Y} = (\mathbf{I} - z\mathbf{A})^{-1}\mathbf{1}y_{n-1}.$$

Using this in

$$\begin{aligned} y_n &= y_{n-1} + \Delta t_n \sum_{i=1}^{s} b_i f(t_{n-1} + \Delta t_n c_i, Y_i) \\ &= y_{n-1} + \lambda \Delta t_n \sum_{i=1}^{s} b_i Y_i, \\ \text{or} \quad y_n &= y_{n-1} + z\mathbf{b}^T\mathbf{Y} \\ &= y_{n-1}\left[1 - z\mathbf{b}^T(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{1}\right]. \end{aligned}$$

But for $\Delta t_n \leftrightarrow z$ sufficiently small
$(\mathbf{I} - z\mathbf{A})^{-1} = \mathbf{I} + z\mathbf{A} + z^2\mathbf{A}^2 + \cdots + z^k\mathbf{A}^k + \cdots$

$$\therefore \qquad y_n = \underbrace{R(z)}_{\text{amplification factor}} \; y_{n-1},$$

where if the RK method is order p,
$R(z) = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots + \frac{z^p}{p!} + \underbrace{\sum_{j>p} z^j \mathbf{b}^T\mathbf{A}^{j-1}\mathbf{1}}_{\sum_{j>p} z^j \mathbf{b}^T\mathbf{A}^{j-2}\mathbf{c}}.$

Note 6. $\mathbf{b}^T\mathbf{A}^k\mathbf{c} = \frac{1}{(k+2)!}$ *(verify !)*
$k = 0, 1, \ldots, p-1$ *if method is order* p.

For an $\underbrace{\text{ERK with } s \text{ stages}}_{\mathbf{A}^{j-1}=\mathbf{0} \text{ for } j>s}$ and order p,

$$R(z) = 1 + z + \frac{z^2}{2!} + \cdots + \frac{z^p}{p!} + \sum_{j=p+1}^{s} z^j \mathbf{b}^T \mathbf{A}^{j-1}\mathbf{1}.$$

Note 7. *(For ERKs)*

- *If* $s = p$, $R(z)$ *does not depend on* $\mathbf{b}^T, \mathbf{A}, \mathbf{c}$.
  $\rightarrow$ *All ERKs with* $s = p$ *have the same absolute stability region !*

- $R(z)$ *is a polynomial of degree* s.
  $\therefore \quad |R(z)| \rightarrow \infty$ *as* $|z| \rightarrow \infty$;
  i.e., stability region is bounded (a problem can always look stiff) *not good stiff solvers.*

- *It is fairly easy to plot a stability region using the contour function in* MATLAB.

- *There is no "perfect" ERK method.*

4.5 Error Estimation and Control

You need stepsize control for both reliability and efficiency.
Any constant stepsize routine will perform poorly if variation of solution is not constant.

- We choose "optimal" stepsize for next step.
- User specifies error tolerance $ETOL$.
- We try to (roughly) equate local errors for each step:

$$\|\mathbf{l}_n\| \approx ETOL.$$

Recall

$$\|\mathbf{l}_n\| = \Delta t_n \|\mathbf{d}_n\|(1 + \mathcal{O}(\Delta t_n))$$
$$\uparrow$$
local truncation error

Note 8. – $\mathbf{l}_n$ *is a vector with same length as* $\mathbf{y}_n$.

* *If $(y_i)_n$ is very large (or very small), you may want a relative tolerance (or an absolute tolerance).*
* *If components of $\mathbf{y}_n$ differ in size, you may want a specific tolerance for each component,*

$$ETOL_i = ATOL_i + |(y_i)_n| RTOL.$$

– *We don't want to use a formula for local error*
e.g., explicit trapezoidal rule (for scalar ODE)

$$l_n \approx \Delta t_n d_n = \frac{(\Delta t_n)^3}{8}\left[\frac{\partial^2 f}{\partial y^2} f^2 + 2\frac{\partial^2 f}{\partial t \partial y} f + \frac{\partial^2 f}{\partial t^2}\right] + \mathcal{O}((\Delta t_n)^4)$$

This defeats the purpose of trying to avoid partial derivatives!

Idea: Look at difference between two solutions $\mathbf{y}_n, \hat{\mathbf{y}}_n$ of different orders to estimate error (usually orders differ by 1).
Design methods so that $\hat{\mathbf{y}}_n - \mathbf{y}_n$ estimates the local error of the less accurate solution $\mathbf{y}_n$.

So if $\|\hat{\mathbf{y}}_n - \mathbf{y}_n\| \leq ETOL$,
step is accepted;
<Compute "optimal" Δt_n for next step>
Else step is rejected;
<Compute new "optimal" Δt_n for this step>
i.e., try again !

- How do you compute the new Δt_n ?
  Suppose we got rejected with stepsize Δt_n with method of order p.
  Then $\|\mathbf{l}_n(\Delta t_n)\| = \|\hat{\mathbf{y}}_n - \mathbf{y}_n\| \approx c(\Delta t_n)^{p+1}$,
  or $\frac{\|\hat{\mathbf{y}}_n - \mathbf{y}_n\|}{c(\Delta t_n)^{p+1}} \approx 1.$

  We want a new stepsize $\widetilde{\Delta t_n}$ such that

$$\|\mathbf{l}_n(\widetilde{\Delta t_n})\| \approx c(\widetilde{\Delta t_n})^{p+1} \approx frac\ ETOL.$$

  $frac \approx 0.9$ is a safety factor to avoid rejections.

  $\therefore \quad \left(\frac{\widetilde{\Delta t_n}}{\Delta t_n}\right)^{p+1} \|\hat{\mathbf{y}}_n - \mathbf{y}_n\| \approx frac\ ETOL.$

  $\rightarrow$ We can compute $\widetilde{\Delta t_n}$ from this.

Note 9. – *In practice, you don't want* Δt_n *to change rapidly from step to step (otherwise local error estimate is unreliable). So restrict*

$$\frac{1}{\alpha}\Delta t_n \leq \widetilde{\Delta t_n} \leq \alpha \Delta t_n, \qquad \alpha \approx 5.$$

– *Added restriction after rejection to avoid multiple rejections*

$$\widetilde{\Delta t_n} \leq \Delta t_n.$$

– *There must be a minimum* Δt_n $(\sim 10^{-14})$ *below which we cannot take a step.*

– *This does not tell you how to take the first step !*

- Embedded ERK methods:
  Idea: Two ERK methods, orders p, $p+1$.
  Recycle as much information as possible:
  $\rightarrow$ Share all the stages.

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} \qquad \begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \hat{\mathbf{b}}^T \end{array} \longrightarrow \begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \hat{\mathbf{b}}^T \\ \hline & \mathbf{b}^T \end{array}$$

order $p+1$ order p $p+1$ (p) pair

Explicit trapezoidal rule and forward Euler: a $2(1)$ embedded pair.

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

Normally: use one extra function evaluation for order $p+1$ over order p.

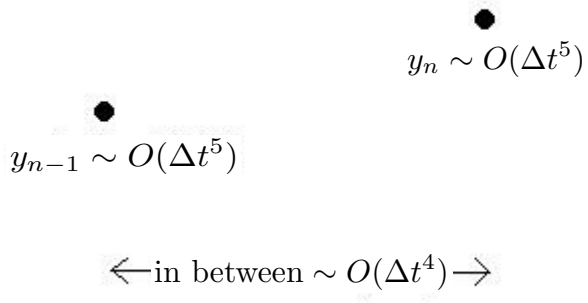
Famous pairs: Fehlberg $4(5)$
Dormand–Prince $5(4)$

Note 10. – *Modern codes advance with the higher-order estimate* $\hat{\mathbf{y}}_n$
This is called local extrapolation.
You add the error $(\hat{\mathbf{y}}_n - \mathbf{y}_n)$ *to the estimate* $\mathbf{y}_n$

$$\mathbf{y}_n + (\hat{\mathbf{y}}_n - \mathbf{y}_n) = \hat{\mathbf{y}}_n,$$

but now (strictly speaking) we don't really have an error estimate anymore !

– *DoPr* $5(4)$ *was designed with this in mind: coefficient of truncation error for* $\hat{\mathbf{y}}_n$ *is minimized.*
  * *also the RK scheme used by* `ode45`
  * *has fourth-order interpolant*



  * *"first same as last" (FSAL)*

- Step doubling:
  A way to estimate error with one method.
  $\rightarrow$ Take one step of $2\Delta t_n$: $\mathbf{y}_{2\Delta t_n}$.
  Take two steps of Δt_n: $\mathbf{y}_{\Delta t_n}$.

  Then $\|\mathbf{y}_{\Delta t_n} - \mathbf{y}(t_n)\| \approx 2\|\mathbf{l}_n(\Delta t_n)\| \approx 2c(\Delta t_n)^{p+1}$.
  $\|\mathbf{y}_{2\Delta t_n} - \mathbf{y}(t_n)\| \approx \|\mathbf{l}_n(2\Delta t_n)\| \approx c(2\Delta t_n)^{p+1}$.

  $\therefore \quad \|\mathbf{y}_{\Delta t_n} - \mathbf{y}_{2\Delta t_n}\| \approx 2(2^p - 1)c(\Delta t_n)^{p+1}$.

$$\rightarrow \quad 2c(\Delta t_n)^{p+1} \approx \frac{\|\mathbf{y}_{\Delta t_n} - \mathbf{y}_{2\Delta t_n}\|}{2^p - 1} = 2\|\mathbf{l}_n(\Delta t_n)\|.$$

Then use $\left(\frac{\tilde{\Delta t_n}}{\Delta t_n}\right)^{p+1} \frac{\|\mathbf{y}_{\Delta t_n} - \mathbf{y}_{2\Delta t_n}\|}{2^p - 1} \approx frac\ ETOL.$

Note 11. *Step doubling gives an accurate error estimate, but it is more expensive than an embedded pair . . . if you have one !*

A word about global error.
After solving a problem, re-solve with same method and all step sizes halved, then estimate error as above.

$\rightarrow$ If all estimates are below $ETOL$, you are done !

$\rightarrow$ If not, you will at least have to refine step sizes in regions where error is too large.

Note 12. *This means keeping track of the entire solution.*

Usually we want to avoid this for IVPs, so we concentrate on controlling the local error.

4.6 Sensitivity Analysis

Model parameters (e.g., g) or initial conditions are rarely known exactly.

In that sense, the exact solution of a given IVP may be viewed as one member of a family of neighbouring trajectories.

$\rightarrow$ It does not make sense to waste resources by solving a given IVP accurately if other errors are much larger.

To assess solution quality, we often need a *sensitivity analysis*:

How much do solutions change when parameters are perturbed ?

Consider
$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f}(t, \mathbf{y}; \mathbf{p}), \\ \mathbf{y}(t_0) &= \mathbf{y}_0, \qquad t_0 < t < t_f. \\ \mathbf{p} \quad &\leftrightarrow \quad \text{(constant) parameters.} \end{aligned}$$

Let solution to this problem be $\mathbf{y}(t)$.

Perturb: $\bar{\mathbf{p}} = \mathbf{p} + \boldsymbol{\phi}$

Call solution $\bar{\mathbf{y}}(t)$

Expand:
$$\begin{aligned}\bar{\mathbf{y}}(t) &= \bar{\mathbf{y}}(t; \mathbf{p} + \boldsymbol{\phi}) \\ &\approx \mathbf{y}(t; \mathbf{p}) + \tfrac{\partial \mathbf{y}}{\partial \mathbf{p}}(t; \mathbf{p})\boldsymbol{\phi}.\end{aligned}$$

Then
$$\|\bar{\mathbf{y}}(t) - \mathbf{y}(t)\| \leq \|\mathbf{P}(t)\boldsymbol{\phi}\|,$$
where $\mathbf{P}(t) = \frac{\partial \mathbf{y}(t;\mathbf{p})}{\partial \mathbf{p}}$.

To calculate $\mathbf{P}$, differentiate
$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}; \mathbf{p}), \quad \mathbf{y}(t_0) = \mathbf{y}_0,$$
with respect to $\mathbf{p}$:

$$\frac{\partial}{\partial \mathbf{p}}(\dot{\mathbf{y}}) = \frac{\partial}{\partial \mathbf{p}}\left(\frac{d\mathbf{y}}{dt}\right) = \frac{d}{dt}\left(\frac{\partial \mathbf{y}}{\partial \mathbf{p}}\right) = \dot{\mathbf{P}}$$

$$\frac{\partial}{\partial \mathbf{p}}\mathbf{f}(t, \mathbf{y}; \mathbf{p}) = \frac{\partial \mathbf{f}}{\partial \mathbf{p}}\underbrace{\frac{\partial \mathbf{p}}{\partial t}}_{0} + \frac{\partial \mathbf{f}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{p}} + \frac{\partial \mathbf{f}}{\partial \mathbf{p}} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}\mathbf{P} + \frac{\partial \mathbf{f}}{\partial \mathbf{p}}$$

$$\frac{\partial}{\partial \mathbf{p}}\mathbf{y}(t_0) = \mathbf{0}$$

$\Rightarrow$ Solve $$\dot{\mathbf{P}} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}\mathbf{P} + \frac{\partial \mathbf{f}}{\partial \mathbf{p}}$$
$$\mathbf{P}(t_0) = \mathbf{0}$$

Note: This is a *linear* problem in $\mathbf{P}$.

In practice, we solve the original IVP with a lax error tolerance and compute $\mathbf{P}$ simultaneously for the same time steps.

This can be done efficiently.

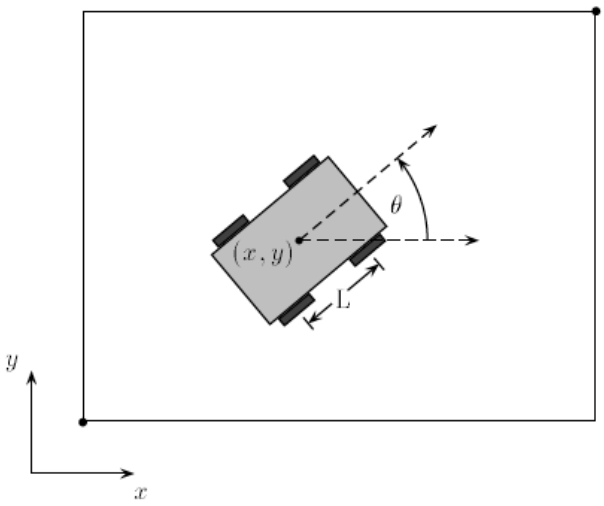
Then bounds on $\|\boldsymbol{\phi}\|$ can be used to compute bounds on $\|\bar{\mathbf{y}}(t) - \mathbf{y}(t)\|$.

Public domain software is available (DASPK3.0).

A similar treatment can be applied to perturbations in the ICs.

Sensitivity of parameters (the effect of parameter uncertainty on solutions) is one of the sources of uncertainty considered in uncertainty quantification.

Example 4. *(Toy car)*



$$\begin{aligned}
\dot{x} &= v\cos\theta \\
\dot{y} &= v\sin\theta \\
\dot{\theta} &= \frac{v\tan\psi}{L} \\
\dot{v} &= a - \gamma v
\end{aligned}$$

ψ *steering angle*

a *acceleration*

take as constant parameters: $a{=}100, \psi{=}1$

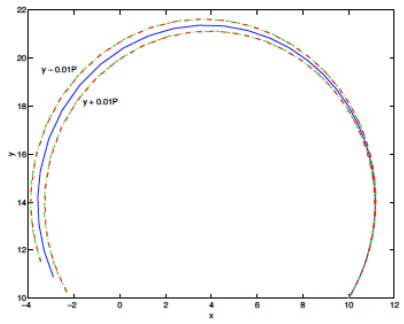
v *velocity*

γ *friction*

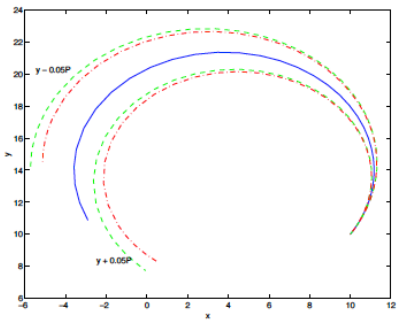
L *car length*

$\rightarrow$ *Investigate sensitivity to perturbation in* ψ.

$$
\begin{aligned}
\dot{P}_1 &= -vP_3 \sin\theta + P_4 \cos\theta \\
\dot{P}_2 &= vP_3 \cos\theta + P_4 \sin\theta \\
\dot{P}_3 &= \frac{1}{L}\left[P_4 \tan\psi + \frac{v}{\cos^2\psi}\right] \\
\dot{P}_4 &= -\gamma P_4
\end{aligned}
$$



(a) $\phi = 0.01$



(b) $\phi = 0.05$

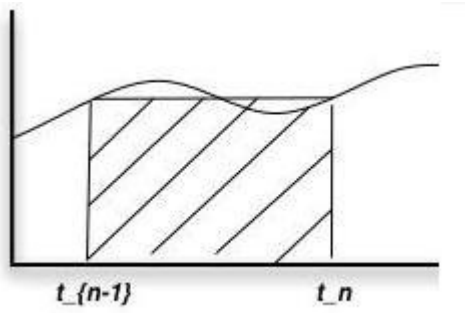
4.7 IRK and Collocation

- IRKs have more free parameters than ERKs.
  $\rightarrow$ For a given s, we can get higher p.
  e.g., implicit midpoint has one stage and order 2.

  $\rightarrow$ $R(z)$ no longer a polynomial (in fact it is now rational function), so IRKs can solve stiff problems.

- IRKs can be designed on quadrature.
  - Pick a quadrature formula. (points, weights)
  - Construct stages to approximate y at the quadrature points.
  - Use these in the quadrature formula.

  Recall implicit midpoint:



$$y_n = y_{n-1} + \underbrace{\int_{t_{n-1}}^{t_n} \dot{y}(t)dt.}_{\approx \Delta t_n f\left(t_{n-\frac{1}{2}}, y_{n-\frac{1}{2}}\right)}$$

Example 5. • *Gauss methods: order* $2s$ *(maximum)*

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

$s = 1, p = 2$

implicit midpoint

$$\begin{array}{c|cc} \frac{3-\sqrt{3}}{6} & \frac{1}{4} & \frac{3-2\sqrt{3}}{12} \\ \frac{3+\sqrt{3}}{6} & \frac{3+2\sqrt{3}}{12} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

$s = 2, p = 4$

• *Radau methods: order* $2s - 1$ *(stiff decay)*

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

$s = 1, p = 1$

backward Euler

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & \frac{-1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}$$

- *If* $\mathbf{A}$ *is nonsingular and* $a_{sj} = b_j$,
$$j = 1, 2, \ldots, s, \quad (c_s = 1)$$
we say the RK method is stiffly accurate (i.e., it has the property of stiff decay).

  Note: *This is a sufficient (but not necessary) condition for stiff decay.*

- *Lobatto methods:* *order* $2s - 2$

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

$s = 2, p = 2$

(Yet another) trapezoidal method.

4.7.1 Collocation

A common idea in numerical analysis.

Idea: The solution to the ODE is probably pretty complicated.
(It may not be expressible in terms of elementary functions, e.g., $\sin t$, e^t, etc.)
Choose to represent the solution, e.g., as a polynomial ($\leftrightarrow$ a "simple space").
Then force the polynomial to satisfy the ODE at a number of points.
$\rightarrow$ Hopefully the polynomial will not deviate too much from the solution away from these points.
This process is called collocation, and the points where the polynomial is forced to satisfy the ODE are the collocation points.

Choose s distinct points

$$0 \leq c_1 < c_2 < \cdots < c_s \leq 1.$$

Find polynomial $\phi(t)$ of degree at most s that collocates the ODE $\dot{y} = f(t, y)$ at these points.

$$\begin{aligned} \phi(t_{n-1}) &= y_{n-1}, \\ \dot{\phi}(t_i) &= f(t_i, y(t_i)) \qquad i = 1, 2, \cdots, s, \end{aligned}$$

where $t_i = t_{n-1} + \Delta t_n c_i$ are the collocation points.

This defines $\phi(t)$ uniquely ($s+1$ equations, $s+1$ unknowns). (verify !)

Then we take $y_n = \phi(t_n)$.
$\rightarrow$ This gives an s-stage implicit RK method.

Let's see how.

First, note $\dot{\phi}(t)$ is a polynomial of degree at most $s-1$ that interpolates s points $f(t_i, \phi(t_i))$.

Define $K_i = \dot{\phi}(t_i)$.

Now write $\dot{\phi}$ as a Lagrange interpolant,

$$\dot{\phi}(t_{n-1} + \tau\Delta t_n) = \sum_{j=1}^{s} L_j(t_{n-1} + \tau\Delta t_n) K_j,$$

where

$$L_j(t_{n-1} + \tau\Delta t_n) = \prod_{\substack{i=1 \\ i\neq j}}^{s} \frac{(\tau - c_i)}{c_j - c_i}.$$

Now

$$\int_{t_{n-1}}^{t_i} \dot{\phi}(t)dt = \phi(t_i) - \phi(t_{n-1}) = \Delta t_n \sum_{j=1}^{s} \underbrace{\left(\int_0^{c_i} L_j(r)dr \right)}_{a_{ij}} K_j,$$

$$\int_{t_{n-1}}^{t_n} \dot{\phi}(t)dt = \phi(t_n) - \phi(t_{n-1}) = \Delta t_n \sum_{j=1}^{s} \underbrace{\left(\int_0^{1} L_j(r)dr \right)}_{b_j} K_j.$$

verify! (HINT: $r = \tau_{n-1} + \tau\Delta t_n$)

Then

$$\begin{aligned} K_i &= f(t_i, \phi(t_i)) \\ &= f\left(t_{n-1} + \Delta t_n c_i, y_{n-1} + \Delta t_n \sum_{j=1}^{s} a_{ij} K_j \right), \end{aligned}$$

and $$y_n = y_{n-1} + \Delta t_n \sum_{j=1}^{s} b_j K_j.$$

Note 13. • *Gauss, Radau, and Lobatto methods are collocation methods given their c_j, a_{ij}, b_j determined as before. $\rightarrow$ Easy to derive !*

• *For an s-stage RK method based on collocation,*

$$s \leq p \leq \underbrace{2s}_{\text{true for all RK methods}}$$

• *Radau generalizes backward Euler:*
$\rightarrow$ A-stable and has stiff decay (good for stiff IVPs).

- *Gauss, Lobatto are symmetric (good for BVPs). (Gauss generalizes midpoint; Lobatto generalizes trapezoidal rule.)*
  Both A-stable.

4.7.2 Implementation of IRKs

IRKs are challenging to implement because of the nonlinear systems that need to be solved at each step.

This also makes them usually less efficient than implicit multistep methods.

Recall the general RK method,

$$\mathbf{Y}_i = \mathbf{y}_{n-1} + \Delta t_n \sum_{j=1}^{s} a_{ij} \mathbf{f}(t_{n-1} + \Delta t_n c_j, \mathbf{Y}_j), \ i = 1, 2, \cdots, s,$$

$$\mathbf{y}_n = \mathbf{y}_{n-1} + \Delta t_n \sum_{i=1}^{s} b_i \mathbf{f}(t_{n-1} + \Delta t_n c_i, \mathbf{Y}_i).$$

At Newton iteration ν, let

$$\delta_i = \mathbf{Y}_i^{(\nu+1)} - \mathbf{Y}_i^{(\nu)}, \ \mathbf{r}_i = \mathbf{Y}_i^{(\nu)} - \mathbf{y}_{n-1} - \Delta t_n \sum_{j=1}^{s} a_{ij} \mathbf{f}(\mathbf{Y}_j^{(\nu)}).$$

Then the Newton iteration is

$$\begin{bmatrix} \mathbf{I} - \Delta t_n a_{11}\mathbf{J}_1 & -\Delta t_n a_{12}\mathbf{J}_2 & \cdots & -\Delta t_n a_{1s}\mathbf{J}_s \\ -\Delta t_n a_{21}\mathbf{J}_1 & \mathbf{I} - \Delta t_n a_{22}\mathbf{J}_2 & \cdots & -\Delta t_n a_{2s}\mathbf{J}_s \\ \vdots & \vdots & \ddots & \vdots \\ -\Delta t_n a_{s1}\mathbf{J}_1 & -\Delta t_n a_{s2}\mathbf{J}_2 & \cdots & \mathbf{I} - \Delta t_n a_{ss}\mathbf{J}_s \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_s \end{bmatrix}$$

$$= - \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_s \end{bmatrix},$$

where $\mathbf{J}_i = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_{n-1} + \Delta t_n c_i, Y_i^{(\nu)}), \quad i = 1, 2, \cdots, s.$

$\rightarrow$ A system of $s \cdot m$ equations.

- Can we take shortcuts to make it faster?
  ♣ Take
  $$\mathbf{J}_i = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t_{n-1}, \mathbf{y}_{n-1});$$
  i.e., evaluate $\mathbf{J}_i$ once.
  $\rightarrow$ Only one LU decomposition required.

  More iterations required, but each is much cheaper!

♢ Freeze $\mathbf{J}_i$ over several steps.
→ More time spent on software development.

♡ Use diagonally implicit RK (DIRK); i.e., choose $\mathbf{A}$ to be lower triangular:

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

If all $a_{ii} = \gamma$, singly diagonally implicit RK (SDIRK).
→ s $(m \times m)$ systems with same iteration matrix

But: ♣ Not collocation anymore.
- ♢ Stage order: 1.
- ♡ Maximum order of s-stage DIRK is $s + 1$.
- ♠ If you want stiff decay, then max order is s.

To increase max stage order, use "explicit" SDIRK (ESDIRK) method ($a_{11} = 0$, all other $a_{ii} = \gamma$).

Popular as implicit part of implicit-explicit (IMEX) RK methods.

4.7.3 Order Reduction

A word about *order reduction* (important for DAEs). Consider a very stiff problem,

$$\dot{y} = \lambda(y - g(t)), \qquad 0 < t < 1,$$

where $$0 < \frac{1}{-\mathcal{R}e\lambda} \ll \Delta t_n \ll 1.$$

$\rightarrow$ Two small parameters: Δt_n and $\frac{1}{\mathcal{R}e(\lambda)}$.

In such cases, the order of RK method reduces to the stage order (or stage order + 1).

$\rightarrow$ DIRKs have stage order 1 and so are not recommended for very stiff problems

Radau is good!

As mentioned, ESDIRKs are popular as well, especially those designed to satisfy so-called DAE order conditions.

4.7.4 SIRK Methods

DIRK methods exploit the lower-triangular structure of $\mathbf{A}$ to improve the efficiency of the linear algebra when solving nonlinear systems of equations.

Seeking lower-triangular matrices $\mathbf{A}$ can be restrictive.

Another approach is to use a similarity transformation $\mathbf{T}$ to take $\mathbf{A}$ to a simple form $\mathbf{S}$:

$$\mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \mathbf{S}.$$

Similarly transforming the variables

$$\hat{\boldsymbol{\delta}}_i = \mathbf{T}^{-1}\boldsymbol{\delta}_i,$$

each block of the Newton iteration matrix becomes

$$(\mathbf{I} - \Delta t_n \mathbf{S}\mathbf{J}_i)\hat{\boldsymbol{\delta}} = -\hat{\mathbf{r}}_i,$$

where $\hat{\mathbf{r}}_i = \mathbf{T}^{-1}\mathbf{r}_i$.

Now any lower-triangular matrix $\mathbf{S}$ yields the DIRK structure for the transformed variables $\hat{\boldsymbol{\delta}}$.

If we require $\mathbf{S} = a\mathbf{I}$, we have singly implicit RK (SIRK) methods.

The name comes from the fact that $\mathbf{S}$ has a single eigenvalue a.

This structure makes the linear algebra of SIRK methods equivalent to that of SDIRK methods.

$\rightarrow$ at most one $m \times m$ matrix is formed and factored per step.

STRIDE (Burridge, Butcher, Chipman) was an almost-famous code that attempted to do this.

The bottleneck is the computation of $\mathbf{T}$.