Runge-Kutta, part 2

Tuesday, November 3, 2020 2:30 PM

The zoo of RK methods: higher-order RK, and which ones to use?

Recall the "Butcher Array"

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_1 & a_{11} & & & & \\ c_2 & a_{21} & a_{22} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_{s-1} & a_{s-1,1} & a_{s-1,2} & \cdots & a_{s-1,s-1} & \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \end{array}$$

meaning

$$\begin{aligned} k_1 &= h f(t_i, w_i) \\ k_2 &= h f(t_i + c_1 h, w_i + a_{11} k_1) \\ k_3 &= h f(t_i + c_2 h, w_i + a_{21} k_1 + a_{22} k_2) \\ &\vdots \\ k_s &= h f(t_i + c_{s-1} h, w_i + a_{s-1,1} k_1 + \cdots) \\ w_{i+1} &= w_i + b_1 k_1 + b_2 k_2 + \cdots + b_s k_s \end{aligned}$$

Let's do another example

Book's "3rd order Heun's Method" (≠ 2nd order Heun's — this is what wikipedia talks about)

$$w_{i+1} = w_i + \frac{h}{4} f(t_i, w_i) + \frac{3}{4} h f\left(t_i + \frac{2}{3}h, \; w_i + \frac{2}{3}h f\left(t_i + \frac{1}{3}h, w_i + \frac{h}{3} f(t_i, w_i)\right)\right)$$

(k_1, $b_1 = 1/4$; $b_2 = 0$; $b_3 = 3/4$; $c_2 = 2/3$; $c_1 = 1/3$; $a_{11} = 1/3$; $a_{22} = 2/3$; $a_{21} = 0$; k_2, k_3)

wow! let's unpack

3 stages

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

(recall 2nd order Heun* was $\begin{array}{c|cc} 0 & & \\ 2/3 & 2/3 & \\ \hline & 1/4 & 3/4 \end{array}$ — 2 stages)

Note: wikipedia calls Heun's 2-stage method "improved" or "modified" Euler, or "Ralston's method" and its "improved Euler" is what we call "modified Euler".

Intuition

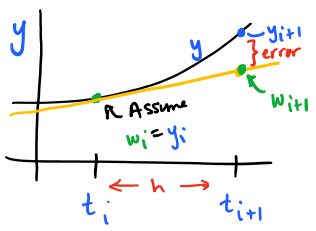
Where do these numbers come from? For higher-order, just come from derivations, but for low-order we have intuition

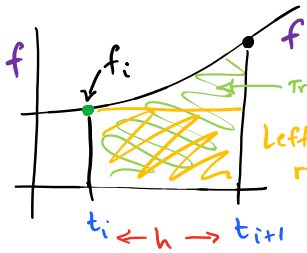
One way to see it: $y_{i+1} = y_i + \int_{t_i}^{t_i+h} y'(s)\, ds$

$= y_i + \int_{t_i}^{t_i+h} f(s, y(s))\, ds$ ... so we're integrating f

For notation, let $f_i = f(t_i, w_i)$

PLOTTING y PLOTTING y'

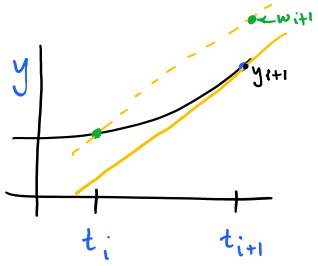


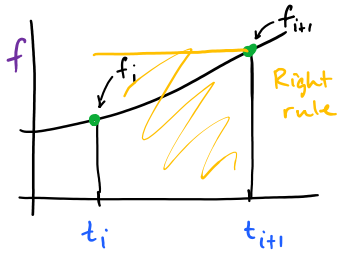


Forward Euler

$$w_{i+1} = w_i + h \cdot f_i$$

estimate of integral





Backward Euler

$$w_{i+1} = w_i + h f_{i+1}$$

Recall

$$f_{i+1} = f(t_{i+1}, w_{i+1}) \ldots$$

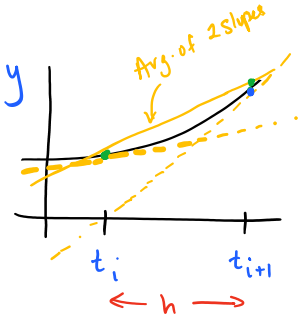
but we don't know w_{i+1} yet!

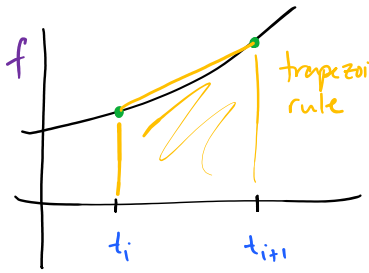
So... must solve a root-finding problem

$$w - w_i - h f(t_{i+1}, w) = 0$$

"IMPLICIT METHOD" and annoying!

But as we'll see later, there are sometimes good reasons to use





$$w_{i+1} = w_i + h\left(\tfrac{1}{2} f_i + \tfrac{1}{2} f_{i+1}\right)$$

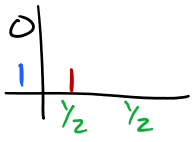
Implicit trapezoidal method

aka

Crank-Nicolson

Still need f_{i+1} so implicit

Now, modified Euler



means

$$k_1 = h \overbrace{f(t_i, w_i)}^{\text{"}f_i\text{"}}$$

$$k_2 = h f(\underbrace{t_i + 1 \cdot h}_{t_{i+1}}, w_i + 1 k_1)$$

$$w_{i+1} = w_i + \tfrac{1}{2} k_1 + \tfrac{1}{2} k_2$$

So... modified Euler is like trapezoid / Crank-Nicolson

$$w_{i+1} = w_i + h\left(\tfrac{1}{2} f_i + \tfrac{1}{2} f_{i+1}\right), \quad f_{i+1} = f(t_{i+1}, w_{i+1})$$

except we approximate f_{i+1} to keep it implicit

$$w_{i+1} = w_i + h\left(\tfrac{1}{2} f_i + \tfrac{1}{2} \tilde{f}_{i+1}\right)$$

$$\tilde{f}_{i+1} = f(t_{i+1}, \underbrace{w_i + h f_i}_{\text{Euler approximation}})$$

More RK examples

There are many 4-stage, 4th order RK, but "the 4th order RK" (most popular) is

"RK4"

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

(has lots of 0's, so saves on computation)

3rd order formula not too common in practice

Which order to use? Why is RK4 popular?

# stages (= # evals of f)	2	3	4	5	6	7	8	9	n ≥ 10
Best possible truncation error	$O(h^2)$	h^3	h^4	h^4	h^5	h^6	h^6	h^7	h^{n-3}

No benefit to 5-stage method over 4-stage

High-order error is nice (usually "worth it") but some disadvantages

- Requires more smoothness in the solution y
- More complicated implementations, especially for adaptive stepsizes and error estimates

... 4-stage is a nice balance

Embedded RK formula

For adaptive stepsizes, we'll want to estimate the error, just as we did for integration, so use higher-order method to estimate error in lower-order method.

So... must call two RK formulas. Want to re-use as much as possible.

"Embedded" RK formulas have the lower-order formula embedded inside the higher-order one

Ex: Bogacki-Shampine (this is in Matlab's ode23)

0				
1/2	1/2			
3/4	0	3/4		
1	2/9	1/3	4/9	
	2/9	1/3	4/9	(A)
	7/24	1/4	1/3	1/8 (B)
	5/72	-1/12	-1/9	1/8 (B)-(A)

the 3-stage 2nd order RK embedded within the 4-stage 3rd order RK formula.

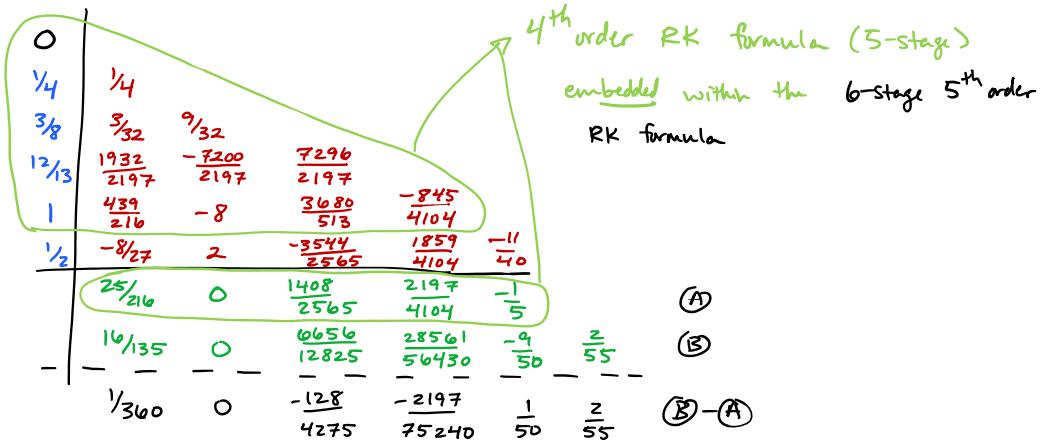
← we'll use later for error estimates

Note: 2/9, 1/3, 4/9 used for k_4, but also for constructing w_{i+1}. Since we evaluated f to create k_4, and $c_3 = 1$, it means we evaluated $f(t_{i+1}, w_{i+1})$

So we already have k_1 computed at the next iteration.

This is the "FSAL" (First Same As Last) property

Ex: Runge-Kutta Fehlberg (RK45)



Matlab's ode45 used to use R-K Fehlberg, though in late '90s switched to a different embedded 4/5th order RK formula due to Dormand and Prince (known as RK5(4)7FM aka DOPRI5 or DP(4,5) or DP45)

DP45 is 7 stages and is especially nice for constructing interpolants for but has FSAL property, so more like 6 stage method

from: “Behind and Beyond the MATLAB ODE Suite”, Ryuichi Ashino, Michihiro Nagase, Remi Vaillancourt, 2000

Embedded pairs of Runge–Kutta methods of orders p and $p+1$ with interpolant are used to control the local error and interpolate the numerical solution between the nodes which are automatically chosen by the step-size control. The difference between the higher and lower order solutions, $y_{n+1}-\hat{y}_{n+1}$, is used to control the local error. A popular pair of methods of orders 5 and 4, respectively, with interpolant due to Dormand and Prince [3] is given in the form of a Butcher tableau.

Table 1: Butcher tableau of the Dormand-Prince pair DP5(4)7M with interpolant.

	c	A						
k_1	0	0						
k_2	$\frac{1}{5}$	$\frac{1}{5}$	0					
k_3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$	0				
k_4	$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$	0			
k_5	$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$	0		
k_6	1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$	0	
k_7	1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
$\hat{y}_{n+1}$	$\hat{b}^T$	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$
y_{n+1}	b^T	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
$y_{n+0.5}$		$\frac{5783653}{57600000}$	0	$\frac{466123}{1192500}$	$-\frac{41347}{1920000}$	$\frac{16122321}{339200000}$	$-\frac{7117}{20000}$	$\frac{183}{10000}$

The number 5 in the designation DP5(4)7M means that the solution is advanced with the solution y_{n+1} of order five (a procedure called *local extrapolation*). The number (4) in parentheses means that the solution $\hat{y}_{n+1}$ of order four is used to obtain the local error estimate. The number 7 means that the method has seven stages. The letter M means that the constant C_6 in the top-order error term has been minimized, while maintaining stability. Six stages are necessary for the method of order 5. The seventh stage is necessary to have an interpolant. However, this is really a six-stage method since the first step at t_{n+1} is the same as the last step at t_n, that is, $k_1^{[n+1]} = k_7^{[n]}$. Such methods are called FSAL (First Step As Last). The upper half of the region of absolute stability of the pair DP5(4)7M comprises the interior of the closed region in the left half-plane and the little round region in the right half-plane shown in the right part of Fig. 1.

Other popular pairs of embedded Runge-Kutta methods are the Runge–Kutta–Verner and Runge–Kutta–Fehlberg methods. For instance, the pair RKF45 of order four and five minimizes the error constant C_5 of the lower order method which is used to advance the solution from y_n to y_{n+1}, that is, without using local extrapolation.

One notices that the matrix A in the Butcher tableau of an explicit Rung–Kutta method is strictly lower triangular. Semi-explicit methods have a lower triangular matrix. Otherwise, the method is implicit. Solving semi-explicit methods for the vector solution y_{n+1} of a system is much cheaper than solving explicit methods.