Embedded RK solver, some implementation aspects

Luca Formaggia

8 July 2021

Runge-Kutta embedded methods are methods for solving (systems of) ordinary differential equations of the form

$$\begin{cases} y'(t) = f(t, y(t)) & t \in (t_0, T] \\ y(0) = y_0 & \text{given} \end{cases}$$

Here, y is in general a vector of $\mathbb{R}^d$. If $d = 1$ we have a *scalar problem.* We mention that indeed y can also be a matrix of a complex number/vector. Here we assume it is a vector of real number (the discussion however extends to scalar problems and even problems in the complex plane.)

Runge-Kutta methods are one-step multistage methods. One step means that the computation of the solution at time $t_{n+1} = t_n + h$ requires the knowledge of the unknown at the previous time t_n only. This, as we will see, simplifies the implementation of adaptive schemes. Multistage means that the computation of the unknown at time t_{n+1} is performed through $s > 1$ intermediate steps.

They are defined through the *extended Butcher's tableau* (or Butcher's array)

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

where $\mathbf{A} \in \mathbb{R}^{s\times s}$, while $\mathbf{c}$, $\mathbf{b}$ and $\hat{\mathbf{b}}$ are vectors of $\mathbb{R}^s$ and $s > 0$ the number of *stages.* For consistency considerations, we must have $c_i = \sum_{j=1}^s A_{ij}$. The presence of the two vectors $\mathbf{b}$ and $\hat{\mathbf{b}}$ characterize the embedded formula (the standard formula has just one vector $\mathbf{b}$).

The solution in a step $t^n \to t^{n+1} = t^n + h$, being h the time step size, is obtained by solving the following problem,

$$\mathbf{K}_i = f(t + c_i h, \mathbf{y}n_+ h \sum_j A_{ij}\mathbf{K}_j) \quad i = 1, \ldots, s, \tag{1}$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{K}\mathbf{b}, \tag{2}$$

$$\hat{\mathbf{y}}_{n+1} = \mathbf{y}_n + h\mathbf{K}\hat{\mathbf{b}}. \tag{3}$$

Here $K \in \mathbb{R}^{d\times s}$, with $\mathbf{K}_i$ its i=th column and $\mathbf{y}n_{\in}\mathbb{R}^d$ the approximated solution at time t^n.

If $\mathbf{A}$ is *strictly* lower diagonal ($A_{ij} = 0$ if $j \geq i$), then the computation of $\mathbf{K}_i$ in (1) is *explicit*, i.e does not require the solution of any non-linear system, but solving in succession, for $i = 1, \ldots s$,

$$\mathbf{K}_i = f(t_n + c_i h, \mathbf{y}n_+ h \sum_{j=1}^{i-1} A_{ij}\mathbf{K}_j).$$

(Note that $\mathbf{K}_1 = f(t_n, \mathbf{y}n_))$.

Otherwise, the method is *implicit* and the step (1) involves, in general, the solution of a non-linear system of $s \times d$ unknowns (the coefficients of $\mathbf{K}$.).

There is an intermediate possibility. When $\mathbf{A}$ is *lower triangular* (but not *strictly lower triangular*) then the RK scheme is said to be *linearly* or *diagonally* implicit, (DIRK scheme). In this case, (1) involves the solution of s nonlinear systems of size d, for the unknown $\mathbf{K}_i$, solved in succession:

$$\mathbf{K}_i - f(t + c_i h, \mathbf{y}n_+ h \sum_{j=1}^{i-1} A_{ij}\mathbf{K}_j + hA_{ii}\mathbf{K}_i) = \mathbf{0} \quad i = 1, \ldots, s$$

Finally, a subclass of the DIRK schemes are the ESDIRK [2] (explicit singly diagonal implicit) schemes, where $\mathbf{A}$ is lower triangular but the first row is zero. It means that the first stage is explicit and you have to solve only $s - 1$ implicit stages at each time step.

Equations (2) and (3) provide two different approximations of $\mathbf{y}(t^{n+1})$, the former, $\mathbf{y}^{n+1}$, is called the *primal*, and is the one actually used to advance the numerical solution, the latter, $\hat{\mathbf{y}}^{n+1}$, sometimes called *secondary*, is instead used to *compute an estimate of the local truncation error (LTE)*, that is the error in a single step (look any good book on numerical analysis for a formal definition of LTE).

1 Computation of the estimate of the LTE

The trick is that the coefficients of the Butcher tableau are given such that relations (2) and (3) provide an approximation of *different order*, typically the two orders differ by 1. So let's assume that the solution of the primal scheme $\mathbf{y}_{n+1}$ is that of *lower order*, and let p indicate the order (we will see later that we can switch the role of primal and secondary solution). Therefore, $\hat{\mathbf{y}}_{n+1}$ is of order $p + 1$.

We recall that the *order* of the method (more precisely the order of consistency) is the order of infinitesimal with respect to the time step of the *truncation error*, which bounds, under condition of regularity of f, the *order of convergence.*

Indeed we have that, indicating with $\tau_p(h)$ the truncation error of the low order scheme, and with N the final step

$$\|\mathbf{y}_N - \mathbf{y}(T)\| \leq C_T \tau_p(h), \tag{4}$$

for a $C_T > 0$. We remind again, that the stated result holds for sufficiently small h and under conditions on the regularity of f. You may also have similar results in other norms. *A scheme is said to be of order p when* $\tau_p = O(h^p)$, therefore if the primal scheme is of order p we have

$$\|\mathbf{y}_N - \mathbf{y}(T)\| \leq C_T h^p,$$

The *local truncation error* (LTE) is the error carried out on a *single step* and is equal to $h\boldsymbol{\tau}_{p,n}(h)$, where $\boldsymbol{\tau}_{p,n}(h)$ is linked to the truncation error by

$$\tau_p = \max_n \|\boldsymbol{\tau}_{p,n}(h)\|. \tag{5}$$

and is defined as

$$\boldsymbol{\tau}_{p,n}(h)h = \mathbf{y}_{n+1} - \mathbf{y}(t^n) = O(h^{p+1}). \tag{6}$$

Analogously, since $\hat{\mathbf{y}}$ is an approximation of order $p + 1$,

$$\hat{\mathbf{y}}_{n+1} - \mathbf{y}(t^n) = O(h^{p+2}). \tag{7}$$

Therefore, subtracting (7) from (6), taking the norm and assuming h sufficiently small so that we can neglect higher order terms, we obtain,

$$\Delta_n = \|\hat{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1}\| \simeq \|\boldsymbol{\tau}_{p,n}(h)\|h = O(h^{p+1}). \tag{8}$$

Therefore, Δ can be used as an *estimate of the local truncation error of the lower order scheme.*

2 Controlling the error

The relations of the previous section will tell us how to control the error using the estimate provided by Δ_n. The expression for the error (4) leads us to the idea of controlling τ_p and (5) to the fact that this can be obtained if at each time step we control $\|\boldsymbol{\tau}_{p,n}$.

Therefore let's call e the *desired* τ_p. The objective is to have $\tau_p \simeq e$. At the n-th step we can compute Δ_n and observe that, from (8),

$$\Delta_n/h \simeq \|\boldsymbol{\tau}_{p,n}(h)\| \simeq Ch^p \tag{9}$$

while the step h_e necessary (under the approximation of the error estimate) to have a final error proportional to e satisfies

$$e \simeq Ch_e^p. \tag{10}$$

So the strategy is as follows. Since we need computable estimates we make the approximation of considering the C in (9) and (10) identical and replace $\simeq$ with equalities.

After computing Δ_n with the latest value of h (at the first step we use an h of choice) we calculate the ratio

$$r = \frac{eh}{\Delta_n}$$

If $r > 1$ the step h is too large. Then (9) and (10) suggest that the "good" h to control the error is

$$h_e = \left(\frac{1}{r}\right)^{1/p} = \left(\frac{eh}{\Delta_n}\right)^{1/p}. \tag{11}$$

So, we *reject* the step and repeat the computations with $h = h_e$, where h_e is computed with the previous formula.

If $r \leq 1$ the step can be accepted. Yet, if $r < 1$ it may be worthwile to increase h since the current one is too small for the given target error. However, expansions are always more critical, since we may end up to a too large h. So, in this case we prefer to control the local truncation error, and not the global one. Thus for $r \leq 1$ we *accept the step* and for the next step we correct h by setting it equal to h_e given by

$$h_e = \left(\frac{1}{r}\right)^{1/(p+1)} = \left(\frac{eh}{\Delta_n}\right)^{1/(p+1)}. \tag{12}$$

The last expression is indeed conservative, since it will increase h less than the use of (11).

Moreover, to be even more conservative, one uses also some safety factors $\alpha_c > 1$ and $\alpha_e < 1$ (typically 1.01 and 0.98) and the algorithm for error control at each time step is as follows

1. Compute Δ_n and $r = \dfrac{eh}{\Delta_n}$.

2. If $r \leq 1$ accept the step, compute $\mathbf{y}_{n+1}$ and set

$$h = \alpha_e \left(\frac{eh}{\Delta_n}\right)^{1/(p+1)}.$$

3. if $r < 1$ *reject* the step and repeat it with

$$h = \alpha_c \left(\frac{eh}{\Delta_n}\right)^{1/p}.$$

3 Choice of the primal scheme

We have seen thet the embedded RK method provides two approximations, one of order p and one of order $p+1$ (in general order $q > p$, but $q = p+1$ is the usual situation). The estimate based on Δ_n is an estimate for the *lower order scheme*, so it would be natural to update the solution using the lower order scheme. In other word, use the lower order scheme as primal.

For instance, the scheme $RK4(5)$ is an embedded RK scheme with primal scheme of 4-th order and it uses a scheme of 5-th order only to compute the error estimate.

Yet, one may wonder whether not to use the higher order scheme as primal. After all we have at disposal a more accurate approximation, so why not use it?

Indeed, the RK5(4) is also an embedded RK method that uses schemes of order 4 and 5, but, differently from RK4(5), uses the latter as primal (i.e. to advance the solution).

Some considerations are however necessary. Consistency error is not the only thing to take into account, we should consider also stability. Depending on how the method has been constructed (there are many RK embedded schemes around, see [1]) the higher order scheme may or may not have worse stability properties than the lower order one.

This is particularly true in DIRK and ESDIRK schemes (which are implicit). One wants to use an implicit scheme to have A or L-stability in order to treat stiff problems. So the choice of which scheme to select as primal really depends on the properties of the specific method and often in implicit DIRK it is the lower order scheme that guarantees A-stability.

References

[1] Runge–Kutta Methods. In *Numerical Methods for Ordinary Differential Equations*, chapter 3, pages 143–331. John Wiley & Sons, Ltd, 2016.

[2] John Bagterp Jørgensen, Morten Rode Kristensen, and Per Grove Thomsen. A Family of ESDIRK Integration Methods. *arXiv:1803.01613 [math]*, March 2018.