2 Runge–Kutta methods

2.1 The family of Runge–Kutta methods

In this section, we will introduce a family of increasingly accurate, and time-efficient, methods called *Runge–Kutta* methods after two German scientists: a mathematician and physicist Carl Runge (1856–1927) and a mathematician Martin Kutta (1867–1944).

The Modified Euler and Midpoint methods of the previous section can be written in a form common to both of these methods:

$$\begin{aligned} &Y_{i+1} = Y_i + (b_1\, k_1 + b_2\, k_2);\\ &k_1 = hf(x_i, Y_i),\\ &k_2 = hf(x_i + c_2\, h, Y_i + a_{21} k_1);\\ &a_{21},\, b_1,\, b_2,\, c_2, \quad \text{are some constants.} \end{aligned} \tag{2.1}$$

The choice of indices in these constants will become clear soon. Now, for the Modified Euler,

$$b_1 = b_2 = \frac{1}{2}, \qquad c_2 = a_{21} = 1; \tag{2.2}$$

while for the Midpoint method (see Questions for self-assessments),

$$b_1 = 0, \quad b_2 = 1, \qquad c_2 = a_{21} = \frac{1}{2}. \tag{2.3}$$

In general, if we require that method (2.1) have the global error $O(h^2)$, we can repeat the calculations we carried out in Section 1.4 for the Modified Euler method and obtain the following *3 equations* for *4 unknown coefficients* a_{21}, b_1, b_2, c_2:

$$b_1 + b_2 = 1, \qquad c_2 b_2 = \frac{1}{2}, \qquad a_{21} b_2 = \frac{1}{2}\,. \tag{2.4}$$

Observations:

- Since there are fewer equations than unknowns in (2.4), then there are infinitely many finite-difference methods whose global error is $O(h^2)$.

- One can generalize form (2.1) and seek methods of higher order (i.e. with the global error of $O(h^k)$ with $k \geq 3$) as follows:

$$\begin{aligned} &Y_{i+1} = Y_i + (b_1 k_1 + b_2 k_2 + b_3 k_3 + \dots);\\ &k_1 = hf(x_i, Y_i),\\ &k_2 = hf(x_i + c_2 h, Y_i + a_{21} k_1),\\ &k_3 = hf(x_i + c_3 h, Y_i + a_{31} k_1 + a_{32} k_2),\\ &\text{etc.} \end{aligned} \tag{2.5}$$

  This family of methods is called the *Runge–Kutta (RK) methods*. The quantities $k_1, \dots k_s$ are called *stages*. (Hence, an s-stage RK method requires s function evaluations.)

- In the above notations, one would always have $c_1 = 0$. There is a more general, than (2.5), form of the Runge–Kutta methods, where this is not necessarily so. This form will be mentioned in Lecture 4.

- A neat, and conventional in the field of Numerical Analysis, arrangement of coefficients a_{mn}, b_m, c_m is presented in Appendix 1 to this Lecture; see Section 2.3.

For example, if one looks for 4th-order methods, one obtains 11 equations for 13 coefficients. Again, this says that there are infinitely many 4th-order methods. Historically, the most popular such method has been

$$\begin{aligned} &Y_{i+1} = Y_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right); \\ &k_1 = hf(x_i, Y_i), \\ &k_2 = hf\left(x_i + \frac{1}{2}h, Y_i + \frac{1}{2}k_1\right), \\ &k_3 = hf\left(x_i + \frac{1}{2}h, Y_i + \frac{1}{2}k_2\right), \\ &k_4 = hf\left(x_i + h, Y_i + k_3\right). \end{aligned} \tag{2.6}$$

We will refer to this as the *classical* Runge–Kutta (cRK) method.

The table below compares the time-efficiency of the cRK and Modified Euler methods and shows that the former method is much more efficient.

Method	Global error	# of function evaluations per step
cRK	$O(h^4)$	4
Modified Euler	$O(h^2)$	2

One of the reasons why the cRK method is so popular is that the number, s, of stages k_1, $\ldots k_s$ (which always equals the number of function evaluations) per step in it equals the order of the method. It is known that RK methods of order $n \geq 5$ require *more than* n function evaluations; i.e. they are less efficient than the cRK and other lower-order RK methods. For example, a 5th-order RK method would require a minimum of 6 function evaluations per step.

2.2 Adaptive methods: Controlling step size for given accuracy

In this subsection, we discuss an important question of how the error of the numerical solution can be controlled and/or kept within a prescribed bound. A more complete and thorough discussion of this issue can be found in a paper by L.F. Shampine, "Error estimation and control for ODEs," SIAM J. of Scientific Computing, **25**, 3–16 (2005). A preprint of this paper is available on the course website.

To begin, we emphasize two important points about error control algorithms.

1. These algorithms control the *local truncation error*, and <u>not</u> the *global error*, of the solution. Indeed, the only way to control the global error is to run the simulations more than once. For example, one can run a simulation with the step h and then repeat it with the step $h/2$ to verify that the difference between the two solutions is within a prescribed accuracy. Although this can be done occasionally (for example, when confirming a key result of one's paper), it is too time-expensive to do so routinely. Therefore, the error control algorithms make sure that the local error at each step is less than a given tolerance (which is in some way related to the prescribed global accuracy), and then just let the user hope that the global accuracy is met. Fortunately, this hope comes true in most cases; but see the aforementioned paper for possible problematic cases.

2. The goal of the error control is not only to control the error but also to optimize the step size used to obtain different portions of the solution. For example, if it is found that the solution changes very smoothly on a subinterval $I_{\rm smooth}$ of the computational interval, then the step size on $I_{\rm smooth}$ can be taken sufficiently large. On the contrary, if one detects that the solution changes rapidly on another interval, $I_{\rm rapid}$, then the step size there should be decreased.

Methods where both the solution and its error are evaluated at each step of the calculation are called *adaptive* methods. They are most useful in problems with abruptly (or rapidly) changing coefficients. One simple example of such a problem is the motion of a skydiver: the air resistance changes abruptly at the moment the parachute opens. This will be discussed in more detail in the homework.

To present the idea of the algorithm used by adaptive methods, assume for the moment that we know the exact solution y_i. Let $\varepsilon_{\rm glob}$ be the maximum desired global error and n be the order of the method. Then the *actual* local truncation error must be $O(h^{n+1})$, or $ch^{n+1} + O(h^{n+2})$ with some constant c. Since the maximum allowed *local* truncation error, $\varepsilon_{\rm loc}$, is not prescribed, it has to be postulated in some plausible manner. The common choice is to take $\varepsilon_{\rm loc} = h\varepsilon_{\rm glob}$. Then, the steps of the **algorithm of an adaptive method** are as follows.

1. At each x_i, compute the actual local truncation error $\epsilon_i = |y_i - Y_i|$ and compare it with $\varepsilon_{\rm loc}$. (The practical implementation of this step is described later.)

2a. If $\epsilon_i < \varepsilon_{\rm loc}$, then accept the solution, multiply the step size by $\kappa(\varepsilon_{\rm loc}/\epsilon_i)^{1/(n+1)}$, (where κ is some numerical coefficient less than 1), and proceed to the next step.

2b. If $\epsilon_i > \varepsilon_{\rm loc}$, then multiply the step size by $\kappa(\varepsilon_{\rm loc}/\epsilon_i)^{1/(n+1)}$, re-calculate the solution, and check the error. If the actual error is acceptable, proceed to the next step. If not, repeat this step again.

Note that with the above step size adjustment, the error at the next step is expected to be approximately

$$c\left(h\cdot\kappa\left(\frac{\varepsilon_{\rm loc}}{\epsilon_i}\right)^{1/(n+1)}\right)^{n+1} \;=\; \varepsilon_{\rm loc}\kappa^{n+1}\,\frac{ch^{n+1}}{\epsilon_i} \;\approx\; \varepsilon_{\rm loc}\kappa^{n+1}\,.$$

The coefficient $\kappa < 1$ (say, $\kappa = 0.9$) is included to avoid the situation where the computed error just slightly exceeds the allowed bound, which would be acceptable to a human, but the computer will have to recalculate the entire step, thereby wasting expensive function evaluations.

Now, in reality, the exact solution of the ODE is not known. Then one can use the following trick. Suppose the numerical method we use is of sufficiently high order (e.g., the order 4 of the cRK method is sufficiently high for all practical purposes). Then we can compute the solution Y_i^h with the step size h and *at each step* compare it with the solution $Y_i^{h/2}$, obtained with the step size being halved. For example, for the cRK method is of fourth order, and hence $Y_i^{h/2}$ should be closer to the exact solution than Y_i^h is by about $2^4 = 16$ times. Then one can declare $Y_i^{h/2}$ *to be* the exact solution, compute $\tilde\epsilon_i = |Y_i^{h/2} - Y_i^h|$, and use $\tilde\epsilon_i$ in place of the ϵ_i above.[1]

However, this way is very inefficient. For example, for the cRK method, it would require 7 additional function evaluations per step (needed to advance $Y^{h/2}$ from x_i to x_{i+1}). Therefore,

[1]Note that even though $\tilde\epsilon_i$ *appears* to be defined as the global error (because $Y_i^{h/2}$ is used in place of the exact solution), it is actually the *local* truncation error. This is because at each step, we *start from the "exact" solution* $Y_i^{h/2}$ rather than using the previous approximations Y_i^h.

people have designed alternative approaches to control the error size. Below we briefly describe the idea behind one of the most well-known such approaches.

Runge–Kutta–Fehlberg method [2]

Idea: Design a 5th-order method that would *share* some of the function evaluations with a 4th-order method. The solution $Y_i^{[5]}$, obtained using the 5th-order method, is expected to be much more accurate than the solution $Y_i^{[4]}$, obtained using the 4th-order method. Then we declare $\tilde{\epsilon}_i = |Y_i^{[5]} - Y_i^{[4]}|$ to be the local truncation error[3] and adjust the step size based on that error relative to the allowed tolerance.

Implementation:

$$
\begin{aligned}
&Y_{i+1}^{[4]} = Y_i + \left(\frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5\right),\\
&Y_{i+1}^{[5]} = Y_i + \left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6\right);\\
&k_1 = hf(x_i, Y_i),\\
&k_2 = hf\left(x_i + \frac{1}{4}h, Y_i + \frac{1}{4}k_1\right),\\
&k_3 = hf\left(x_i + \frac{3}{8}h, Y_i + \frac{3}{32}k_1 + \frac{9}{32}k_2\right),\\
&k_4 = hf\left(x_i + \frac{12}{13}h, Y_i + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right),\\
&k_5 = hf\left(x_i + h, Y_i + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right),\\
&k_6 = hf\left(x_i + \frac{1}{2}h, Y_i - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right),\\
&\text{where } Y_i = Y_i^{[5]}.
\end{aligned}
\tag{2.7}
$$

Altogether, there are only 6 function evaluations per step, because the 4th- and 5th-order methods share 5 function evaluations (one of them is for finding k_2, which does not explicitly enter the expressions for $Y_i^{[4]}$ and $Y_i^{[5]}$ but is used in the evaluation of k_3). The RK-Fehlberg method is an instance of an *embedded* method, because it *embeds* the computation of the 4th-order solution into the computation of a higher- — 5th- — order solution.

Just for historical record, in Appendix 3 we present another adaptive method, which had been proposed almost 20 years before RK–Fehlberg's one and which uses a different idea; this other adaptive method is no longer used today.

To conclude the presentation of adaptive RK methods, we must specify what solution is taken at x_{i+1}. For example, for the RK–Fehlberg method, we have the choice between setting Y_{i+1} to either $Y_{i+1}^{[4]}$ or $Y_{i+1}^{[5]}$. The common sense suggests setting $Y_{i+1} = Y_{i+1}^{[5]}$, because, after all, it is $Y_{i+1}^{[5]}$ that we have declared to be our "exact" solution. This choice does work in most

[2] It is interesting to note that while the cRK method was developed in early 1900's, its extension by Fehlberg was proposed only in 1970.

[3] Just as in the footnote on the previous page, this implies that as the initial value at *each* step, here one uses the more accurate of the two solutions, $Y_i^{[5]}$.

circumstances, although there are important exceptions (see the paper by L. Shampine). Thus, what the RK–Fehlberg method does is *compute a 5th-order-accurate solution while controlling the error of a less accurate 4-th-order solution related to it.*

2.3 Appendix 1: Butcher's tableau

The coefficients a_{mn}, b_m, c_m in (2.5) can be arranged in a matrix and vectors with respective entries. For example, for the Modified Euler method, one has (compare with Eqs. (2.2)):

$$\mathbf{A} \equiv \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \qquad \mathbf{b} \equiv \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}, \qquad \mathbf{c} \equiv \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.8}$$

Similarly, for the Midpoint method, one has:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 \\ 1/2 & 0 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad \mathbf{c} = \begin{pmatrix} 0 \\ 1/2 \end{pmatrix}. \tag{2.9}$$

In advanced texts on Numerical Analysis, it is common to present a RK scheme by presenting its *Butcher tableau*,[4] which is just a certain way to display matrix $\mathbf{A}$ and vectors $\mathbf{b}$ and $\mathbf{c}$. The form of the Butcher tableau is this:

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} \tag{2.10}$$

where $\mathbf{b}^T$ is the row made by transposing the column vector $\mathbf{b}$. For example, the Butcher tableau for the Midpoint method would usually be listed as:

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array} \tag{2.11}$$

Note that the convention is to *not* list zeros in the $\mathbf{A}$ matrix, leaving blank spaces where zeros are supposed to be. Zeros, however, *are* explicitly shown in the $\mathbf{b}$ and $\mathbf{c}$ vectors.

You may legitimately wonder if it is ever possible to have $c_1 \neq 0$ or to have the first row of matrix $\mathbf{A}$, i.e. entries a_{11}, a_{12}, etc., nonzero. Indeed, in the latter case, the second line in Eqs. (2.5) would read:

$$k_1 = h\, f(x_i, Y_i + a_{11}k_1 + a_{12}k_2)\,; \tag{2.12}$$

then, how would it be possible to find the k_1 on the left hand side if one needs the very same quantity on the right hand side, not to mention a yet unknown k_2? It turns out that methods with $c_1 \neq 0$ and $a_{1n} \neq 0$ do, actually, exist. They are called *implicit* RK methods and have a certain advantage over the *explicit* RK methods considered in this Lecture. We will learn about this advantage (albeit for a slightly different method) in Lecture 4. (Of course, implicit RK methods also have the disadvantage mentioned earlier in this paragraph.)

[4] Named after J.C. Butcher, a numerical analyst from New Zealand, who contributed significantly in the development of the theory of Runge–Kutta methods.

2.4 Appendix 2: Computing the solution at intermediate points

In some applications, one wants to know the solution not only at the grid points $x_i = x_0+(i-1)h$, but also at some points $x_{i,\theta} = x_i + \theta h \in (x_i, x_{i+1})$, $0 < \theta < 1$, between grid points. A generalization of RK methods that allows one to perform such computations efficiently is called *Dense output methods*. Its basic idea is somewhat similar to that of the embedded methods (of which the RK–Fehlberg method is an example). Namely, given an s-stage, n-th order (i.e., with local truncation error $O(h^{n+1})$) RK method with stages $k_1,\ldots,k_s$, one seeks a related dense output method in the form:

$$Y_{i,\theta} = Y_i + \sum_{i=1}^{s^*} b_i(\theta)\, k_i \qquad \text{with some coefficients } b_i(\theta). \tag{2.13}$$

If the local error of $Y_{i,\theta}$ is required to also be $O(h^{n+1})$ (as opposed to being of a lower order), then one needs to have $s^* \geq s+1$; i.e., a dense output method requires more stages than the original RK method with the same order of local error. However, stages $k_1,\ldots,k_s$ of the original method can be designed to be embedded into the stages $k_1,\ldots,k_{s^*}$ of the dense output method; therefore, the number of extra function evaluations can be made small.

More details can be found, e.g., in the book by E. Hairer, S.P. Nørsett, G. Wanner, "Solving Ordinary Differential Equations I (Nonstiff problems)," 2nd Ed., Springer, 1993, Sec. II.6.

2.5 Appendix 3: Runge–Kutta–Merson method

Idea: For certain choices of the auxiliary functions k_1, k_2, etc., the local truncation error of, say, a 4th order RK method can be made equal to $C_5 h^5 y^{(5)}(x_i) + O(h^6)$ with some known coefficient C_5. (Note that this local truncation error is proportional to the $(n+1)$-st derivative of the solution, where n is the order of the method. We observed a similar situation earlier for the simple Euler method; see Eq. (1.3).) On the other hand, a certain linear combination of the k's can also be chosen to equal $C_5 h^5 y^{(5)}(x_i) + O(h^6)$ *for a certain class of functions*. Namely, this occurs for linear functions:

$$f(x,y) = a\cdot x + b\cdot y + c, \tag{2.14}$$

where $a, b, c = \text{const}$. Thus, we can obtain both an approximate solution and an estimate for its error. This estimate is *not* expected to be accurate unless we a solving an ODE with the r.h.s. (2.14), but it will likely still give one an indication of where the actual error decreases or increases significantly due to the behavior of the exact solution. We can then use that estimate to adjust the step size so as to always make the (estimate for the) local truncation error below a prescribed maximum value.

For example, if one computes the solution Y_i using the cRK method and then, in addition, evaluates

$$k_5 = hf\left(x_i + \frac{3}{4}h, Y_i + \frac{1}{32}\left[5k_1 + 7k_2 + 13k_3 - k_4\right]\right), \tag{2.15}$$

then it can be shown (with a great deal of algebra) that

$$\text{Local truncation error } \sim \frac{2}{3}h\left(-k_1 + 3k_2 + 3k_3 + 3k_4 - 8k_5\right) + O(h^6)\,. \tag{2.16}$$

Here the sign '$\sim$' is used instead of the '$=$' because the equality holds only for $f(x,y)$ of the form (2.14). Thus, again, by evaluating function f just one extra time compared to the cRK

method, one obtains both the numerical solution and a *crude* estimate for its error. Then this error estimate can be used as the actual error ϵ_i in the algorithm of the corresponding adaptive method.

Implementation: More popular than the method described by (2.15) and (2.16), however, is another method based on the same idea and called the *Runge–Kutta–Merson* method:

$$
\begin{aligned}
&Y_{i+1} = Y_i + \frac{1}{6}\,(k_1 + 4k_4 + k_5);\\
&k_1 = hf(x_i, Y_i),\\
&k_2 = hf\left(x_i + \frac{1}{3}h, Y_i + \frac{1}{3}k_1\right),\\
&k_3 = hf\left(x_i + \frac{1}{3}h, Y_i + \frac{1}{6}(k_1 + k_2)\right),\\
&k_4 = hf\left(x_i + \frac{1}{2}h, Y_i + \frac{1}{8}(k_1 + 3k_3)\right),\\
&k_5 = hf\left(x_i + h, Y_i + \frac{1}{2}(k_1 - 3k_3 + 4k_4)\right),\\
&\text{Local truncation error } \sim \tfrac{1}{30}(2k_1 - 9k_3 + 8k_4 - k_5) \propto h^5\, y^{(5)}.
\end{aligned}
\tag{2.17}
$$

Once again, one should note that the last line above is only a crude estimate for the truncation error (valid only when $f(x, y)$ is a linear function of x and y). Indeed, if it had been valid for any $f(x, y)$, then we would have a contradiction with a statement found at the end of Sec. 2.1. (Which statement is that?)

2.6 Questions for self-assessment

1. Verify the statements about Eqs. (2.2) and (2.3) made at the beginning of Section 2.1. For example, for the Modified Euler, begin by comparing the first line in (2.1) with the last line of Eq. (1.22) in Lecture 1. This, along with the definition of k_1 in (2.1), will tell you what b_1 and b_2 are. The value of c_2 should easily follow from comparison of the last line of (1.22) and third line in (2.1). Finally, comparison of the latter line with the second line of (1.22) should tell you what a_{21} is. Do not forget to repeat this process for the Midpoint method.

2. List the 13 coefficients mentioned in the paragraph after Eq. (2.5). Do not write the 11 equations.

3. If the step size is reduced by a factor of 2, how much will the error of the cRK and the Modified Euler methods be reduced? Which of these methods is more accurate?

4. Suppose $f = f(x)$ (on the r.h.s. of the ODE); that is, f does not depend on y but only on x. What numerical integration method (studied in Calculus 2) does the cRK method reduce to? [*Hint*: Rewrite Eq. (2.6) for $f = f(x)$.]

5. List the 7 function evaluations mentioned in the paragraph before the title 'Runge–Kutta–Fehlberg method'.

6. Describe the idea behind the Runge–Kutta–Fehlberg method.

7. One of the built-in ODE solvers in MATLAB is called `ode45` . What do you think the origin of this name is? Without reading the description of this solver under MATLAB's help browser, can you guess what order this method is?

8. Write the Butcher tableau for the Modified Euler method.

9. Describe the idea behind the Runge–Kutta–Merson method.

10. Which statement is meant in the paragraph following Eq. (2.17)?

11. Is the Runge–Kutta–Merson an embedded method? (You do not need to understand the mechanics of this method to answer this question. All you need is to compare Eq. (2.17) with the definition of an embedded method, found earlier in these notes.)