Numerical Differential Equations
Final lectures

Prof. Jacob Bedrossian
University of Maryland, College Park

These notes will supplement the lectures on numerical differential equations. Roughly speaking, they are a distillation of Iserles' book on the topic.

1 Runge-Kutta

RK methods are based on using quadrature methods. Indeed, if we have a quadrature scheme consisting of points c_j and weights b_j with $1 \leq j \neq \nu$ of order p, we could approximate the integral via:

$$\int_{t_n}^{t_{n+1}} f(\tau, y(\tau))d\tau = h\sum_{j=1}^{\nu} b_j f(t_n + hc_j, y(t_n + hc_j)) + O(h^{p+1}). \tag{1}$$

Since

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(\tau, y(\tau))d\tau, \tag{2}$$

we have the approximation:

$$y(t_{n+1}) = y(t_n) + h\sum_{j=1}^{\nu} b_j f(t_n + hc_j, y(t_n + hc_j)) + O(h^{p+1}). \tag{3}$$

Of course, for this method to work as a numerical scheme, we would somehow have to know not just have an approximation for $y(t_n)$ but also $y(t_n + hc_j)$ for $1 \leq j \leq \nu$. Hence, a numerical method based on a quadrature scheme like above, will need to also make approximations for this sub-steps. This suggests a general scheme of the following type (which, like most math, looks scarier than it really is):

$$\xi_1 = y_n + \sum_{j=1}^{\nu} ha_{1,j} f(t_n + c_j h, \xi_j) \tag{4}$$

$$\vdots = \vdots \tag{5}$$

$$\xi_\nu = y_n + \sum_{j=1}^{\nu} ha_{\nu,j} f(t_n + c_j h, \xi_j) \tag{6}$$

$$y_{n+1} = y_n + h\sum_{j=1}^{\nu} b_j f(t_n + c_j h, \xi_j). \tag{7}$$

The ξ_i's are usually called *stages* and a scheme of this type is a called a *ν-stage Runge-Kutta method*. We should interpret as satisfying $\xi_j \approx y(t_n + c_j h)$ the j-th sub-step. Note that we first solve for the ξ_j's and then we solve for y_{n+1} by a quadrature scheme using the ξ_j's in place of the $y(t_n + c_j h)$'s. The most important sub-set of possible schemes are those for which we do not have to solve any

nonlinear equations – the *explicit Runge-Kutta schemes* (usually called ERK methods):

$$\xi_1 = y_n \tag{8}$$
$$\xi_2 = y_n + ha_{2,1}f(t_n, \xi_1) \tag{9}$$
$$\vdots \tag{10}$$
$$\xi_i = y_n + h\sum_{j=1}^{i-1} a_{i,j}f(t_n + c_j h, \xi_j) \tag{11}$$
$$\vdots \tag{12}$$
$$\xi_\nu = y_n + \sum_{j=1}^{\nu-1} ha_{\nu,j}f(t_n + c_j h, \xi_j) \tag{13}$$
$$y_{n+1} = y_n + h\sum_{j=1}^{\nu} b_j f(t_n + c_j h, \xi_j). \tag{14}$$

In particular, note that for ξ_i depends only on ξ_j for $j < i$ and hence we can explicitly write all the substeps as simple assignments rather than nonlinear equation solves. Usually if you say "Runge-Kutta" it is normally assumed that you mean an ERK method. The implicit Runge-Kuttas are much harder to implement and so will usually only be used when absolutely necessary. The one stage ERK method is just forward Euler:

$$\xi_1 = y_n \tag{15}$$
$$y_{n+1} = y_n + hf(t_n, \xi_1). \tag{16}$$

It turns out there is a whole family of 2-stage ERK methods:

$$\xi_1 = y_n \tag{17}$$
$$\xi_2 = y_n + ha_{2,1}f(t_n, \xi_1) \tag{18}$$
$$y_{n+1} = y_n + h\sum_{j=1}^{2} b_j f(t_n + c_j h, \xi_j). \tag{19}$$

(Automatically $c_1 = 0$ by $\xi_1 = y_n$). Let us derive conditions on a, b, c to imply a local truncation error of $O(h^3)$. That is, if $y(t)$ is the exact solution to the ODE, then we want to show:

$$y(t_{n+1}) - y(t_n) - hb_1 f(t_n, y(t_n)) - hb_2 f(t_n + c_2 h, y(t_n) + ha_{2,1}f(t_n + c_1 h, y(t_n))) = O(h^3). \tag{20}$$

Taylor expanding everything around t_n gives

$$y(t_{n+1}) = y + y'h + \frac{h^2}{2}y'' + O(h^3) \tag{21}$$
$$f(t_n + c_2 h, y(t_n) + ha_{2,1}f(t_n, y(t_n))) = f + c_2 h f_t + h f_y a_{2,1} f + O(h^2) \tag{22}$$

and then from the ODE $y'(t) = f(t, y(t))$ we obtain

$$y'' = f_t(t, y(t)) + f_y(t, y(t))f(t, y(t)) = f_t + f f_y. \tag{23}$$

Plugging everything into the local truncation error

$$y(t_{n+1}) - y(t_n) - hb_1 f(t_n, y(t_n)) - hb_2 f(t_n + c_2 h, y(t_n) + ha_{2,1} f(t_n + c_1 h, y(t_n))) = \tag{24}$$

$$= fh + \frac{h^2}{2}(f_t + f_y f) - hb_1 f - hb_2 \left(f + c_2 h f_t + h f_y a_{2,1} f\right) + O(h^3) \tag{25}$$

$$= hf\left(1 - b_1 - b_2\right) + \frac{h^2}{2}\left(f_t - 2b_2 c_2 f_t + f_y f - 2a_{2,1} b_2 f_y f\right) + O(h^3). \tag{26}$$

Therefore, in order to have local truncation error of $O(h^3)$, it is necessary and sufficient to have

$$b_1 + b_2 = 1 \tag{27}$$

$$b_2 c_2 = \frac{1}{2} \tag{28}$$

$$a_{2,1} b_2 = \frac{1}{2}. \tag{29}$$

This does not quite uniquely determine 2nd order 2-stage ERK methods. Two examples are the explicit trapezoidal rule already suggested in class:

$$\xi_1 = y_n \tag{30}$$

$$\xi_2 = y_n + hf(t_n, \xi_1) \tag{31}$$

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, \xi_1) + \frac{h}{2} f(t_n + h, \xi_2). \tag{32}$$

This method corresponds to the choices:

$$b_1 = b_2 = 1/2 \tag{33}$$

$$c_2 = 1 \tag{34}$$

$$a_{2,1} = 1. \tag{35}$$

Another set of parameters which satisfy all the necessary conditions is:

$$b_2 = 1 \tag{36}$$

$$c_2 = a_{2,1} = \frac{1}{2}. \tag{37}$$

Write out this scheme and convince you it is an analogue of the midpoint rule in numerical quadrature. The above calculation hints at the heart of RK methods: the accuracy of the sub-steps does not need to be quite as high as the desired error of the RK method, because these corrections come with additional powers of h. However, it is also clear that building higher order methods will require increasingly accurate approximations for building ξ_j's. Figuring out exactly how to balance this necessity to derive higher order methods is difficult. Getting 3rd or 4th order are still "tractable" by direct Taylor expansions like above, though unpleasant, it certainly something that you could do given enough time and effort. Moreover, there are systematic algorithms for deriving higher order methods (though much more than 4th order is rarely necessary). The most important schemes are

the classical ERK-3 method which has global error of 3rd order:

$$c = \begin{pmatrix} 0 \\ \frac{1}{2} \\ 1 \end{pmatrix}, \tag{38}$$

$$b = \begin{pmatrix} \frac{1}{6} \\ \frac{2}{3} \\ \frac{1}{6} \end{pmatrix}, \tag{39}$$

$$A = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ -1 & 2 & 0 \end{pmatrix}, \tag{40}$$

and the classical ERK-4:

$$c = \begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \\ 1 \end{pmatrix}, \tag{41}$$

$$b = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{6} \end{pmatrix}, \tag{42}$$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{43}$$

which has global error of 4th order (despite being only slightly more difficult to implement than forward Euler!). The ERK-3 and ERK-4 are both based on Simpson's rule, which recall is a 4-th order quadrature method. The ERK-3 method is limited only to 3rd order due to the fact that the approximation of the mid-point being not accurate enough. This short-coming is corrected in ERK-4 which builds a better approximation for the mid-point (the midpoint approximation in ERK-4 is the average of stages ξ_2 and ξ_3). All explicit RK methods are limited such that the order p satisfies $p \leq \nu$, so that a 3 stage scheme can be at most 3rd order. It turns out that this problem multiplies for higher order explicit methods: for all $p > 4$, one needs more than p stages to obtain p-order global accuracy (another reason that it is annoying to go to higher order than 4).

1.1 Implicit RK (IRK) methods

The implicit trapezoidal method is an example of an implicit RK scheme. Indeed, we can write it in the form:

$$\xi_1 = y_n \tag{44}$$

$$\xi_2 = y_n + \frac{h}{2} f(t_n, \xi_1) + \frac{h}{2} f(t_n + h, \xi_2) \tag{45}$$

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, \xi_1) + \frac{h}{2} f(t_n + h, \xi_2). \tag{46}$$

In fact, if one uses Gaussian quadrature, then we get a class of ν-stage, order 2ν methods, known as Gauss-Legendre IRK methods. The first method of this type is the *implicit midpoint rule*:

$$\xi_1 = y_n + \frac{h}{2} f(t_n + h/2, \xi_1) \tag{47}$$

$$y_{n+1} = y_n + h f(t_n + h/2, \xi_1), \tag{48}$$

which is second order accurate and comparable in cost to the implicit trapezoidal. Using two Gaussian quadrature points yields a *4-th* order accurate method:

$$\xi_1 = y_n + \frac{h}{4} f(t_n + c_1 h, \xi_1) + h\left(\frac{1}{4} - \frac{\sqrt{3}}{6}\right) f(t_n + c_2 h, \xi_2) \tag{49}$$

$$\xi_2 = y_n + h\left(\frac{1}{4} + \frac{\sqrt{3}}{6}\right) f(t_n + c_1 h, \xi_1) + \frac{h}{4} f(t_n + c_2 h, \xi_2 \tag{50}$$

$$y_{n+1} = y_n + \frac{h}{2} f(t_n + c_1 h, \xi_1) + \frac{h}{2} f(t_n + c_2 h, \xi_2), \tag{51}$$

with

$$c_1 = h\left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right) \tag{52}$$

$$c_2 = h\left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right). \tag{53}$$

This scheme is 4th order accurate despite only being 2-stage (and is the only RK method with this property). However, the implementation is noticeably more difficult than implicit trapezoidal or implicit midpoint: the unknowns ξ_1 and ξ_2 must be solved for *simultaneously*, which means if $y(t) \in \mathbb{R}^d$, every time-step we need to solve a nonlinear system of $2d$ variables (in particular, for d large, this is *much* harder than solving 2 nonlinear systems in d variables). Hence, this 4-th order method will be much harder than ERK-4 (we will see below that there are still contexts in which the Gauss-Legendre method will be preferrable to the ERK-4). There is another class of IRK methods called *diagonally implicit RK* methods (DIRK). The DIRK methods are the versions of IRK which come in the form:

$$\xi_1 = y_n + h a_{1,1} f(t_n + c_1 h, \xi_1) \tag{54}$$

$$\vdots = \vdots \tag{55}$$

$$\xi_i = y_n + \sum_{j=1}^{i} h a_{i,j} f(t_n + c_j h, \xi_j) \tag{56}$$

$$\vdots = \vdots \tag{57}$$

$$\xi_\nu = y_n + \sum_{j=1}^{\nu} h a_{\nu,j} f(t_n + c_j h, \xi_j) \tag{58}$$

$$y_{n+1} = y_n + h \sum_{j=1}^{\nu} b_j f(t_n + c_j h, \xi_j). \tag{59}$$

That is, stage ξ_i is written in terms of stages ξ_j, $1 \leq j \leq i$. Hence, each stage requires solving a nonlinear system, but you don't need to solve for all stages at the same time like Gauss-Legendre methods. These methods will be easier and potentially more efficient than Gauss-Legendre for problems for which d is large, even though more stages will be required to obtain the same order of accuracy (for example, the 2-stage DIRK methods are 3rd order accurate, not 4th order accurate). For more information on DIRK schemes, see Alexander, R. Diagonaly implicit Runge-Kutta methods for stiff O.D.E.'s, SIAM J on Numerical Analysis, Vol 14, No 6 (1977).

2 Linear systems of ODEs

In ODEs you should have learned the basic methods for solving linear systems of ODEs. That is, let $A \in \mathbb{R}^{n \times n}$ and consider the ODE:

$$\frac{d}{dt}y(t) = Ay(t) \tag{60}$$

$$y(0) = y_0. \tag{61}$$

Suppose that A is diagonalizable, that is, there exists an invertible matrix $P \in \mathbb{C}^{n \times n}$ and a set of *eigenvalues* $\lambda_j \in \mathbb{C}$ with $1 \leq j \leq n$ such that

$$A = P \begin{pmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & \cdots & 0 & \lambda_n \end{pmatrix} P^{-1}. \tag{62}$$

Note that this is the same as saying that there exists a set of linearly independent $\{p_1, ..., p_n\} \subset \mathbb{C}^n$ such that $Ap_j = \lambda_j p_j$ so that the p_j are the columns of P. Unfortunately, we really do need to allow λ, P to be complex-valued (there are lots of matrices which are diagonalizable, but not diagonalizable over $\mathbb{R}^{n \times n}$). Now, set

$$P^{-1}y(t) = x(t). \tag{63}$$

Hence, the x now solves the ODE

$$\frac{d}{dt}x(t) = P^{-1}APx(t) \tag{64}$$

$$x(0) = P^{-1}y_0. \tag{65}$$

We see that $P^{-1}AP$ is actually the diagonal matrix with $(\lambda_1, ..., \lambda_n)$ along the diagonal. Hence, every component $1 \leq j \leq n$ satisfies

$$\frac{d}{dt}x_j(t) = \lambda_j x_j(t), \tag{66}$$

so we can solve

$$x_j(t) = x_j(0)e^{\lambda_j t}. \tag{67}$$

Then we recover $y(t) = Px(t)$. What the matrix P is doing is changing variables into n decoupled unknowns for which we can solve the problem explicitly, and then at the end we transform the problem back to the original unknowns. It is quite slick, but it is usually faster and easier numerically to solve the original linear system $y' = Ay$ using a numerical ODE solver than it is to diagonalize the matrix and use the explicit solution (at least if $n \geq 100$ or so).

3 Stiffness and A-stability

Consider the very simple scalar linear equation:

$$\frac{d}{dt}y(t) = -100y(t) \tag{68}$$

$$y(0) = 1. \tag{69}$$

The exact solution is $y(t) = e^{-100t}$. Consider now solving this with Euler's method:

$$y_{n+1} = y_n - 100hy_n = (1 - 100h)y_n, \tag{70}$$

and hence Euler's method gives the solution

$$y_n = (1 - 100h)^n. \tag{71}$$

Hence, we see a fundamental problem: if $1 - 100h < -1$, that is, if $h > 1/50$, then the numerical solution is actually *growing* (and oscillating!). This looks absolutely nothing like the rather boring exact solution, which is, for intents and purposes, essentially zero for $t > 1/50$. This is an example of what is called a *numerical instability*: an "instability" observed numerically which is not actually there physically. Notice that there is nothing wrong with Euler's method per se: the method really is convergent, so when you make h small enough, the issue goes away. The problem is appearing for h which is small, but not quite small enough to "resolve" the real dynamics. This problem is called *stiffness*, and problems which have this kind of character are called *stiff* (we'll see below for a technical definition). Stiffness is the fundamental reason why anyone cares about implicit methods. Indeed, consider now implicit trapezoidal:

$$y_{n+1} = y_n - 100\frac{h}{2}y_n - 100\frac{h}{2}y_{n+1}, \tag{72}$$

and hence

$$y_{n+1} = \left(\frac{1 - 50h}{1 + 50h}\right) y_n, \tag{73}$$

so trapezoidal gives

$$y_n = \left(\frac{1 - 50h}{1 + 50h}\right)^n. \tag{74}$$

Since for all $h > 0$,

$$\left|\frac{1 - 50h}{1 + 50h}\right| < 1, \tag{75}$$

the implicit trapezoidal method does *not* suffer from the same problem as forward Euler. For all $h > 0$ you at least get a decaying solution. As a general rule, implicit methods handle stiffness better than explicit methods. In fact, no explicit method can handle stiffness very well – this is the main advantage of implicit trapezoidal method over the much simpler and cheaper explicit trapezoidal method. A problem may be so stiff that it is actually more efficient to use the slow and hard to implement implicit methods, which allow a larger time-step, than it is to use a simpler explicit methods (which would require prohibitively small time-steps).

3.1 A-Stability

For some $\lambda \in \mathbb{C}$,

$$\frac{d}{dt}y(t) = \lambda y(t) \tag{76}$$

$$y(0) = 1. \tag{77}$$

Note the exact solution is, denoting $\lambda = \lambda_r + i\lambda_i$:

$$y(t) = e^{\lambda t}y_0 = e^{\lambda_r t}(\cos\lambda_i t + i\sin\lambda_i t)y_0. \tag{78}$$

Let us assume that $\lambda_r < 0$, which implies that the solution is decaying exponentially fast. Suppose we are going to use Euler's method to solve this:

$$y_{n+1} = (1 + \lambda h)y_n, \tag{79}$$

and hence

$$y_n = (1 + \lambda h)^n y_0. \tag{80}$$

We see that if $|1 + \lambda h| = \sqrt{(1 + h\lambda_r)^2 + (h\lambda_i)^2} > 1$, then y_n is actually *growing*, regardless of the sign of λ_r! The *linear stability domain* of a numerical method is the set $\lambda h \in \mathcal{D} \subset \mathbb{C}$ with $\lambda_r < 0$ such that the numerically computed solution to (76) is decaying. Hence, the linear stability domain for Euler is:

$$\mathcal{D}_E = \{z \in \mathbb{C} : |1 + z| < 1\}, \tag{81}$$

whereas the linear stability domain for the implicit trapezoidal rule is

$$\mathcal{D}_{IT} = \{z \in \mathbb{C} : \mathrm{Re}\, z < 0\}. \tag{82}$$

A numerical method with a linear stability domain $\mathcal{D} = \{z \in \mathbb{C} : \mathrm{Re}\, z < 0\}$ is callled *A-stable*. It is impossible for an explicit method to be A-stable, whereas all of the implicit RK methods we have discussed are A-stable. It is worth mentioning that not every useful implicit method in existence is A-stable, but generally, implicit methods have larger linear stability domains than their explicit counter-parts. The exact shape of the linear stability domain can be important for some applications. Computing the domains for higher order methods can be rather difficult without the aid of a computer but there are some tricks for doing so (developed because it is a very important property to understand).

3.2 A-stability of Runge-Kutta methods

Recall, the RK methods are schemes of the general type:

$$\xi_1 = y_n + h\sum_{j=1}^{\nu} a_{1,j} f(t_n + c_j h, \xi_j) \tag{83}$$

$$\vdots = \vdots \tag{84}$$

$$\xi_\nu = y_n + h\sum_{j=1}^{\nu} a_{\nu,j} f(t_n + c_j h, \xi_j) \tag{85}$$

$$y_{n+1} = y_n + h\sum_{j=1}^{\nu} b_j f(t_n + c_j h, \xi_j). \tag{86}$$

In order to find the linear stability domain, we want to apply this method to the simple scalar problem (76). Hence,

$$\xi_1 = y_n + h\lambda \sum_{j=1}^{\nu} a_{1,j}\xi_j \tag{87}$$

$$\vdots = \vdots \tag{88}$$

$$\xi_\nu = y_n + h\lambda \sum_{j=1}^{\nu} a_{\nu,j}\xi_j \tag{89}$$

$$y_{n+1} = y_n + h\lambda \sum_{j=1}^{\nu} b_j\xi_j. \tag{90}$$

The ξ_j's form a linear system:

$$\begin{pmatrix} \xi_1 \\ \vdots \\ \xi_\nu \end{pmatrix} = \begin{pmatrix} y_n \\ \vdots \\ y_n \end{pmatrix} + h\lambda A \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_\nu \end{pmatrix} \tag{91}$$

At least for h small, we can always solve this system to get

$$\begin{pmatrix} \xi_1 \\ \vdots \\ \xi_\nu \end{pmatrix} = (I - h\lambda A)^{-1} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} y_n. \tag{92}$$

Plugging this into the definition of y_{n+1}, we get

$$y_{n+1} = y_n + \lambda h b^T (I - h\lambda A)^{-1} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} y_n \tag{93}$$

$$= \left(1 + \lambda h b^T (I - h\lambda A)^{-1} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right) y_n. \tag{94}$$

Denote the complex function

$$r(z) = \left(1 + z b^T (I - zA)^{-1} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right). \tag{95}$$

We can then solve (94) for y_n and we get

$$y_n = (r(\lambda h))^n y_0. \tag{96}$$

Hence, the linear stability domain is the set of $h\lambda$ such that $|r(\lambda h)| < 1$:

$$D = \{\lambda h \in \mathbb{C} : \operatorname{Re}\lambda < 0, |r(\lambda h)| < 1\}. \tag{97}$$

Hence, the difficult part is now analyzing $r(z)$ (which indeed is quite difficult to do by hand in general). Let us deduce a bit more information however.

Lemma 1. *For all RK schemes, $r(z)$ is a rational function. That is $r(z) = \frac{p_1(z)}{p_2(z)}$ where p_j are ν degree polynomials with real coefficients. Moreover, if the RK method is explicit, then $r(z) = p_1(z)$ for p_1 a ν degree polynomial with real coefficients.*

Proof. This comes from recalling the theoretical expresssion for the inverse of a matrix (an expression which is completely impractical for real matrices, but is sometimes useful for theory, as it is here):

$$(I - zA)^{-1} = \frac{1}{\det(I - zA)} \text{adj}(I - zA), \tag{98}$$

where $\text{adj}(I - zA)$ is the adjunct matrix, that is the matrix whose (i, j) element is the determinant of the $\nu - 1 \times \nu - 1$ matrix arising from deleting the (i, j)-th row of $I - zA$. This formula is complicated enough such that you will never use it except for 2×2 systems, however, since $\det(I - zA)$ is a polynomial in z, and every entry in $\text{adj}(I - zA)$ is a polynomial in z, it tells us that indeed $r(z)$ is a rational function as claimed.

If the RK method is explicit, then A is strictly lower triangular, and hence $I - zA$ is a lower triangular matrix with 1's along the diagonal. Hence, $\det(I - zA) = 1$. Therefore,

$$(I - zA)^{-1} = \text{adj}(I - zA), \tag{99}$$

and hence every entry in $(I - zA)^{-1}$ is just a polynomial, not a rational function. Hence, $r(z)$ in this case is just a polynomial. □

The next theorem shows that no explicit RK scheme can be A-stable.

Theorem 1. *No explicit RK scheme is A-stable.*

Proof. Observe from the definition that $r(0) = 1$ and that $r(z)$ is a non-constant polynomial for all explicit RK methods which converge. The linear stability domain is the set of z such that $|r(z)| < 1$ with $\text{Re}\, z < 0$. However, the only polynomials which are bounded on the entire complex half-space are the constants, and so we would need $r(z) = 1$, which is not possible. □

Remark 1. If we think about the above proof more, it also tells us several more specific things, the most important for practical relevance is the fact that for all explicit RK methods, there exists an $M > 0$ such that for $\lambda \in \mathbb{R}$, the following set $\{\lambda h \in Real : \lambda h < -M\}$ is not in the linear stability domain. This is important because problems of the type $y' = -Ay$ where A is SPD are very common problem in practice (and often times, these problems are often very stiff, e.g. there are eigenvalues of A which are very large in magnitude and other eigenvalues which are not large).

By using some additional complex analysis, we can analyze the stability of implicit RK methods, but it gets a bit messy. Let me just state the following theorem:

Theorem 2. *All Gauss-Legendre methods are A-stable.*

This tells us at the only real disadvantage of Gauss-Legendre methods is that they are difficult to implement for the programmer and, much worse, slow and difficult for the computer to execute. Hence, despite being optimal in terms of stability and accuracy, they are not very commonly used.

3.3 A minor aside about A-stability and nonlinear problems

First of all, many important physical problems are of the general type (for example, wave-like systems and quantum phenomena are often in this form, or are equivalent to this form),

$$y'(t) = iAy(t), \tag{100}$$

where $A \in \mathbb{R}^{n\times n}$ is SPD and $i = \sqrt{-1}$. This complex valued linear system has purely imaginary eigenvalues (e.g. the real part of all the eigenvalues are zero). Our notion of linear stability domain does not really consider the behavior of ODE methods for these problems, and some methods perform better than others (for example ERK-4 is significantly better than Euler's method in terms of stability).

For a nonlinear problem

$$\frac{d}{dt}y(t) = f(y(t)) \tag{101}$$

$$y(0) = y_0, \tag{102}$$

it is not clear how to relate A-stability to anything in particular about the performance of the ODE solver on this problem. We'll only discuss this heuristically. Let $\tilde{y}$ solve

$$\frac{d}{dt}\tilde{y}(t) = f(\tilde{y}(t)) \tag{103}$$

$$\tilde{y}(0) = \tilde{y}_0, \tag{104}$$

then, for $\|y - \tilde{y}\|$ initial small,

$$\frac{d}{dt}\left(y(t) - \tilde{y}(t)\right) = \partial_y f(y(t))(y(t) - \tilde{y}) + O(\|y(t) - \tilde{y}(t)\|^2), \tag{105}$$

where $\partial_y f$ is the Jacobian matrix

$$(\partial_y f)_{ij} = \frac{\partial f_i}{\partial y_j}. \tag{106}$$

Because the computer will always introduce some errors, this suggests that we won't be able to stably compute y unless $\lambda_j h$ is in the linear stability domain for all λ_j eigenevalues of $\partial_y f(y(t))$ (we haven't proved anything, its just a heuristic). For many interesting nonlinear problems, you will sometimes have $\operatorname{Re}\lambda_j > 0$ and then it isn't clear that one can compute anything accurately at all. Indeed, one cannot in general – positive eigenvalues of $\partial_y f$ are often indicative of chaotic behavior such as seen in e.g. weather models etc (hence, why you can't accurately predict the weather far in advance). Suffice it to say that, as a general rule, A-stability is just one aspect of stability: for nonlinear problems, one might be interested in other notions, and for problems with imaginary eigenvalues, one needs to consider more carefully.

3.4 Time-scales and why its important to know theory in the age of computers

Let us return to the simple ODE we started with:

$$\frac{d}{dt}y(t) = -100y(t) \tag{107}$$

$$y(0) = 1. \tag{108}$$

When we solved this, we said it was stiff because in order to integrate until time 1, we would have to use lots and lots of time-steps. That's true, but...the solution is zero in machine precision after about $t > 1/50$ or so. Why would you actually want to integrate this to time 1? The only interesting time-scale is more like $t \leq 1/100$. Indeed, the physics students among you should point out: *we wrote the ODE in the wrong units* – we are using the wrong units of time to study the dynamics. We should study the unkown $x(t) = y(t/100)$ (which is y slowed down by 100 units of time) and study the perfectly easy to solve

$$\frac{d}{dt}x(t) = -x(t) \tag{109}$$

$$x(0) = 1 \tag{110}$$

until time $t = 1$ or 2 or so. No stiffness at all! That is true, in some sense, the example that motivated the entire above discussion is a bit facetious. The real problem comes when you study a system like:

$$\frac{d}{dt}\begin{pmatrix} u(t) \\ v(t) \end{pmatrix} = \frac{1}{2}\begin{pmatrix} -1001 & 999 \\ 999 & -1001 \end{pmatrix}\begin{pmatrix} u(t) \\ v(t) \end{pmatrix} = A\begin{pmatrix} u(t) \\ v(t) \end{pmatrix}. \tag{111}$$

This matrix has the orthogonal diagonalization:

$$A = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} -1 & 0 \\ 0 & -1000 \end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \tag{112}$$

Hence, there is a change of variables so that the system becomes two decoupled variables but where one variable decays like e^{-t} and another decays like e^{-1000t}. For this problem, there are hence two widely separated time-scales. If you want to look at a time-scale over which both variables move, you have to solve till around $t = 1$, but then you will have a big stiffness problem (Euler will need time-steps like 1/500). In order to solve this sytem till time $t = 1$, it might make much more sense to solve using an implicit method. For other problems, it might also be that you aren't quite sure what the precise time-scales are, because you haven't factorized A, or even more likely, the problem is nonlinear and hence the exact stiffness of your problem is going to be hard to quantify exactly.

As a last comment, let me mention that understanding numerical instability, especially when one goes into the murkier waters of nonlinear ODEs and nonlinear PDEs, can be subtle and tricky. It has happened multiple times that scientific groups have excitedly announced the prediction of dazzling, potentially paradigm shifting physics, only to be crushed later with the knowledge that all their predictions were trash because their code was unstable and was producing non-physical behavior. Such groups are often suffering from a lack of understanding of theory, which would've helped them to identify what kind of behavior they can and cannot expect from their models. This is one of several reasons why a serious numerical scientist needs to have a solid grounding in the theoretical understanding of differential equations as well as numerical analysis.