Chapter 1

NUMERICAL METHODS FOR ODE INITIAL VALUE PROBLEMS

A scalar first order *ordinary differential equation* (ODE) takes the form

$$y' = f(t, y) \ , \tag{1.1}$$

where $y = y(t)$ is a function tobe determined. It can be illustrated by a vector field, as shown in Figure 1.1 for the case of

$$y' = t^2 + y^2 \ . \tag{1.2}$$

At each location in a (t, y)-plane, equation (1.2) tells in which direction a solution curve should go. The analytic solution is a continuous function that everywhere follows the required directions. Even for very simple-looking ODEs, it may be impossible to find closed-form expressions for solutions. In the case of (1.2), together with the *initial condition* (IC) $y(0) = 0$, it happens to be possible:

$$y(t) = - \frac{t \ \left(J_{-3/4}(\frac{1}{2}t^2) - Y_{-3/4}(\frac{1}{2}t^2) \right)}{J_{1/4}(\frac{1}{2}t^2) - Y_{1/4}(\frac{1}{2}t^2)} \ ,$$

but its complexity makes it of limited value (J and Y denote here Bessel functions of fractional orders; this solution is shown as the smooth curve starting at the origin in Figure 1.1).

In complete contrast to analytical methods, numerical techniques for ODEs do not get any more complicated if the ODE is nonlinear rather than linear, or if it is

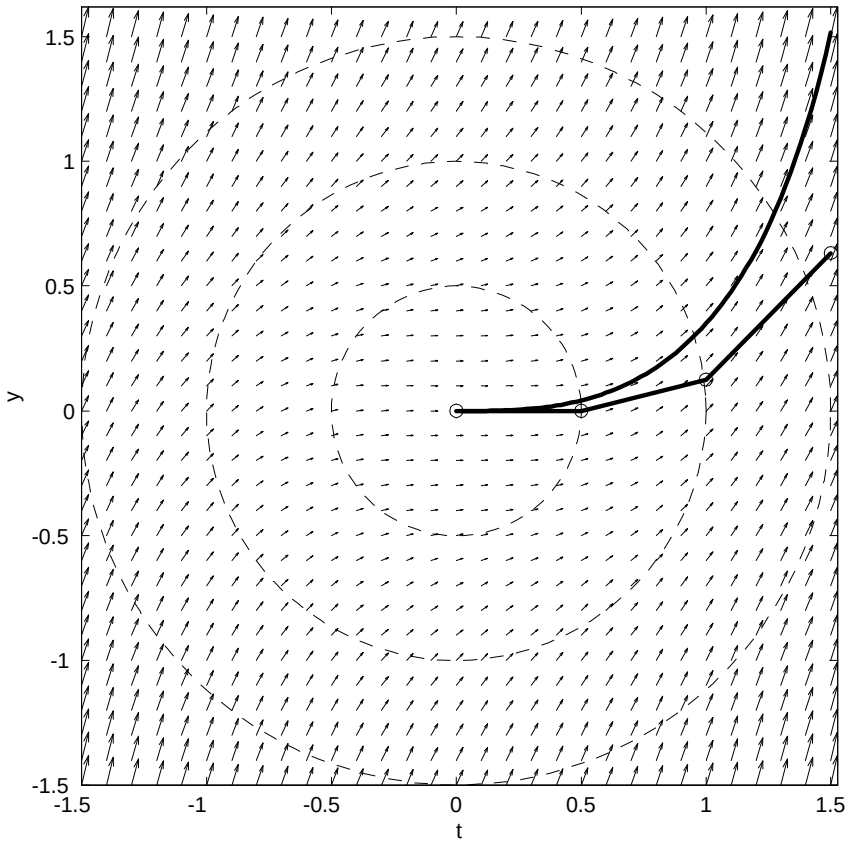


Figure 1.1: Illustration of solutions to the ODE (1.2). The arrows show the direction field. The smooth and the piecewise linear curves show the analytic solution and the Euler approximation (with step $k = 0.5$) in the case of the IC $y(0) = 0$. The dashed circles are examples of *isoclines* - curves along which the slopes are constant.

generalized from one scalar ODE to a system of many coupled ODEs, for ex.

$$\begin{cases} y_1' &= f_1(t, y_1, y_2, \ldots, y_n) \\ y_2' &= f_2(t, y_1, y_2, \ldots, y_n) \\ \vdots & \\ y_n' &= f_n(t, y_1, y_2, \ldots, y_n) \end{cases} \quad \text{with IC} \quad \begin{cases} y_1(0) &= a_1 \\ y_2(0) &= a_2 \\ \vdots & \\ y_n(0) &= a_n \end{cases} . \tag{1.3}$$

The ease with which we will soon see that first order systems can be handled numerically makes it attractive to turn higher order equations into coupled systems of first order ones. For example, instead of

$$y''' = f\ (t, y, y', y'') \ \text{ with IC } \ y(0) = a_0,\ y'(0) = a_1,\ y''(0) = a_2,$$

we would introduce extra dependent variables through $y_0(t) = y(t)$, $y_1(t) = y'(t)$, $y_2(t) = y''(t)$ and then solve

$$\begin{cases} y_0' &= y_1 \\ y_1' &= y_2 \\ y_2' &= f\ (t, y_0, y_1, y_2) \end{cases} \quad \text{with IC} \quad \begin{cases} y_0(0) &= a_0 \\ y_1(0) &= a_1 \\ y_2(0) &= a_2 \end{cases} .$$

In Section 1.1, we introduce the simplest possible numerical ODE scheme, *Forward Euler* (FE). In its basic form, it is very inaccurate (as seen already in Figure 1.1), and it is rarely used. We give in Section 1.2 numerous examples of linear multistep (LM) methods, which represent one direction for generalizing FE towards higher accuracy and efficiency. It will transpire that some of these schemes that are described are very efficient while others, appearing equally plausible, in fact are entirely useless. The analysis that is needed to address this involves four key concepts: accuracy, consistency, stability and stability domain, and is given in Section 1.3. With the help of this additional background, we discuss in Section 1.4 backward differentiation methods, in Section 1.5 predictor-corrector methods, in Section 1.6 Runge-Kutta methods and in Section 1.7 Taylor series methods. In the concluding Section 1.8, we describe stiff ODEs and how stability domains give insights into how to choose between different ODE methods.

1.1 Forward Euler (FE) scheme

The numerical solution produced by this scheme is made up of piecewise straight line segments, each one with the slope of the direction field at its start location. A step from time t to time $t + k$ amounts to using the first two terms of a Taylor

expansion of the solution at time t:

$$\underbrace{y(t+k) = y(t) + k\, y'(t)}_{\text{Use for Forward Euler}} \quad \underbrace{+ \frac{k^2}{2} y''(t) + \ldots}_{\text{Local error } O(k^2)} \tag{1.4}$$

Substituting the ODE (1.1) for y', we get the Forward Euler method

$$y(t+k) = y(t) + f(t,y)\ . \tag{1.5}$$

A very convenient way to graphically illustrate the stencil of this scheme is as follows:

$$\begin{array}{lccccl} & y & f & & & \\ \text{time level} & \Downarrow & \Downarrow & & & \\ \text{new: } t+k & \boxdot & & & \boxdot / \boxplus & \text{unknown/known values for } y \\ - \quad + \quad - \quad - & & & & \odot / \oplus & \text{unknown/known values for } f \\ \text{old: } t & \boxplus & \oplus & & & \end{array} \tag{1.6}$$

or, more compactly, $\begin{bmatrix} \boxdot & \\ \boxplus & \oplus \end{bmatrix}$.

Each time step, the local error is of size $O(k^2)$. In order to advance over a time interval of fixed duration, we need to take $O(1/k)$ time steps. It is therefore not surprising that the total error becomes of the size $O(k^2) \cdot O(1/k) = O(k^1)$ as $k \to 0$. The FE scheme is therefore known as a *first order* scheme.

1.2 Examples of linear multistep (LM) methods

The general idea behind LM methods is to extend the computational stencil shown for FE in (1.6) backwards to still earlier time levels. In the examples below, we do this in various ways. In all the cases, once the shape of the stencil is decided on, we find the actual coefficients to use (giving the highest local accuracy) most easily by means of the two-line Padé-based Mathematica algorithm described in Section ??.??:

```
t = Pade[x^s*Log[x]^m,{x,1,n,d}];
{CoefficientList[Numerator[t],x],CoefficientList[Denominator[t],x]}
```

The numbers m, s, n and d that are needed to get the desired coefficients are shown below for each method. The parameter m is here always one since we are approximating a first derivative; the other three numbers describe the 'shape' of the stencil, as is described in Section ??.??. and is also illustrated in Figure 1.2.

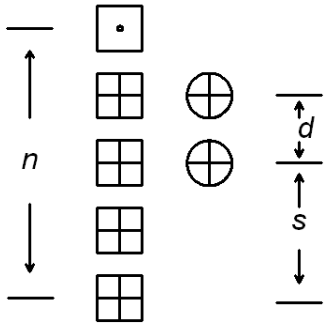
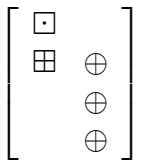


Figure 1.2: Illustration of how the shape of a LM stencil is described in terms of the parameters s, n and d. If the right column of entries descends further down than the left column, s will be negative. While n and d need to be integers, we will in a later context consider time-staggared stencils for which s is a half-integer.

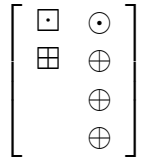
Example 1:



Scheme: $y(t+k) = y(t) + k\ \left[\frac{23}{12}y'(t) - \frac{4}{3}y'(t-k) + \frac{5}{12}y'(t-2k)\right]$
Find coeff: $m = 1,\ s = -2,\ n = 1,\ d = 2$
Accuracy: 3^{rd} order

In this and the following examples, $y'(\cdot)$ should be replaced by $f\ (\cdot, y(\cdot))$, i.e. the RHS of the ODE at time level "$\cdot$". In this scheme, the stencil is extended backwards two additional steps in the right column compared to the FE scheme. Schemes of this general type (extending backwards in the right column only) are known as Adams-Bashforth schemes. Every step backwards relative to the FE scheme increases the order by one. Due to its third order of accuracy, we denote the present scheme as AB3. □

Example 2:



Scheme: $y(t+k) = y(t) +$
$+k\ \left[\frac{3}{8}y'(t+k) + \frac{19}{24}y'(t) - \frac{5}{24}y'(t-k) + \frac{1}{24}y'(t-2k)\right]$
Find coeff: $m = 1,\ s = -2,\ n = 1,\ d = 3$
Accuracy: 4^{th} order

This is an example of an *implicit* scheme (as opposed to the AB-type schemes, which are *explicit*). At each time step, both $y(t+k)$ and $y'(t+k) = f\ (t+k, y(t+k))$ are

unknown. However, the scheme provides a relation between these two quantities, from which we can solve for $y(t+k)$ (maybe requiring Newton's method in case $f(t,y)$ is a nonlinear function of y). In the next Section 1.3, we will see that, depending on the ODE, this inconvenience may be well worth it. Schemes of this type (two levels for y, while extending different distances backwards for y') are known as Adams-Moulton schemes. If the right column contains only the entry $\odot$, the resulting first order scheme AM1 is also known as *Backward Euler* (BE). Again, the accuracy increases by one for every further entry in the right column. □

Example 3:

$$\begin{bmatrix} \boxdot & \\ \boxplus & \oplus \\ \boxplus & \oplus \\ \boxplus & \oplus \end{bmatrix}$$

Scheme: $y(t+k) = [-18y(t) + 9y(t-k) + 10y(t-2k)] - +k\ [9y'(t) + 18y'(t-k) + 3y'(t-2k)]$

Find coeff: $m = 1,\ s = 0,\ n = 3,\ d = 2$

Accuracy: 5^{th} order

Here, we are attempting (and we indeed succeed) to reach a very high formal order of accuracy by extending the stencil backwards in both of its columns. However, as we will see soon, the result is nevertheless a disaster. When letting $k \to 0$, the numerical solution will explode to infinity. Although the scheme is useless for all practical work, it will serve as a good motivation for why we need the convergence theory that is summarized in Section 1.3. □

Example 4:

$$\begin{bmatrix} \boxdot & \odot \\ \boxplus & \\ \boxplus & \\ \boxplus & \end{bmatrix}$$

Scheme: $y(t+k) = \left[\frac{18}{11}y(t) - \frac{18}{22}y(t-k) + \frac{1}{3}y(t-2k)\right] + - k\ \frac{6}{11}y'(t+k)$

Find coeff: $m = 1,\ s = 3,\ n = 3,\ d = 0$

Accuracy: 3^{rd} order

This scheme generalizes BE by extending two more levels backwards in the left column (and its accuracy is consequently two orders higher than that of BE). Schemes of this type are called Backward Differentiation (BD) methods, with this case being abbreviated as BD3. Again, as the analysis in Section 1.3 will show, this class of implicit schemes can be very attractive in certain situations. Although AB and AM schemes can be successfully brought to any order by just pushing increasingly far back in the second column, there will turn out to be a barrier against increasing BD schemes past order 6. If that is attempted, the same disaster will arise as in Example 3. □

The four examples above have illustrated three important families of LM methods: AB, AM, and BD. In the Section 1.5, we will see how the AB and AM classes can be combined in a particularly effective way.

We conclude this section by showing how one can determine the accuracy of a LM scheme if its coefficients are given (and not necessarily are the optimal ones with regard to accuracy). One example suffices to illustrate the procedure.

Example 5:

$$\begin{bmatrix} \boxdot & \\ \boxplus & \oplus \\ \boxplus & \oplus \end{bmatrix} \quad \begin{array}{ll} \text{Scheme:} & y(t+k) = [3y(t) - 2y(t-k)] + \\ & \quad + k\ \left[\frac{1}{2}y'(t) - \frac{3}{2}y'(t-k)\right] \\ \text{Accuracy:} & \text{To be determined} \end{array}$$

Since each LM formula is an example of a finite difference formula, relating y and y'-values, we can obtain the scheme's order simply by checking for how high powers of t the formula is exact. We test the error $E(t) = \{LHS\} - \{RHS\}$ of the present scheme as follows:

$$\begin{cases} y(t) = 1,\ y'(t) = 0 \\ E(t) = \{1\} - \{[3-2] + k[\frac{1}{2}0 - \frac{3}{2}0]\} = 0 \end{cases}$$

$$\begin{cases} y(t) = t,\ y'(t) = 1 \\ E(t) = \{t+k\} - \{[3t - 2(t-k)] + k[\frac{1}{2}1 - \frac{3}{2}1]\} = 0 \end{cases}$$

$$\begin{cases} y(t) = t^2,\ y'(t) = 2t \\ E(t) = \{(t+k)^2\} - \{[3t^2 - 2(t-k)^2] + k[\frac{1}{2}2t - \frac{3}{2}2(t-k)]\} = 0 \end{cases}$$

$$\begin{cases} y(t) = t^3,\ y'(t) = 3t^2 \\ E(t) = \{(t+k)^3\} - \{[3t^3 - 2(t-k)^3] + k[\frac{1}{2}3t^2 - \frac{3}{2}3(t-k)^2]\} = \frac{7}{2}k^3 \neq 0 \end{cases}$$

Given that the scheme fails to be exact for $y(t) = t^3$, it is a second order scheme. Since the result in this type of test will not depend on the value of k, one can simplify the algebra by first setting $k = 1$. □

1.3 Key numerical ODE concepts

1.3.1 Convergence

The tkey issues when using a numerical ODE solver are

1. Will the numerical solution converge to the true solution when $k \to 0$ and, if so,
2. How fast is the rate of convergence?
3. How small does the time step k need to be for the scheme to give a qualitatively correct answer?

The first of these questions is answered by the following theorem

Theorem: A numerical ODE scheme converges to the true solution of an ODE (linear or nonlinear) if and only if both consistency and stability hold.

Consistency is a very easy concept to check. A scheme is consistent if the *local* error, when the true ODE solution is substituted into the scheme, goes to zero when $k \to 0$. In particular, any ODE scheme that is of at least of first order accuracy is automatically consistent. No scheme that we would ever consider would fail this condition. Assuming from now on that consistency is satisfied, a scheme will therefore converge if and only if it is stable.

1.3.2 Stability

Another key result states that, in order to check stability, it suffices to test the scheme in case of the trivially simple ODE $y'(t) = 0$. To first illustrate what can go wrong, let us consider the scheme in Example 5. When applied to $y'(t) = 0$, it becomes

$$y(t+k) - 3y(t) + 2y(t-k) = 0 \, . \tag{1.7}$$

This is a 3-term recursion relation, which is most easily solved by forming its characteristic equation

$$r^2 - 3r + 2 = 0 \, ,$$

with roots $r_1 = 1$ and $r_2 = 2$. The general solution to (1.7) is therefore

$$y(t + n\ k) = c_1 \cdot 1^n + c_2 \cdot 2^n$$

where c_1 and c_2 are some constants ($c_1 = 2y(t) - y(t+k)$ and $c_2 = y(t+k) - y(t)$). For every time step forward, the term $c_2 \cdot 2^n$ doubles in size. The smaller k is made, the more steps need to be taken, and the numerical solution will thus diverge to infinity (given that machine rounding errors are always present and that we simplified the ODE by setting its RHS to zero, it will not happen in practice that c_2 is *exactly*

equal to zero). The cause of the divergence is that the characteristic equation has a root (here $r_2 = 2$) outside the unit circle $|r| = 1$. Omitting a few further details, this leads us to the *root condition*:

Theorem: A LM scheme is stable if and only if all the roots to its characteristic equation lie inside or on the periphery of the unit circle. If any root is on the periphery, it needs to be a simple root.

We can note that whenever a scheme is accurate to first order or better, it will admit the exact solution $y(t) \equiv 1$ to the ODE $y'(t) = 0$. From this follows that $r_1 = 1$ is always a root to the characteristic equation. All other roots are 'spurious', and what matters is that powers of these spurious roots must not grow. In the graphical stencils we use to illustrate LM schemes, stability (as opposed to accuracy) depends only on the entries in the left column.

1.3.3 Stability domain

The concept of *stability domain* (or *domain of absolute stability*) has very little to do with stability, as defined in the previous Section 1.3.2, so the traditional naming convention is unfortunate. To every ODE method corresponds a stability domain (in a complex plane, which we will soon define). For ODEs, this stability domain provides a guide to how small time step one need to take in order to get a 'reasonable' numerical solution. Stability domains provide often a good guide to what type of numerical method to choose for different ODEs, as discussed in Section 1.8. A related application area of stability domains will arise when we, in Chapter ??, apply ODE methods for numerically solving PDEs. In that context, the stability domain will tell whether schemes will converge or diverge when both time and space steps are refined - much like how stability (as tested by the root condition) is the key issue for convergence in the case of ODEs.

We will start the discussion of stability domains by an example which illustrates the need for a practical guide in choosing an appropriate time step:

Example 6: Determine how small we need to choose k in order to get a qualitatively reasonable approximation when using FE to solve the ODE

$$y'(t) = -10y(t) \tag{1.8}$$

The analytic solution $y(t) = e^{-10\,t}y(0)$ decays rapidly for increasing t. Suppose we try to solve (1.8) with FE using the time step $k = 1$. The scheme becomes

$y(t+1) = y(t) + 1 \cdot (-10y(t)) = -9y(t)$. Obviously, stepping forward in this way will lead to an oscillatory and rapidly divergent numerical solution, bearing no relation to the analytical one. Since FE is consistent and satisfies the root condition (with $r_1 = 1$ as the only root), we know that all will be well in the limit of $k \to 0$. However, in practice one needs to calculate with a finite value of k, so a natural question to ask is how small k need to be so that the numerical solution will not grow in time. For a general value of k, the FE scheme for (1.8) becomes $y(t+k) = y(t) + k \cdot (-10y(t))$ $= (1 - 10k)\, y(t)$, so the answer becomes $k \leq 1/5$. □

Linearization of a general ODE

The general ODE (1.1) can, locally around a point $(t_0, y_0) = (t_0, y(t_0))$, be linearized in both t and y:

$$y' \approx \underbrace{f(t_0, y_0) + f_t(t_0, y_0)(t - t_0)}_{g(t)} + \underbrace{f_y(t_0, y_0)}_{\lambda} \underbrace{(y - y_0)}_{v(t)}$$

i.e. $v'(t) \approx \lambda\, v(t) + g(t)$. In the present context, it turns out that we can further ignore $g(t)$, and only the value of $\lambda = f_y(t_0, y_0)$ will remain significant.

Stability domains for AB-type LM methods

In the case of FE (AB1), we can compute the *stability domain* as follows:

Example 7: Calculate the stability domain for FE.

For the ODE $y' = \lambda y$, FE becomes $y(t+k) = y(t) + \lambda k\, y(t)$. For all ODE methods (not just as here for FE), the variables λ and k will at this point only enter in the combination $\xi = \lambda k$. Thus $y(t+k) = (1+\xi)y(t)$, and $y(t+nk) = (1+\xi)^n y(t)$. The condition for no growing solutions therefore becomes $|1 + \xi| \leq 1$, i.e. a circle in a complex ξ-plane centered at $\xi = -1$ and with radius 1, shown by the label "1" in Figure 1.3 a. □

In the case of Example 6, we have $\lambda = -10$. The stability domain for FE tells that we have no-growth if $-2 \leq \xi \leq 0$. Given that $\xi = \lambda k$, we have thus again arrived at the previous condition $k \leq 1/5$.

Even for real-valued ODEs, we are also interested in complex values of ξ. If we for example want to solve a system of ODEs such as

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

(which can arise directly in applications or as the result of rewriting a higher order equation as a first order system; in this case $y'' + y = 0$), a simple change of variable decouples the two equations, and the eigenvalues to consider are $\lambda_{1,2} = \pm i$.

Example 8: Calculate the stability domain for AB2.

Applying $y(t+k) = y(t) + k\,[\frac{3}{2}y'(t) - \frac{1}{2}y'(t-k)]$ to $y' = \lambda y$ and substituting $\xi = \lambda k$ gives

$$y(t+k) - (1 + \tfrac{3}{2}\xi)\, y(t) + \tfrac{1}{2}\,\xi\, y(t-k) = 0. \tag{1.9}$$

This is a 3-term linear recursion relation, which is most easily solved via its characteristic equation

$$r^2 - (1 + \tfrac{3}{2}\xi)\, r + \tfrac{1}{2}\,\xi = 0. \tag{1.10}$$

We now want to plot, in the complex ξ-plane, the domain with the property that both roots r_1 and r_2 to (1.10) lie inside or on the unit circle - the condition for (1.9) not to have growing solutions. Trying to solve the quadratic equation (1.10) for r turns out to be a bad idea. A much better one is to realize that the boundary of the stability domain (in the ξ-plane) must be characterized by one root r to (1.10) being on the edge of the unit circle. Thus, we solve (1.10) for ξ

$$\xi = \frac{2r\,(r-1)}{3r-1} \tag{1.11}$$

and obtain the stability domain by letting $r = e^{i\theta}$ and plotting ξ as θ runs from 0 to 2π. In Matlab, the complete code for this becomes

```
r = exp(i*linspace(0,2*pi));
plot(2*r.*(r-1)./(3*r-1));
```

The resulting domain is labeled "2" Figure 1.3 a. □

The procedure in Example 8 for AB2 works for all LM methods. In case of AB3, we get in place of (1.11)

$$\xi = \frac{12r^2\,(r-1)}{23r^2 - 16r + 5}$$

and for AB4

$$\xi = \frac{24r^3\,(r-1)}{55r^3 - 59r^2 + 37r - 9}.$$

At this point (of AB4) enters a minor complication in that the curve that is traced out in the ξ-plane intersects itself. The curve marks where one root to the characteristic equation changes from $|r| < 1$ to $|r| > 1$. It can happen that another root

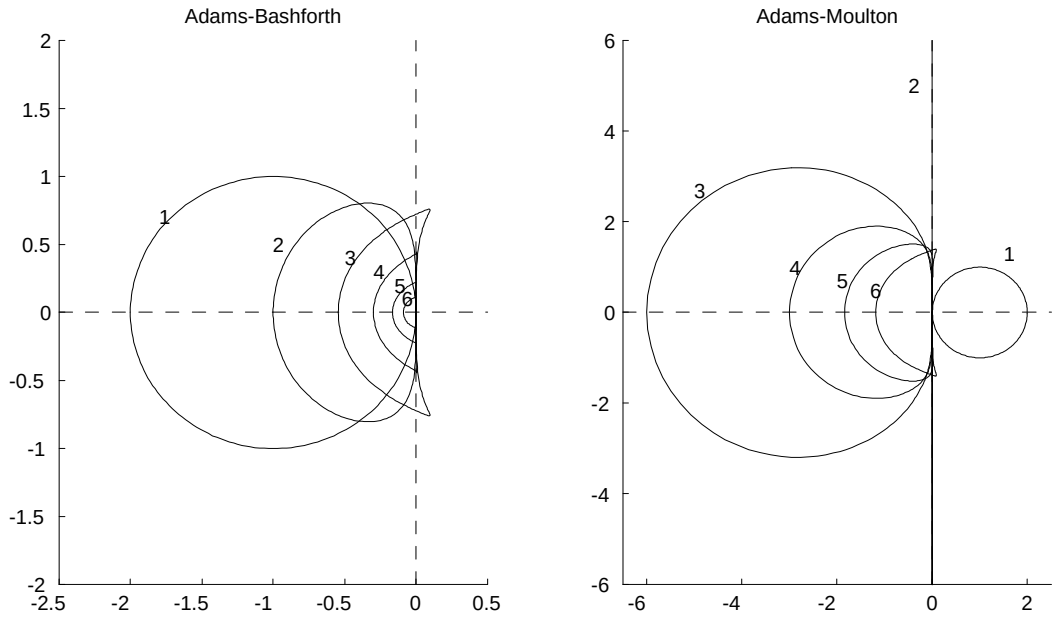


Figure 1.3: Stability domains for (a) AB and (b) AM methods of orders 1-6. The stability domains in all cases include the regions immediately to the left of the origin, i.e. for AB1 it is the domain outside the shown circle, and for AB2 the left halfplane. In all other cases, the regions are bounded. Note that the scale differs by a factor of three between the two pictures.

is larger than one, so that both sides of the traced curve are outside the stability domain. In such cases of intersecting loops, one simply chooses any ξ-value inside the loop, and solves for all the roots - that will tell if the loop is part of the stability region or not.

Figures 1.3 a and b summarize the stability domains for AB and AM methods of increasing orders. In both cases, the stability domains get smaller when the order increases. For each fixed order, the domain for the AM method is much larger than the one for the AB method. Dependent on the ODE that is solved, this advantage may outweigh the disadvantage of the AM methods being implicit.

In some important applications discussed in Chapter ??, all eigenvalues of the ODE will be purely imaginary, and it is then essential to know if a method's stability domain will include an interval of the imaginary axis around the origin. For AB methods, this turns out to happen if the order is {3,4}, {7,8}, {11,12}, etc. and the opposite for AM methods, i.e. there is some imaginary axis coverage for orders {1,2}, {5,6}, {9,10}, etc. [..].

1.3.4 The first Dahlquist barrier

The concepts of stability (root condition) and of stability domain are not entirely unrelated for LM methods, as shown by the Dahlquist stability barriers.

First barrier: The order of accuracy p of a stable s-step LM formula satisfies

$$p \leq \begin{cases} s+2 & \text{if } s \text{ is even} \\ s+1 & \text{if } s \text{ is odd} \\ s & \text{if the formula is explicit} \end{cases} .$$

For example, the AB3 scheme illustrated in Example 1 is explicit, and features $p = s = 3$ (note that s here is not the same s as in the Mathematica code to generate LM schemes). From this first barrier follows that any attempt to increase the order of accuracy by introducing further entries also down the left stencil column (as in Example 3) must cause the root condition to become violated. For explicit LM schemes, the AB class features as high orders of accuracy as is possible, given any value of s.

1.4 Backward differentiation (BD) methods

Example 4 in Section 1.2 provided an example of a BD scheme. The schemes BD1-BD6 differ only in the number of back levels that are employed for y. Figure 1.4 a show their stability domains and part b gives a more detailed picture near the origin. The stability domains are this time what falls outside (not inside) the closed curves. For all these schemes, the whole negative real axis is included in the domains. For BD schemes of orders $p = 7$ and above, not only does this property get lost; the schemes will also fail the root condition.

1.4.1 The second Dahlquist barrier

The BD methods feature stability domains of infinite extent. It is natural to want a method to include the whole negative half plane (Re $\xi < 0$) in the domain, since the analytical solutions to $y' = \lambda y$ decay when $\mathrm{Re}\,\lambda < 0$. Such a scheme is called A-stable.

Second barrier: The following two results hold:

1. An explicit LM method can never be A-stable, and

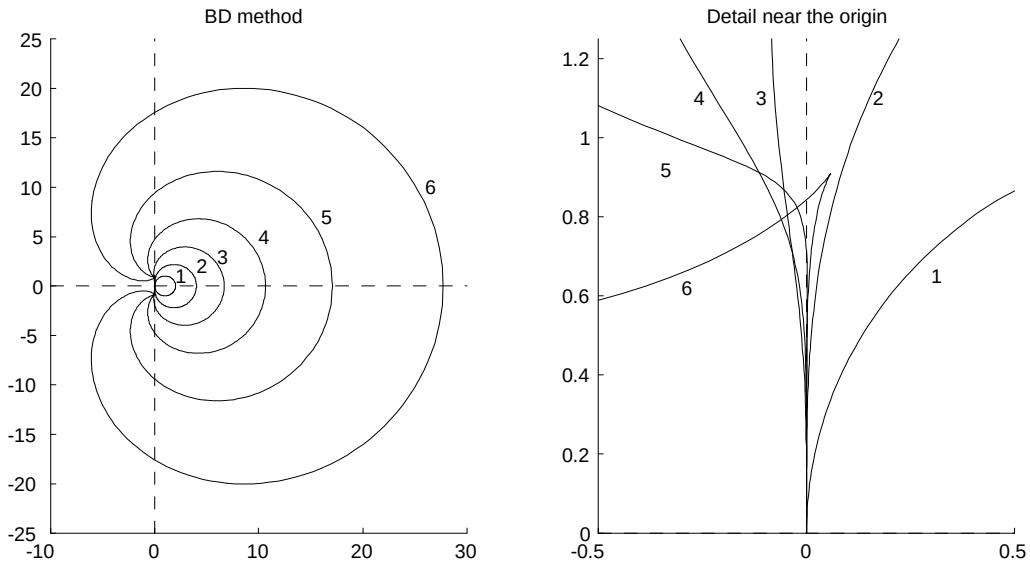


Figure 1.4: Stability domains of BD methods of orders 1-6. The domains are in all cases on the outside of the shown boundaries.(a) the complete boundaries (b) detailed boundary structures near the origin.

2. An implicit A-stable method can be at most second order accurate.

The AM1 (also described as BE and BD1), AM2 and BD2 methods are all examples of A-stable schemes. In most applications, A-stability is a more restrictive requirement than what is needed, cf. the discussion about stiff systems in Section 1.8.

1.5 Predictor-corrector methods

An interesting idea for combining the key strength of AB methods (being explicit) with that of AM methods (much larger stability domains) is to use them in succession as a *predictor/corrector* pair. One of several possible strategies to obtain such a combined method of order p is the following:

1. use AB of order $p-1$ to get a predicted value at the new time level: $\widehat{y}(t+k)$,
2. use $\widehat{y}(t+k)$ to calculate a predicted value $\widehat{f}(t+k, \widehat{y}(t+k))$,

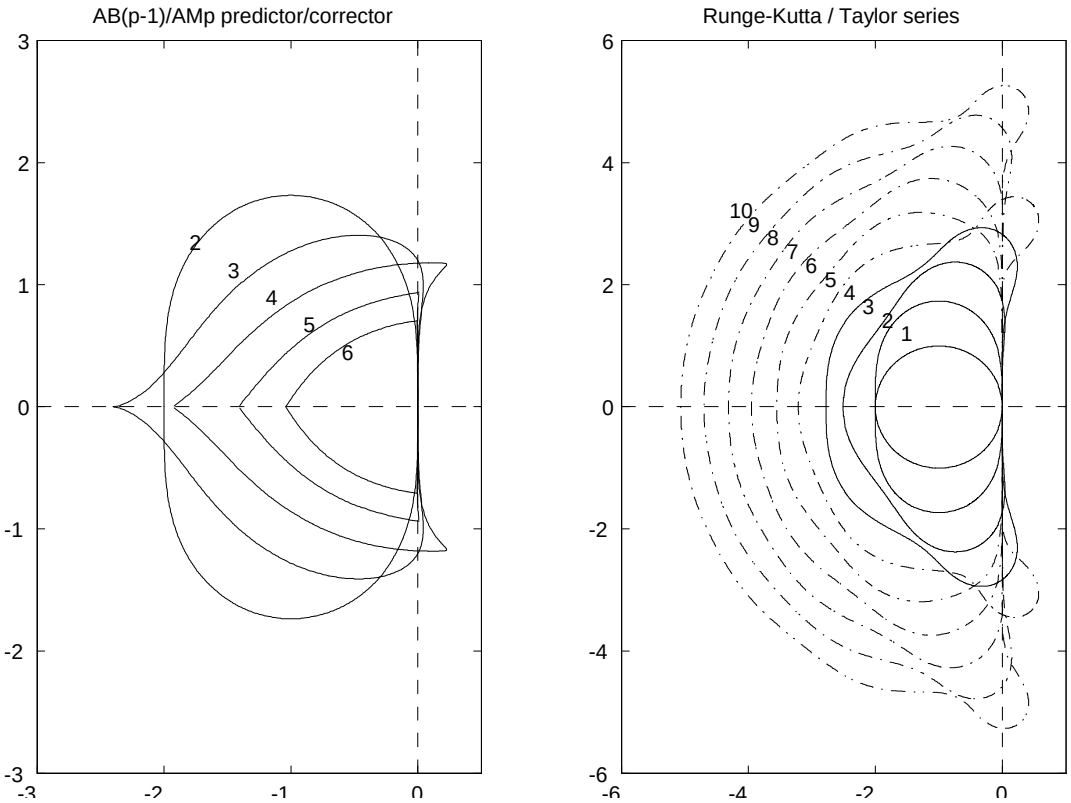


Figure 1.5: Stability domains for (a) AB(p-1)/AMp predictor/corrector methods for $p = 2, 3, \ldots, 6$. (b) Solid curves: RK methods with $s = p = 1, 2, 3, 4$; Solid and dash-dot curves: TS methods of orders $p = 1, 2, \ldots, 10$. Note that the scale differs by a factor of two between the two plots.

3. use $\widehat{f}(t+k, \widehat{y}(t+k))$ in the RHS of an AM scheme of order p, to obtain $y(t+k)$.

4. calculate a corrected value of $f(t+k, y(t+k))$ (to use for subsequent time steps).

This combined scheme can be shown to be accurate of order p, is entirely explicit, and will have considerably larger stability domain than corresponding AB methods, as seen by comparing Figures 1.3 a, b with Figure 1.5 a. It can be shown that all schemes of this type will have some imaginary axis coverage near the origin. A disadvantage with the schemes is that they require two (rather than one) function evaluations per time step.

The method to calculate a predictor/corrector scheme's stability domain is very similar to that for a regular LM scheme:

Example 9: Calculate the stability domain for the AB2/AM3 scheme.

As always when calculating stability domains, we consider the ODE $y' = \lambda y$. The two steps can be written

$$\begin{cases} \widehat{y}(t+k) = y(t) + \lambda k \, [\frac{3}{2}y(t) - \frac{1}{2}y(t-k)] \\ y(t+k) = y(t) + \lambda k \, [\frac{5}{12}\widehat{y}(t+k) + \frac{2}{3}y(t) - \frac{1}{12}y(t-k)] \; . \end{cases}$$

After the substitutions of $\lambda k = \xi$ and of $\widehat{y}(t+k)$ from the first equation into the second one, we get a linear 3-term recursion relation with characteristic equation

$$r^2 - \left(1 + \tfrac{13}{12}\xi + \tfrac{5}{8}\xi^2\right) \, r + \left(\tfrac{1}{12}\xi + \tfrac{5}{24}\xi^2\right) = 0 \; . \tag{1.12}$$

To trace out the edge of the stability domain, we again let r run around the periphery of the unit circle, and follow how ξ then moves. Although we, for each r-value can do this by solving a quadratic equation for ξ, an easier strategy is to start by noting that $r = 1$ always corresponds to $\xi = 0$. As we step along in r, we can use Newton's method to find the corresponding ξ-value (using the ξ-value from the last computed case as the start approximation for the next one). □

Figure 1.5 a shows the stability domains for AB(p-1)/AMp schemes when $p = 2, 3, 4, 5, 6$.

1.6 Runge-Kutta (RK) methods

Runge-Kutta methods are more convenient than LM methods in that one does not need any previous time levels to get started. The lowest order RK scheme is FE, which we now write as

$$\begin{array}{ll} s = 1, \; p = 1 & d^{(1)} = k \; f(t, y(t)) \\ & - - - - - - - - \\ & y(t+k) = y(t) + [d^{(1)}] \end{array}$$

For higher order RK methods, each time step is split into s internal *stages*, requiring one function evaluation each. The following are examples of methods of increasing numbers of stages s and of correspondingly increasing accuracies p:

$s = 2,\ p = 2$
$$\begin{aligned} d^{(1)} &= k\ f(t, y(t)) \\ d^{(2)} &= k\ f(t + \tfrac{k}{2}, y(t) + \tfrac{1}{2}d^{(1)}) \\ &- - - - - - - - \\ y(t + k) &= y(t) + [d^{(2)}] \end{aligned}$$

$s = 3,\ p = 3$
$$\begin{aligned} d^{(1)} &= k\ f(t, y(t)) \\ d^{(2)} &= k\ f(t + \tfrac{k}{3}, y(t) + \tfrac{1}{3}d^{(1)}) \\ d^{(3)} &= k\ f(t + \tfrac{2k}{3}, y(t) + \tfrac{2}{3}d^{(2)}) \\ &- - - - - - - - \\ y(t + k) &= y(t) + [\tfrac{1}{4}d^{(1)} + \tfrac{3}{4}d^{(3)}] \end{aligned}$$

The best known of all RK schemes is probably

$s = 4,\ p = 4$
$$\begin{aligned} d^{(1)} &= k\ f(t, y(t)) \\ d^{(2)} &= k\ f(t + \tfrac{k}{2}, y(t) + \tfrac{1}{2}d^{(1)}) \\ d^{(3)} &= k\ f(t + \tfrac{k}{2}, y(t) + \tfrac{1}{2}d^{(2)}) \\ d^{(4)} &= k\ f(t + k, y(t) + d^{(3)}) \\ &- - - - - - - - \\ y(t + k) &= y(t) + \tfrac{1}{6}[d^{(1)} + 2d^{(2)} + 2d^{(3)} + d^{(4)}]\ . \end{aligned}$$

Although there are many RK schemes with the same values for s and p, it turns out that whenever $s = p$, the stability domains are described by the relation

$$r = \sum_{n=0}^{p} \frac{\xi^n}{n!}\ , \tag{1.13}$$

giving the domains shown by solid lines (numbered 1-4) in Figure 1.5 b. For $s = p = 1$, we recognize the domain for FE. In contrast to the AB and AM methods, the stability domains increase in size when the order increases. In particular, we can note that the domains for RK3 and RK4 cover sections of both the negative real axis and the imaginary axis. For higher orders than $p = 4$, a few complications arise:

1. computation of coefficients within the stages becomes extremely complicated (however, schemes with orders up to around $p = 10$ are tabulated in the literature, and need not be re-derived by users),

2. there are no longer any schemes with $s = p$, only with $s > p$,

3. the stability domains become dependent on the RK coefficients (i.e. no longer dependent only on $p = s$), and

4. the accuracy may be lower for systems of PDEs than for scalar ODEs.

Numerical ODE packages that are based on RK methods usually employ RK variations which provide not only a value at the new time level, but also an error estimate for each step. The step lengths can then be adjusted automatically in order to meet a specified error tolerance.

Apart from explicit RK methods, there are also implicit RK methods. These can reach order $2s$ with only s stages, and can also have perfect stability domains - precisely the left half plane (matching where the solutions to the ODE $y' = \lambda y$ feature no growth). However, the cost required for solving the systems of equations that arise often make them less practical.

1.7 Taylor series (TS) methods

The TS approach is in some cases the most powerful technique available, but it is somewhat limited in that

- The RHS of the ODE (1.1) must be composed only of analytic functions (e.g. of powers, fractions, exponentials, and trigonometric functions), and
- Quite advanced software is needed if one wants to automatically translate (compile) an ODE into executable code. In contrast, for LM and RK methods, the ODE solver itself does not need to be altered between different ODEs.

Recalling from (1.4) that FE corresponds to truncating after the second term in a Taylor series

$$y(t+k) = y(t) + k\, y'(t) + \frac{k^2}{2!}y''(t) + \frac{k^3}{3!}y'''(t) + \frac{k^4}{4!}y''''(t) + \dots\,, \tag{1.14}$$

the idea is simply to truncate at a much later level. That requires that we somehow can determine higher derivatives of y. Surprisingly many numerical analysis text books mention and then dismiss this approach after considering only an inefficient way to go about this - repeated differentiation. Given $y'(t) = f(t, y(t))$, differentiation (based on the chain rule) gives

$$\begin{aligned}
y'' &= f\,\tfrac{\partial f}{\partial y} + \tfrac{\partial f}{\partial t} \\
y''' &= f^2\tfrac{\partial^2 f}{\partial y^2} + f\,\left\{2\tfrac{\partial^2 f}{\partial t\partial y} + \left(\tfrac{\partial f}{\partial y}\right)^2\right\} + \tfrac{\partial f}{\partial t}\cdot\tfrac{\partial f}{\partial y} + \tfrac{\partial^2 f}{\partial t^2} \\
y'''' &= f^3\tfrac{\partial^3 f}{\partial y^3} + f^2\left\{3\tfrac{\partial^3 f}{\partial t\partial y^2} + 4\tfrac{\partial f}{\partial y}\cdot\tfrac{\partial^2 f}{\partial y^2}\right\} + \tfrac{\partial f}{\partial t}\cdot\left(\tfrac{\partial f}{\partial y}\right)^2 + \tfrac{\partial^3 f}{\partial t^3} + 3\tfrac{\partial f}{\partial t}\cdot\tfrac{\partial^2 f}{\partial t\partial y} + \tfrac{\partial^2 f}{\partial t^2}\cdot\tfrac{\partial f}{\partial y} + \\
&\quad + f\,\left\{\left(\tfrac{\partial f}{\partial y}\right)^3 + 5\tfrac{\partial^2 f}{\partial t\partial y}\cdot\tfrac{\partial f}{\partial y} + 3\tfrac{\partial^3 f}{\partial t^2\partial y} + 3\tfrac{\partial f}{\partial t}\cdot\tfrac{\partial^2 f}{\partial y^2}\right\} \\
\cdots &\quad \text{etc.}
\end{aligned}$$

Not only does the number of terms increase very rapidly, the individual derivatives also get very complex quickly for all but the simplest functions $f(t, y)$. It is much more efficient to determine the expansion coefficients recursively. The following brief Mathematica script illustrates how, if given a value y at time t, one can generate the coefficients a_1, a_2, ... in the expansion

$$y(t+k) = y(t) + a_1 k + a_2 k^2 + a_3 k^3 + \dots \tag{1.15}$$

In the case of $n = 8$ and the ODE (1.2), the script would be

```
n = 8;                    (* Specify number of coefficients *)
y[k_] := y+Sum[a[i] k^i,{i,1,n}]+O[k]^(n+1)
f[t_,y_] := t^2+y^2       (* Specify the RHS of the ODE *)
LogicalExpand[y'[k]==f[t+k,y[k]]]
```

giving the output

```
t^2+y^2-a[1]==0 && 2t+2y a[1]-2a[2]==0 && 1+a[1]^2+2y a[2]-3a[3]==0 &&
2a[1] a[2]+2y a[3]-4a[4]==0 && a[2]^2+2 a[1] a[3]+2y a[4]-5a[5]==0 &&
2a[2] a[3]+2a[1] a[4]+2y a[5]-6a[6]==0 &&
a[3]^2+2a[2] a[4]+2a[1] a[5]+2y a[6]-7a[7]==0 &&
2a[3] a[4]+2a[2] a[5]+2a[1] a[6]+2y a[7]-8a[8]==0
```

Even for general (analytic) functions $f(t, y(t))$, the result will again become simple algebraic recursions that explicitly provide the successive coefficients. A similar approach to obtain the same recursions starts with considering

$$\frac{dy(t+k)}{dk} = f(t+k, y(t+k)).$$

Every time a truncated version of (1.15) is substituted into the RHS $f(t+k, y(t+k))$ and is integrated with respect to k, we gain one more term in (1.15). By always truncating to no more terms than what are known to be correct, the algebra remains moderate and well suited for automation without the need of advanced symbolic packages such as Mathematica. In the TS method, the recursion relations are generated as a preprocessing step, and then compiled into the actual solution algorithm. Like for the other numerical ODE approaches, it is usually most convenient to rely on library software when using the TS approach.

The stability domains for the TS methods are also described by (1.13). For RK methods, this formula was valid only when $s = p = 1, 2, 3, 4$. The TS method continues this same formula up to any value of p. The dashed curves in Figure 1.5 b show how the domains continue to grow as p increases.

One intuitive way to understand why TS methods have so much larger stability domains (and also are much more accurate) than AB or AM methods of corresponding orders is to note that the latter pick up their 'extra' information from very old and 'obsolete' back values of y', essentially relying on the notoriously ill-conditioned concept of using one-sided high order polynomial approximations (cf. the discussion of the Runge phenomenon in Section ...). In contrast, a TS method brings up the order by means of extracting much more analytic information from the ODE itself, right at the most recent time position.

The TS method has been extended to a continued fraction method, with the ability go to through or near to poles in a complex t-plane without any adverse effects on numerical step size k - an entirely unique feature among numerical ODE solvers [..].

1.8 Stiff ODEs

A *stiff* system of ODEs is one in which some processes happen extremely fast while others are much slower. A typical example could be a chemical process with different coupled reactions. Generalizing the linearization procedure described in Section 1.3.3 to a system of ODEs leads us to consider

$$\underline{v}'(t) \approx A\,\underline{v}(t) + \underline{b}\,.$$

If the eigenvalues λ of A differ in size by several orders of magnitude, especially if some are negative and very large in magnitude, we have a stiff system. In order not to be forced to use an extremely small time step (given that $\xi = \lambda k$ for all the λ have to fall within the stability domain of the ODE solver), it is essential that the stability domain extends far out to the left in the complex ξ-plane. Of the methods that we have discussed, the BD methods excel in this regard.