

## Ordinary Differential Equations

### 8.1. Introduction

My section 8.1 will cover the material in sections 8.1 and 8.2 in the book. Read the book sections on your own.

I don't like the order of things in the textbook. I will cover the sections in the following order: 8.1, 8.2, 8.4, 8.5, 8.6, 8.19, 8.8, 8.9, 8.10, 8.11. Read section 8.7 on your own, and skip the rest.

The material on *Theory of ODES* in this section should have been covered in a previous course on ordinary differential equations. It is included to refresh your memory, not to teach you the material.

**8.1.1. Theory of ODEs.** An ordinary differential equation (ODE) is an equation involving an unknown function  $y(t)$  and its derivatives. The *order* of the ODE is the order of the highest derivative. Assuming that we can always solve for this highest derivative, the most general form of an  $n$ th order ODE is

$$y^{(n)}(t) = f(t, y(t), y'(t), \dots, y^{(n-1)}(t)).$$

The function  $f$  can be quite complicated.

The functions  $y$  and  $f$  could also be vectors

$$\mathbf{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \dots \\ y_m(t) \end{pmatrix}, \quad \mathbf{f}(t, \mathbf{y}(t), \mathbf{y}'(t), \dots, \mathbf{y}^{(n-1)}(t)) = \begin{pmatrix} f_1(t, y_1(t), \dots, y_m(t), y_1'(t), \dots, y_m^{(n-1)}(t)) \\ f_2(t, y_1(t), \dots, y_m(t), y_1'(t), \dots, y_m^{(n-1)}(t)) \\ \dots \\ f_m(t, y_1(t), \dots, y_m(t), y_1'(t), \dots, y_m^{(n-1)}(t)) \end{pmatrix}.$$

The resulting equation

$$\mathbf{y}^{(n)}(t) = \mathbf{f}(t, \mathbf{y}(t), \mathbf{y}'(t), \dots, \mathbf{y}^{(n-1)}(t)).$$

is called a *vector ODE* or a *system of ODEs*.

The solution of an ODE is not a number, but a set of functions defined on some interval. To solve an  $n$ th order ODE, you have to perform  $n$  integrations, and you pick up  $n$  arbitrary constants in the process. The solution is an  $n$ -parameter family of curves.

To find a unique solution function, you need  $n$  extra pieces of information. If these extra conditions are given in terms of the values of  $y$ ,  $y'$ ,  $y''$ , etc. at some point  $t_0$ , this is called an *initial value problem* (IVP). The extra conditions are then called *initial conditions* (IC).

If the extra conditions involve values of  $y$ ,  $y'$ , etc. at two or more points, we have a *boundary value problem* (BVP), and the conditions are the *boundary conditions* (BC). In BVP, the independent variable is often called  $x$  instead of  $t$ , since typically it represents a space variable. In IVP,  $t$  usually represents time.

In this course, we will concentrate only on IVP. The numerical techniques for IVP and BVP are quite different. The techniques used for BVP are similar to the techniques used for solving partial differential equations, and really belong in a course on numerical solution of PDE.

In a course on ODEs, great emphasis is placed on the distinction between linear and non-linear ODEs. Linear ODEs are much easier to solve by hand. For the numerical methods covered in this chapter, there is no difference between linear and nonlinear equations.

As a preparation for numerical solution, we can transform any higher order IVP into a first order system.

**Example:** Suppose we want to solve the IVP

$$\begin{aligned}u'''(t) &= -u'(t)u^2(t) + \sin t \\u(0) &= -1 \\u'(0) &= 1 \\u''(0) &= 2\end{aligned}$$

We introduce new variables

$$\begin{aligned}y_1 &= u \\y_2 &= u' \\y_3 &= u''\end{aligned}$$

and get the equations

$$\begin{aligned}y_1' &= y_2 \\y_2' &= y_3 \\y_3' &= -y_2y_1^2 + \sin t \\y_1(0) &= -1 \\y_2(0) &= 1 \\y_3(0) &= 2\end{aligned}$$

or, in mathematical shorthand,

$$\begin{aligned}\mathbf{y}' &= \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(0) &= \mathbf{y}_0\end{aligned}$$

where

$$\begin{aligned}\mathbf{f}(t, \mathbf{y}) &= \begin{pmatrix} f_1(t, y_1, y_2, y_3) \\ f_2(t, y_1, y_2, y_3) \\ f_3(t, y_1, y_2, y_3) \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \\ -y_2y_1^2 + \sin t \end{pmatrix}, \\ \mathbf{y}_0 &= \begin{pmatrix} -1 \\ 1 \\ 2 \end{pmatrix}.\end{aligned}$$

□

Any IVP can be transformed in this way into a first order system. For example, we could transform a system of 3 second order ODEs into a system of 6 first order ODEs.

As I mentioned in an earlier chapter, the basic questions a mathematician asks when confronted with a problem are

**Existence:** Is there a solution?

**Uniqueness:** Is there only one solution?

**Algorithms:** How do you find the solution?

The first two questions have been answered in the theory of ODEs:

**THEOREM 8.1.** *Consider the first-order system*

$$\begin{aligned}\mathbf{y}'(t) &= \mathbf{f}(t, \mathbf{y}(t)) \\ \mathbf{y}(t_0) &= \mathbf{y}_0\end{aligned}$$

(a) (Peano) *If  $\mathbf{f}$  is continuous, the equation has a solution.*

(b) (Picard-Lindelöf) *If  $\mathbf{f}$  has continuous partial derivatives with respect to the  $y_j$  and is continuous in  $t$ , the solution exists and is unique. (There are less restrictive conditions under which the solution is unique, but that is getting technical).*

Both of these facts are *local* statements: if  $f$  is continuous (or differentiable) in a neighborhood of the point  $(t_0, y_0)$ , the solution is only guaranteed to exist (or to be unique) in a small neighborhood of this point.

**Example:** The equation

$$\begin{aligned}y'(t) &= 2\sqrt{|y(t)|} \\ y(0) &= 0\end{aligned}$$

has the solution  $y(t) = t^2$ . It also has the solution  $y(t) = 0$ . There is actually an infinite number of solutions:

$$y(t) = \begin{cases} 0 & \text{for } 0 \leq t \leq a \\ (t-a)^2 & \text{for } t > a \end{cases}$$

where  $a$  is arbitrary. In words: you follow the  $t$ -axis from the origin for a while, and wherever you feel like, you take a branch of a parabola upwards.

Notice that  $f(t, y) = 2\sqrt{|y|}$  is continuous, but is differentiable in  $y$  only when  $y$  is non-zero. Therefore, there is always a solution, but where  $y$  is zero (on the  $t$ -axis), the solution may not be unique. After you leave the  $t$ -axis, there are no more problems.  $\square$

**Example:** Consider

$$\begin{aligned}y'(t) &= y^2(t) \\ y(0) &= 1\end{aligned}$$

This  $f$  is continuously differentiable, so we know that the solution exists and is unique. The solution is not hard to find:

$$y(t) = \frac{1}{1-t}$$

However, this solution “blows up” as  $t$  approaches 1.

The moral is that even if a solution exists locally, it may not exist for all  $t$ .  $\square$

**8.1.2. Numerical Methods.** As shown above, we can restrict attention to solving first order systems. To make life even easier, we will only investigate methods for the one-dimensional case

$$\begin{aligned}y'(t) &= f(t, y(t)) \\ y(t_0) &= y_0\end{aligned}$$

It turns out that everything in this chapter that works for the one-dimensional case works the same way for vectors. We will do some vector examples, so you can see how to do that, but all the derivations will be done for the one-dimensional case.

We assume that  $f$  is at least continuous, so that a solution always exists. We ignore the possibilities that the solution is not unique, or that it “blows up” in finite time. These problems must be dealt with by examining the original problem you are trying to solve in more detail.

All of the methods we will consider are *marching methods*. That means that we start at the point  $t_0$ , where the initial condition is given, calculate the approximate solution at some point  $t_1$ , then at  $t_2$ , and so on. We are “marching” along the graph of the solution in the direction of increasing  $t$ .

This is in contrast to BVP and PDE, where you usually solve the equation globally. That means you discretize the interval on which you want to solve the equation at  $t_0, \dots, t_n$  and set up a system of equations to be solved simultaneously.

The classes of methods we will consider are

- Taylor series methods (including Euler’s method)
- Runge-Kutta methods
- multistep methods (predictor-corrector)

Other types of methods that are frequently used are *extrapolation methods* and *multi-value methods*. We will not discuss them here.

**8.1.3. Errors.** In the error analysis, we have to distinguish between the *local error* and the *global error*. The local error is the error committed in one step, from  $t_{n-1}$  to  $t_n$ . The global error is the accumulated error from all the previous steps. In order to keep things straight, we have to introduce some notation.

$y(t)$  is the true solution of the IVP. Thus,  $y(t_n)$  is the value of the true solution at the point  $t_n$ .

$y_n$  is the value of the approximate solution at the point  $t_n$ . The numerical and true solutions agree at  $t_0$ , so that  $y(t_0) = y_0$ , but probably nowhere else.

The global error  $E_n$  at the point  $t_n$  is the difference between true and numerical solution:

$$E_n = y(t_n) - y_n.$$

To define the local error  $e_n$  at the point  $t_n$ , we need to ignore everything that happened before  $t_{n-1}$ , and re-start the ODE at that point. Let  $z(t)$  be the true solution of the IVP re-started at the point  $t_{n-1}$ . That is,  $z(t)$  satisfies

$$\begin{aligned} z'(t) &= f(t, z(t)) \\ z(t_{k-1}) &= y_{k-1}. \end{aligned}$$

Then the local error  $e_n$  at the point  $t_n$  is defined as

$$e_n = z(t_n) - y_n.$$

In order to visualize this, you have to look not only at the solution curve you are after, but at the entire family of all solution curves.

For example, consider the equation  $y' = y$ . The solution family consists of all curves of the form

$$y(t) = ce^t,$$

where  $c$  is an arbitrary constant. Every time you take a step, you step off the solution curve you were on, and land on a nearby curve, with a different  $c$ . The local error is the difference between the last two curves. The global error is the difference between the last curve and the original one (see figure).

The local error is determined by the method you use, and is usually fairly easy to estimate. The global error is determined partly by the method you use and also partly by the geometry of the family of solution curves.

Consider again the test equation

$$y'(t) = y(t)$$

The solution curves are  $y(t) = ce^t$ , which are exponentially diverging. No matter what method you use, your global error will grow exponentially.

In contrast, the equation

$$y'(t) = -y(t)$$

has the family of solution curves  $y(t) = ce^{-t}$ . Here, the curves are going to zero at an exponential rate, and any decent numerical solution should do the same. Global errors actually get smaller.

These test problems are sometimes referred to as *stable* and *unstable* equations, respectively. The names are a little misleading, in my opinion. It is not necessarily bad if the errors grow exponentially, as long as the solution does the same. If you look at the *relative* errors, they are actually the same in both test problems.

## 8.2. Stable and Unstable Equations, Numerical Methods

This section in the book is covered in my section 8.1.

### 8.4. Euler's Method

We want to solve the IVP

$$y'(t) = f(t, y(t))$$

$$y(t_0) = y_0$$

numerically at a sequence of points  $t_0, t_1, t_2, \dots$ . To make life easier, we use a constant stepsize  $h$ , so that

$$t_n = t_0 + hn.$$

Assuming that  $y$  is sufficiently often differentiable,

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \dots$$

If we truncate this series at a certain point, we get a *Taylor series* method. We will talk about them in more detail in section 8.19. Right now, we just want to look at the simplest one.

If we stop the expansion after the second term, we get *Euler's method*. It is given by

$$\begin{aligned} y_{n+1} &= y_n + hy'_n \\ &= y_n + hf(t_n, y_n). \end{aligned}$$

**Example:** Consider the equation

$$y'(t) = -2ty^2(t)$$

$$y(0) = 1$$

with exact solution

$$y(t) = \frac{1}{1+t^2}$$

With stepsize  $h = 0.5$ , we find

$$\begin{aligned} t_0 &= 0, & y_0 &= 1 \\ t_1 &= 0.5, & y_1 &= y_0 + hf(t_0, y_0) = 1 + 0.5(-2 \cdot 0 \cdot 1^2) = 1 \\ t_2 &= 1, & y_2 &= y_1 + hf(t_1, y_1) = 1 + 0.5(-2 \cdot 0.5 \cdot 1^2) = 0.5 \\ t_3 &= 1.5, & y_3 &= 0.25 \\ t_4 &= 2, & y_4 &= 0.15625 \end{aligned}$$

The true value is  $y(2) = 0.2$ , so the global error at  $t = 2$  is  $E_4 = 0.04375$ .

With stepsize  $h = 0.25$ , we get

$$\begin{aligned} t_0 &= 0, & y_0 &= 1 \\ t_1 &= 0.25, & y_2 &= 1 \\ t_2 &= 0.5, & y_3 &= 0.875 \\ t_4 &= 1, & y_4 &= 0.508356094 \\ t_8 &= 2, & y_8 &= 0.181628009 \end{aligned}$$

The global error at the same point  $t = 2$  is now  $E_8 = 0.018371991$ .

With stepsize  $h = 0.125$ , you get

$$\begin{aligned} t_0 &= 0, & y_0 &= 1 \\ t_8 &= 1, & y_8 &= 0.504548613 \\ t_{16} &= 2, & y_{16} &= 0.191547485 \end{aligned}$$

The global error at  $t = 2$  is now  $E_{16} = 0.008452515$ .

Observe that when we cut the stepsize in half, the global error is also approximately cut in half. We will prove this in the next section  $\square$

### 8.5. Accuracy and Stability

My section 8.5 will cover the material in sections 8.5 and 8.6 in the book. Read the book sections on your own.

**8.5.1. Accuracy.** Let us investigate the local and global errors of Euler's method in detail.

The local error is easy to find: we expand  $z(t)$  in a Taylor series around the point  $t_{n-1}$  and evaluate at  $t_n$

$$\begin{aligned} z(t_n) &= z(t_{n-1}) + hz'(t_{n-1}) + \frac{h^2}{2}z''(t_{n-1}) + \dots \\ &= y_{n-1} + hf(t_{n-1}, y_{n-1}) + \frac{h^2}{2}z''(t_{n-1}) + \dots \end{aligned}$$

while

$$y_n = y_{n-1} + hf(t_{n-1}, y_{n-1}).$$

Subtracting gives

$$\text{local error } e_n = z(t_n) - y_n = \frac{h^2}{2}z''(t_{n-1}) + \dots = O(h^2).$$

A quick and dirty estimate for the global error is this: assume the global error at the point  $t$  is the sum of all local errors. It takes  $(t - t_0)/h$  steps to reach  $t$ , and each step has a local error  $O(h^2)$ , so the global error is  $O(h)$ . This is not a mathematical proof, but it is correct reasoning: if the local error is  $O(h^{p+1})$ , the global error is  $O(h^p)$ . We saw the same argument for numerical quadrature before.

A method for solving ODEs numerically is said to be *of order p* if

$$\begin{aligned} \text{local error} &= O(h^{p+1}), \\ \text{global error} &= O(h^p). \end{aligned}$$

Remember that the only way to verify the order experimentally is to do the calculation with two different values of  $h$ . If we cut  $h$  in half, then the error should go down by a factor of 2 for a first order method, by a factor of  $2^2 = 4$  for a second order method, and so on.

I have tried to do that in the examples. For example, when I cut the stepsize for Euler's method in half, the final error is approximately cut in half. Unfortunately, this does not work in all cases. The reason is that the geometry of the solution curves also plays a role. However, if you took smaller and smaller  $h$ , eventually you would be able to observe this behavior in all cases.

A more accurate estimate of the error for Euler's method is the following.

$$\begin{aligned} E_{n+1} &= y(t_{n+1}) - y_{n+1} \\ &= [y(t_n) + hf(t_n, y(t_n)) + O(h^2)] - [y_n + hf(t_n, y_n)] \\ &= (y(t_n) - y_n) + h[f(t_n, y(t_n)) - f(t_n, y_n)] + O(h^2) \\ &\approx E_n + h \frac{\partial f}{\partial y} [y(t_n) - y_n] + O(h^2) \\ &\approx (1 + h\lambda)E_n + O(h^2), \end{aligned}$$

where we assume that the  $y$ -derivative of  $f$  is approximately equal to some constant  $\lambda$ . As a prototype equation, think of

$$y'(t) = \lambda y(t)$$

with the solution  $y(t) = ce^{\lambda t}$ .

Let me repeat the final formula:

$$E_{n+1} \approx (1 + h\lambda)E_n + O(h^2)$$

The interpretation of this formula is the following: The new global error  $E_{n+1}$  is the old global error  $E_n$ , magnified by a factor of  $(1 + h\lambda)$ , plus the new local error  $O(h^2)$ . If  $|1 + h\lambda| > 1$ , the global error grows exponentially. If  $|1 + h\lambda| < 1$ , the global error stays small.

The recursive equation for  $E_n$  can be solved in closed form, and again leads to  $E_n = O(h)$ .

**8.5.2. Stability.** A numerical method for solving ODEs is called *stable* (for a particular stepsize  $h$  and for a particular  $\lambda$ ) if the numerical solution to the model problem

$$y'(t) = \lambda y(t)$$

remains bounded as  $t \rightarrow \infty$ .

You should note a few things about this definition:

- Stability is different from accuracy. The error, especially the relative error, may still be large, but at least things don't blow up.
- We are mostly interested in negative values of  $\lambda$ . If  $\lambda$  is positive, the solution itself blows up, and we expect the numerical solution to do the same. Actually, what we are really interested in is complex  $\lambda$  with negative real part. I assume this sort of stuff comes up in EE (oscillating signals with exponentially decaying amplitude).
- Stability depends on the stepsize  $h$  and on  $\lambda$ . Thus, a method may be stable for some choices of  $h$  and  $\lambda$  and not for others. In practice, stability always depends only on the product  $h\lambda$ , not on  $h$  or  $\lambda$  separately. This makes sense, because a change of unit in  $t$  will affect both  $h$  and  $\lambda$ , but  $h\lambda$  will remain the same.

For Euler's method, the stability behavior is easy to see. Euler's method for the test problem reduces to

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

The method is stable if  $|1 + h\lambda| \leq 1$ , which means for real  $\lambda$  that

$$-2 \leq h\lambda \leq 0.$$

The *region of stability* of a method is defined as the set of all  $h\lambda$  for which the method is stable. The region of stability is a set in the complex plane.

The true solution of the test problem goes to zero if  $\Re(\lambda) < 0$ , and blows up if  $\Re(\lambda) > 0$ . A numerical method with ideal stability characteristics has the same properties: the numerical solution goes to zero if  $\Re(h\lambda) < 0$ , and blows up if  $\Re(h\lambda) > 0$ . Thus, the ideal region of stability is the left half plane.

For most methods, the region of stability is a small subset of that. For Euler's method, the region of stability is a circle (see figure). This means that for a fixed  $\lambda$ , the method is stable for some  $h$ , but not for others. For some  $\lambda$ , there may not be any stable  $h$ .

In some cases, the region of stability extends into the right half plane. This is not desirable, since it means that in some cases the numerical solution goes to zero, even though the true solution blows up.

**Example:** Solve the equation

$$\begin{aligned} y' &= -y \\ y(0) &= 1 \end{aligned}$$

by Euler's method with step sizes  $h = 0.5$  and  $h = 3$ . The results are

	$h = 0.5$	$h = 3$
$y_0$	1	1
$y_1$	0.5	-2
$y_2$	0.25	4
$y_3$	0.125	-8
$y_4$	0.0625	16

Clearly, the choice  $h = 0.5$  is stable, while  $h = 3$  is not.  $\square$

We will discuss the stability behavior of other methods in section 8.9.3.

### 8.6. Order of an Integration Method

This section in the book is covered in my section 8.5.

### 8.19. Taylor Series and Runge-Kutta Methods

**8.19.1. Taylor Series Methods.** The basic idea was mentioned before, when we derived Euler's method. Assuming that  $y$  is sufficiently often differentiable,

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \dots$$

If we truncate this series at a certain point and replace  $y(t_n)$  by  $y_n$ , we get a Taylor series method.

The simplest case is Euler's method, which we already covered. The next higher Taylor series method uses the first three terms, so

$$y_{n+1} = y_n + hy'_n + \frac{h^2}{2}y''_n$$

The second derivative is obtained by differentiating the ODE once more.

**Example:** For our standard example,  $f(t, y) = -2ty^2$ , so

$$\begin{aligned} y''(t) &= \frac{d}{dt}y'(t) = \frac{d}{dt}(-2ty^2) = -2y^2 - 4tyy' \\ &= -2y^2 - 4ty(-2ty^2) = -2y^2 + 8t^2y^3. \end{aligned}$$

We get the method

$$y_{n+1} = y_n + h[-2t_n y_n^2] + \frac{h^2}{2}[-2y_n^2 + 8t_n^2 y_n^3]$$

We find for  $h = 0.5$

$$\begin{aligned}y_0 &= 1 \\y_1 &= 0.75 \\y_2 &= 0.43359375 \\y_3 &= 0.280106485 \\y_4 &= 0.192250483\end{aligned}$$

with a global error  $E_4 = 0.007749517$ .

For  $h = 0.25$ , we get

$$\begin{aligned}y_0 &= 1 \\y_4 &= 0.487213029 \\y_8 &= 0.199720953\end{aligned}$$

with a global error  $E_8 = 0.000279047$ .

For  $h = 0.125$ , we get

$$\begin{aligned}y_0 &= 1 \\y_8 &= 0.4972455756 \\y_{16} &= 0.199994786\end{aligned}$$

with a global error  $E_{16} = 0.000005214$ .  $\square$

You can observe that this method is more accurate than Euler's method, and also that the errors decrease faster. This method should be of order 2, even though the error seems to decrease faster than that.

Something else you will observe if you try this on a more complicated problem, or for a higher-order method, is that this differentiation business gets harder pretty fast. This is the main reason that other methods are more popular. Runge-Kutta and multistep methods give the same accuracy as higher order Taylor series methods, without the hassle of taking all these derivatives.

**8.19.2. Runge-Kutta Methods.** Runge-Kutta methods (I will call them RK methods from now on) were developed in an effort to achieve the accuracy of higher order Taylor series methods without having to calculate a lot of derivatives. This is done by inserting intermediate points between  $t_n$  and  $t_{n+1}$ .

An  $m$ -stage explicit RK method is given by

$$\begin{aligned}k_1 &= hf(t_n, y_n) \\k_2 &= hf(t_n + \alpha_2 h, y_n + \beta_{21} k_1) \\k_3 &= hf(t_n + \alpha_3 h, y_n + \beta_{31} k_1 + \beta_{32} k_2) \\&\dots\dots \\k_m &= hf(t_n + \alpha_m h, y_n + \beta_{m1} k_1 + \dots + \beta_{m,m-1} k_{m-1}) \\y_{n+1} &= y_n + \gamma_1 k_1 + \gamma_2 k_2 + \dots + \gamma_m k_m\end{aligned}$$

The  $k_j$  are something like intermediate points we put between  $t_n$  and  $t_{n+1}$ . The  $\alpha_j$ ,  $\beta_{ij}$ ,  $\gamma_j$  are chosen to produce a method of as high an order as possible, by matching the RK formula with the Taylor series expansion of the true solution.

Note that the calculation of each  $k_j$  involves other known  $k_i$ . An *implicit RK method* also uses  $k_i$  that are not known yet. An implicit method requires the solution of a system of nonlinear equations at each step. In between the two, we have *diagonally implicit RK methods*, where each  $k_j$  involves itself, but not higher ones. A diagonally implicit method requires the solution of a sequence of one-dimensional nonlinear equations, one for each  $k_j$ . We won't discuss implicit methods.

Consider the general case of two-stage explicit RK methods. We set up

$$\begin{aligned}k_1 &= hf(t_n, y_n) \\k_2 &= hf(t_n + \alpha h, y_n + \beta k_1) \\y_{n+1} &= y_n + \gamma_1 k_1 + \gamma_2 k_2\end{aligned}$$

We want to determine  $\alpha$ ,  $\beta$ ,  $\gamma_1$  and  $\gamma_2$  to minimize the local error.

Let us use the shorthand notation

$$\begin{aligned} f &= f(t_n, y_n) \\ f_y &= \frac{\partial f}{\partial y}(t_n, y_n) \\ f_t &= \frac{\partial f}{\partial t}(t_n, y_n) \end{aligned}$$

Note that by the chain rule,

$$\frac{d}{dt}f(t, y(t)) = f_t + f \cdot f_y.$$

Let  $z(t)$  solve

$$\begin{aligned} z'(t) &= f(t, z(t)) \\ z(t_n) &= y_n \end{aligned}$$

Do a Taylor series expansion

$$\begin{aligned} z(t_{n+1}) &= z(t_n) + hz'(t_n) + \frac{h^2}{2}z''(t_n) + \dots \\ &= y_n + hf + \frac{h^2}{2}[f_t + f \cdot f_y] + \dots \end{aligned}$$

For the RK formula, we have

$$\begin{aligned} k_1 &= hf \\ k_2 &= hf(t_n + \alpha h, y_n + \beta k_1) = h[f + \alpha h f_t + \beta k_1 f_y + \dots] \\ &= hf + \alpha h^2 f_t + \beta h^2 f \cdot f_y + \dots \\ y_{n+1} &= y_n + \gamma_1 k_1 + \gamma_2 k_2 \\ &= y_n + h[\gamma_1 f + \gamma_2 f] + h^2[\gamma_2 \alpha f_t + \gamma_2 \beta f \cdot f_y] + \dots \end{aligned}$$

Thus, the local error is

$$\begin{aligned} e_{n+1} &= z(t_{n+1}) - y_{n+1} \\ &= h[f - \gamma_1 f - \gamma_2 f] + h^2 \left[ \left( \frac{1}{2} f_t + \frac{1}{2} f \cdot f_y \right) - (\gamma_2 \alpha f_t + \gamma_2 \beta f \cdot f_y) \right] + \dots \end{aligned}$$

We can eliminate the coefficients of  $h$  and  $h^2$  by demanding

$$\begin{aligned} \gamma_1 + \gamma_2 &= 1 \\ \gamma_2 \alpha &= 1/2 \\ \gamma_2 \beta &= 1/2 \end{aligned}$$

These are 3 equations in 4 unknowns, so we have one degree of freedom left. If we choose  $\alpha$  as a parameter, we get

$$\begin{aligned} \beta &= \alpha \\ \gamma_1 &= 1 - 1/(2\alpha) \\ \gamma_2 &= 1/(2\alpha) \end{aligned}$$

and the general two-stage explicit RK method:

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \alpha h, y_n + \alpha k_1) \\ y_{n+1} &= y_n + \left(1 - \frac{1}{2\alpha}\right)k_1 + \frac{1}{2\alpha}k_2 \end{aligned}$$

Each choice of  $\alpha$  leads to a different method. There is no choice of  $\alpha$  which allows you to also match the coefficient of  $h^3$ . Thus, the local error is of order 3, the order of the method is 2 for all  $\alpha$ .

For  $\alpha = 0.5$  you get the *midpoint method*, for  $\alpha = 1$  you get the *modified Euler method*.

**Example:** Use the midpoint method with  $h = 0.5$  on our standard example

$$\begin{aligned}y'(t) &= -2ty^2(t) \\ y(0) &= 1\end{aligned}$$

We have here  $\alpha = \beta = 0.5$ ,  $\gamma_1 = 0$ ,  $\gamma_2 = 1$ . The equations are

$$\begin{aligned}k_1 &= hf(t_n, y_n) \\ k_2 &= hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \\ y_{n+1} &= y_n + k_2\end{aligned}$$

We start with  $t_0 = 0$ ,  $y_0 = 1$ . The first step is

$$\begin{aligned}k_1 &= hf(t_0, y_0) = 0 \\ k_2 &= hf(t_0 + h/2, y_0 + k_1/2) = hf(0.25, 1) = -0.25 \\ y_1 &= y_0 + k_2 = 0.75\end{aligned}$$

The next step is

$$\begin{aligned}k_1 &= hf(t_1, y_1) = -0.28125 \\ k_2 &= hf(t_1 + h/2, y_1 + k_1/2) = hf(0.75, 0.609375) = -0.278503418 \\ y_2 &= y_1 + k_2 = 0.4714965820\end{aligned}$$

and so on. We get

$$\begin{aligned}y_3 &= 0.309188574 \\ y_4 &= 0.2104856219\end{aligned}$$

The error at the end is  $E_4 = 0.0104856219$ .  $\square$

**Example:** As an example of how do this with a system, consider the problem

$$\begin{aligned}y'(t) &= y(t) + 1/z(t) \\ z'(t) &= -t/y(t) \\ y(1) &= e \\ z(1) &= 1/e\end{aligned}$$

with true solution

$$\mathbf{y}(t) = \begin{pmatrix} y(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} te^t \\ e^{-t} \end{pmatrix}$$

Again, we use  $\alpha = 0.5$ ,  $h = 0.5$ .

The first step goes like this:

$$\begin{aligned}\mathbf{k}_1 &= h\mathbf{f}(t_0, \mathbf{y}_0) = 0.5\mathbf{f}(1, e, 1/e) \\ &= 0.5 \begin{pmatrix} e + 1/(1/e) \\ -1/e \end{pmatrix} = \begin{pmatrix} 2.7182818285 \\ -0.1839397206 \end{pmatrix} \\ \mathbf{k}_2 &= h\mathbf{f}\left(t_0 + \frac{1}{2}h, \mathbf{y}_0 + \frac{1}{2}\mathbf{k}_1\right) \\ &= 0.5\mathbf{f}\left(1.25, e + \frac{1}{2}2.718\dots, 1/e - \frac{1}{2}0.183\dots\right) = \begin{pmatrix} 3.8508992570 \\ -0.1532831005 \end{pmatrix} \\ \mathbf{y}_1 &= \mathbf{y}_0 + \mathbf{k}_2 = \begin{pmatrix} e \\ 1/e \end{pmatrix} + \mathbf{k}_2 = \begin{pmatrix} 6.5691810854 \\ 0.2145963407 \end{pmatrix}\end{aligned}$$

The true values at  $t = 1.5$  are 6.722533605 and 0.223130160.

In later steps, we get

$$\begin{aligned} \mathbf{k}_1 &= h\mathbf{f}(t_1, \mathbf{y}_1) = \begin{pmatrix} 5.6145463957 \\ -0.1141694817 \end{pmatrix} \\ \mathbf{k}_2 &= h\mathbf{f}\left(t_1 + \frac{1}{2}h, \mathbf{y}_1 + \frac{1}{2}\mathbf{k}_1\right) = \begin{pmatrix} 7.8625965252 \\ -0.0933188574 \end{pmatrix} \\ \mathbf{y}_2 &= \mathbf{y}_1 + \mathbf{k}_2 = \begin{pmatrix} 14.4317776107 \\ 0.1212774833 \end{pmatrix} \\ \mathbf{y}_3 &= \begin{pmatrix} 30.2538910932 \\ 0.0653104260 \end{pmatrix} \\ \mathbf{y}_4 &= \begin{pmatrix} 62.2742345985 \\ 0.0322934446 \end{pmatrix} \end{aligned}$$

□

Higher order RK methods are derived in a similar way, except that the details can get quite tricky.

The current record, according to the Guinness book of world records, is held by A.R. Curtis, for an 18 stage formula of order 10 in 1975. (Thanks to Roger Alexander for this information).

Make sure you don't confuse "order" with "number of stages". A RK method of order  $n$  is one in which the local error is of order  $O(h^{n+1})$ . An  $n$ -stage method is one that uses  $k_1$  through  $k_n$ . The two happen to be the same for  $n = 2$ ,  $n = 3$  and  $n = 4$ , but not in general.

The most popular method is probably the four-stage fourth-order RK with constants  $\alpha_2 = \alpha_3 = 1/2$ ,  $\alpha_4 = 1$ ,  $\beta_{21} = \beta_{32} = 1/2$ ,  $\beta_{31} = \beta_{41} = \beta_{42} = 0$ ,  $\beta_{43} = 1$ ,  $\gamma_1 = \gamma_4 = 1/6$ ,  $\gamma_2 = \gamma_3 = 1/3$ . Written out, this gives you

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \\ k_3 &= hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right) \\ k_4 &= hf(t_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

This method is popular probably because it is nice and symmetric, and several  $\beta$  are zero. When somebody talks about "the" Runge-Kutta method, they probably mean this one.

**Example:** Again, our example

$$\begin{aligned} y'(t) &= -2ty^2(t) \\ y(0) &= 1 \end{aligned}$$

We again use  $h = 0.5$ ,  $t_0 = 0$ ,  $y_0 = 1$ . The first complete step is

$$\begin{aligned} k_1 &= 0 \\ k_2 &= -0.25 \\ k_3 &= -0.19140625 \\ k_4 &= -0.3269119263 \\ y_1 &= 0.7983792623 \end{aligned}$$

and the following values

$$\begin{aligned} y_2 &= 0.4997015229 \\ y_3 &= 0.3081669121 \\ y_4 &= 0.2004056722 \end{aligned}$$

so the global error is  $E_4 = 0.0004056722$ .

If we repeat the process with  $h = 0.25$ , we get

$$\begin{aligned} y_0 &= 1 \\ y_1 &= 0.941154013 \\ &\dots\dots \\ y_4 &= 0.5000135525 \\ &\dots\dots \\ y_8 &= 0.2000271443 \end{aligned}$$

with a global error  $E_8 = 0.0000271443$ .

The ratio of errors is  $14.9 \approx 16$ , confirming that this is indeed a fourth order method.  $\square$

We will leave even simple examples of this method for systems to the computers.

### 8.8. Implicit Methods

An *explicit method* calculates the numerical solution at  $t_{n+1}$  from values at  $t_n$  or earlier. An *implicit method* uses also values at  $t_{n+1}$ . Implicit methods may require that you solve a nonlinear equation at every step, but they are more accurate and more stable.

To derive a couple of simple implicit methods, rewrite

$$y' = f(t, y(t))$$

in the form

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t)) dt,$$

and replace the integral by a simple quadrature formula. With the notation

$$f_n = f(t_n, y_n)$$

we get the following methods:

Approximating the integral by a rectangle, using the value at the left endpoint, we get

$$y_{n+1} = y_n + hf_n,$$

which is Euler's method.

Using the right endpoint,

$$y_{n+1} = y_n + hf_{n+1},$$

which is called the *backward Euler* method.

Using the trapezoidal rule,

$$y_{n+1} = y_n + \frac{h}{2} [f_n + f_{n+1}],$$

which is called the *trapezoidal method*.

The local error is the error in the integration formula, so backward Euler has local error  $O(h^2)$  and is a first-order method, and the trapezoidal rule has local error  $O(h^3)$  and is a second-order method.

How do we actually take a step in an implicit method? If the equation is linear, we can solve for  $y_{n+1}$  by hand (see example below). Any method from chapter 7 (for a single equation) or chapter 9 (for systems of equations) can be used, but that is only necessary if the equation is very badly behaved. Most of the time, we use the *predictor-corrector* technique outlined in the next section.

**Example:** Consider the test equation

$$\begin{aligned} y' &= -y \\ y(0) &= 1 \end{aligned}$$

Using backward Euler, we get

$$y_{n+1} = y_n - hy_{n+1},$$

or

$$y_{n+1} = \frac{1}{1+h} y_n$$

Obviously, this is stable for any  $h$ , since  $y_n \rightarrow 0$  as  $n \rightarrow \infty$ .  $\square$

### 8.9. Multi-step Methods

My section 8.9 will cover the material in sections 8.9 through 8.11 in the book. Read the book sections on your own.

A *k-step method* is defined as a method which uses information from the last  $k$  values of  $y$  and  $f$  to find the next value. All methods discussed so far have been one-step methods.

The general form of an *explicit k-step method* is

$$y_{n+1} = \alpha_1 y_n + \alpha_2 y_{n-1} + \cdots + \alpha_k y_{n-k+1} + h [\beta_1 f_n + \beta_2 f_{n-1} + \cdots + \beta_k f_{n-k+1}]$$

The general form of an *implicit k-step method* is

$$y_{n+1} = \alpha_0 y_{n+1} + \alpha_1 y_n + \cdots + \alpha_k y_{n-k+1} + h [\beta_0 f_{n+1} + \beta_1 f_n + \cdots + \beta_k f_{n-k+1}]$$

**8.9.1. The Predictor-Corrector Scheme.** As mentioned above, implicit methods are usually both more accurate and more stable than explicit methods, but they are harder to apply. To understand the so-called *predictor-corrector* approach, we need to take a little detour.

Suppose we want to solve an equation of the form

$$x = g(x),$$

where  $g(x)$  is some function. A solution of this equation is called a *fixed point* of  $g$ . We could try to set up an iteration of the form

$$x_0 = \text{initial guess}$$

$$x_1 = g(x_0)$$

$$x_2 = g(x_1)$$

...

and so on, and hope that this converges to a fixed point. This method is called *fixed point iteration*. Sometimes it works, sometimes it doesn't.

From the material in chapter 7 it follows that *for small enough  $h$ , fixed point iteration will work for solving the equations in implicit k-step methods.*

The setup for a predictor-corrector method is this:

- We use an explicit method to get a good guess for  $y_{n+1}$ . This is called the *predictor step*.
- We calculate  $f_{n+1} = f(t_{n+1}, y_{n+1})$ . This is the *evaluation step*.
- We use an implicit method with these guesses for  $y_{n+1}$  and  $f_{n+1}$  on the right-hand side. This is the *corrector step* (one step of fixed point iteration, really). This gives us a new and improved  $y_{n+1}$ .
- We calculate a new and improved  $f_{n+1}$  with our new and improved  $y_{n+1}$  from the corrector step.

We could repeat the last two steps over and over again until we get convergence. We know that this will happen for small enough  $h$ . In practice, this is not efficient. Even if we found the exact  $y_{n+1}$ , this would still not be the true  $y(t_{n+1})$ . There are two components of error: one from replacing the ODE with a numerical method, and one from the fixed point iteration. The best place to stop is when both are about equal. One corrector step is usually sufficient for that. If you need more accuracy, it is more efficient to choose a smaller stepsize.

The method outlined above is referred to as a *PECE* scheme (predict-evaluate-correct-evaluate). Other schemes are possible, like *PEC*, *P(EC)<sup>2</sup>E*, and so on. *PECE* is the most commonly used.

**Example:** Apply the predictor-corrector scheme to our standard example

$$y'(t) = -2ty^2(t)$$

$$y(0) = 1$$

using Euler's method as a predictor, the trapezoidal method as a corrector. For stepsize  $h = 0.5$ , we find

$$\begin{array}{ll}
 t_0 = 0, & y_0 = 1 \\
 & f_0 = -2t_0y_0^2 = 0 \\
 t_1 = 0.5, & y_1 = y_0 + hf_0 = 1 \quad \text{(P, Euler's method)} \\
 & f_1 = -2t_1y_1^2 = -1 \quad \text{(E)} \\
 & y_1 = y_0 + \frac{h}{2}[f_0 + f_1] = 0.75 \quad \text{(C, trapezoidal method)} \\
 & f_1 = -2t_1y_1^2 = -0.5625 \quad \text{(E)} \\
 t_2 = 1, & y_2 = y_1 + hf_1 = 0.46875 \\
 & f_2 = -2t_2y_2^2 = -0.439453125 \\
 & y_2 = y_1 + \frac{h}{2}[f_1 + f_2] = 0.499511719 \\
 & f_2 = -2t_2y_2^2 = -0.499023914
 \end{array}$$

and so on.  $\square$

**8.9.2. The Adams-Bashforth-Moulton Methods.** The most commonly used multistep methods are *Adams-Bashforth* (predictor) and *Adams-Moulton* (corrector). Both are based on the same idea as the methods in section 8.8.

Replace the ODE

$$y'(t) = f(t, y(t))$$

by

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t)) dt,$$

replace  $f$  by an interpolating polynomial and integrate.

**Example:** Let us derive the two-step Adams-Bashforth predictor method in detail. Since  $k = 2$ , we use the last 2 values of  $f$ , namely  $f_{n-1}$  and  $f_n$ , and put a polynomial through the points  $(t_{n-1}, f_{n-1})$  and  $(t_n, f_n)$ . In this case, the polynomial is a straight line. We get

$$\begin{aligned}
 f(t, y(t)) &\approx \frac{t - t_n}{t_{n-1} - t_n} f_{n-1} + \frac{t - t_{n-1}}{t_n - t_{n-1}} f_n \\
 \int_{t_n}^{t_{n+1}} f(t, y(t)) dt &\approx \int_{t_n}^{t_{n+1}} \left[ \frac{t - t_n}{t_{n-1} - t_n} f_{n-1} + \frac{t - t_{n-1}}{t_n - t_{n-1}} f_n \right] dt \\
 &= \left[ \frac{(t - t_n)^2}{-2h} f_{n-1} + \frac{(t - t_{n-1})^2}{2h} f_n \right]_{t_n}^{t_{n+1}} \\
 &= \frac{3h}{2} f_n - \frac{h}{2} f_{n-1}.
 \end{aligned}$$

The method is therefore

$$y_{n+1} = y_n + h \left[ \frac{3}{2} f_n - \frac{1}{2} f_{n-1} \right]$$

$\square$

The most commonly used methods are 4-step Adams-Bashforth as a predictor

$$y_{n+1} = y_n + \frac{h}{24} [55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}]$$

and 3-step Adams-Moulton as a corrector

$$y_{n+1} = y_n + \frac{h}{24} [9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}]$$

Both of them are of order 4.

In general, the  $k$ -step Adams-Bashforth method is of order  $k$ , the  $k$ -step Adams-Moulton method is of order  $k + 1$ . You can convince yourself of that as follows: The  $k$ -step Adams-Bashforth method uses interpolation at  $k$  points. The interpolation error is  $O(h^k)$ . When we integrate, we pick up another factor of  $h$  (from the interval width). Thus, the local error is  $O(h^{k+1})$ , and the global error is  $O(h^k)$ . The  $k$ -step Adams-Moulton method uses  $(k + 1)$  points. By the same reasoning, it is of order  $(k + 1)$ .

**Example:** We use our standard example again,

$$\begin{aligned}y'(t) &= -2ty^2(t) \\ y(0) &= 1\end{aligned}$$

We use  $h = 0.25$ , the fourth order Adams-Bashforth-Moulton method (4-step predictor, 3-step corrector). startup. To get going, we need values for the first 4 points. I used the Runge-Kutta method to get those.

$$\begin{array}{ll}y_0 = 1, & f_0 = 0 \\ y_1 = 0.9411540130, & f_1 = -0.442885438 \\ y_2 = 0.7999481032, & f_2 = -0.639916968 \\ y_3 = 0.6399738841, & f_3 = -0.614349858\end{array}$$

Then we get the predictor step

$$y_4 = y_3 + \frac{h}{24} [55f_3 - 59f_2 + 37f_1 - 9f_0] = 0.5105894849,$$

the evaluation step

$$f_4 = -0.521403244,$$

the corrector step

$$y_4 = y_3 + \frac{h}{24} [9f_4 + 19f_3 - 5f_2 + f_1] = 0.4982178726$$

and finally another evaluation step

$$f_4 = -0.496442097$$

The next values are

$$\begin{array}{ll}y_5 &= 0.3862529501 \quad (\text{predictor}) \\ &= 0.390328576 \quad (\text{corrector}) \\ y_6 &= 0.3090162813 \quad (\text{corrector}) \\ y_7 &= 0.2472511935 \quad (\text{corrector}) \\ y_8 &= 0.2007863546 \quad (\text{corrector})\end{array}$$

The global error at the end is  $E_8 \approx 0.000786$ .  $\square$

### 8.9.3. Stability.

Recall the general form of a  $k$ -step method

$$y_{n+1} = \alpha_0 y_{n+1} + \alpha_1 y_n + \cdots + \alpha_k y_{n-k+1} + h [\beta_0 f_{n+1} + \beta_1 f_n + \cdots + \beta_k f_{n-k+1}].$$

If  $\alpha_0 = \beta_0 = 0$ , the method is explicit, otherwise implicit.

If we apply this method to the test problem

$$y' = \lambda y,$$

we get

$$y_{n+1} = (\alpha_0 + h\lambda\beta_0)y_{n+1} + (\alpha_1 + h\lambda\beta_1)y_n + (\alpha_2 + h\lambda\beta_2)y_{n-1} + \cdots + (\alpha_k + h\lambda\beta_k)y_{n-k+1}.$$

This is a so-called *difference equation*. Difference equations are like discrete versions of differential equations, in many respects, and have a well-developed theory. The solutions of difference equations are infinite sequences  $\{y_j\}$ .

Define the *characteristic equation* of the difference equation as

$$r^k = (\alpha_0 + h\lambda\beta_0)r^k + (\alpha_1 + h\lambda\beta_1)r^{k-1} + (\alpha_2 + h\lambda\beta_2)r^{k-2} + \cdots + (\alpha_k + h\lambda\beta_k)r^0$$

and the *reduced characteristic equation* as the characteristic equation with  $h\lambda = 0$

$$r^k = \alpha_0 r^k + \alpha_1 r^{k-1} + \alpha_2 r^{k-2} + \cdots + \alpha_k r^0.$$

The theory of difference equations says that a  $k$ th order difference equation has  $k$  linearly independent solutions  $y_j$ ,  $j = 1, \dots, k$  of the form

$$(y_j)_n = r_j^n$$

where  $r_j$  is a root of the characteristic equation. The general solution is

$$y = c_1 y_1 + \cdots + c_k y_k,$$

where the  $c_j$  are arbitrary constants. This is very similar to the theory of constant coefficient ODEs.

**Example:** Euler's method is

$$y_{n+1} = y_n + hf(t_n, y_n).$$

For the sample problem, this becomes

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n.$$

The characteristic equation is

$$r = 1 + h\lambda,$$

the reduced characteristic equation is

$$r = 1.$$

□

**Example:** The trapezoidal method

$$y_{n+1} = y_n + \frac{h}{2} [y_n + y_{n+1}]$$

has characteristic equation

$$r = 1 + \frac{h\lambda}{2}(1 + r)$$

and reduced characteristic equation

$$r = 1.$$

□

**Example:** The second order Runge-Kutta method with  $\alpha = 0.5$  is given by

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf(t_n + h/2, y_n + k_1/2)$$

$$y_{n+1} = y_n + k_2$$

For the sample problem, this becomes

$$k_1 = h\lambda y_n$$

$$k_2 = h\lambda[y_n + k_1/2]$$

$$= h\lambda y_n + (h\lambda)^2 y_n/2$$

$$y_{n+1} = y_n + k_2 = y_n + h\lambda y_n + (h\lambda)^2 y_n/2$$

The characteristic equation is

$$r = 1 + h\lambda + (h\lambda)^2/2,$$

the reduced characteristic equation is

$$r = 1.$$

□

**Example:** For the 4-step Adams-Bashforth method

$$y_{n+1} = y_n + \frac{h}{24} [55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}]$$

we have for the sample problem

$$y_{n+1} = y_n + \frac{h\lambda}{24} [55y_n - 59y_{n-1} + 37y_{n-2} - 9y_{n-3}]$$

The characteristic equation is

$$r^4 = r^3 + \frac{h\lambda}{24} [55r^3 - 59r^2 + 37r - 9],$$

the reduced characteristic equation is

$$r^4 = r^3.$$

□

Theory now tells us the following facts:

- The method is stable at stepsize  $h$  for a given  $\lambda$  if all solutions of the characteristic equation have absolute value less than or equal to 1, and all multiple solutions have absolute value less than 1. The method is unstable if one or more of the solutions have absolute value greater than 1. Note that we allow complex  $\lambda$ , so this is the complex absolute value.
- The reduced characteristic equation always has the solution  $r = 1$ . The method is stable for small values of  $h\lambda$  if all the other solutions have absolute value less than 1.

Again, the *region of stability* of a method is defined as the set of all points  $(h\lambda)$  in the complex plane for which the method is stable.

Let us consider our examples again:

**Example:** Euler's method has reduced characteristic equation

$$r = 1$$

This equation has the solution  $r = 1$ , as predicted, and no others. Euler's method is therefore stable for small values of  $h\lambda$  (we already knew that).

The full characteristic equation is

$$r = 1 + h\lambda$$

It has exactly one solution, namely  $r = 1 + h\lambda$ , so Euler's method is stable whenever

$$|1 + h\lambda| < 1.$$

We knew that, too. The region of stability is a circle of radius 1 around the point  $(-1)$ . The real part of it is the interval  $[-2, 0]$ .  $\square$

**Example:** The trapezoidal method has the reduced characteristic equation

$$r = 1,$$

so it is stable for small values of  $h\lambda$ . The full characteristic equation

$$r = 1 + \frac{h\lambda}{2}(1 + r)$$

has solution

$$r = \frac{1 + h\lambda/2}{1 - h\lambda/2}$$

It is stable whenever

$$\left| \frac{1 + h\lambda/2}{1 - h\lambda/2} \right| < 1.$$

It can be shown that this is exactly the left half plane.  $\square$

**Example:** The second order RK method has the same reduced characteristic equation

$$r = 1$$

and is also stable for small  $h\lambda$ . (Remark: any one-step method has reduced characteristic equation  $r = 1$ . Think about it).

The full characteristic equation is

$$r = 1 + h\lambda + (h\lambda)^2/2$$

The method is stable whenever

$$|1 + h\lambda + (h\lambda)/2| < 1.$$

The region of stability is harder to find here, but looks similar to the circle we got for Euler's method. The real part of it is again the interval  $[-2, 0]$  (see picture).  $\square$

**Example:** 4-step Adams-Bashforth has the reduced characteristic equation

$$r^4 = r^3.$$

It has one solution  $r = 1$ , as predicted, and the triple solution  $r = 0$ . All solutions other than  $r = 1$  have absolute values less than 1, so this method is again stable for small  $h\lambda$ .

The characteristic equation itself is a fourth degree polynomial

$$r^4 = r^3 + \frac{h\lambda}{24}[55r^3 - 59r^2 + 37r - 9].$$

For each complex  $h\lambda$ , we need to find the four (complex) solutions, take the absolute value of each and decide whether they are all less than or equal to 1 or not. Then we can decide whether this particular  $h\lambda$  belongs in the region of stability or not.

Fortunately, somebody has done this before (see picture).  $\square$

Some regions of stability are pictured below. Notice that sometimes the region sticks out into the right half plane a little. What that means in practice is that the method is a little too stable. The true solution is exponentially growing, while the numerical solution goes to zero.

Also notice that all of these methods have fairly small regions of stability, only up to -2 or -3.

### 8.10. Order and Error of a Multi-step Method

This section in the book is covered in my section 8.9.

### 8.11. Stability for Multi-step Methods

This section in the book is covered in my section 8.9.

### 8.7. Subroutine SDRIV2

Read this section for yourself.