1 Eulers method.

Let us start this introduction to the numerical solution of *ordinary differential equations* (ODEs) by something familiar. Given a scalar (one equation only) ODE

$$y' = f(t,y), \qquad t_0 \le t \le t_{end}, \qquad y(t_0) = y_0, \tag{1}$$

in which the function f, the integration interval $[t_{0,}t_{end}]$ and the initial value y_0 is assumed to be given. The *solution* of this initial value problem (IVP) is a function $y(t)$ on the interval $[t_0, t_{end}]$.

Example 1.1. *The ODE/IVP*

$$y' = -2ty, \qquad 0 \le t \le 1, \qquad y(0) = 1.0$$

has as solution the function

$$y(t) = e^{-t^2}.$$

But in many practical situations, it is not possible to express the solution $y(t)$ in closed form, even if a solution exist. In these cases, a numerical algorithm can give an *approximation* to the exact solution. Let us start with Eulers method, which should be known from some calculus classes. Divide the interval $[t_0, t_{end}]$ into $Nstep$ equal subintervals, each of size $h = (t_{end} - t_0)/Nstep$, and let $t_n = t_0 + nh$. Euler's method can be derived by several means. One possibility is to use the first few terms of the Taylor expansion of the exact solution, which is given by

$$y(t_0+h) = y(t_0)+hy'(t_0)+\frac{1}{2}h^2y''(t_0)+\cdots+\frac{1}{p!}h^py^{(p)}(t_0)+\frac{1}{(p+1)!}h^{p+1}y^{(p+1)}(\xi), \tag{2}$$

where ξ is somewhere between t_0 and t_{end}. The integer $p \ge 1$ is a number of our own choice, but we have to require y to be sufficiently differentiable, in this case that $y^{(p+1)}$ exist and is continuous. If h is small, we may assume that the solution will be completely dominated by the first two terms, thus

$$y(t_0 + h) \approx y(t_0) + hy'(t_0) = y_0 + hf(t_0, y_0),$$

and we call this approximate solution y_1. Starting from the point $t_1 = t_0+h$ and y_1 we can repeat the process. We have now developed Euler's method, given by

$$y_{n+1} = y_n + hf(t_n, y_n), \qquad n = 0, 1, \cdots, Nsteps - 1,$$

resulting in approximations $y_n \approx y(t_n)$.

Example 1.2. *Eulers method with* $h = 0.1$ *applied to the ODE of Example 1.1 gives*

t_n	y_n
0.0	1.0000
0.1	1.0000
0.2	0.9800
0.3	0.9408
0.4	0.8844
0.5	0.8136
0.6	0.7322
0.7	0.6444
0.8	0.5542
0.9	0.4655
1.0	0.3817

.

In this case we know the exact solution, $y(1.0) = e^{-1.0^2} = 0.3679$ *and the error at the endpoint is* $e_{10} = y(1.0) - y_{10} = -1.38 \cdot 10^{-2}$. *If we repeat this experiment (write a MATLAB program to do so) with different stepsizes, and measure the error at the end of the interval, we get*

h	$e_{Nstep} = y(1.0) - y_{Nstep}$
0.1	$-1.38 \cdot 10^{-2}$
0.05	$-6.50 \cdot 10^{-3}$
0.025	$-3.16 \cdot 10^{-3}$
0.0125	$-1.56 \cdot 10^{-3}$

.

From this example, it might look like the error at the endpoint $e_{Nstep} \sim h$, where $h = (tend - t_0)/Nstep$. But is this true for all problems, and if yes, can we prove it? To do so, we need to see what kind of errors we have and how they behave. This is illustrated in Figure 1. For each step an error is made, and these errors are then propagated til the next steps and accumulate at the endpoint.

Definition 1.3. *The* local truncation error d_{n+1} *is the error done in one step when starting at the exact solution* $y(t_n)$. *The* global error *is the difference between the exact and the numerical solution at point* t_n, *thus* $e_n = y(t_n) - y_n$.

The local truncation error of Euler's method is

$$d_{n+1} = y(t_n + h) - y(t_n) - hf(t_n, y(t_n)) = \frac{1}{2}h^2 y''(\xi),$$

where $\xi \in (t_n, t_{n+1})$. This is given from the Taylor-expansion of $y(t_n+h)$ around t_n with $p = 1$. To see how the global error propagates from one step to the next, the trick is: We have

$$\begin{aligned} y(t_n + h) &= y(t_n) + hf(t_n, y(t_n)) + d_{n+1}, \\ y_{n+1} &= y_n + hf\left(t_n, y_n\right). \end{aligned}$$

Take the difference of these two, and get

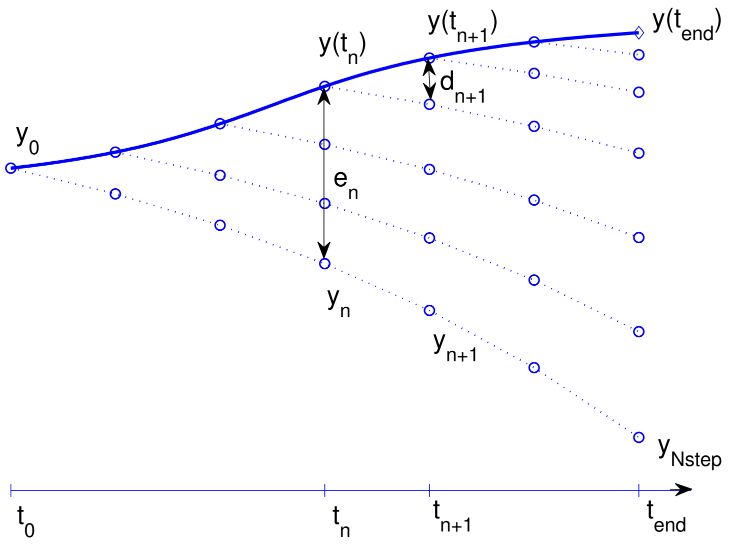

Figure 1: Lady Windermere's Fan

$$e_{n+1} = e_n + h\left(f\left(t_n, y(t_n)\right) - f\left(t_n, y_n\right)\right) + d_{n+1} = (1 + hf_y(t_n, v))e_n + d_{n+1}, \quad (3)$$

where v is somewhere between $y(t_n)$ and y_n. We have here used the mean value theorem (Theorem 1.8 in Burden and Faires) for $f\left(t_n, y(t_n)\right) - f\left(t_n, y_n\right)$. This is about as far as we get with exact calculations, since ξ in d_{n+1} as well as v in f_y are unknown, and will also change from one step to the next. So we will look for an *upper bound* of the global error. We will first assume upper bounds for our unknown, that is, we assume there exist positive constants D and L so that

$$\frac{1}{2}\left|y''\right| \le D \text{ for all } t \in (t_0, t_{end}) \qquad \text{and} \qquad |f_y| \le L \text{ for all } t \in [t_0, t_{end}] \text{ and for all } y.$$

Taken the absolute value of both sides of (3) and using the triangle inequality gives

$$|e_{n+1}| \le (1 + hL)\,|e_n| + Dh^2.$$

Since $e_0 = 0$ (there is no error at the initial point) we can use this formula recursively to get an upper bound for the error at the endpoint:

$$\begin{aligned}
|e_1| &\le Dh^2, \\
|e_2| &\le (1 + hL)Dh^2 + Dh^2 \\
&\vdots \\
|e_{Nstep}| &\le \sum_{i=0}^{Nstep-1} (1 + hL)^i Dh^2 = \frac{(1 + hL)^{Nstep} - 1}{1 + hL - 1} Dh^2.
\end{aligned}$$

Using the fact that $1 + hL \leq e^{hL}$ (why?) and $h \cdot Nstep = t_{end} - t_0$ we finally reach the conclusion

$$|e_{Nstep}| \leq \frac{\left(e^{hL}\right)^{Nstep} - 1}{L} Dh = \frac{e^{L(t_{end}-t_0)} - 1}{L} D \cdot h = C \cdot h.$$

The constant $C = \left(e^{hL} - 1\right) D/L$ depends only on the problem, and we have proved convergence

$$|y(t_{end}) - y_{Nstep}| \to 0 \text{ when } \; h \to 0 \text{ (or } Nstep \to \infty).$$

Summary: In this section we have

1. Formulated the problem.
2. Developed an algorithm.
3. Implemented and tested it.
4. Proved convergence.

This is fairly much what this course in Numerical Mathematics is about. To be more precise, for each class of problems (ODEs, integrals, linear and nonlinear equations,$\cdots$) we will roughly do the following

1. Formulate the problem. What do we know about it? What about existence and uniqueness results?
2. Develop an algorithm to find a solution. Can it be generalised?
3. Implement and test the algorithm. Test it on problems with known solutions. Do we get a reasonable result?
4. Prove convergence. How accurate is the algorithm?
5. Verify the theoretical results by numerical experiments.
6. Try the algorithm on harder problems. Do we get unexpected results? If yes, can they be explained? Can we get around them?
7. Can we make our algorithm to automatically adjust itself to fulfil requirements given by the user (adaptivity)?

The last two points do not apply to all the problems we are going to discuss. We will also derive mathematical results that are useful for either the theoretical analysis, or for development of methods.