Stiff differential equations.

Anne Kværnø

Nov 3, 2020

With revisions by **Markus Grasmair**.

1 Introduction

When an ODE is solved by an adaptive solver we will expect that more steps are required for stricter tolerances. Specifically, the step size control is based on the assumption that the local error estimate $\mathbf{le}_{n+1}$ satisfies

$$\|\mathbf{le}_{n+1}\| \approx D h_n^{p+1} \approx \text{Tol},$$

where p is the order of the lowest order method, and D is independent of the step size h. The constant D depends on the solution point $(x, \mathbf{y}_n)$, but it will usually not change much from one step to the next.

If we use different tolerances Tol_1 and Tol_2 for the solution of the same problem with the same adaptive method, we will therefore expect that the step sizes h_1 and h_2 near the same point will behave like

$$\text{Tol}_1 \approx D h_1^{p+1}, \qquad \text{Tol}_2 \approx D h_2^{p+1},$$

so that

$$\frac{h_1}{h_2} \approx \left(\frac{\text{Tol}_1}{\text{Tol}_2}\right)^{\frac{1}{p+1}} \approx \frac{N_2}{N_1}.$$

where N_1 and N_2 are the total number of steps used for the two tolerances.

In the case of the Heun-Euler scheme, the lower order is $p = 1$. By reducing the tolerance by a factor $1/100$ we will expect that the number of steps increases by a factor of 10.

Numerical example 1: Given the following system of two ODEs

$$\begin{aligned} y_1' &= -2y_1 + y_2 + 2\sin(x), & y_1(0) &= 2, \\ y_2' &= (a-1)y_1 - a y_2 + a\big(\cos(x) - \sin(x)\big), & y_2(0) &= 3, \end{aligned}$$

where a is some positive parameter. The exact solution, which is independent of the parameter, is

$$y_1(x) = 2e^{-x} + \sin(x), \qquad y_2(x) = 2e^{-x} + \cos(x).$$

Solve this problem now with some adaptive ODE solver, for instance the Heun-Euler scheme.

Now try the tolerances $\text{Tol} = 10^{-2}$, 10^{-4}, 10^{-6}, and perform the experiment with two different values of the parameters, $a = 2$ and $a = 999$.

```python
# Numerical example 1s
# Define the function
def f(x, y):
    a = 2
    dy = array([-2*y[0]+y[1]+2*sin(x),
                (a-1)*y[0]-a*y[1]+a*(cos(x)-sin(x))])
    return dy

# Initial values and integration interval
y0 = array([2, 3])
x0, xend = 0, 10
h0 = 0.1

tol = 1.e-2
# Solve the ODE using different tolerances
for n in range(3):
    print('\nTol = {:.1e}'.format(tol))
    x_num, y_num = ode_adaptive(f, x0, xend, y0, h0, tol, method=heun_euler)

    if n==0:
        # Plot the solution
        subplot(2,1,1)
        plot(x_num, y_num)
        ylabel('y')
        subplot(2,1,2)

    # Plot the step size control
    semilogy(x_num[0:-1], diff(x_num), label='Tol={:.1e}'.format(tol));

    tol = 1.e-2*tol         # Reduce the tolerance by a factor 0.01.
xlabel('x')
ylabel('h')
legend(loc='center left', bbox_to_anchor=(1, 0.5));
```

For $a = 2$ the expected behaviour is observed. But the example $a = 999$ requires much more steps, and the step size seems almost independent of the tolerance, at least for $\mathrm{Tol} = 10^{-2}, 10^{-4}$.

The example above with $a = 999$ is a typically example of a *stiff ODE*. What defines these types of ODEs is that there are (at least) two different time scales at play at the same time: a slow time scale that dominates the time evolution of the solution of the ODE, and a fast time scale at which small perturbations of the solution may occur. In physical systems, this might be due to very strong damping of selected components of the system.

If we consider for instance the ODE in the numerical example above, then we obtain, after some computation, that the general solution is

$$\mathbf{y}(x) = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} e^{-x} + c_2 \begin{pmatrix} -1 \\ a-1 \end{pmatrix} e^{-(a+1)x} + \begin{pmatrix} \sin(x) \\ \cos(x) \end{pmatrix}$$

for some constants c_1, c_2. The terms e^{-x}, $\sin(x)$, and $\cos(x)$ evolve at a time scale of order 1. In contrast, the term $e^{-(a+1)x}$ reverts back to being essentially equal to zero at a time scale of order $1/(a+1)$.

When a stiff ODE is solved by some explicit adaptive method like the Heun-Euler scheme, an unreasonably large number of steps is required, and this number seems independent of the tolerance. The problem is that, for explicit methods, the local error is dominated by what is happening at the fast time scale, and the step length will be adapted to that time scale as well. Even worse, any larger step size will lead to instabilities and exponentially increasing oscillations in the numerical solution.

In the remaining part of this note we will explain why this happens, and how we can overcome the problem. For simplicity, the discussion is restricted to linear problems, but also nonlinear ODEs can be stiff, and often will be.

Exercise 1: Repeat the experiment on the Van der Pol equation

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 2, \\ y_2' &= \mu(1-y_1^2)y_2 - y_1, & y_2(0) &= 0. \end{aligned}$$

Use $\mu = 2$, $\mu = 5$ and $\mu = 50$.

1.1 Linear stability analysis

Motivation. We are given a system of m differential equation of the form

$$\mathbf{y}' = A\mathbf{y} + \mathbf{g}(x). \tag{*}$$

Such systems have been discussed in Mathematics 3, and the technique for finding the exact solution will shortly be repeated here:

Solve the homogenous system $\mathbf{y}' = A\mathbf{y}$, that is, find the eigenvalues λ_i and the corresponding eigenvectors $\mathbf{v}_i$ satisfying

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \qquad i = 1, 2, \ldots, m. \tag{**}$$

Assume that A has a full set of linearly independent (complex) eigenvectors $\mathbf{v}_i$ with corresponding (complex) eigenvalues λ_i. Let $V = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$, and $\Lambda = \operatorname{diag}\{\lambda_1, \ldots, \lambda_m\}$. Then V is invertible and

$$AV = V\Lambda \qquad \text{and therefore} \qquad V^{-1}AV = \Lambda.$$

The ODE (*) can thus be rewritten as

$$V^{-1}\mathbf{y}' = V^{-1}AVV^{-1}\mathbf{y} + V^{-1}\mathbf{g}(x).$$

Let $\mathbf{z} = V^{-1}\mathbf{y}$ and $\mathbf{q}(x) = V^{-1}\mathbf{g}(x)$ such that the equation can be decoupled into a set of independent scalar differential equations

$$\mathbf{z}' = \Lambda\mathbf{z} + \mathbf{q}(x) \qquad \text{or, equivalently} \qquad z_i' = \lambda_i z_i + q_i(x), \quad i = 1, \ldots, m.$$

The solution of such equations has been discussed in Mathematics 1. When these solutions are found, the exact solution is given by

$$\mathbf{y}(x) = V\mathbf{z}(x),$$

and possible integration constants are given by the initial values.

As it turns out, the eigenvalues $\lambda_i \in \mathbb{C}$ are the key to understanding the behaviour of the adaptive integrators. So we will discuss the stability properties of the very simplified, though complex, linear test equation

$$y' = \lambda y.$$

The discussion below is also relevant for nonlinear ODEs $\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x))$, in which case we have to consider the eigenvalues of the Jacobian $\mathbf{f}_\mathbf{y}$ of $\mathbf{f}$ with respect to $\mathbf{y}$.

Example 1: We now return to the introductory example. There, the ODE can be written as

$$\mathbf{y}' = A\mathbf{y} + \mathbf{g}(x),$$

with

$$A = \begin{pmatrix} -2 & 1 \\ a-1 & -a \end{pmatrix}, \qquad \mathbf{g}(x) = \begin{pmatrix} \sin(x) \\ a(\cos(x) - \sin(x)) \end{pmatrix}.$$

The eigenvalues of the matrix A are $\lambda_1 = -1$ and $\lambda_2 = -(a+1)$. The general solution is given by

$$\mathbf{y}(x) = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} e^{-x} + c_2 \begin{pmatrix} -1 \\ a-1 \end{pmatrix} e^{-(a+1)x} + \begin{pmatrix} \sin(x) \\ \cos(x) \end{pmatrix}.$$

In the introductory example, the initial values were chosen such that $c_1 = 2$ and $c_2 = 0$. However, for large values of a, the term $e^{-(a+1)x}$ will still go to 0 almost immediately, even if $c_2 \neq 0$. It is this term that creates problems for the numerical solution.

Stability functions and stability regions. We consider the linear test equation

$$y' = \lambda y, \qquad y(0) = y_0,$$

where the parameter $\lambda \in \mathbb{C}$ satisfies

$$\Re\lambda < 0.$$

Here, and in the following, $\Re\lambda$ will denote the real part of λ, and $\Im\lambda$ will denote the imaginary part. The analytic solution of this problem is

$$y(x) = y_0\, e^{\lambda x} = y_0\, e^{\Re\lambda\, x}\big(\cos(\Im\lambda\, x) + i\sin(\Im\lambda\, x)\big).$$

Since $\Re\lambda < 0$, the solution $y(x)$ tends to zero as $x \to \infty$. We want a similar behaviour for the numerical solution, that is $|y_n| \to 0$ when $n \to \infty$.

One step of some Runge–Kutta method applied to the linear test equation can always be written as

$$y_{n+1} = R(z)y_n, \qquad z = \lambda h.$$

The function $R(z)\colon \mathbb{C} \to \mathbb{C}$ is called the *stability function* of the method.

Example 2: The application of Euler's method for the linear test equation results in the iteration

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n = (1+z)y_n \qquad \text{with } z = h\lambda.$$

The stability function of Euler's method is therefore the function

$$R(z) = 1 + z.$$

For a comparison, Heun's method for this test equation is

$$\begin{aligned} k_1 &= \lambda y_n, \\ k_2 &= \lambda(y_n + hk_1), \\ y_{n+1} &= y_n + \frac{h}{2}(k_1 + k_2), \end{aligned}$$

which can be rewritten as

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

As a consequence, the stability function for Heun's method is

$$R(z) = 1 + z + \frac{z^2}{2}.$$

We now return back to the analysis of the behaviour of an arbitrary Runge-Kutta method with stability function R. Taking the absolute value on each side of the expression

$$y_{n+1} = R(z)y_n,$$

we see that there are three possible outcomes:

$$\begin{array}{llllllll} |R(z)| < 1 & \Rightarrow & |y_{n+1}| < |y_n| & \Rightarrow & y_n \to 0 & \text{(stable)} \\ |R(z)| = 1 & \Rightarrow & |y_{n+1}| = |y_n| & & & \\ |R(z)| > 1 & \Rightarrow & |y_{n+1}| > |y_n| & \Rightarrow & |y_n| \to \infty & \text{(unstable)} \end{array}$$

The *stability region* of a method is defined by

$$\mathcal{S} = \{z \in \mathbb{C} \ :\ |R(z)| \le 1\}.$$

To get a stable numerical solution, we have to choose the step size h such that $z = \lambda h \in \mathcal{S}$.

Example 2, continued: In the case of Euler's method, we have obtained the stability function

$$R(z) = 1 + z.$$

As a consequence, the stability region for Euler's method is

$$\mathcal{S} = \{z \in \mathbb{C} \,:\, |1+z| \leq 1\}.$$

This is a ball in the complex plane, which is centered at -1 and has a radius of 1.

Numerical example 2: We have already discussed the stability function and stability region for Euler's method in the example above. We now solve the introductory problem

$$\mathbf{y}' = \begin{pmatrix} -2 & 1 \\ a-1 & -a \end{pmatrix} \mathbf{y} + \begin{pmatrix} \sin(x) \\ a(\cos(x) - \sin(x)) \end{pmatrix}, \qquad \mathbf{y}(0) = \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \qquad a > 0.$$

by Euler's method. We know that the eigenvalues of the matrix A are $\lambda_1 = -1$ and $\lambda_2 = -(1+a)$.

For the numerical solution to be stable for both eigenvalues, we have to require that the step length h satisfies

$$|1 + \lambda_1 h| \leq 1 \qquad \text{and} \qquad |1 + \lambda_2 h| \leq 1.$$

Since both eigenvalues in this case are real and negative, we see after a short computation that this results in the requirement that

$$h \leq \frac{2}{1+a}.$$

Try $a = 9$ and $a = 999$. Choose step sizes a little bit over and under the stability boundary, and you can experience that the result is sharp. If h is just a tiny bit above, you may have to increase the interval of integration to see the unstable solution.

```python
# Numerical example 2s
def f(x, y):
    # y' = f(x,y) = A*y+g(x)
    a = 9
    dy = array([-2*y[0]+y[1]+2*sin(x),
                (a-1)*y[0]-a*y[1]+a*(cos(x)-sin(x))])
    return dy

# Initial value and integration interval
y0 = array([2, 3])
x0, xend = 0, 10
h = 0.19

x_num, y_num = ode_solver(f, x0, xend, y0, h, method=euler)
plot(x_num, y_num);
```

It is the term corresponding to the eigenvalue $\lambda_2 = -(a+1)$ which makes the solution unstable. And the solution oscillate since $R(z) < -1$ for $h > 2/(1+a)$.

Exercise 2:

1. Find the stability region for Heun's method.
2. Repeat the experiment in Example 2 using Heun's mehod.

NB! Usually the error estimation in adaptive methods will detect the unstability and force the step size to stay inside or near the stability interval. This explains the behaviour of the experiment in the introduction of this note.

2 $A(0)$-stable methods.

In an ideal world, we would prefer the stability interval to satisfy

$$\mathcal{S} \supset \mathbb{C}^- := \{z \in \mathbb{C} \;:\; \Re z \leq 0\},$$

such that the method is stable for all $\lambda \in \mathbb{C}$ with $\Re\lambda \leq 0$ and for all h. Such methods are called $A(0)$-stable. For all explicit methods, like Euler's and Heun's, the stability function will be a polynomial, and $|R(z)| \to \infty$ as $\Re z \to -\infty$. Explicit methods can never be $A(0)$-stable, and we therefore have to search among implicit methods. The simplest of those is the implicit, or backward, Euler's method, given by

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

Applied to the linear test equation $y' = \lambda y$, this results in the update

$$y_{n+1} = y_n + h\lambda y_{n+1} \qquad \text{or} \qquad y_{n+1} = \frac{1}{1-h\lambda} y_n.$$

We therefore see that we have the stability function

$$R(z) = \frac{1}{1-z}.$$

The stability region for the implicit Euler function is thus

$$\mathcal{S} = \left\{ z \in \mathbb{C} \;:\; \left| \frac{1}{1-z} \right| \leq 1 \right\} = \{z \in \mathbb{C} \;:\; |1-z| \geq 1\}.$$

This is the whole complex plan apart from an open ball with center $+1$ and radius 1. Thus the method is $A(0)$-stable, as every complex number z with $\Re z \leq 0$ is contained in $\mathcal{S}$.

2.1 Implementation of implicit Euler's method

For simplicity, we will only discuss the implementation of implicit Euler's method for linear systems of the form

$$\mathbf{y}' = A\mathbf{y} + \mathbf{g}(x),$$

where A is a constant matrix. In this case, one step of implicit Euler is given by

$$\mathbf{y}_{n+1} = \mathbf{y}_n + hA\mathbf{y}_{n+1} + h\mathbf{g}(x_{n+1}).$$

Thus a linear system

$$(I - hA)\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{g}(x_{n+1})$$

has to be solved with respect to $\mathbf{y}_{n+1}$ for each step.

In the implementation below, the right hand side of the ODE is implemented as a function `rhs`, returning the matrix A and the vector $\mathbf{g}(x)$ for each step. The function `implicit_euler` does one step with implicit Euler. It has the same interface as the explicit method, so that the function `ode_solve` can be used as before.

```
def implicit_euler(rhs, x, y, h):
    '''
    One step of the implicit Euler's method on the problem
                y' = Ay + g(x)
    The function rhs should return A and g for each x
        A, gx = rhs(x)
    '''

    A, gx = rhs(x+h)
    d = len(gx)                   # The dimension of the system
    M = eye(d)-h*A                # M = I-hA
    b = y + h*gx                  # b = y + hf(x)
    y_next = solve(M, b)          # Solve M y_next = b
    x_next = x+h
    return x_next, y_next
```

Numerical example 3: Solve the test equation with

$$A = \begin{pmatrix} -2 & 1 \\ a-1 & -a \end{pmatrix}, \qquad \mathbf{g}(x) = \begin{pmatrix} \sin(x) \\ a(\cos(x) - \sin(x)) \end{pmatrix},$$

by the implicit Euler method. Choose $a = 2$ and $a = 999$, and try different stepsizes like $h = 0.1$ and $h = 0.01$. Are there any stability issues in this case?

```
# Numerical example 3s
def rhs(x):
    # The right hand side (rhs) of y' = Ay + g(x)
    a = 9
    A = array([[-2, 1],[a-1, -a]])
    gx = array([2*sin(x), a*(cos(x)-sin(x))])
    return A, gx

# Initial values and integration interval
y0 = array([2, 3])
x0, xend = 0, 10
h = 0.2             # Initial stepsize

x_num, y_num = ode_solver(rhs, x0, xend, y0, h, method=implicit_euler)
plot(x_num, y_num);
```

Exercise 2: The trapezoidal rule is an implicit method which for a general ODE $\mathbf{y}'(x) = f(x, \mathbf{y}(x))$ is given by

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{2}\Big(\mathbf{f}(x_n, \mathbf{y}_n) + \mathbf{f}(x_{n+1}, \mathbf{y}_{n+1})\Big).$$

1. Find the stability function to the trapezoidal rule, and prove that it is $A(0)$-stable.
2. Implement the method, and repeat the experiment above.

2.2 Adaptive methods.

Implicit Euler is a method of order 1, and the trapezoidal rule of order 2. Thus, these can be used for error estimation: Perform one step with each of the methods, use the difference between the solutions as an error estimate, and use the solution from the trapezoidal rule to advance the solution. This has been implemented in the function `trapezoidal_ieuler`. The interface is as for the embedded pair `heun_euler`, so the adaptive solver `ode_adaptive` can be used as before.

```
def trapezoidal_ieuler(rhs, x, y, h):
    '''
    One step with the combination of implicit Euler and the trapezoidal rule
    for ODEs on the form
                y' = Ay + f(x)
    The function rhs should return A and g for each x:
        A, fx = rhs(x)
    '''

    A, gx1 = rhs(x+h)
    A, gx0 = rhs(x)
    d = len(gx1)

    # One step with implicit Euler
    M = eye(d)-h*A
    b = y + h*gx1
    y_ie = solve(M, b)
    # One step with the trapezoidal rule
    M = eye(d)-0.5*h*A
    b = y + 0.5*h*dot(A,y) + 0.5*h*(gx0+gx1)
    y_next = solve(M, b)                       # The solution in the next step

    error_estimate = norm(y_next-y_ie)
    x_next = x + h
    p = 1                                       # The order
    return x_next, y_next, error_estimate, p
```

Numerical example 4: Repeat the experiment from the introduction, using `trapezoidal_euler`.

```
# Numerical example 4s
def rhs(x):
    # The right hand side of the ODE y' = Ay+g(x)
    a = 9
    A = array([[-2, 1],[a-1, -a]])
    gx = array([2*sin(x), a*(cos(x)-sin(x))])
    return A, gx

# Initial values and integration interval
y0 = array([2, 3])
x0, xend  = 0, 10
h0 = 0.1                    # Initial stepsize

tol = 1.e-2                 # Tolerance

rcParams['figure.figsize'] = 8, 8
# Solve the equation by different stepsizes.
for n in range(3):
    print('\nTol = {:.1e}'.format(tol))
    x_num, y_num = ode_adaptive(rhs, x0, xend, y0, h0, tol, method=trapezoidal_ieuler)

    if n==0:
        # Plot the solution
        subplot(2,1,1)
        plot(x_num, y_num)
        ylabel('y')
        subplot(2,1,2)

    # Plot the step size sequence
    semilogy(x_num[0:-1], diff(x_num), label='Tol={:.1e}'.format(tol));

    tol = 1.e-2*tol         # Reduce the tolerance by a factor 1/100

# Decorations
xlabel('x')
ylabel('h')
legend(loc='center left', bbox_to_anchor=(1, 0.5));
```

We observe that there are no longer any step size restriction because of stability. The algorithm behaves as expected.

Comment: Implicit methods can of course also be applied for nonlinear ODEs. Implicit Euler's method will be

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(x, \mathbf{y}_{n+1}),$$

which is a nonlinear system which has to be solved for each step. Similar for the trapezoidal rule. Usually these equations are solved by Newton's method or some simplification of it.

Summary.

Linear test equation:

$$y' = \lambda y, \qquad \lambda < 0.$$

Stability function $R(z)$, given by the method applied to the test problem:

$$y_{n+1} = R(z) y_n, \qquad z = \lambda h.$$

Stability region $\mathcal{S}$:

$$\mathcal{S} = \{z \in \mathbb{C}, \quad |R(z)| \leq 1\}.$$

$A(0)$-stability:

$$\mathcal{S} \supset \mathbb{C}^- = \{z \in \mathbb{C} \ : \ \Re z \leq 0\},$$

which is the same as the requirement that

$$|R(z)| \leq 1 \qquad \text{for all } z \text{ with } \Re z \leq 0.$$