1 ▶ Order criteria

Consider an explicit two-stage Runge-Kutta method given by the Butcher table

\[
\begin{array}{c|cc}
0 & 0 & 0 \\
\hline
c & a & 0 \\
\end{array}
\]

Write down equations for the parameters such that the method is consistent of order two. Show that the order of consistency cannot be higher.

This Runge-Kutta method reads

\[
y_n(t_0 + h) = y_0 + h b_1 f(t_0, y_0) + h b_2 f(t_0 + h c, y_0 + h a f(t_0, y_0))
\]

Let's choose \( a = c \), otherwise it will be difficult. (Ref. Lemma 2.7)

Then

\[
y_0 + h a f(t_0, y_0) = y_0 + h c y'(t_0) = y'(t_0 + h c) + O(h^2), \text{ so that, if } f \text{ is sufficiently smooth}
\]

\[
h b_1 f(t_0 + h c, y_0 + h a f(t_0, y_0)) = h b_1 f(t_0 + h c, y'(t_0 + h c)) + O(h^2)
\]

\[
= h b_2 y'(t_0 + h c) + O(h^3)
\]

\[
= h b_2 y'(t_0) + h^2 b_2 c y''(t_0) + O(h^3).
\]

Hence

\[
y_n(t_0 + h) = y_0 + (b_1 + b_2) y'(t_0) h + b_2 c y''(t_0) h^2 + O(h^3).
\]

The terms up to order two should equal the ones in the Taylor expansion of \( y(t) \), that is,

\[
\begin{align*}
\frac{b_2 c}{2} = \frac{1}{2} \\
\frac{b_1 + b_2}{2} = 1
\end{align*}
\]

For \( a = c = \frac{1}{2} \) we get \( b_2 = 1, b_1 = 0 \), i.e. Runge's method.

Consistency of order three is impossible.

Argument is like in Problem 1.6): For the IVP

\[
y' = y, \quad y(t_0) = y(0)
\]

a two-step method will only produce a polynomial of degree two.
2 ▶ Simpler order conditions for simple ODEs

Let \((A, b, c)\) be an explicit \(s\)-stage Runge Kutta method, \(s \geq p\), which is invariant under autonomization, i.e., \(c = Ae\).

(a) Consider the simplest autonomous initial value problem

\[
\dot{y}(t) = By(t), \quad y(t_0) = y_0 \in \mathbb{R}^d, \quad B \in \mathbb{R}^{d \times d}.
\]

Show that the Runge-Kutta method has consistency order \(p\) if and only if

\[
b^T A^{(\beta)} = \frac{1}{(\beta!)} \quad \text{for all linear trees } \beta = [[\cdots [\circ] \cdots]] \text{ with } \#\beta \leq p.
\]

(b) Consider the quadrature problem

\[
\dot{y}(t) = f(t), \quad y(t_0) = y_0 \in \mathbb{R}^d
\]

of a function \(f \in C^\infty([0, T])\). Show that the Runge-Kutta method has consistency order \(p\) if and only if

\[
b^T A^{(\beta)} = \frac{1}{\#\beta} \quad \text{for all one-level trees } \beta = [\circ, \ldots, \circ] \text{ with } \#\beta \leq p.
\]
3 ▶ Local vs. Global Error

In this exercise, we will investigate the difference between the local and the global error. We consider the ODE

\[ y'(t) = 1 - t + 3y, \quad y(t_0) = y_0 \]

with exact solution

\[ y(t) = Ce^{3t} - \frac{2}{9} + \frac{t}{3}, \quad C(t_0, y_0) = \frac{y_0 - t_0^2 + \frac{2}{9}}{e^{3y_0}}. \]

1. Create a plot similar to Figure 2.1 (Lady Windemere's fan) using an explicit Euler method on the interval \( T = [0, 1] \) with stepsize \( h = 0.2 \) and initial condition \( y(0) = 1 \). To do this, you have to calculate the exact solutions \( y(t) \) with initial conditions \( y(t_i) = y_i \) obtained from the \( i \)th step of the explicit Euler scheme.

2. Plot the global error \( |y(1) - y_N| \), where \( y_N \) is the last step of the Euler scheme and compare it to the maximum local error, both as functions of the stepsize \( h \). What is the convergence order that you obtain for the local and the global error?
function windemere()

f = @(t,y) 1 - t + 3*y;

sol = @(t) 1/9 * (3*t + 11*exp(3*t) - 2);
T = [0,1];
y0 = 1;
[t,y] = ode45(f, T, y0);

hold on
for i = 2:5
    [t5,y5] = expeuler( f, T, y0, 5);
    plot(t5,y5, '-o', 'Color', [0.5,0.5,0.5])
end

set(0, 'defaultlinelinewidth', 1.2)
set(gca, 'fontsize', 14)
plot(t5,y5, '-o')
xlabel('time t')
ylabel('y')
plot(t,y, '-k')
plot(t,sol(t))
ylim([0,25])

solC = @(t,C) C*exp(3*t) - 2/9 + t/3;

N = round(logspace(1,3,5));
for i=1:5
    [t, y] = expeuler( f, T, y0, N(i) );
    y_local = zeros(length(t)-2,1);
    for j=2:length(t)-1
        C = (y(j) - t(j)/3 + 2/9 )/exp(3*t(j));
        y_local(j-1) = solC(t(j+1), C);
    end
    yex = sol(t);
    err_local(i) = max(abs(y(3:end) - y_local));
    err_global(i) = abs(yex(end) - y(end));
    err_sum_local(i) = sum(abs(y(3:end) - y_local));
end

figure
loglog( N, err_local)
set(gca, 'fontsize', 14)
hold on
loglog( N, err_global)
loglog( N, err_sum_local)
loglog( N, 1./N, '--k')
loglog( N, 1./(N.^2), ':k')
xlabel('N')
ylabel('error')
legend('local', 'global', 'sum_local', 'O(1/N)', 'O(1/N^2)')
hold off
end

function [t, y] = expeuler( f, T, y0, N );
t = linspace( T(1), T(2), N+1 )';
h = t(2) - t(1);
y = zeros( N+1, 1 );
y(1) = y0;
for i = 1:N
    y(i+1) = y(i) + h * f(t(i), y(i));
end
4 ► Order isn’t everything\textsuperscript{1}

Solve the initial value problem

\[ y'(t) = |1.1 - y| + 1, \quad t \in [0, 0.1], \quad y(0) = 1, \]

both analytically, and numerically using the explicit Euler method, Runge’s method, and the classical RK4 method. Plot errors of all three methods against time.

Then write a script which does the same for various step sizes \( N = 2^k, \ k = 1, 2, \ldots, 10, \) and measures the runtime of each method (use the commands \texttt{tic} and \texttt{toc}). For all three methods, plot the runtime versus the achieved final accuracy at \( T = 0.1 \) (use a double logarithmic scale). You may compare to a plot of the required number of function evaluations versus the final achieved accuracy.

\textit{Hint.} The exact solution is

\[ y(t) = \begin{cases} -1.1e^{-t} + 2.1, & 0 \leq t \leq \ln 1.1, \\ \frac{10}{11}e^t + 0.1, & \ln 1.1 \leq t \leq 0.1. \end{cases} \]

\textsuperscript{1}Title and exercise taken from Deuflhard, Bornemann: Scientific Computing with Ordinary Differential Equations, Springer, 2002.
% Exercise 4 Problem 3
% ====================

function ex4problem3

% set parameters
f = @(t,y) abs(1.1 - y) + 1;
T = 0.1;
tspan = [0,T];
y0 = 1;
N = 1024;

% exact solution
y = @(t) (2.1-1.1*exp(-t)).*(t<log(1.1)) + (0.1+10/11*exp(t)).*(t>=log(1.1));

% run expeuler, runge, and RK4
[t1,y1] = expeuler(f,tspan,y0,N);
[t2,y2] = runge(f,tspan,y0,N);
[t3,y3] = rk4(f,tspan,y0,N);

% errors
e1 = abs(y(t1) - y1);
e2 = abs(y(t2) - y2);
e3 = abs(y(t3) - y3);

% plot all in one figure
scrsz = get(0,'ScreenSize');
figure('Position',[1 scrsz(4)/2-100 scrsz(3)/2 scrsz(4)/2])
subplot(2,2,1)
semilogy(t1,e1,t2,e2,t3,e3);
legend('expeuler','Runge','rk4')
title(sprintf('Semilog error vs. time, N=%i',N))

% performance test
for i=1:1:10
    N = 2^i;
    % exp euler
    tic;
    [t1,y1] = expeuler(f,tspan,y0,N);
    runtime1(i) = toc;
    funeval1(i) = N;
    flops1(i) = 3*N;
    final_error1(i) = abs(y(T) - y1(N + 1));
    % Runge
    tic;
    [t2,y2] = runge(f,tspan,y0,N);
    runtime2(i) = toc;
    funeval2(i) = 2*N;
    flops2(i) = 7*N;
    final_error2(i) = abs(y(T) - y2(N + 1));
    % RK4
    tic;
    [t3,y3] = rk4(f,tspan,y0,N);
    runtime3(i) = toc;
    funeval3(i) = 4*N;
    flops3(i) = 19*N;
    final_error3(i) = abs(y(T) - y3(N + 1));
end

subplot(2,2,2)
loglog(final_error1(runtime1,final_error2(runtime2,final_error3(runtime3);
legend('expeuler','Runge','rk4')
title('tic_toc_time_vs_accuracy')
```
subplot(2,2,3)
loglog(final_error1,funeval1,final_error2,funeval2,final_error3,funeval3);
legend('expeuler','Runge','rk4')
title('Function_evaluations_vs_accuracy')

subplot(2,2,4)
loglog(final_error1,flops1,final_error2,flops2,final_error3,flops3);
legend('expeuler','Runge','rk4')
title('flops+functions_evaluations_vs_accuracy')
end

% Required functions
% ------------------

% Explicit Euler
function [t,y] = expeuler(f,tspan,y0,N)
    y(:,1) = y0;
    h = (tspan(2)-tspan(1))/N;
    t = tspan(1)*ones(1,N+1) + h*(0:N);
    for i =2:N+1
        y(:,i) = y(:,i-1) + h*f(t(i-1),y(:,i-1));
    end
end

% Runge's method
function [t,y] = runge(f,tspan,y0,N)
    y(:,1) = y0;
    h = (tspan(2)-tspan(1))/N;
    t = tspan(1)*ones(1,N+1) + h*(0:N);
    for i =2:N+1
        k1 = f( t(i-1), y(:,i-1) );
        k2 = f( t(i-1) + h/2, y(:,i-1) + h/2*k1);
        y(:,i) = y(:,i-1) + h*k2;
    end
end

% Classical RK4
function [t,y] = rk4(f,tspan,y0,N)
    h = (tspan(2)-tspan(1))/N;
    t = tspan(1)*ones(1,N+1) + h*(0:N);
    t(1) = tspan(1);
    y(:,1) = y0;
    g=[1/6 1/3 1/3 1/6];
    for i=2:N+1
        k1 = f( t(i-1), y(:,i-1) );
        k2 = f( t(i-1)+h/2, y(:,i-1)+(h/2)*k1 );
        k3 = f( t(i-1)+h/2, y(:,i-1)+(h/2)*k2 );
        k4 = f( t(i-1)+h, y(:,i-1)+h*k3 );
        y(:,i) = y(:,i-1) + h*(g(1)*k1+g(2)*k2+g(3)*k3+g(4)*k4);
    end
end
```