

Problem Sheet 12

Problem 1 Three-stage Runge-Kutta method (core problem)

The most widely used class of numerical integrators for IVPs is that of *explicit* Runge-Kutta (RK) methods as defined in [1, Def. 11.4.9]. They are usually described by giving their coefficients in the form of a Butcher scheme [1, Eq. (11.4.11)].

(1a) ☞ Implement a header-only C++ class RKIntegrator

```
1  template <class State>
2  class RKIntegrator {
3  public:
4      RKIntegrator(const Eigen::MatrixXd & A,
5                  const Eigen::VectorXd & b) {
6          // TODO: given a Butcher scheme in A,b, initialize
6          //          RK method for solution of an IVP
7      }
8
9      template <class Function>
10     std::vector<State> solve(const Function &f, double T,
11                             const State & y0,
12                             unsigned int N) const {
13         // TODO: computes N uniform time steps for the ODE
13         //           $y'(t) = f(y)$  up to time T of RK method with
13         //          initial value y0 and store all steps (y_k) into
13         //          return vector
14     }
15 private:
16     template <class Function>
```

```


17 void step(const Function &f, double h,
18           const State &y0, State &y1) const {
19     // TODO: performs a single step from y0 to y1 with
20           step size h of the RK method for the IVP with rhs f
21 }
22 // TODO: hold data for RK methods
23 };

```

which implements a generic RK method given by a Butcher scheme to solve the autonomous initial value problem $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(t_0) = \mathbf{y}_0$.

HINT: See `rkintegrate_template.hpp` for more details about the implementation.

Solution: See `rkintegrate.hpp`.

(1b)  Test your implementation of the RK methods with the following data. As autonomous initial value problem, consider the predator/prey model (cf. [1, Ex. 11.1.9]):

$$\dot{y}_1(t) = (\alpha_1 - \beta_1 y_2(t))y_1(t) \quad (82)$$

$$\dot{y}_2(t) = (\beta_2 y_1(t) - \alpha_2)y_2(t) \quad (83)$$

$$\mathbf{y}(0) = [100, 5] \quad (84)$$

with coefficients $\alpha_1 = 3, \alpha_2 = 2, \beta_1 = \beta_2 = 0.1$.

Use a Runge-Kutta single step method described by the following *Butcher scheme* (cf. [1, Def. 11.4.9]):

$$\begin{array}{c|ccc}
0 & 0 & & \\
\frac{1}{3} & \frac{1}{3} & 0 & \\
\frac{2}{3} & 0 & \frac{2}{3} & 0 \\
\hline
\frac{3}{3} & \frac{1}{4} & 0 & \frac{3}{4}
\end{array} \quad (85)$$

Compute an approximated solution up to time $T = 10$ for the number of steps $N = 2^j, j = 7, \dots, 14$.

Use, as reference solution, $\mathbf{y}(10) = [0.319465882659820, 9.730809352326228]$.

Tabulate the error and compute the experimental order of algebraic convergence of the method.


HINT: See `rk3prey_template.cpp` for more details about the implementation.

Solution: See `rk3prey.cpp`.

Problem 2 Order is not everything (core problem)

In [1, Section 11.3.2] we have seen that Runge-Kutta single step methods when applied to initial value problems with sufficiently smooth solutions will converge algebraically (with respect to the maximum error in the mesh points) with a rate given by their intrinsic order, see [1, Def. 11.3.21].

In this problem we perform empiric investigations of orders of convergence of several explicit Runge-Kutta single step methods. We rely on two IVPs, one of which has a perfectly smooth solution, whereas the second has a solution that is merely piecewise smooth. Thus in the second case the smoothness assumptions of the convergence theory for RK-SSMs might be violated and it is interesting to study the consequences.

(2a)  Consider the scalar autonomous ODE

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0, \quad (86)$$

where $\mathbf{f}: \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $\mathbf{y}_0 \in \mathbb{R}^n$. Using the class `RKIntegrate` of Problem 1 write a C++ function

```
1 template <class Function>
2 void errors(const Function &f, const double &T, const
   Eigen::VectorXd &y0, const Eigen::MatrixXd &A,
3 const Eigen::VectorXd &b)
```

that computes an approximated solution \mathbf{y}_N of (86) up to time T by means of an explicit Runge-Kutta method with $N = 2^k$, $k = 1, \dots, 15$, uniform timesteps. The method is defined by the Butcher scheme described by the inputs `A` and `b`. The input `f` is an object with an evaluation operator (e.g. a lambda function) for arguments of type `const VectorXd &` representing \mathbf{f} . The input `y0` passes the initial value \mathbf{y}_0 .

For each k , the function should show the error at the final point $E_N = \|\mathbf{y}_N(T) - \mathbf{y}_{2^{15}}(T)\|$, $N = 2^k$, $k = 1, \dots, 13$, accepting $\mathbf{y}_{2^{15}}(T)$ as exact value. Assuming algebraic convergence for $E_N \approx CN^{-r}$, at each step show an approximation of the order of convergence r_k (recall that $N = 2^k$). This will be an expression involving E_N and $E_{N/2}$.

Finally, compute and show an approximate order of convergence by averaging the relevant r_{Ns} (namely, you should take into account the cases before machine precision is reached in the components of $\mathbf{y}_N(T) - \mathbf{y}_{2^{15}}(T)$).

Solution: Let us find an expression for the order of convergence. Set $N_k = 2^k$. We readily


derive

$$\frac{E_{N_{k-1}}}{E_{N_k}} \approx \frac{C2^{-(k-1)r_k}}{C2^{-kr_k}} = 2^{r_k}, \quad r_k \approx \log\left(\frac{E_{N_{k-1}}}{E_{N_k}}\right) / \log(2).$$

A reasonable approximation of the order of convergence is given by

$$r \approx \frac{1}{\#K} \sum_{k \in K} r_k, \quad K = \{k = 1, \dots, 15 : E_{N_k} > 5n \cdot 10^{-14}\}. \quad (87)$$

See file `errors.hpp` for the implementation.

(2b)  Calculate the analytical solutions of the logistic ODE (see [1, Ex. 11.1.5])

$$\dot{y} = (1 - y)y, \quad y(0) = 1/2, \quad (88)$$

and of the initial value problem

$$\dot{y} = |1.1 - y| + 1, \quad y(0) = 1. \quad (89)$$

Solution: As far as (88) is concerned, the solution is $y(t) = (1 + e^{-t})^{-1}$ (see [1, Eq. (11.1.7)]).

Let us now consider (89). Because of the absolute value on the right hand side of the differential equation, we have to distinguish two cases $y(t) < 1.1$ and $y(t) > 1.1$. Since the initial condition is given by $y(0) = 1 < 1.1$, we start with the case $y(t) < 1.1$. For $y(t) < 1.1$, the differential equation is $\dot{y} = 2.1 - y$. Separation of variables

$$\int_1^{y(t)} \frac{1}{2.1 - \tilde{y}} d\tilde{y} = \int_0^t d\tilde{t}$$

yields the solution


$$y(t) = 2.1 - 1.1e^{-t}, \quad \text{for } y(t) < 1.1.$$

For $y(t) > 1.1$, the differential equation is given by $\dot{y} = y - 0.1$ with initial condition $y(\ln(\frac{11}{10})) = 1.1$, where the initial time t^* was derived from the condition $y(t^*) = 2.1 - 1.1e^{-t^*} \stackrel{!}{=} 1.1$. Separation of variables yields the solution for this IVP

$$y(t) = \frac{10}{11}e^t + 0.1.$$

Together, the solution of the initial IVP is given by

$$y(t) = \begin{cases} 2.1 - 1.1e^{-t}, & \text{for } t < \ln(1.1) \\ \frac{10}{11}e^t + 0.1, & \text{for } t > \ln(1.1). \end{cases}$$

(2c)  Use the function `errors` from (2a) with the ODEs (88) and (89) and the methods:

- the explicit Euler method, a RK single step method of order 1,
- the explicit trapezoidal rule, a RK single step method of order 2,
- an RK method of order 3 given by the Butcher tableau

$$\begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 1 & -1 & 2 & \\ \hline & 1/6 & 2/3 & 1/6 \end{array}$$

- the classical RK method of order 4.

(See [1, Ex. 11.4.13] for details.) Set $T = 0.1$.

Comment the calculated order of convergence for the different methods and the two different ODEs.

Solution: Using the expression for the order of convergence given in (87) we find for (88):

- Eul: 1.06
- RK2: 2.00
- RK3: 2.84
- RK4: 4.01

This corresponds to the expected orders. However, in the case of the ODE (89) we obtain

- Eul: 1.09
- RK2: 1.93
- RK3: 1.94
- RK4: 1.99

The convergence orders of the explicit Euler and Runge–Kutta 2 methods are as expected, but we do not see any relevant improvement in the convergence orders of RK3 and RK4. This is due to the fact that the right hand side of the IVP is not continuously differentiable: the convergence theory breaks down.

See file `order_not_all.cpp` for the implementation.

Problem 3 Integrating ODEs using the Taylor expansion method

In [1, Chapter 11] of the course we studied single step methods for the integration of initial value problems for ordinary differential equations $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, [1, Def. 11.3.5]. Explicit single step methods have the advantage that they only rely on point evaluations of the right hand side \mathbf{f} .

This problem examines another class of methods that is obtained by the following reasoning: if the right hand side $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ of an autonomous initial value problem

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) , \quad \mathbf{y}(0) = \mathbf{y}_0 , \quad (90)$$

with solution $\mathbf{y} : \mathbb{R} \rightarrow \mathbb{R}^n$ is smooth, also the solution $\mathbf{y}(t)$ will be regular and it is possible to expand it into a Taylor sum at $t = 0$, see [1, Thm. 2.2.15],


$$\mathbf{y}(t) = \sum_{n=0}^m \frac{\mathbf{y}^{(n)}(0)}{n!} t^n + R_m(t) , \quad (91)$$

with remainder term $R_m(t) = O(t^{m+1})$ for $t \rightarrow 0$.

A single step method for the numerical integration of (90) can be obtained by choosing $m = 3$ in (91), neglecting the remainder term, and taking the remaining sum as an approximation of $\mathbf{y}(h)$, that is,

$$\mathbf{y}(h) \approx \mathbf{y}_1 := \mathbf{y}(0) + \frac{d\mathbf{y}}{dt}(0)h + \frac{1}{2} \frac{d^2\mathbf{y}}{dt^2}(0)h^2 + \frac{1}{6} \frac{d^3\mathbf{y}}{dt^3}(0)h^3 .$$

Subsequently, one uses the ODE and the initial condition to replace the temporal derivatives $\frac{d^l\mathbf{y}}{dt^l}$ with expressions in terms of (derivatives of) \mathbf{f} . This yields a single step integration method called *Taylor (expansion) method*.

(3a)  Express $\frac{d\mathbf{y}}{dt}(t)$ and $\frac{d^2\mathbf{y}}{dt^2}(t)$ in terms of \mathbf{f} and its Jacobian \mathbf{Df} .

HINT: Apply the chain rule, see [1, § 2.4.5], then use the ODE (90).

Solution: For the first time derivative of \mathbf{y} , we just use the differential equation:

$$\frac{d\mathbf{y}}{dt}(t) = \mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t)).$$

For the second derivative, we use the previous equation and apply chain rule and then once again insert the ODE:

$$\frac{d^2\mathbf{y}}{dt^2}(t) = \frac{d}{dt}\mathbf{f}(\mathbf{y}(t)) = \mathbf{Df}(\mathbf{y}(t)) \cdot \mathbf{y}'(t) = \mathbf{Df}(\mathbf{y}(t)) \cdot \mathbf{f}(\mathbf{y}(t)).$$

Here $\mathbf{Df}(\mathbf{y}(t))$ is the Jacobian of \mathbf{f} evaluated at $\mathbf{y}(t)$.

(3b) ☒ Verify the formula

$$\frac{d^3 \mathbf{y}}{dt^3}(0) = \mathbf{D}^2 \mathbf{f}(\mathbf{y}_0)(\mathbf{f}(\mathbf{y}_0), \mathbf{f}(\mathbf{y}_0)) + \mathbf{Df}(\mathbf{y}_0)^2 \mathbf{f}(\mathbf{y}_0). \quad (92)$$

HINT: this time we have to apply both the product rule [1, (2.4.9)] and chain rule [1, (2.4.8)] to the expression derived in the previous sub-problem.

To gain confidence, it is advisable to consider the scalar case $d = 1$ first, where $f : \mathbb{R} \rightarrow \mathbb{R}$ is a real valued function.

Relevant for the case $d > 1$ is the fact that the first derivative of \mathbf{f} is a linear mapping $\mathbf{Df}(\mathbf{y}_0) : \mathbb{R}^n \rightarrow \mathbb{R}^n$. This linear mapping is applied by multiplying the argument with the Jacobian of \mathbf{f} . Similarly, the second derivative is a *bilinear* mapping $\mathbf{D}^2 \mathbf{f}(\mathbf{y}_0) : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$. The i -th component of $\mathbf{D}^2 \mathbf{f}(\mathbf{y}_0)(\mathbf{v}, \mathbf{v})$ is given by

$$\mathbf{D}^2 \mathbf{f}(\mathbf{y}_0)(\mathbf{v}, \mathbf{v})_i = \mathbf{v}^T \mathbf{H}f_i(\mathbf{y}_0) \mathbf{v},$$

where $\mathbf{H}f_i(\mathbf{y}_0)$ is the Hessian of the i -th component of \mathbf{f} evaluated at \mathbf{y}_0 .

Solution: We follow the hint and first have a look at the scalar case. Here, the Jacobian reduces to $f'(y(t))$. Thus, we have to calculate

$$\frac{d^3 y}{dt^3}(t) = \frac{d}{dt}(f'(y(t))f(y(t))).$$

Product rule and chain rule give

$$\frac{d}{dt}(f'(y(t))f(y(t))) = f''(y(t))y'(t)f(y(t)) + f'(y(t))f'(y(t))y'(t).$$

Inserting the ODE $y'(t) = f(y(t))$ once again yields

$$\frac{d^3 y}{dt^3}(t) = f''(y(t))f(y(t))^2 + f'(y(t))^2 f(y(t)).$$

This already resembles Formula 92. The first term is quadratic in $f(y(t))$ and involves the second derivative of f , whereas the second term involves the first derivative of f in quadratic form.

To understand the formula for higher dimensions, we verify it componentwise. For each component $y_i(t)$ we have a function $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$.

For the first derivative of $y_i(t)$, this is straightforward:

$$y'_i(t) = f_i(\mathbf{y}(t)).$$

Thus, the second derivative is

$$\begin{aligned} y''_i(t) &= \frac{d}{dt}(f_i(\mathbf{y}(t))) = \sum_{j=1}^n \partial_{y_j} f_i(\mathbf{y}(t)) y'_j(t) \\ &= (\mathbf{grad} f_i(\mathbf{y}(t)))^T \cdot \mathbf{y}'(t) = (\mathbf{grad} f_i(\mathbf{y}(t)))^T \cdot \mathbf{f}(\mathbf{y}(t)). \end{aligned}$$

Building up all components gives us what we have already obtained in (3a):

$$\mathbf{y}''(t) = \mathbf{Df}(\mathbf{y}(t)) \cdot \mathbf{f}(\mathbf{y}(t)),$$

with the Jacobian $\mathbf{Df}(\mathbf{y}(t))$, which contains the gradients of the components of f row-wise.

Now, we apply product rule to $y''_i(t)$ to obtain

$$y'''_i(t) = \left(\frac{d}{dt} (\mathbf{grad} f_i(\mathbf{y}(t)))^T \right) \cdot \mathbf{y}'(t) + (\mathbf{grad} f_i(\mathbf{y}(t)))^T \cdot \mathbf{y}''(t).$$

The second term of the sum again builds up to

$$\mathbf{Df}(\mathbf{y}(t)) \cdot \mathbf{y}''(t) = \mathbf{Df}(\mathbf{y}(t)) \cdot \mathbf{Df}(\mathbf{y}(t)) \cdot \mathbf{f}(\mathbf{y}(t)) = \mathbf{Df}(\mathbf{y}(t))^2 \cdot \mathbf{f}(\mathbf{y}(t)).$$

For the first term, we first write the scalar product of the two vectors as a sum and then interchange the order of derivatives. This is possible as long as functions are sufficiently differentiable.

$$\left(\frac{d}{dt} (\mathbf{grad} f_i(\mathbf{y}(t)))^T \right) \cdot \mathbf{y}'(t) = \sum_{j=1}^n \left(\partial_{y_j} \frac{d}{dt} f_i(\mathbf{y}(t)) \right) f_j(\mathbf{y}(t)).$$

Now we apply the chain rule:

$$\begin{aligned} \sum_{j=1}^n \left(\partial_{y_j} \frac{d}{dt} f_i(\mathbf{y}(t)) \right) f_j(\mathbf{y}(t)) &= \sum_{j=1}^n \left(\partial_{y_j} \sum_{l=1}^n \partial_{y_l} f_i(\mathbf{y}(t)) y'_l(t) \right) f_j(\mathbf{y}(t)) \\ &= \sum_{j,l=1}^n (\partial_{y_j} \partial_{y_l} f_i(\mathbf{y}(t))) f_l(\mathbf{y}(t)) f_j(\mathbf{y}(t)) \\ &= \mathbf{f}(\mathbf{y}(t))^T \cdot \mathbf{H} f_i(\mathbf{y}(t)) \cdot \mathbf{f}(\mathbf{y}(t)). \end{aligned}$$

This is the desired result.

(3c) ☐ We now apply the Taylor expansion method introduced above to the *predator-prey* model (97) introduced in Problem 1 and [1, Ex. 11.1.9].

To that end write a header-only C++ class `TaylorIntegrator` for the integration of the autonomous ODE of (97) using the Taylor expansion method with uniform time steps on the temporal interval $[0, 10]$.

HINT: You can copy the implementation of Problem 1 and modify only the `step` method to perform a single step of the Taylor expansion method.

HINT: Find a suitable way to pass the data for the derivatives of the r.h.s. function \mathbf{f} to the `solve` function. You may modify the signature of `solve`.

HINT: See `taylorintegrator_template.hpp`.

Solution: For our particular example, we have

$$\mathbf{y} = \begin{pmatrix} u \\ v \end{pmatrix}, \quad \mathbf{f}(\mathbf{y}) = \begin{pmatrix} (\alpha_1 - \beta_1 y_2) y_1 \\ -(\alpha_2 - \beta_2 y_1) y_2 \end{pmatrix},$$

$$\mathbf{Df}(\mathbf{y}) = \begin{pmatrix} \alpha_1 - \beta_1 y_2 & -\beta_1 y_1 \\ \beta_2 y_2 & -(\alpha_2 - \beta_2 y_1) \end{pmatrix},$$

$$\mathbf{H}f_1(\mathbf{y}) = \begin{pmatrix} 0 & -\beta_1 \\ -\beta_1 & 0 \end{pmatrix}, \quad \mathbf{H}f_2(\mathbf{y}) = \begin{pmatrix} 0 & \beta_2 \\ \beta_2 & 0 \end{pmatrix}.$$

See `taylorintegrator.hpp`.

(3d) ☐ Experimentally determine the order of convergence of the considered Taylor expansion method when it is applied to solve (97). Study the behaviour of the error at final time $t = 10$ for the initial data $\mathbf{y}(0) = [100, 5]$.

As a reference solution use the same data as Problem 1.

HINT: See `taylorprey_template.cpp`.

Solution: From `taylorprey.cpp`, we see cubic algebraic convergence $O(h^3)$.

(3e) ☐ What is the disadvantage of the Taylor method compared with a Runge-Kutta method?

Solution: As we can see in the error table, the error of the studied Runge-Kutta method and Taylor's method are practically identical. The obvious disadvantage of Taylor's method in comparison with Runge-Kutta methods is that the former involves rather complicated higher derivatives of f . If we want higher order, those formulas get even more complicated, whereas explicit Runge-Kutta methods work with only a few evaluations of f itself,


yielding results which are comparable. Moreover, the Taylor expansion method cannot be applied for f , when this is given in procedural form.

Problem 4 System of ODEs

Consider the following initial value problem for a second-order system of ordinary differential equations:

$$\begin{aligned}
 2\ddot{u}_1 - \ddot{u}_2 &= u_1(u_2 + u_1), \\
 -\ddot{u}_{i-1} + 2\ddot{u}_i - \ddot{u}_{i+1} &= u_i(u_{i-1} + u_{i+1}), \quad i = 2, \dots, n-1, \\
 2\ddot{u}_n - \ddot{u}_{n-1} &= u_n(u_n + u_{n-1}), \\
 u_i(0) &= u_{0,i} \quad i = 1, \dots, n, \\
 \dot{u}_i(0) &= v_{0,i} \quad i = 1, \dots, n,
 \end{aligned} \tag{93}$$

in the time interval $[0, T]$.

(4a)  Write (93) as a first order IVP of the form $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$ (see [1, Rem. 11.1.23]).

Solution: The second order IVP can be rewritten as a first order one by introducing \mathbf{v} :

$$\begin{aligned}
 \dot{u}_i &= v_i \quad i = 1, \dots, n, \\
 2\dot{v}_1 - \dot{v}_2 &= u_1(u_2 + u_1), \\
 -\dot{v}_{i-1} + 2\dot{v}_i - \dot{v}_{i+1} &= u_i(u_{i-1} + u_{i+1}) \quad i = 2, \dots, n-1, \\
 2\dot{v}_n - \dot{v}_{n-1} &= u_n(u_n + u_{n-1}), \\
 u_i(0) &= u_{0,i} \quad i = 1, \dots, n, \\
 v_i(0) &= v_{0,i} \quad i = 1, \dots, n.
 \end{aligned}$$

The ODE system can be written in vector form


$$\dot{\mathbf{u}} = \mathbf{v}, \quad \mathbf{C}\dot{\mathbf{v}} = \mathbf{g}(\mathbf{u}) := \begin{bmatrix} u_1(u_2 + u_1) \\ u_i(u_{i-1} + u_{i+1}) \\ u_n(u_n + u_{n-1}) \end{bmatrix},$$

where $\mathbf{C} \in \mathbb{R}^{n,n}$ is

$$\mathbf{C} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}.$$

In order to use standard interfaces to RK-SSM for first order ODEs, collect \mathbf{u} and \mathbf{v} in a $(2n)$ -dimensional vector $\mathbf{y} = [\mathbf{u}; \mathbf{v}]$. Then the system reads

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) := \begin{bmatrix} y_{n+1} \\ \vdots \\ y_{2n} \\ \mathbf{C}^{-1} \mathbf{g}(y_1, \dots, y_l) \end{bmatrix}.$$

(4b)  Apply the function `errors` constructed in Problem 2 to the IVP obtained in the previous subproblem. Use

$$n = 5, \quad u_{0,i} = i/n, \quad v_{0,i} = -1, \quad T = 1,$$

and the classical RK method of order 4. Construct any sparse matrix encountered as a sparse matrix in EIGEN. Comment the order of convergence observed.

Solution: See file `system.cpp` for the implementation. We observe convergence of order 4.00: this is expected since the function \mathbf{f} is smooth.

Issue date: 03.12.2015

Hand-in: 10.12.2015 (in the boxes in front of HG G 53/54).

Version compiled on: December 10, 2015 (v. 1.0).