

Problem Sheet 13

Problem 1 Matrix-valued Differential Equation (core problem)

First we consider the *linear* matrix differential equation

$$\dot{\mathbf{Y}} = \mathbf{A}\mathbf{Y} =: \mathbf{f}(\mathbf{Y}) \quad \text{with} \quad \mathbf{A} \in \mathbb{R}^{n \times n}. \quad (50)$$

whose solutions are *matrix-valued functions* $\mathbf{Y} : \mathbb{R} \rightarrow \mathbb{R}^{n \times n}$.

(1a) \square Show that for *skew-symmetric* \mathbf{A} , i.e. $\mathbf{A} = -\mathbf{A}^\top$ we have:

$$\mathbf{Y}(0) \text{ orthogonal} \implies \mathbf{Y}(t) \text{ orthogonal} \quad \forall t.$$

HINT: Remember what property distinguishes an orthogonal matrix. Thus you see that the assertion we want to verify boils down to showing that the bilinear expression $t \mapsto \mathbf{Y}(t)^\top \mathbf{Y}(t)$ does not vary along trajectories, that is, its time derivative must vanish. This can be established by means of the product rule [1, Eq. (2.4.9)] and using the differential equation.

(1b) \square Implement three C++ functions

(i) a single step of the explicit Euler method:

```
1 Eigen::MatrixXd eeulstep(const Eigen::MatrixXd & A,  
    const Eigen::MatrixXd & Y0, double h);
```

(ii) a single step of the implicit Euler method:

```
1 Eigen::MatrixXd ieulstep(const Eigen::MatrixXd & A,  
    const Eigen::MatrixXd & Y0, double h);
```

(iii) a single step of the implicit mid-point method:

```
1 Eigen::MatrixXd impstep(const Eigen::MatrixXd & A,
    const Eigen::MatrixXd & Y0, double h);
```

which determine, for a given initial value $\mathbf{Y}(t_0) = \mathbf{Y}_0$ and for given step size h , approximations for $\mathbf{Y}(t_0 + h)$ using one step of the corresponding method for the approximation of the ODE (50)

(1c) \square Investigate numerically, which one of the implemented methods preserves orthogonality in the sense of sub-problem (1a) for the ODE (50) and which one doesn't. To that end, consider the matrix

$$\mathbf{M} := \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 9 & 9 & 2 \end{bmatrix}$$

and use the matrix \mathbf{Q} arising from the QR-decomposition of \mathbf{M} as initial data \mathbf{Y}_0 . As matrix \mathbf{A} , use the skew-symmetric matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}.$$

To that end, perform $n = 20$ time steps of size $h = 0.01$ with each method and compute the Frobenius norm of $\mathbf{Y}(T)' \mathbf{Y}(T) - \mathbf{I}$. Use the functions from subproblem (1b).

From now we consider a non-linear ODE that is structurally similar to (50). We study the initial value problem

$$\dot{\mathbf{Y}} = -(\mathbf{Y} - \mathbf{Y}^\top) \mathbf{Y} =: f(\mathbf{Y}) \quad , \quad \mathbf{Y}(0) = \mathbf{Y}_0 \in \mathbb{R}^{n,n}, \quad (51)$$

whose solution is given by a *matrix-valued function* $t \mapsto \mathbf{Y}(t) \in \mathbb{R}^{n \times n}$.

(1d) \square Write a C++ function

```
1 Eigen::MatrixXd matode(const Eigen::MatrixXd & Y0,
    double T)
```

which solves (51) on $[0, T]$ using the C++ header-only class `ode45` (in the file `ode45.hpp`). The initial value should be given by a $n \times n$ EIGEN matrix `Y0`. Set the absolute tolerance to 10^{-10} and the relative tolerance to 10^{-8} . The output should be an approximation of $\mathbf{Y}(T) \in \mathbb{R}^{n \times n}$.

HINT: The `ode45` class works as follows:

1. Call the constructor, and specify the r.h.s function f and the type for the solution and the initial data in `RhsType`, example:

```
ode45<StateType> O(f);
```

with, for instance, `Eigen::VectorXd` as `StateType`.

2. (optional) Set custom options, modifying the `struct options` inside `ode45`, for instance:

```
O.options.<option_you_want_to_change> = <value>;
```


3. Solve the IVP and store the solution, e.g.:

```
std::vector<std::pair<Eigen::VectorXd, double>> sol
    = O.solve(y0, T);
```

Relative and absolute tolerances for `ode45` are defined as `rtol` resp. `atol` variables in the `struct options`. The return value is a sequence of states and times computed by the adaptive single step method.

HINT: The type `RhsType` needs a vector space structure implemented with operators `*`, `*`, `*=`, `+=` and assignment/copy operators. Moreover a norm method must be available. Eigen vector and matrix types, as well as fundamental types are eligible as `RhsType`.

HINT: Have a look at the public interface of `ode45.hpp`. Look at the template file `matrix_ode_template.cpp`.

(1e)  Show that the function $t \mapsto \mathbf{Y}^\top(t)\mathbf{Y}(t)$ is constant for the exact solution $\mathbf{Y}(t)$ of (51).

HINT: Remember the general product rule [1, Eq. (2.4.9)].

(1f) ☐ Write a C++ function

```
bool checkinvariant(const Eigen::MatrixXd & M, double T);
```

which (numerically) determines if the statement from (1e) is true, for $t = T$ and for the output of `matode` from sub-problem (1d). You must take into account round-off errors. The function's input should be the same as that of `matode`.

HINT: See `matrix_ode_template.cpp`.

(1g) ☐ Use the function `checkinvariant` to test whether the invariant is preserved by `ode45` or not. Use the matrix `M` defined above and $T = 1$.

Problem 2 Stability of a Runge-Kutta Method (core problem)

We consider a 3-stage Runge-Kutta single step method described by the Butcher-Tableau

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1/2 & 1/4 & 1/4 & 0 \\ \hline & 1/6 & 1/6 & 2/3 \end{array} \quad (52)$$

(2a) ☐ Consider the prey/predator model

$$\dot{y}_1(t) = (1 - y_2(t))y_1(t) \quad (53)$$

$$\dot{y}_2(t) = (y_1(t) - 1)y_2(t) \quad (54)$$

$$\mathbf{y}(0) = [100, 1]. \quad (55)$$

Write a C++ code to approximate the solution up to time $T = 1$ of the IVP. Use a RK-SSM defined above. Numerically determine the convergence order of the method for uniform steps of size 2^{-j} , $j = 2, \dots, 13$.

Use, as a reference solution, an approximation with 2^{14} steps.

What do you notice for big step sizes? What is the maximum step size for the solution to be stable?

HINT: You can use the `rkintegrator.hpp` implemented in Problem Sheet 12. See `stabrk_template.cpp`.

(2b) ☐ Calculate the stability function $S(z)$, $z = h\lambda$, $\lambda \in \mathbb{C}$ of the method given by the table (52).

Problem 3 Initial Condition for Lotka-Volterra ODE

Introduction. In this problem we will face a situation, where we need to compute the derivative of the solution of an IVP with respect to the initial state. This paragraph will show how this derivative can be obtained as the solution of another differential equation. Please read this carefully and try to understand every single argument.

We consider IVPs for the autonomous ODE

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \quad (56)$$

with smooth right hand side $\mathbf{f}: D \rightarrow \mathbb{R}^d$, where $D \subseteq \mathbb{R}^d$ is the state space. We take for granted that for all initial states, solutions exist for all times (global solutions, see [1, Ass. 11.1.38]).

By its very definition given in [1, Def. 11.1.39]), the evolution operator

$$\Phi: \mathbb{R} \times D \rightarrow D, \quad (t, \mathbf{y}) \mapsto \Phi(t, \mathbf{y})$$

satisfies

$$\frac{\partial \Phi}{\partial t}(t, \mathbf{y}) = \mathbf{f}(\Phi(t, \mathbf{y})).$$

Next, we can differentiate this identity with respect to the state variable \mathbf{y} . We assume that all derivatives can be interchanged, which can be justified by rigorous arguments (which we won't do here). Thus, by the chain rule, we obtain and after swapping partial derivatives $\frac{\partial}{\partial t}$ and $D_{\mathbf{y}}$

$$\frac{\partial D_{\mathbf{y}} \Phi}{\partial t}(t, \mathbf{y}) = D_{\mathbf{y}} \frac{\partial \Phi}{\partial t}(t, \mathbf{y}) = D_{\mathbf{y}}(\mathbf{f}(\Phi(t, \mathbf{y}))) = D \mathbf{f}(\Phi(t, \mathbf{y})) D_{\mathbf{y}} \Phi(t, \mathbf{y}).$$

Abbreviating $\mathbf{W}(t, \mathbf{y}) := D_{\mathbf{y}} \Phi(t, \mathbf{y})$ we can rewrite this as the non-autonomous ODE

$$\dot{\mathbf{W}} = D \mathbf{f}(\Phi(t, \mathbf{y})) \mathbf{W}. \quad (57)$$

Here, the state \mathbf{y} can be regarded as a parameter. Since $\Phi(0, \mathbf{y}) = \mathbf{y}$, we also know $\mathbf{W}(0, \mathbf{y}) = \mathbf{I}$ (identity matrix), which supplies an initial condition for (57). In fact, we can even merge (56) and (57) into the ODE

$$\frac{d}{dt} [\mathbf{y}(\cdot), \mathbf{W}(\cdot, \mathbf{y}_0)] = [\mathbf{f}(\mathbf{y}(t)), D \mathbf{f}(\mathbf{y}(t)) \mathbf{W}(t, \mathbf{y}_0)], \quad (58)$$

which is autonomous again.

Now let us apply (57)/(58). As in [1, Ex. 11.1.9], we consider the following autonomous Lotka-Volterra differential equation of a predator-prey model

$$\begin{aligned}\dot{u} &= (2 - v)u \\ \dot{v} &= (u - 1)v\end{aligned}\tag{59}$$

on the state space $D = \mathbb{R}_+^2$, $\mathbb{R}_+ = \{\xi \in \mathbb{R} : \xi > 0\}$. All the solutions of (59) are periodic and their period depends on the initial state $[u(0), v(0)]^T$. In this exercise we want to develop a numerical method which computes a suitable initial condition for a given period.

- (3a) ☐ For fixed state $\mathbf{y} \in D$, (57) represents an ODE. What is its state space?
- (3b) ☐ What is the right hand side function for the ODE (57), in the case of the $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ given by the Lotka-Volterra ODE (59)? You may write $u(t), v(t)$ for solutions of (59).
- (3c) ☐ From now on we write $\Phi: \mathbb{R} \times \mathbb{R}_+^2 \rightarrow \mathbb{R}_+^2$ for the evolution operator associated with (59). Based on Φ derive a function $\mathbf{F}: \mathbb{R}_+^2 \rightarrow \mathbb{R}_+^2$ which evaluates to zero for the input \mathbf{y}_0 if the period of the solution of system (59) with initial value

$$\mathbf{y}_0 = \begin{bmatrix} u(0) \\ v(0) \end{bmatrix}$$

is equal to a given value T_P .

- (3d) ☐ We write $\mathbf{W}(T, \mathbf{y}_0)$, $T \geq 0$, $\mathbf{y}_0 \in \mathbb{R}_+^2$ for the solution of (57) for the underlying ODE (59). Express the Jacobian of \mathbf{F} from (3c) by means of \mathbf{W} .

- (3e) ☐ Argue, why the solution of $\mathbf{F}(\mathbf{y}) = 0$ will, in general, not be unique. When will it be unique?

HINT: Study [1, § 11.1.21] again. Also look at [1, Fig. 319].


- (3f) ☐ A C++ implementation of an adaptive embedded Runge-Kutta method is available, with a functionality similar to MATLAB's `ode45` (see Problem 1). Relying on this implement a C++ function

```
1 std::pair<Vector2d, Matrix2d> PhiAndW(double u0, double
    v0, double T)
```

that computes $\Phi(T, [u_0, v_0]^T)$ and $\mathbf{W}(T, [u_0, v_0]^T)$. The first component of the output pair should contain $\Phi(T, [u_0, v_0]^T)$ and the second component the matrix $\mathbf{W}(T, [u_0, v_0]^T)$. See `LV_template.cpp`.

HINT: As in (58), both ODEs (for Φ and W) must be combined into a single autonomous differential equation on the state space $D \times \mathbb{R}^{d \times d}$.

HINT: The equation for W is a matrix differential equation. These cannot be solved directly using `ode45`, because the solver expects the right hand side to return a vector. Therefore, transform matrices into vectors (and vice-versa).

(3g)  Using `PhiAndW`, write a C++ routine that determines initial conditions $u(0)$ and $v(0)$ such that the solution of the system (59) has period $T = 5$. Use the multi-dimensional *Newton method* for $F(y) = 0$ with F from (3c). As your initial approximation, use $[3, 2]^T$. Terminate the *Newton method* as soon as $\|F(y)\| \leq 10^{-5}$. Validate your implementation by comparing the obtained initial data y with $\Phi(100, y)$.

HINT: Set relative and absolute tolerances of `ode45` to 10^{-14} and 10^{-12} , respectively. See file `LV_template.cpp`.

HINT: The correct solutions are $u(0) \approx 3.110$ and $v(0) = 2.081$.

Problem 4 Exponential integrator

A modern class of single step methods developed for special initial value problems that can be regarded as perturbed linear ODEs are the exponential integrators, see

M. HOCHBRUCK AND A. OSTERMANN, *Exponential integrators*, Acta Numerica, 19 (2010), pp. 209–286.


These methods fit the concept of single step methods as introduced in [1, Def. 11.3.5] and, usually, converge algebraically according to [1, (11.3.20)].


A step with size h of the so-called *exponential Euler* single step method for the ODE $\dot{y} = f(y)$ with continuously differentiable $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ reads

$$y_1 = y_0 + h \varphi(h Df(y_0)) f(y_0), \quad (60)$$

where $Df(y) \in \mathbb{R}^{d,d}$ is the Jacobian of f at $y \in \mathbb{R}^d$, and the matrix function $\varphi : \mathbb{R}^{d,d} \rightarrow \mathbb{R}^{d,d}$ is defined as $\varphi(Z) = (\exp(Z) - \text{Id}) Z^{-1}$. Here $\exp(Z)$ is the matrix exponential of Z , a special function $\exp : \mathbb{R}^{d,d} \rightarrow \mathbb{R}^{d,d}$, see [1, Eq. (12.1.32)].

The function φ is implemented in the provided file `ExpEul_template.cpp`. When plugging in the exponential series, it is clear that the function $z \mapsto \varphi(z) := \frac{\exp(z)-1}{z}$ is analytic on \mathbb{C} . Thus, $\varphi(Z)$ is well defined for all matrices $Z \in \mathbb{R}^{d,d}$.


(4a)  Is the exponential Euler single step method defined in (60) consistent with the ODE $\dot{y} = f(y)$ (see [1, Def. 11.3.10])? Explain your answer.


(4b)  Show that the exponential Euler single step method defined in (60) solves the linear initial value problem

$$\dot{\mathbf{y}} = \mathbf{A} \mathbf{y}, \quad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{R}^d, \quad \mathbf{A} \in \mathbb{R}^{d,d},$$

exactly.

HINT: Recall [1, Eq. (12.1.32)]; the solution of the IVP is $\mathbf{y}(t) = \exp(\mathbf{A}t)\mathbf{y}_0$. To facilitate formal calculations, you may assume that \mathbf{A} is regular.


(4c)  Determine the region of stability of the exponential Euler single step method defined in (60) (see [1, Def. 12.1.49]).

(4d)  Write a C++ function

```
1 template <class Function, class Function2>
2 Eigen::VectorXd ExpEulStep(Eigen::VectorXd y0, Function
   f, Function2 df, double h)
```

that implements (60). Here f and df are objects with evaluation operators representing the ODE right-hand side function $f: \mathbb{R}^d \rightarrow \mathbb{R}^d$ and its Jacobian, respectively.

HINT: Use the supplied template `ExpEul_template.cpp`.

(4e)  What is the order of the single step method (60)? To investigate it, write a C++ routine that applies the method to the scalar logistic ODE

$$\dot{y} = y(1 - y), \quad y(0) = 0.1,$$

in the time interval $[0, 1]$. Show the error at the final time against the stepsize $h = T/N$, $N = 2^k$ for $k = 1, \dots, 15$. As in Problem 2 in Problem Sheet 12, for each k compute and show an approximate order of convergence.

HINT: The exact solution is

$$y(t) = \frac{y(0)}{y(0) + (1 - y(0))e^{-t}}.$$

Issue date: 10.12.2015

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

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

References

- [1] R. Hiptmair. *Lecture slides for course "Numerical Methods for CSE"*.
<http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE/NumCSE15.pdf>. 2015.