Prof. R. Hiptmair
G. Alberti,
F. Leonardi

AS 2015

ETH Zürich
D-MATH

Numerical Methods for CSE

Problem Sheet 13

Problem 1 Matrix-valued Differential Equation (core problem)

First we consider the *linear* matrix differential equation

$$\dot{\mathbf{Y}} = \mathbf{A}\mathbf{Y} =: \mathbf{f}(\mathbf{Y}) \quad \text{with} \quad \mathbf{A} \in \mathbb{R}^{n\times n}. \tag{94}$$

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

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

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

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.

Solution: Let us consider the time derivative of $\mathbf{Y}^\top\mathbf{Y}$:

$$\begin{aligned}\frac{d}{dt}(\mathbf{Y}^\top\mathbf{Y}) &= \dot{\mathbf{Y}}^\top\mathbf{Y} + \mathbf{Y}^\top\dot{\mathbf{Y}}\\ &= (\mathbf{A}\mathbf{Y})^\top\mathbf{Y} + \mathbf{Y}^\top\mathbf{A}\mathbf{Y}\\ &= -\mathbf{Y}^\top\mathbf{A}\mathbf{Y} + \mathbf{Y}^\top\mathbf{A}\mathbf{Y}\\ &= 0\end{aligned}$$

This implies $\mathbf{Y}(t)^\top\mathbf{Y}(t)$ is constant. From this, it follows that the orthogonality is preserved, as claimed. In fact $\mathbf{I} = \mathbf{Y}(0)^\top\mathbf{Y}(0) = \mathbf{Y}(t)^\top\mathbf{Y}(t)$.

(1b) ⊡ Implement three C++ functions

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

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

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

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

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

```
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 (94)

Solution: The explicit Euler method is given by

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{f}(t_k, \mathbf{Y}_k).$$

For the given differential equation we therefore obtain

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{A}\mathbf{Y}_k.$$

The implicit Euler method is given by

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{f}(t_{k+1}, \mathbf{Y}_{k+1}).$$

For the given differential equation this yields

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{A}\mathbf{Y}_{k+1} \quad \Longrightarrow \quad \mathbf{Y}_{k+1} = (\mathbf{I} - h\mathbf{A})^{-1}\mathbf{Y}_k$$

Finally, the implicit mid-point method is given by

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + h\mathbf{f}\big(\tfrac{1}{2}(t_k + t_{k+1}), \tfrac{1}{2}(\mathbf{Y}_k + \mathbf{Y}_{k+1})\big),$$

hence

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + \tfrac{h}{2}\mathbf{A}(\mathbf{Y}_k + \mathbf{Y}_{k+1}) \quad \Longrightarrow \quad \mathbf{Y}_{k+1} = \big(\mathbf{I} - \tfrac{h}{2}\mathbf{A}\big)^{-1}\big(\mathbf{Y}_k + \tfrac{h}{2}\mathbf{A}\mathbf{Y}_k\big).$$

(1c) ⊡ Investigate numerically, which one of the implemented methods preserves orthogonality in the sense of sub-problem (1a) for the ODE (94) 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 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).

Solution: Orthogonality is preserved only by the implicit midpoint rule. Explicit and implicit Euler methods do not preserve orthogonality. See `matrix_ode.cpp`.

From now we consider a non-linear ODE that is structurally similar to (94). We study the initial value problem

$$\dot{\mathbf{Y}} = -(\mathbf{Y} - \mathbf{Y}^\top)\mathbf{Y} =: f(\mathbf{Y}) \quad , \quad \mathbf{Y}(0) = \mathbf{Y}_0 \in \mathbb{R}^{n,n}, \tag{95}$$

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

(1d) ⊡ Write a C++ function

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

which solves (95) 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`.

Solution: The class `ode45` can take `Eigen::MatrixXd` as `StateType`. Alternatively, one can transform Matrices to Vectors (and viceversa), using the `Eigen::Map` function (similar to MATLAB own `reshape` function). See `matrix_ode.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 (95).

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

Solution: By the product rule and using the fact that $\mathbf{Y}$ is a solution of the IVP we obtain

$$\begin{aligned}\frac{d}{dt}(\mathbf{Y}^\top(t)\mathbf{Y}(t)) =& \frac{d}{dt}(\mathbf{Y}^\top(t))\mathbf{Y}(t) + \mathbf{Y}^\top(t)\frac{d}{dt}(\mathbf{Y}(t)) \\ =& (-(\mathbf{Y}(t) - \mathbf{Y}^\top(t))\mathbf{Y}(t))^\top\mathbf{Y}(t) + \mathbf{Y}(t)(-(\mathbf{Y}(t) - \mathbf{Y}^\top(t))\mathbf{Y}(t)) \\ =& -\mathbf{Y}^\top(t)\mathbf{Y}^\top(t)\mathbf{Y}(t) + \mathbf{Y}^\top\mathbf{Y}(t)\mathbf{Y}(t) \\ & -\mathbf{Y}^\top\mathbf{Y}(t)\mathbf{Y}(t) + \mathbf{Y}^\top(t)\mathbf{Y}^\top(t)\mathbf{Y}(t) = 0.\end{aligned}$$

implying that the map is constant.

(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`.

Solution: Let $\mathbf{Y}_k$ be the output of `matode`. We compute $\mathbf{Y}_0^\top\mathbf{Y}_0 - \mathbf{Y}_k^\top\mathbf{Y}_k$ and check if the (Frobenius) norm is smaller than a constant times the machine `eps`. Even for an orthogonal matrix, we have to take into account round-off errors. See `matrix_ode.cpp`.

(1g) ⚀ Use the function `checkinvariant` to test wether the invariant is preserved by `ode45` or not. Use the matrix $\mathbf{M}$ defined above and and $T = 1$.

Solution: The invariant is not preserved. See `matrix_ode.cpp`.

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

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1/2 & 1/4 & 1/4 & 0 \\ \hline & 1/6 & 1/6 & 2/3 \end{array} \tag{96}$$

(2a) ⊡ Consider the prey/predator model

$$\dot{y}_1(t) = (1 - y_2(t))y_1(t) \tag{97}$$
$$\dot{y}_2(t) = (y_1(t) - 1)y_2(t) \tag{98}$$
$$\mathbf{y}(0) = [100, 1]\,. \tag{99}$$

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 $2^{-j}, j = 2, \ldots, 13$.

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`.

Solution: The scheme is of order 3. With big step sizes the scheme is unstable. At least 64 steps are needed for the solution to be stable. See `stabrk.cpp`.

(2b) ⊡ Calculate the stability function $S(z)$, $z = h\lambda$, $\lambda \in \mathbb{C}$ of the method given by the table (96).

Solution: We obtain the stability function $S(z)$ by applying our method to the model problem $\dot{y}(t) = \lambda y$, $y(0) = y_0$ and by writing the result in the form $y_1 = S(z)y_0$, where $z := h\lambda$. For the increment, we obtain

$$\begin{aligned} k_1 &= \lambda y_0, \\ k_2 &= \lambda(y_0 + hk_1) \\ &= \lambda y_0(1 + z), \\ k_3 &= \lambda(y_0 + \tfrac{h}{4}(k_1 + k_2)) \\ &= \lambda y_0(1 + \tfrac{1}{2}z + \tfrac{1}{4}z^2), \end{aligned}$$

and for the update

$$\begin{aligned} y_1 &= y_0 + \frac{h\lambda y_0}{6}\left(1 + 1 + z + 4\left(1 + \frac{1}{2}z + \frac{1}{4}z^2\right)\right) \\ &= y_0 \underbrace{\left(1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^6\right)}_{=:S(z)}. \end{aligned}$$

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

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}) \tag{100}$$

with smooth right hand side $\mathbf{f}: D \to \mathbb{R}^d$, 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

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

satisfies

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

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 $\mathsf{D}_{\mathbf{y}}$

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

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

$$\dot{\mathbf{W}} = \mathsf{D}\,\mathbf{f}(\boldsymbol{\Phi}(t, \mathbf{y}))\mathbf{W}. \tag{101}$$

Here, the state $\mathbf{y}$ can be regarded as a parameter. Since $\boldsymbol{\Phi}(0, \mathbf{y}) = \mathbf{y}$, we also know $\mathbf{W}(0, \mathbf{y}) = \mathbf{I}$ (identity matrix), which supplies an initial condition for (101). In fact, we can even merge (100) and (101) into the ODE

$$\frac{d}{dt}\left[\mathbf{y}(\cdot)\,,\ \mathbf{W}(\cdot, \mathbf{y}_0)\right] = \left[\mathbf{f}(\mathbf{y}(t))\,,\ \mathsf{D}\,\mathbf{f}(\mathbf{y}(t))\mathbf{W}(t, \mathbf{y}_0)\right]\,, \tag{102}$$

which is autonomous again.

Now let us apply (101)/(102). As in [1, Ex. 11.1.9], we consider the following autonomous Lotka-Volterra differential equation of a predator-prey model

$$\begin{aligned} \dot{u} &= (2-v)u \\ \dot{v} &= (u-1)v \end{aligned} \tag{103}$$

on the state space $D = \mathbb{R}^2_+$, $\mathbb{R}_+ = \{\xi \in \mathbb{R} : \xi > 0\}$. All the solutions of (103) 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 $\mathbf{y} \in D$, (101) represents an ODE. What is its state space?

Solution: By construction, $\mathbf{\Phi}$ is a function with values in $\mathbb{R}^d$. Thus, $\mathsf{D}_\mathbf{y}\mathbf{\Phi}$ has values in $\mathbb{R}^{d,d}$, and so the state space of (101) is $\mathbb{R}^{d,d}$, a space of matrices.

(3b) ⊡ What is the right hand side function for the ODE (101), in the case of the $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ given by the Lotka-Volterra ODE (103)? You may write $u(t), v(t)$ for solutions of (103).

Solution: Writing $\mathbf{y} = [u, v]^T$, the map $\mathbf{f}$ associated to (103) is $\mathbf{f}(\mathbf{y}) = [(2-v)u, (u-1)v]^T$. Therefore

$$\mathsf{D}\mathbf{f}(\mathbf{y}) = \begin{bmatrix} 2-v & -u \\ v & u-1 \end{bmatrix}.$$

Thus, (103) becomes

$$\dot{\mathbf{W}} = \begin{bmatrix} 2-v(t) & -u(t) \\ v(t) & u(t)-1 \end{bmatrix} \mathbf{W}. \tag{104}$$

This is a non-autonomous ODE.

(3c) ⊡ From now on we write $\mathbf{\Phi}: \mathbb{R} \times \mathbb{R}^2_+ \to \mathbb{R}^2_+$ for the evolution operator associated with (103). Based on $\mathbf{\Phi}$ derive a function $\mathbf{F} : \mathbb{R}^2_+ \to \mathbb{R}^2_+$ which evaluates to zero for the input $\mathbf{y}_0$ if the period of the solution of system (103) with initial value

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

is equal to a given value T_P.

Solution: Let $\mathbf{F}$ be defined by

$$\mathbf{F}(\mathbf{y}) := \mathbf{\Phi}(T_P, \mathbf{y}) - \mathbf{y}.$$

If the solution of the system (103) with initial value $\mathbf{y}_0$ has period T_P, then we have $\mathbf{\Phi}(T_P, \mathbf{y}_0) = \mathbf{y}_0$, so $\mathbf{F}(\mathbf{y}_0) = 0$.

(3d) ⚀ We write $\mathbf{W}(T, \mathbf{y}_0)$, $T \geq 0$, $\mathbf{y}_0 \in \mathbb{R}^2_+$ for the solution of (101) for the underlying ODE (103). Express the Jacobian of $\mathbf{F}$ from (3c) by means of $\mathbf{W}$.

Solution: By definition of $\mathbf{W}$ we immediately obtain

$$\mathsf{D}\,\mathbf{F}(\mathbf{y}) = \mathbf{W}(T_P, \mathbf{y}) - \mathbf{I}$$

where $\mathbf{I}$ is the identity matrix.

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

HINT: Study [1, § 11.1.21] again. Also look at [1, Fig. 354].

Solution: If $\mathbf{y}_0 \in \mathbb{R}^2_+$ is a solution, then every state on the trajectory $\mathbf{y}_0$ will also be a solution. Only if the trajectory collapses to a point, that is, if $\mathbf{y}_0$ is a stationary point, $\mathbf{f}(\mathbf{y}_0) = 0$, we can expect uniqueness.

(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

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

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

HINT: As in (102), both ODEs (for $\boldsymbol{\Phi}$ and $\mathbf{W}$) must be combined into a single autonomous differential equation on the state space $D \times \mathbb{R}^{d \times d}$.

HINT: The equation for $\mathbf{W}$ 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).

Solution: Writing $\mathbf{w} = [u, v, W_{11}, W_{21}, W_{12}, W_{22}]^T$, by (103) and (104) we have that

$$\begin{aligned}
\dot{w}_1 &= (2 - w_2)w_1 & \dot{w}_3 &= (2 - w_2)w_3 - w_1 w_4 & \dot{w}_5 &= (2 - w_2)w_5 - w_1 w_6 \\
\dot{w}_2 &= (w_1 - 1)w_2 & \dot{w}_4 &= w_2 w_3 + (w_1 - 1)w_4 & \dot{w}_6 &= w_2 w_5 + (w_1 - 1)w_6
\end{aligned}$$

with initial conditions

$$w_1(0) = u_0, \quad w_2(0) = v_0, \quad w_3(0) = 1, \quad w_4(0) = 0, \quad w_5(0) = 0, \quad w_6(0) = 1,$$

since $\mathbf{W}(0, \mathbf{y}) = \mathsf{D}_{\mathbf{y}}\, \mathbf{\Phi}(0, \mathbf{y}) = \mathsf{D}_{\mathbf{y}}\, \mathbf{y} = \mathbf{I}$.

The implementation of the solution of this system is given in the file `LV.cpp`.

(3g) ⚄ Using `PhiAndW`, write a C++ routine that determines initial conditions $u(0)$ and $v(0)$ such that the solution of the system (103) has period $T = 5$. Use the multi-dimensional *Newton method* for $\mathbf{F}(\mathbf{y}) = 0$ with $\mathbf{F}$ from (3c). As your initial approximation, use $[3, 2]^T$. Terminate the *Newton method* as soon as $\|\mathbf{F}(\mathbf{y})\| \leq 10^{-5}$. Validate your implementation by comparing the obtained initial data $\mathbf{y}$ with $\mathbf{\Phi}(100, \mathbf{y})$.

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 $v(0) = 2.081$.

Solution: See file `LV.cpp`.

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 $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ with continuously differentiable $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^d$ reads

$$\mathbf{y}_1 = \mathbf{y}_0 + h\, \varphi\big(h\, \mathsf{D}\,\mathbf{f}(\mathbf{y}_0)\big)\, \mathbf{f}(\mathbf{y}_0), \tag{105}$$

where $\mathsf{D}\,\mathbf{f}(\mathbf{y}) \in \mathbb{R}^{d,d}$ is the Jacobian of $\mathbf{f}$ at $\mathbf{y} \in \mathbb{R}^d$, and the matrix function $\varphi : \mathbb{R}^{d,d} \to \mathbb{R}^{d,d}$ is defined as $\varphi(\mathbf{Z}) = (\exp(\mathbf{Z}) - \mathsf{Id})\, \mathbf{Z}^{-1}$. Here $\exp(\mathbf{Z})$ is the matrix exponential of $\mathbf{Z}$, a special function $\exp : \mathbb{R}^{d,d} \to \mathbb{R}^{d,d}$, 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, $\varphi(\mathbf{Z})$ is well defined for all matrices $\mathbf{Z} \in \mathbb{R}^{d,d}$.

(4a) ⚀ Is the exponential Euler single step method defined in (105) consistent with the ODE $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ (see [1, Def. 11.3.10])? Explain your answer.

Solution: In view of (105), consistency is equivalent to $\varphi\big(h\,\mathsf{D}\mathbf{f}(\mathbf{y}_0)\big)\,\mathbf{f}(\mathbf{y}_0) = \mathbf{f}(\mathbf{y}_0)$ for $h = 0$. Since φ is not defined for the zero matrix, this should be intended in the limit as $h \to 0$. By definition of the matrix exponential we have $e^{hZ} = \mathsf{Id} + hZ + O(h^2)$ as $h \to 0$. Therefore

$$\varphi(h\,\mathsf{D}\mathbf{f}(\mathbf{y}_0)) = (\exp(h\,\mathsf{D}\mathbf{f}(\mathbf{y}_0)) - \mathsf{Id})(h\,\mathsf{D}\mathbf{f}(\mathbf{y}_0))^{-1} = (\mathsf{D}\mathbf{f}(\mathbf{y}_0)) + O(h))\,\mathsf{D}\mathbf{f}(\mathbf{y}_0)^{-1},$$

whence

$$\lim_{h\to 0} \varphi\big(h\,\mathsf{D}\mathbf{f}(\mathbf{y}_0)\big)\,\mathbf{f}(\mathbf{y}_0) = \mathbf{f}(\mathbf{y}_0),$$

as desired.

(4b) ⚀ Show that the exponential Euler single step method defined in (105) solves the linear initial value problem

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

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.

Solution: Given $\mathbf{y}(0)$, a step of size t of the method gives

$$\begin{aligned}
\mathbf{y}_1 &= \mathbf{y}(0) + t\,\varphi\big(t\ \mathsf{D}\mathbf{f}(\mathbf{y}(0))\big)\,\mathbf{f}(\mathbf{y}(0)) \\
&= \mathbf{y}(0) + t\,\varphi\big(t\,\mathbf{A}\big)\,\mathbf{A}\,\mathbf{y}(0) \\
&= \mathbf{y}(0) + t\,\big(\exp(\mathbf{A}t) - \mathsf{Id}\big)\,(t\mathbf{A})^{-1}\,\mathbf{A}\,\mathbf{y}(0) \\
&= \mathbf{y}(0) + \exp(\mathbf{A}t)\,\mathbf{y}(0) - \mathbf{y}(0) \\
&= \exp(\mathbf{A}t)\,\mathbf{y}(0) \\
&= \mathbf{y}(t).
\end{aligned}$$

(4c) ⚀ Determine the region of stability of the exponential Euler single step method defined in (105) (see [1, Def. 12.1.49]).

Solution: The discrete evolution $\boldsymbol{\Psi}^h$ associated to (105) is given by

$$\boldsymbol{\Psi}^h(y) = y + h\varphi(h\,\mathsf{D}\mathbf{f}(\mathbf{y}))\mathbf{f}(\mathbf{y}).$$

Thus, for a scalar linear ODE of the form $\dot{y} = \lambda y$ for $\lambda \in \mathbb{C}$ we have ($f(y) = \lambda y$, $Df(y) = \lambda$)

$$\boldsymbol{\Psi}^h(y) = y + h\frac{e^{h\lambda} - 1}{h\lambda}\lambda y = e^{h\lambda}y.$$

Thus, the stability function $S:\mathbb{C} \to \mathbb{C}$ is given by $S(z) = e^z$. Therefore, the region of stability of this method is

$$\mathcal{S}_{\boldsymbol{\Psi}} = \{z \in \mathbb{C} : |e^z| < 1\} = \{z \in \mathbb{C} : e^{\operatorname{Re} z} < 1\} = \{z \in \mathbb{C} : \operatorname{Re} z < 0\}.$$

It is useful to compare this with [1, Ex. 12.1.51].

Alternatively, one may simply appeal to (4b) to see that the method has the "ideal" region of stability as introduced in [1, § 12.3.33].

(4d) ⚀ Write a C++ function

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

that implements (105). Here `f` and `df` are objects with evaluation operators representing the ODE right-hand side function $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^d$ and its Jacobian, respectively.

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

Solution: See `ExpEul.cpp`.

(4e) ⚄ What is the order of the single step method (105)? 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 $k = 1, \ldots, 15$. As in Problem 2 in Problem Sheet 12, for each k compute and show an approximate order of convergence.

HINT: The exact solution is

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

Solution: Error = $O(h^2)$. See `ExpEul.cpp`.

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

Version compiled on: December 17, 2015 (v. 1.0).