Exam (Solutions)		
EPFL	**Course:**	Numerical Analysis - Sections SIE-GC
	Lecturer:	Daniel Kressner
	Date:	29/01/2025 **Duration:** 3h
Sciper:	**Student:**	**Section:**

Evaluation table							
Ex.1	**Ex.2**	**Ex.3**	**Ex.4**	**Ex.5**	**Ex.6**	**Ex.7**	**Total**

Do not turn the page before the start of the exam. Read carefully the instructions below.

EXAM RULES

- Write everything with a blue or black pen. Do **NOT** use a pencil.
- Please write your surname, name, and Sciper number on **EVERY PAGE** of this document!
- All results (the plots if they are requested, too) and Python code (when requested) **MUST BE TRANSCRIBED ON THESE SHEETS**, which must be submitted at the end of the exam.
- It is **NOT** possible to submit the Python/Jupyter code electronically.
- Scratch paper is provided for your personal notes, but it will not be collected and not be considered for the corrections.
- Numerical results involving floating point numbers need to be reported to at least three significant decimal digits. For example, if the result is 1.234657809E-2, write 1.23E-2 or 0.0123.
- Please note that the exam is **DOUBLE-SIDED**.

AUTHORIZED MATERIAL

The only authorized material is **1 A4 cheat sheet (front and back) handwritten with pen/pencil**. No additional sheets, notes or books (paper or electronics), or calculator, mobile phone, tablet, laptop or other electronic devices are admitted. The access to the Internet (e-mail, websites) is prohibited.

VIRTUAL MACHINE LOGIN INSTRUCTIONS

Before the start of the exam:

1) Sign in with your **GASPAR username and password** (same as for IS-Academia, Moodle, ...);
2) Choose the **MATH-251a** virtual machine client;

***After* the exam has started:**

3) Open the **Jupyter Notebook** application by double clicking the icon on the Desktop;
4) In the Jupyter Notebook file browser, go to **Desktop** and then click on **exam2025.ipynb**.

We provide you with a copy of the exam notebook called **exam2025_copy.ipynb**. If you make some unwanted modifications and do not know how to revert them, you can compare with the copy.

Sciper: **Student:** **Section:**

Exercise 1 (0.7 Points in total)

Given some function $f : [-1, 1] \to \mathbb{R}$, consider the approximation of the integral

$$I(f) = \int_{-1}^{1} f(x)\ \mathrm{d}x$$

by the quadrature formula

$$Q(f) = \sum_{i=1}^{2} \alpha_i f(x_i) \tag{1}$$

for the quadrature weights $\alpha_1 = 1$ and $\alpha_2 = 1$, and quadrature nodes $x_1 = -\frac{1}{3}$ and $x_2 = +\frac{1}{3}$.

Question 1 (0.3 Points) Determine the degree of exactness of the quadrature formula (1). *Provide full justification for your answer, incuding all derivations.*

Solution: *We check for which monomials $f(x) = x^s$, $s = 0, 1, \ldots$ the quadrature rule gives the exact integral:*

$$I(1) = \int_{-1}^{1} 1\ \mathrm{d}x = 2$$
$$Q(1) = 1 + 1 = 2$$

$$I(x) = \int_{-1}^{1} x\ \mathrm{d}x = 0$$
$$Q(x) = -\frac{1}{3} + \frac{1}{3} = 0$$

$$I(x^2) = \int_{-1}^{1} x^2\ \mathrm{d}x = \frac{2}{3}$$
$$Q(x^2) = \frac{1}{9} + \frac{1}{9} = \frac{2}{9}$$

Since the quadrature formula integrates monomials up to degree 1 exactly, but no longer monomials of degree 2, the degree of exactness is 1.

Point distribution:

- +0.1 for the correct derivation of each monomial;
- +0.05 for the correct degree of exactness;
- −0.05 if all derivations are correct but the degree of exactness is incorrect.

Question 2 (0.4 Points) Consider the integrals

$$\int_{-2}^{1} f_1(x)\,\mathrm{d}x \quad \text{for} \quad f_1(x) = \sin(x),$$

and

$$\int_{0}^{1} f_2(x)\,\mathrm{d}x \quad \text{for} \quad f_2(x) = \sqrt{x}.$$

The function `composite_quadrature` in the Jupyter notebook implements the composite quadrature formula $Q_h(f)$ corresponding to (1) for a given function f on an arbitrary interval $[a, b]$ with n sub-intervals. For both integrals, use this Python function to: (i) compute the error of the composite quadrature formula for $n = 10$ sub-intervals, and (ii) numerically determine the order of convergence of the composite quadrature formula. Complete the table below with the errors rounded to three significant decimal digits.

Hint: The primitives of f_1 and f_2 are $F_1(x) = -\cos(x)$ and $F_2 = \frac{2}{3}\sqrt{x}^3$.

Coefficient	$Q_h(f_1)$	$Q_h(f_2)$
(i) error for $n = 10$	1.58×10^{-2} or 0.0158	2.83×10^{-3} or 0.00283
(ii) order of convergence	1	1

Point distribution:

- +0.1 for each correct answer;
- rounding errors are tolerated.

Sciper: **Student:** **Section:**

Exercise 2 (1.2 Points in total)

This is a multiple choice exercise. **One, and only one, answer is correct for every question**. Clearly mark your answer choice with a cross. Use correction fluid (Tipp-Ex) to remove a cross if you have accidentally crossed an answer. If needed, you can request Tipp-Ex from the proctors. 0.2 points are given if the correct answer is crossed and none of the other three answers is crossed. In all other cases, 0 points are given.

Question 1 (0.2 Points) Consider the data

$$\begin{array}{ll} x_1 = -1 & y_1 = 3 \\ x_2 = 1 & y_2 = 0 \\ x_3 = 2 & y_3 = \beta \\ x_4 = 4 & y_4 = \beta. \end{array}$$

Let p_3 be the degree 3 interpolating polynomial for this data, that is, $p_3(x_i) = y_i$ for $i = 1, \ldots, 4$. Determine the value of β for which $p_3(0) = -1$ holds.

☐ $-\frac{3}{11}$

☐ $\frac{4}{3}$

☒ 3

☐ 6

Solution: *The degree 3 interpolating polynomial is given by the Lagrange interpolation formula (lecture notes, Proposition 2.1)*

$$\begin{aligned} p_3(x) &= y_1 \frac{(x-x_2)(x-x_3)(x-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)} + y_2 \frac{(x-x_1)(x-x_3)(x-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)} \\ &\quad + y_3 \frac{(x-x_1)(x-x_2)(x-x_4)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)} + y_4 \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)} \\ &= 3\frac{(x-1)(x-2)(x-4)}{(-1-1)(-1-2)(-1-4)} + \beta \frac{(x+1)(x-1)(x-4)}{(2+1)(2-1)(2-4)} + \beta \frac{(x+1)(x-1)(x-2)}{(4+1)(4-1)(4-2)} \\ &= -\frac{1}{10}(x-1)(x-2)(x-4) + -\frac{\beta}{6}(x+1)(x-1)(x-4) + \frac{\beta}{30}(x+1)(x-1)(x-2). \end{aligned}$$

Enforcing the condition $p_3(0) = -1$ results in the value

$$p_3(0) = \frac{4}{5} + -\frac{2\beta}{3} + \frac{\beta}{15} = \frac{4}{5} - \frac{9\beta}{15} = -1 \quad \Longleftrightarrow \quad \beta = 3.$$

Point distribution:

- +0.2 for the correct answer;
- 0 points if crosses are ambiguous or multiple boxes are crossed.

Question 2 (0.2 Points) The following Python function implements an unknown composite quadrature formula for approximating an integral $\int_a^b f(x)\,\mathrm{d}x$.

```
def unknown_composite_quadrature(a, b, n, f):
    # Unknown composite quadrature formula
    # - a,b: lower/upper bounds of integration interval
    # - n: number of sub-intervals
    # - f: function to integrate
    h = (b - a) / n
    xi = np.linspace(a, b, n + 1)  # sub-interval boundaries
    alphai = (h / 3) * np.hstack((0.5, np.ones(n - 1), 0.5))  # weights at xi
    ci = np.linspace(a + h / 2, b - h / 2, n)  # sub-interval mid-points
    betai = (2 * h / 3) * np.ones(n)  # weights at ci
    Qh = np.dot(alphai, f(xi)) + np.dot(betai, f(ci))
    return Qh
```

Which composite quadrature formula does this function compute?

☐ Composite midpoint formula

☐ Composite trapezoidal formula

☒ Composite Simpson formula

☐ Composite Gaussian formula

Solution: *This identical code appears in the course notes (lecture notes, page 64).*

Point distribution:

- +0.2 for the correct answer;
- 0 points if crosses are ambiguous or multiple boxes are crossed.

Question 3 (0.2 Points) Suppose that the Newton method is applied to the system of equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ with

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} x_1 + \cos(x_2) \\ 5 - x_2 \end{bmatrix}$$

and initial point $\mathbf{x}^{(0)} = [x_1^{(0)}, x_2^{(0)}]^\top$. Then the iterates $\mathbf{x}^{(k)} = [x_1^{(k)}, x_2^{(k)}]^\top$, $k = 1, 2, \ldots$, produced by the Newton method satisfy the recursion:

☒ $\begin{cases} x_1^{(k+1)} = \sin(x_2^{(k)})(5 - x_2^{(k)}) - \cos(x_2^{(k)}) \\ x_2^{(k+1)} = 5 \end{cases}$

☐ $\begin{cases} x_1^{(k+1)} = \cos(x_2^{(k)}) + \sin(x_2^{(k)})(x_2^{(k)} - 5) \\ x_2^{(k+1)} = 5 \end{cases}$

☐ $\begin{cases} x_1^{(k+1)} = x_1^{(k)} - x_1^{(k+1)} - \cos(x_2^{(k+1)}) - \sin(x_2^{(k+1)})(x_2^{(k+1)} - 5) \\ x_2^{(k+1)} = x_2^{(k)} + 5 - x_2^{(k+1)} \end{cases}$

☐ $\begin{cases} x_1^{(k+1)} = \cos(x_2^{(k)}) + \sin(x_2^{(k)})(x_2^{(k)} - 5) - x_1^{(k)} \\ x_2^{(k+1)} = 5 - x_2^{(k)} \end{cases}$

Solution: *The Jacobian of* $\mathbf{f}$ *at* $\mathbf{x}$ *satisfies*

$$J_{\mathbf{f}}(\mathbf{x}) = \begin{bmatrix} 1 & -\sin(x_2) \\ 0 & -1 \end{bmatrix} \quad \Longrightarrow \quad J_{\mathbf{f}}(\mathbf{x})^{-1} = \begin{bmatrix} -1 & \sin(x_2) \\ 0 & 1 \end{bmatrix}.$$

Therefore, the Newton method iterates $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J_{\mathbf{f}}(\mathbf{x}^{(k)})^{-1}\mathbf{f}(\mathbf{x}^{(k)})$ (lecture notes, page 28) are

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} + \begin{bmatrix} -1 & \sin(x_2^{(k)}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1^{(k)} + \cos(x_2^{(k)}) \\ 5 - x_2^{(k)} \end{bmatrix} = \begin{bmatrix} -\cos(x_2^{(k)}) + \sin(x_2^{(k)})(5 - x_2^{(k)}) \\ 5 \end{bmatrix}.$$

Point distribution:

- +0.2 for the correct answer;
- 0 points if crosses are ambiguous or multiple boxes are crossed.

Question 4 (0.2 Points) Consider the following linear system of ordinary differential equations:

$$\frac{\mathrm{d}\mathbf{u}(t)}{\mathrm{d}t} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 4 & -2 & -2 \end{bmatrix} \mathbf{u}(t) + \mathbf{b}(t), \text{ with } t > 0, \ \mathbf{u}(0) = \mathbf{u}_0, \tag{2}$$

and some $\mathbf{b} : \mathbb{R} \to \mathbb{R}^3$, $\mathbf{u}_0 \in \mathbb{R}^3$. The backward (implicit) Euler method applied to (2) with fixed time step size $\Delta t > 0$ is

☐ absolutely stable if and only if $0 < \Delta t < 1$.

☐ absolutely stable if and only if $0 < \Delta t < 2$.

☒ absolutely stable for every $\Delta t > 0$ (unconditionally absolutely stable).

☐ not absolutely stable for any $\Delta t > 0$.

Solution: *By Lemma 6.4 (lecture notes), the backward Euler method applied to any linear system is unconditionally absolutely stable.*

Point distribution:

- +0.2 for the correct answer;
- 0 points if crosses are ambiguous or multiple boxes are crossed.

Question 5 (0.2 Points) Given a smooth function $f : \mathbb{R} \to \mathbb{R}$ and $\bar{x} \in \mathbb{R}$, consider the finite differences formula

$$\delta_h f(\bar{x}) := \frac{f(\bar{x} - h) - 2f(\bar{x}) + f(\bar{x} + 2h)}{h}.$$

Of which order is this formula for approximating the derivative of the function f at $\bar{x} \in \mathbb{R}$?

☐ 0

☒ 1

☐ 2

☐ 3

Solution: *The Taylor expansions around $\bar{x}$ are*

$$f(\bar{x} - h) = f(\bar{x}) - hf'(\bar{x}) + \frac{h^2}{2} f''(\bar{x}) + \mathcal{O}(h^3)$$
$$f(\bar{x} + 2h) = f(\bar{x}) + 2hf'(\bar{x}) + 2h^2 f''(\bar{x}) + \mathcal{O}(h^3).$$

Inserting these expansions into the finite differences formula, we get

$$\delta_h f(\bar{x}) = f'(\bar{x}) + \frac{5h}{2} f''(x) + \mathcal{O}(h^2).$$

Hence, the method is of order 1.

Point distribution:

- +0.2 for the correct answer;
- 0 points if crosses are ambiguous or multiple boxes are crossed.

Question 6 (0.2 Points) Consider a linear system $A\mathbf{x} = \mathbf{b}$ with

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

and some $\mathbf{b} \in \mathbb{R}^3$. For which values of $\omega > 0$ does the Richardson method defined by

$$P\mathbf{x}^{(k+1)} = (P - A)\mathbf{x}^{(k)} + \mathbf{b}, \quad k = 0, 1, 2, \ldots,$$

with $P = \omega I$ converge to the solution $\mathbf{x}$ for every starting vector $\mathbf{x}^{(0)}$ and right-hand side $\mathbf{b}$?

☐ For $\omega > 1$ but not for $0 < \omega < 1$.

☐ For $\omega > \frac{1}{2}$ but not for $0 < \omega < \frac{1}{2}$.

☒ For $\omega > \frac{3}{2}$ but not for $0 < \omega < \frac{3}{2}$.

☐ There is no such $\omega > 0$.

Solution: *By Theorem 5.2 (lecture notes), the Richardson method converges if and only if the spectral radius $\rho(B)$ of $B = I - P^{-1}A$ is smaller than 1. The eigenvalues of B are clearly*

$$1 - \frac{1}{\omega}, 1 - \frac{2}{\omega}, \text{ and } 1 - \frac{3}{\omega}.$$

Then, for $\omega > 3/2$, $\rho(B) < 1$ and for $\omega < 3/2$, $\rho(B) > 1$.

Point distribution:

- +0.2 for the correct answer;
- 0 points if crosses are ambiguous or multiple boxes are crossed.

Exercise 3 (0.3 Points in total)

The following Python function implements a fixed-point iteration.

```
def fixed_point(x0, tol, nmax):
    """
    Fixed point iteration.

    Parameters
    ----------
    x0 : float
        The initial guess for the fixed point.
    tol : float
        The desired tolerance for the fixed point.
    nmax : int
        The maximum number of iterations.

    Returns
    -------
    x_seq : array-like
        The successive values of the fixed point iterations.
    res : array-like
        The value of the residual at each iteration.
    niter : int
        The number of iterations performed.
    """

    niter = 0
    x_seq = []
    x_seq.append(x0)
    xt = 2 * x0 - (x0 - 1) ** 3 - 2 * np.log(x0) - 1

    res = []
    res.append(x0 - xt) # this measures "how far x0 is from the fixed point"
    while (abs(res[-1]) > tol) and (niter < nmax):
        niter = niter + 1
        x_seq.append(xt)
        x0 = xt
        xt = 2 * x0 - (x0 - 1) ** 3 - 2 * np.log(x0) - 1
        res.append(abs(x0 - xt))
        if niter ≥ nmax:
            print("maximum iterations reached without achieving desired tolerance")

    # convert from list to array
    x_seq = np.array(x_seq)
    res = np.array(res)

    return x_seq, res, niter
```

Question 1 (0.2 Points) Write down the fixed-point iteration. That is, determine the function ϕ that produces the sequence $x^{(k+1)} = \phi(x^{(k)})$, $k = 0, 1, 2, \ldots$, corresponding to the output `x_seq` of the function `fixed_point`.

Solution: *We look at the line* `xt = 2 * x0 - (x0 - 1) ** 3 - 2 * np.log(x) - 1` *to see that the function ϕ is given by*

$$\underline{\underline{\phi(x) = 2x - (x-1)^3 - 2\log(x) - 1.}}$$

Point distribution:

- $+0.2$ for correct expression ϕ;
- -0.1 if ϕ isn't defined but the correct recurrence is given;
- Expression in terms of NumPy functions are tolerated.

Question 2 (0.1 Point) For the fixed point $\alpha = 1$ and ϕ from Question 1, determine theoretically the order of the fixed-point iteration $x^{(k+1)} = \phi(x^{(k)})$, $k = 0, 1, 2, \ldots$. *Provide full justification for your answer, incuding all derivations.*

Solution: *By Theorem 1.2 from the lecture notes, since ϕ is smooth in a neighborhood of $\alpha = 1$, we can check the derivatives of the function ϕ evaluated at the fixed point $\alpha = 1$:*

$$\begin{aligned}
\phi'(x) &= 2 - 3(x-1)^2 - \frac{2}{x} \quad \Longrightarrow \quad \phi'(1) = 0 \\
\phi''(x) &= -6(x-1) + \frac{2}{x^2} \quad \Longrightarrow \quad \phi''(1) = 2 \neq 0 \\
\phi'''(x) &= -6 - \frac{4}{x^3} \quad \Longrightarrow \quad \phi'''(1) = -10 \neq 0 \\
\phi^{(s)}(x) &= -\frac{2(s-1)!}{x^s}, s \geq 4 \quad \Longrightarrow \quad \phi^{(s)}(1) \neq 0, s \geq 4.
\end{aligned}$$

Since $\phi^{(s)}(\alpha) = 0$ for $s = 1$ but $\phi^{(s)}(\alpha) \neq 0$ for $s \geq 2$, the method is of order 2.

Point distribution:

- $+0.05$ if the derivatives $\phi'(1)$ and $\phi''(1)$ are correctly calculated;
- $+0.05$ point if based on the calculation the correct order is determined;
- -0.05 if $\phi''(1) \neq 0$ is not checked;
- 0 points in any other case (missing derivative computations, wrong justification, ...).

Exercise 4 (0.3 Points in total)

Consider the Hilbert matrix $A \in \mathbb{R}^{n\times n}$, which has the entries

$$a_{ij} = \frac{1}{i+j-1}, \quad i,j = 1,2,\ldots,n.$$

The function `hilbert_matrix` in the Jupyter notebook generates this matrix.

Question 1 (0.1 Point) Numerically compute the (spectral) condition number $\kappa(A) = \|A^{-1}\|\|A\|$ of the Hilbert matrix for $n = 12$, where $\|\cdot\|$ denotes the spectral norm of a matrix.

Computed value of $\kappa(A)$ (rounded to three significant decimal digits):

Solution: *Using* `np.linalg.cond`*, we get* $\underline{\kappa(A) = 1.75 \times 10^{16}}$.

Point distribution:

- +0.1 for correct values of $\kappa(A)$;
- small deviations from the correct value are tolerated.

Question 2 (0.1 Point) Let $\mathbf{x}$ solve the system $A\mathbf{x} = \mathbf{b}$ for some nonzero vector $\mathbf{b} \in \mathbb{R}^{12}$, where A is the Hilbert matrix from above for $n = 12$. Let $\widehat{\mathbf{x}}$ solve the system $A\widehat{\mathbf{x}} = \widehat{\mathbf{b}}$ for a different non-zero vector $\widehat{\mathbf{b}} \in \mathbb{R}^{12}$. Suppose that the entries of $\mathbf{b}, \widehat{\mathbf{b}}$ satisfy $|\hat{b}_i - b_i| \leq \epsilon |b_i|$ for $i = 1,\ldots,12$ and some $\epsilon > 0$. Provide a value for α such that the inequality

$$\frac{\|\widehat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \alpha\epsilon \tag{3}$$

holds. Briefly justify your choice.

Solution: *By Lemma 4.3 from the lecture notes,* $\underline{\alpha = \kappa(A) = 1.75 \times 10^{16}}$ *verifies this inequality. To see this, realize that* $|\hat{b}_i - b_i| \leq \epsilon|b_i| \implies \hat{b}_i - b_i = s_i\epsilon b_i \iff \hat{b}_i = b_i(1 + s_i\epsilon)$ *for some* $s_i \in [-1,1]$, $i = 1,2,\ldots,12$. *Hence, Lemma 4.3 with* $\epsilon_{\max} = \max_{i=1,2,\ldots,12} |s_i\epsilon| \leq \epsilon$ *yields the inequality.*

Point distribution:

- +0.1 if either $\alpha = \kappa(A)$ or the value from Question 1 is given.

Question 3 (0.1 Point) Validate numerically that the inequality (3) holds for the Hilbert matrix A for $n = 12$ and the value of α you determined in Question 2. For this purpose, choose $\mathbf{x} = [1, 2, \ldots, 12]^\top \in \mathbb{R}^{12}$ and compute $\mathbf{b} := A\mathbf{x}$. Further, let $\widehat{\mathbf{x}}$ be the solution produced by the NumPy function `np.linalg.solve(A, b)` and compute $\widehat{\mathbf{b}} := A\widehat{\mathbf{x}}$. Then compute and write down the left- and right-hand side of (3), rounded to three significant decimal digits.

Solution: *We get*

$$\frac{\|\widehat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \underline{\underline{4.77 \times 10^{-2} \text{ or } 0.0477}},$$

and

$$\max_i \frac{|\hat{b}_i - b_i|}{|b_i|} \kappa(A) = \underline{\underline{3.34}}.$$

Clearly, the inequality holds for this specific example.

Point distribution:

- +0.05 if left-hand side is correctly computed;
- +0.05 if right-hand side is correctly computed;
- tolerated if correct value for ε not just 10^{-16} but justified it with something like $\frac{\|b-\hat{b}\|_2}{\|b\|_2}$.

Exercise 5 (0.7 Points in total)

Question 1 (0.3 Points) Complete the following Python function such that it implements the Heun method.

```
def heun(f, u_0, T, N):
    """
    Solves an ordinary differential equation
        u' = f(t,u), t in (0, T],
        u(0) = u_0
    using the Heun method on an equispaced grid of stepsize dt = T / N.

    Parameters
    ----------
    f : function
    u_0 : float
        Starting value.
    T : float > 0
        End point.
    N : int
        Number of time steps

    Returns
    -------
    u : NumPy array
        Approximate solution np.array([u_0, u_1, u_2,...,u_N])
    """

    u = np.zeros(N + 1)

    ### BEGIN SOLUTION
    u[0] = u_0
    dt = T / N
    ### END SOLUTION

    for n in range(N):

        ### BEGIN SOLUTION
        u_FE = u[n] + dt * f(n * dt, u[n])
        u[n + 1] = u[n] + dt / 2 * f(n * dt, u[n]) + dt / 2 * f((n + 1) * dt, u_FE)
        ### END SOLUTION

    return u
```

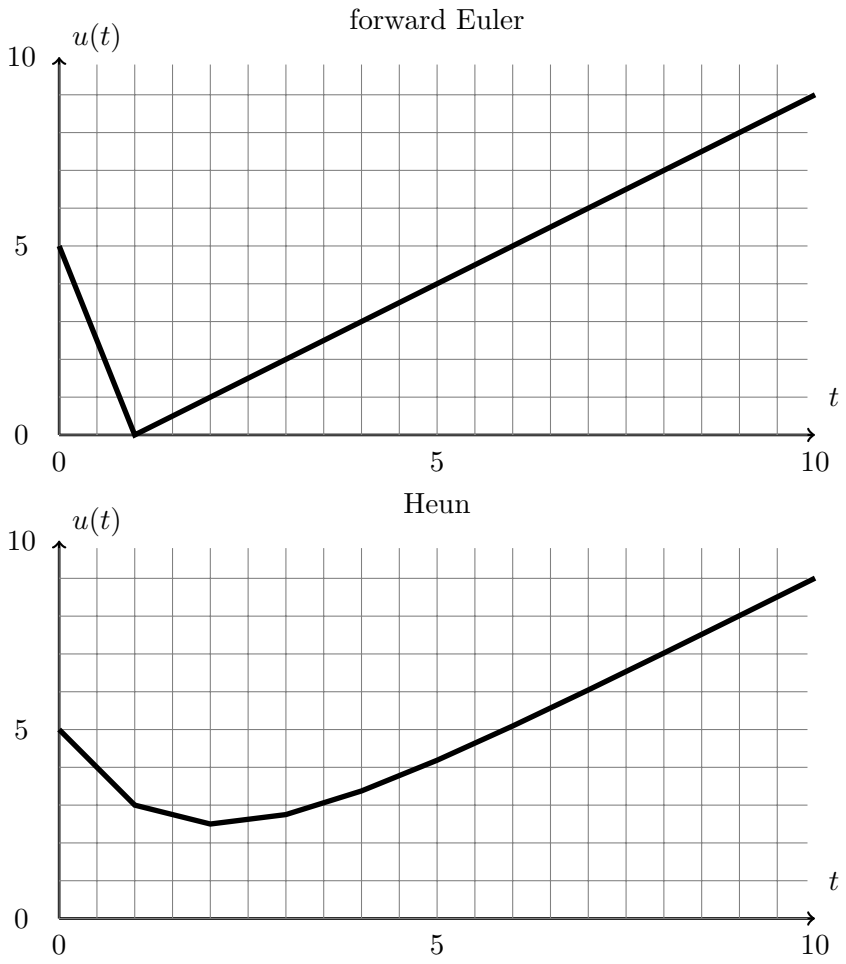
Point distribution:

- −0.1 for correct variable initialization;
- +0.1 for correct intermediate step (forward Euler);
- +0.1 for correct next iterate;
- also fine if intermediate is used directly in the next iterate, and the formula as a whole is correct;
- small syntactic errors in the code (wrongly placed parantheses, ...) are tolerated.

Question 2 (0.2 Points) Consider the ordinary differential equation

$$\begin{cases} \dfrac{\mathrm{d}u(t)}{\mathrm{d}t} = t - u(t), \quad t \in (0, T] \\ u(0) = 5. \end{cases} \tag{4}$$

The Jupyter notebook provides the function `forward_euler`, which implements the forward (explicit) Euler method. Use this implementation of the forward Euler method and your implementation of the Heun method from Question 1 to solve (4) for $T = 10$ and $N = 10$ steps. Sketch the two approximations of the solution $u(t)$ in the plots below. *Do not forget to label the y-axis.*



Point distribution:

- +0.1 per plot if the general shape is correct, the axes are labeled and values approximately correspond to the solution;
- If code from Question 1 was wrong and the graph reflects that mistake, tolerate it as a follow-up mistake;
- −0.1 if graphs are correct, but order is swapped.

Question 3 (0.2 Points) Numerically determine the order of convergence of the forward Euler and Heun method applied to problem (4).

Hint: The exact solution of (4) is $u(t) = 6\exp(-t) + t - 1$.

	forward Euler	Heun
order of convergence	1	2

Point distribution:

- +0.1 for every correct order.

Exercise 6 (0.6 Points in total)

Consider the function

$$f(x) = (1 - x)\cos(x), \quad x \in [0, 1]. \tag{5}$$

Question 1 (0.2 Points) Use Python to compute the degree 5 interpolating polynomial p of this function f at the uniformly spaced nodes $x_i = \frac{i}{5}$ for $i = 0, 1, \ldots, 5$. Write down the Python code below and report the value of $p(0.4)$.

Python code:

Solution:

```
### BEGIN SOLUTION
f = lambda x: (1 - x) * np.cos(x)
x = np.linspace(0, 1, 6)

p = np.polyfit(x, f(x), 5)
p_eval = np.polyval(p, 0.4)
print("Evaluation: p(0.4) = {:.3f}".format(p_eval))
### END SOLUTION
```

Value of $p(0.4)$ rounded to three decimal digits:

$p(0.4) = 5.53 \times 10^{-1}$ or 0.553

Point distribution:

- $+0.2$ if everything is correct;
- -0.1 if there is a small mistake;
- 0 otherwise.

Question 2 (0.2 Points) Use Python to compute the derivative p' of the interpolating polynomial p from Question 1. Write down the Python code below and report the error $|p'(0.4) - f'(0.4)|$.

Python code:

```
### BEGIN SOLUTION
p_der = p[:-1] * np.arange(5, 0, -1)
p_der_eval = np.polyval(p_der, 0.4)
f_der_eval = - np.cos(0.4) - (1 - 0.4) * np.sin(0.4)
error = abs(p_der_eval - f_der_eval)
print("Error: |p'(0.4) - f'(0.4)| = {:.3f}".format(error))
### END SOLUTION
```

Value of $|p'(0.4) - f'(0.4)|$ rounded to three decimal digits:

$\underline{\underline{|p'(0.4) - f'(0.4)| = 1.24 \times 10^{-5} \text{ or } 0.0000124}}$

Point distribution:

- +0.2 if everything is correct (polyder and finite differences accepted);
- −0.1 if there is a small mistake (e.g., order, coefficients);
- 0 otherwise.

Question 3 (0.2 Points) Use Python to compute the integral $I(p) = \int_0^1 p(x)\ \mathrm{d}x$ of the interpolating polynomial p from Question 1. Write down the Python code below and report the error $|I(p) - I(f)|$. *Hint*: $I(f) = \int_0^1 f\ \mathrm{d}x = 1 - \cos(1)$.

Python code:

```
### BEGIN SOLUTION
p_int = np.sum(p * 1 / np.arange(6, 0, -1))
f_int = 1 - np.cos(1)
error = abs(p_int - f_int)
print("Error: |I(p) - I(f)| = {:.3f}".format(error))
### END SOLUTION
```

Value of $|I(f) - I(p)|$ rounded to three decimal digits:

$\underline{\underline{|I(p) - I(f)| = 7.00 \times 10^{-7} \text{ or } 0.0000007}}$

Point distribution:

- +0.2 if everything is correct (polyint accepted);
- −0.1 if there is a small mistake (e.g., order, coefficients);
- 0 otherwise.

Exercise 7 (0.7 Points in total)

Given data $x_i \in \mathbb{R}$, $y_i \in \mathbb{R}$, with $i = 1, 2, \ldots, n+1$, we aim at approximating this data by a function of the form

$$q(x) = a_0 + a_1 x + a_2 e^x$$

for some real coefficients $a_0, a_1, a_2 \in \mathbb{R}$. More precisely, we want to find the least-squares approximation, i.e., the coefficients $a_0, a_1, a_2 \in \mathbb{R}$ that minimize the least-squares error:

$$\min_{a_0,a_1,a_2 \in \mathbb{R}} \sum_{i=1}^{n+1} (y_i - (a_0 + a_1 x_i + a_2 e^{x_i}))^2. \tag{6}$$

Question 1 (0.3 Points) Rewrite the minimization problem (6) in matrix-vector form. More precisely, determine a matrix $V \in \mathbb{R}^{(n+1)\times 3}$ such that the vector $\boldsymbol{a} = [a_0, a_1, a_2]^\top \in \mathbb{R}^3$ containing the solution a_0, a_1, a_2 of (6) satisfies the minimization problem

$$\min_{\mathbf{a} \in \mathbb{R}^3} \|\mathbf{y} - V\mathbf{a}\|^2, \tag{7}$$

with $\mathbf{y} = [y_1, y_2, \ldots, y_{n+1}]^\top \in \mathbb{R}^{n+1}$. Write down explicit formulas for the entries of V:

Solution: *The matrix has entries*

$$V = \begin{bmatrix} 1 & x_1 & e^{x_1} \\ 1 & x_2 & e^{x_2} \\ 1 & x_3 & e^{x_3} \\ \vdots & \vdots & \vdots \\ 1 & x_{n+1} & e^{x_{n+1}} \end{bmatrix}$$

or

$$\begin{aligned} V_{i1} &= 1, i = 1, 2, \ldots, n+1 \\ V_{i2} &= x_i, i = 1, 2, \ldots, n+1 \\ V_{i3} &= e^{x_i}, i = 1, 2, \ldots, n+1. \end{aligned}$$

Point distribution:

- $+0.3$ if matrix or entries of the matrix are correctly written down;
- -0.2 if only case $n = 2$ is given;
- -0.1 points if V is starting with x_0 or ending with x_n;
- 0 points if only standard Vandermonde matrix (monomial basis) is given.

Hint: Note that this is *not* a least-squares polynomial approximation problem. Still, the correct matrix V can be found in a fashion anologous to the derivation of the Vandermonde matrix for least-squares polynomial approximation problems.

Question 2 (0.2 Points) Using your results from Question 1, complete the following Python function such that it computes the least-squares solution $\mathbf{a} = [a_0, a_1, a_2]^\top \in \mathbb{R}^3$ of (7) for given data vectors $\mathbf{x} = [x_1, x_2, \ldots, x_{n+1}]^\top \in \mathbb{R}^{n+1}$ and $\mathbf{y} = [y_1, y_2, \ldots, y_{n+1}]^\top \in \mathbb{R}^{n+1}$.

```
def least_squares(x, y):
    """
    Computes the vector of coefficients for the least-squares approximant
    q(x) = a_0 + a_1 x + a_2 e^x given data x_i, y_i, i=1, 2, ..., n + 1.

    Parameters
    ----------
    x : NumPy array of shape (n+1, )
        x-values of the data.
    y : NumPy array of shape (n+1, )
        y-values of the data.

    Returns
    -------
    a : NumPy array of shape (3, )
        Array of coefficients a_0, a_1, a_2.
    """

    ### BEGIN SOLUTION
    n = len(x) - 1
    V = np.ones((n + 1, 3))
    V[:, 1] = x
    V[:, 2] = np.exp(x)
    a = np.linalg.solve(V.T @ V, V.T @ y)
    ### END SOLUTION

    return a
```

Point distribution:

- $+0.1$ if matrix V is correctly assembled;
- $+0.1$ if normal equations are correctly computed and solved (`np.linalg.lstsq` is also tolerated);
- -0.05 if $V^\top$ instead of V is assembled and then used as V;
- -0.05 if the shape of V is $(n+1) \times (n+1)$ instead of $(n+1) \times 3$;
- -0.05 if matrix V is only assembled for $n = 2$;
- small indexing errors are tolerated;
- 0 points if `np.polyfit` is used, since it computes the least squares approximant in a different basis.

Question 3 (0.2 Points) Consider the data points $x_i = \frac{i}{4}$ and $y_i = \sin(x_i)\cos(x_i) + x_i^2 - 1$ for $i = 1, 2, \ldots, 10$. Provide the Python code that utilizes the function from Question 2 to solve the least-squares problem (7) for this data. Report the least-squares error (7).

Python code:

```
### BEGIN SOLUTION
x_data = np.arange(1, 11) / 4
y_data = np.sin(x_data) * np.cos(x_data) + x_data ** 2 - 1

a = least_squares(x_data, y_data)

y_ls = a[0] + a[1] * x_data + a[2] * np.exp(x_data)

ls_error = np.sum((y_data - y_ls) ** 2)
print("Least squares error: E_ls = {:.2e}".format(ls_error))
### END SOLUTION
```

Value of the least-squares error (7) rounded to three decimal digits.

Solution: $\underline{\min_{\mathbf{a}\in\mathbb{R}^3}\|\mathbf{y} - V\mathbf{a}\|^2 = 2.90 \times 10^{-2} \text{ or } 0.0290}$

Point distribution:

- +0.1 if least-squares coefficients are correctly computed, even if an incorrectly implemented `least_squares` from previous question is referenced;
- +0.1 if the least-squares error (7) is correctly computed and written down;
- −0.05 if the value of the error is not given.