Final Exam (A)

EPFL	**Course:**	Numerical Analysis - Sections SIE-GC
	Lecturer:	Pablo Antolín, Espen Sande
	Date:	15/01/2024 **Duration:** 3h 00
Sciper:	**Student:**	**Section:**

Evaluation table				
Ex.1	**Ex.2**	**Ex.3**	**Ex.4**	**Total**

PLEASE READ CAREFULLY ALL THE INSTRUCTIONS OF THE EXAM

WRITE YOUR SURNAME, NAME, SCIPIER on every page, PLEASE!

All Python code and all the results (the plots and images if they are requested, too) 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 considered for the evaluation of the exam. It must be returned at the end of the exam.

Exercise 3 is of multiple-choice type and this is the only case throughout this exam where it is not necessary to justify your answer.

Write everything with a blue or black pen.

AUTHORIZED MATERIAL

The only authorized material is: an handwritten formulary on a double-sized A4 sheet.

No other sheets, notes or books (paper or electronics), or calculator, mobile phone, tablet, laptop or other electronic devices. The access to Internet (e-mail, websites) is obviously prohibited.

Informations for identification (log-in):

Username and password: `Your GASPAR account`

Virtual machine:

`SB-MATH-EXAM`

Exercise 1 (1.4 points)

Let us define the following tridiagonal matrix:

$$A = \begin{bmatrix} 4 & \theta & & & \\ 1 & 4 & \theta & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & \theta \\ & & & 1 & 4 \end{bmatrix} \in \mathbb{R}^{n\times n}$$

where $\theta \geq 0$, which can be constructed in Python (after having defined `theta` and `n`) with the following commands:

```
import numpy as np
A = 4*np.eye(n) + np.diag(np.ones(n-1),-1) + theta*np.diag(np.ones(n-1),1)
```

We want to solve the linear system $A\mathbf{x} = \mathbf{b}$ with $\mathbf{b} = (1, ..., 1)^\top \in \mathbb{R}^n$.

a) Compute (**by hand**) for which values of θ the Jacobi iterative method converges to the true solution. *Hint: the eigenvalues of a tridiagonal matrix*

$$M = \begin{bmatrix} a & b & & & \\ c & a & b & & \\ & \ddots & \ddots & \ddots & \\ & & c & a & b \\ & & & c & a \end{bmatrix} \in \mathbb{R}^{n\times n}$$

are $\lambda_i(M) = a + 2\sqrt{bc}\cos\left(\frac{\pi i}{n+1}\right)$, $i = 1, ..., n$.

Then, compute the spectral radius of the iteration matrix of the Jacobi method for the different values $n = 5, 100$ and $\theta = 2, 5$. Fill out the following table with the obtained results. Do the results agree with the convergence condition for θ obtained previously?

	$n = 5$	$n = 100$
$\theta = 2$		
$\theta = 5$		

b) Let us set $n = 50$ and $\theta = 2$. Solve the linear system using the Jacobi method (function `jacobi` provided on the next page) by setting the initial guess to the vector $\mathbf{x}_0 = \mathbf{0}$, the tolerance to 10^{-6} and the maximum number of iteration to 500. Write down the number of iterations needed to reach convergence and the residual at the final iteration.

c) Now, set $\theta = 1$, $\mathbf{b} = (5, 6, 6, ..., 6, 6, 5)^\top$ and the initial guess vector $\mathbf{x}_0 = \mathbf{0}$, and consider the different values $n = 4, 8, 12, ..., 200$. Knowing that the exact solution of the linear system is $\mathbf{x} = (1, ..., 1)^\top$ and denoting by $\mathbf{x}_J$ the approximated solution obtained with the Jacobi method, plot the relative error $\|\mathbf{x} - \mathbf{x}_J\|/\|\mathbf{x}\|$ and the normalized residual $\|\mathbf{b} - A\mathbf{x}_J\|/\|\mathbf{b}\|$ as a function of n in a graph in double logarithmic scale (report the graph qualitatively on the exam sheet). Is the residual a good estimator of the error? Why?

```python
import numpy as np

def jacobi(A,b,x0,nmax,tol):
    """
    Jacobi iterative method.

    x, niter, res = jacobi(a, b, x0, nmax, tol) attempts to solve the system of linear
        equations A*x=b
    for x using the Jacobi method.

    Input:
        A: n-by-n coefficient matrix A. It must be not singular
        b: right-hand-side column vector. It must have length n.
        x0: initial solution guess. It must have length n.
        nmax: Maximum number of iterations to perform.
        tol: Tolerance of the method. Used for stopping the iterations by comparing
            the norm of the residual with the residual for x0.

    Output:
        x: compute solution vector. It has length n.
        niter: Number of iterations performed until convergence was reached.
        res: Norm of the residual at each iteration, including the first one.
    """

    P = np.diag(np.diag(A))

    x = x0.copy()
    r = b - A @ x
    r0 = np.linalg.norm(r)
    res = [r0]
    niter = 0

    while res[-1]/r0 > tol and niter < nmax:
        z = np.linalg.solve(P, r)
        x = x + z
        r = b - A @ x
        niter = niter + 1
        res.append(np.linalg.norm(r))

    return x, niter, res
```

Exercise 2 (1.4 points)

a) For a continuous function $f : [a, b] \to \mathbb{R}$, write the composite Simpson's quadrature formula $Q_h^{simp}(f)$ over M sub-intervals of length $h = (b - a)/M$ to approximate the integral $I(f) = \int_a^b f(x)\,dx$, and recall its degree of exactness.

b) We recall that the order of a composite quadrature formula $Q_h(f)$ is the largest integer p such that

$$\left| \int_a^b f(x)\,dx - Q_h(f) \right| \leq Ch^p, \tag{1}$$

where $h = (b - a)/M$, for a constant $C > 0$ that depends on f and is otherwise independent of h. What is the order of the composite Simpson's formula and which derivatives of f appear in the constant C of the estimate (1)?

c) We now consider the function $f(x) = e^x \sin(x)$ and the bounds $a = 0$ and $b = 1$. Knowing that the exact value of the integral over the interval $[0, 1]$ is

$$I(f) = \frac{1}{2}\left[1 + \exp(1)\Big(\sin(1) - \cos(1)\Big)\right],$$

approximate the integral $I(f)$ by the composite Simpson's formula (function `simp` provided on the next page) using $M = 2, 2^2, 2^3, 2^4, 2^5$ sub-intervals of the same length $h = (b - a)/M$, and calculate for each value of h the error

$$\epsilon_h = \left|I(f) - Q_h^{simp}(f)\right|.$$

Then plot the graph of the error ϵ_h as a function of h on a logarithmic scale (command `loglog`). What is the order of convergence that we observe?

d) Repeat point c) using $f(x) = \sqrt{x}$, knowing that $I(f) = \frac{2}{3}$. Comment on the results obtained.

e) Repeat point c) using $f(x) = \dfrac{x^3 - 7x^2 + 14x - 8}{x - 4}$, knowing that $I(f) = \frac{5}{6}$. Comment on the results obtained.

```python
import numpy as np

def simp(a,b,m,f):
    """
    Composite Simpson formula.

    intsimp = simp(a, b, m, f) computes the integral of the function f in the interal [a
        , b]
    using m sub-intervals.

    Input:
        a: Start of the interval.
        b: End of the interval.
        m: Number of sub-intervals.
        f: function to integrate. This function must be evaluable as f(x),
            with x being a numpy array.

    Output:
        intsimp: Computed integral.
    """

    m = int(m)

    h = (b - a) / m

    x = np.linspace(a, b, m + 1)

    xmid = 0.5 * (x[:-1] + x[1:])

    return h/6.0 * np.sum(f(x[:-1])+ 4.0*f(xmid) + f(x[1:]))
```

Exercise 3 - Multiple choice questions (1.2 points)

This is a multiple-choice exercise.
One, and only one, answer is correct for every question.
Mark your answer with a cross or a circle in a legible way.
Each correct answer is worth $\frac{1}{5}$, each "blank" question is worth 0 and each wrong answer is worth $-\frac{1}{20}$.

MC1 We want to find an approximation of the solution of the ordinary differential equation

$$\begin{cases} u'(t) &= -2u(t) + (t+1) \\ u(0) &= u_0, \end{cases}$$

by using the following scheme:

$$\begin{cases} u^{n+1} &= \frac{1}{2\Delta t+1} u^n + \Delta t \frac{(n+1)\Delta t+1}{2\Delta t+1} \\ u^0 &= u_0. \end{cases}$$

Which scheme is it?

a) Forward Euler (explicit).

b) Backward Euler (implicit).

c) Crank-Nicolson.

d) Heun.

e) None of the above.

MC2 Let us consider the data

$$\begin{array}{ll} x_1 = -1 & y_1 = 12 \\ x_2 = 0 & y_2 = 12 \\ x_3 = 2 & y_3 = 6. \end{array}$$

The derivative at $x = 1$ of a polynomial that interpolates the data (x_i, y_i), $i = 1, 2, 3$, is equal to

a) -3

b) 0

c) 1

d) 5

e) 11

MC3 The Newton method for solving the system of equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ with

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \sin(x_1) + x_1 \\ 2x_1 - x_2 + 5 \end{bmatrix}.$$

is:

a) $\begin{cases} x_1^{(k+1)} = x_1^{(k)} - \frac{\sin(x_1^{(k)})+x_1^{(k)}}{\cos(x_1^{(k)})+1} \\ x_2^{(k+1)} = 2x_1^{(k+1)} + 5 \end{cases}$

b) $\begin{cases} x_1^{(k+1)} = x_1^{(k)} - \frac{\cos(x_1^{(k)})+1}{\sin(x_1^{(k)})+x_1^{(k)}} \\ x_2^{(k+1)} = 2x_1^{(k)} + 5 \end{cases}$

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

d) $\begin{cases} x_1^{(k+1)} = x_1^{(k)} - \frac{\cos(x_1^{(k)})+1}{\sin(x_1^{(k)})+x_1^{(k)}} \\ x_2^{(k+1)} = 2x_1^{(k+1)} + 5 \end{cases}$

e) $\begin{cases} x_1^{(k+1)} = x_1^{(k)} - \frac{\sin(x_1^{(k)})+x_1^{(k)}}{\cos(x_1^{(k)})+1} \\ x_2^{(k+1)} = 2x_1^{(k)} + 5 \end{cases}$

MC4 Consider the first-order linear differential equation system

$$\mathbf{X}'(t) = A\mathbf{X}(t) + \mathbf{b}(t), \quad \text{with } A = \begin{pmatrix} -1 & 3 & 2 \\ 0 & -1 & 0 \\ -1 & 1 & -3 \end{pmatrix}. \tag{2}$$

The forward Euler method to approximate the problem (2) is absolutely stable if and only if

a) $0 < \Delta t \leq 2$

b) $0 < \Delta t < \frac{4}{5}$

c) $0 < \Delta t < \frac{1}{2}$

d) $0 < \Delta t < 1$

e) $\Delta t > 0$ (unconditionally absolutely stable)

MC5 Let $f : \mathbb{R}_+^* \to \mathbb{R}$ be defined by $f(x) = (x-1)^3 + 2\log(x) + 1$. We are interested in the unique fixed point $\alpha = 1$ of f. Under what condition on ω is the composite fixed point method defined for $\omega \in \mathbb{R}$ by

$$x^{(k+1)} = \phi(x^{(k)}) = (1-\omega)x^{(k)} + \omega f(x^{(k)})$$

of order 2?

a) $\omega = -1$

b) $\omega = 0$

c) $\omega = 1$

d) $\omega = 1/2$

e) none of the values of ω proposed above

MC6 Given $f \in C^4([0,1])$, let $s_{3,h}$ be a cubic spline that interpolates f at the equally distributed nodes $0 = x_1 < x_2 < \ldots < x_{n+1} = 1$, where $h = 1/n$ is the length of each interval $I_i = [x_i, x_{i+1}]$, $i = 0, \ldots, n$. What is the order of convergence for the error $\max_{x\in[0,1]} |f''(x) - s''_{3,h}(x)|$ as h goes to zero?

a) 4

b) 3

c) 2

d) 1

e) 0 (no convergence order).

Exercise 4 - Implementation (0.5 points)

CODE1 Complete the following Python function which implements the Heun method

```
import numpy as np
def heun(f, I, u0, N):
    """
    Solve the differential equation:
        u'=f(t,u), t in (t0,T],
        u(t0)=u0
    using the Heun's method on an equispaced grid of stepsize dt=(T-t0)/N.

    Input:
        f: right-hand-side function. It can be evaluated as f(x),
            where x is a scalar value.
        I: integration interval [t0,T].
        u0: initial condition.
        N: number of sub-intervals.

    Output:
        t: vector of time snapshots [t0,t1,...,tN] (with tN=T).
        u: approximation of the solution [u0,u1,...,uN].
        dt: time step.
    """

    dt = (I[1] - I[0]) / N
    u = np.zeros(N+1)


    t =                     # TO COMPLETE


    u[0] =                  # TO COMPLETE


    for n in range(N):

        # TO COMPLETE below




        u[n+1] =                                        # TO COMPLETE




    return t, u, dt
```

CODE2 Complete the following Python function which implements the bisection method.

```
import numpy as np

def bisection(f, a, b, tol, nmax):
    """
    Tries to find a zero of the continuous function f in the interval [a, b]
    using the bisection method. This function assumes that a zero is found when the
    length of the bisection interval semilength at a certain iteration is smaller
    than the given tolerance.

    Input:
        f: function whose zero is sought. It must be evaluable as f(x),
            where x is a scalar value.
        a, b: Start and end of the interval. They must be such that the values
            of the function f at them have opposite sign.
        tol: Tolerance to be used.
        nmax: Maximum number of iterations to perform.

    Output:
        zero: Computed zero.
        fzero: Value of f evaluated at the computed zero.
        niter: Number of iterations performed.
    """

    fa = f(a)
    fb = f(b)
    niter = 0

    if fa * fb > 0:
        raise Exception("The sign of f at a and b must be different")

    elif np.abs(fa) < tol:
        zero  =                    # TO COMPLETE
        res   =                    # TO COMPLETE
        return zero, res, niter
    elif np.abs(fb) < tol:
        zero  =                    # TO COMPLETE
        res   =                    # TO COMPLETE
        return zero, res, niter

    I = 0.5 * (b - a)  # This is the interval semilength, i.e., the error estimate.

    while I > tol and niter < nmax:
        # TO COMPLETE below













    return zero, f(zero), niter
```