School of Computing: assessment brief

Module title	Numerical computation
Module code	COMP/XJCO2421
Assignment title	Final coursework
Assignment type and description	Coursework. Exploring methods for solving differential equations.
Rationale	Testing the understanding of learning outcomes in practical situation
Word limit and guidance	Suggested word limit given in each section totalling 650 words
Weighting	80%
Submission deadline	2pm Wed 20 Dec
Submission method	Gradescope
Feedback provision	Written group feedback
Learning outcomes assessed	- Use, data-based arguments to justfy choosing a computational algorithm appropriately, accounting for issues of accuracy, reliability and efficiency; - Understand how to assess/measure the error in a numerical algorithm and be familiar with how such errors are controlled; - Implement simple numerical algorithms accurately and present results in a variety of forms.
Module lead	Thomas Ranner
Other Staff contact	(COMP) Yongxing Wang, (XJCO) Zhiguo Long

1. **Assignment guidance**

   In this coursework, you will use software libraries to explore different numerical schemes and you will analyse the methods and results.

2. **Assessment tasks**

   In this coursework, you will be writing a report for your boss as BigNumComp Inc helping them to choose a numerical method. You should assume she has the knowledge of second year undergraduate Computer Science student at the University of Leeds.

   Problem

   Your boss wants to find good numerical methods for solving predator-prey models. Predator-prey models describe the evolution of two different co-existing species: one is the predator and one is the prey.

   The problem can be described through the following system of differential equations for $x(t)$, the prey, and $y(t)$ the predator:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy + f(t)$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = \delta xy - \gamma y + g(t),$$

   where $\alpha, \beta, \gamma, \delta$ are positive real numbers and f and g are given functions of t which relate to external migration into or out of the system.

   Your boss, helpfully, suggests two test cases for you to consider:

   (a) Use the parameter set $\alpha = \beta = \gamma = \delta = 1$,

$$f(t) = -\sin(t) - (\cos(t))^2 - \cos(t) \quad g(t) = \sin(t) + (\cos(t))^2 - \cos(t),$$

   and initial condition $x(t = 0) = 2$ and $y(t = 0) = 0$, which has exact solution

$$x(t) = 1 + \cos(t) \quad y(t) = 1 - \cos(t).$$

   (b) Use the parameter set $\alpha = 2/3, \beta = 4/3, \gamma = 1, \delta = 1$, $f(t) = 0$, $g(t) = 0$ and $x(t = 0) = 0.9, y(t = 0) = 0.9$. For this case there is no nice form for the exact solution but your boss tells you that the solution should repeat itself and the maxima of each population should be consistent (i.e. not change) over time.

   You are tasking with finding the best solver in terms of accuracy and efficiency. You should write a report (following the template laid out below) evidencing your recommendations and reasoning.

   Task

   You recognise this problem as being solvable as a system of differential equations. You have seen some methods in the course but know there are better methods available. In

your preliminary research you find a company internal solver for systems of differential equations (attached as `solvers.py`). *You will solve and analyse the results of solving the predator prey model for two different methods within* ***`solvers.py`*** *and one method from the module which you think is the most appropriate for this problem.*

You should include all the sections in the template in your report. There is guidance of what to include in each section and a guidance word limit for each section too. Code, tables and equations do not count in the word limit. The word limit is only guidance and no penalties will be introduced for going over the limit. You should aim to write less than the word count to ensure your writing is concise and understandable to your audience.

You should submit a jupyter notebook including all computations as your report. You should write in full sentences throughout to guide the reader through what you are doing. There is no need to include the file `solvers.py` in your solution. You should write text in `Markdown` blocks and include all code for the implementation and generating results in `Code` blocks.

Library documentation

`solvers` Solve differential equations.

Solve the differential equation(s) specified by

$$y'(t) = f(t, y) \quad \text{subject to} \quad y(t_0) = y_0.$$

The problem is solved using a specified method from t_0 to a final time T using a time step dt.

Parameters

`rhs` A python function describing the right hand side function f of the differential equation. The function takes two arguments: the first represents `t` for time and the second represents the solution `y` which may be either a floating point type or a numpy array for the case of multiple differential equations.

`y0` The starting value of y - accepts either a floating point type or a numpy array for the case of multiple differential equations.

`t0` The starting time t_0.

`dt` The time step dt.

`T` The final or stopping time.

`method` The method used to advance the solver given as a string. The method should be one of

- `"Heun"`
- `"Ralston"`
- `"Van De Houwen"`

- `"SSPRK3"`
- `"Runge-Kutta"`
- `"3/8-rule"`
- `"Ralston-4"`.

Returns

`t` The time points where the solution was found as a list

`y` The estimate of the solution at each time point as a list

Sample code

Download the file `solver.py` and place it in the same folder as your code.

An example for solving the a single differential equation:

```
from solvers import solver

def rhs(t, y):
    return -y

y0 = 1.0
t0 = 0.0
dt = 0.1
T = 1.0

t, y = solver(rhs, y0, t0, dt, T)
```

An example for solving the a system of differential equation:

```
import numpy as np

from solvers import solver

def rhs(t, y):
    return np.array([-y[1], y[0]])

y0 = np.array([1.0, 0.0])
t0 = 0.0
dt = 0.1
T = 1.0

t, y = solver(rhs, y0, t0, dt, T)
```

Solution template

(a) Implementation. Write code to be able to run three methods you have chosen for arbitrary initial conditions $(x(t=0), y(t=0))$, time step $(\mathrm{d}t)$, model parameters $(\alpha, \beta, \gamma, \delta)$ and functions f and g. [50 words]

(b) Results. Simulate and show results for each of the test cases suggested by your boss for a range of time steps until a final time T. For test case 2a, you should use $T = 2.5\pi$ and (at least) $\mathrm{d}t = T/100, T/200, T/400, T/800, T/1600$. For test case 2b, you should use $T = 30$ and (at least) $\mathrm{d}t = T/100, T/200, T/400, T/800, T/1600$. You should demonstrate how solutions look for each method, and the accuracy and efficiency of each approach. [50 words]

(c) Analysis. Comment on the efficiency and accuracy of each approach. [250 words]

(d) Conclusion. Compare the methods that you have results for, and any other relevant methods from the module, and make a recommendation of which method you think is best. [300 words]

3. **General guidance and study support**

Examples of how to approach each aspect of this coursework is given in lectures and using the online notes.

You may wish to read more about predator prey models (also known as Lotka Volterra equations) to help with your analysis:

- Wikipedia: Lotka Volterra equations
- Mathworld: Lotka Volterra equations

Further support for this assessment is given through the MS Class Team. Details of further support sessions will be given closer to the deadline.

4. **Assessment criteria and marking process**

Your work will be assessed on your code implementation, your results and their presentation, your analysis of the method and results, and your writing quality. Work will be marked as a final assessment for this module so your mark will only be given back as part of your final grade.

5. **Presentation and referencing**

The quality of written English will be assessed in this work - further details in the Rubric below. As a minimum, you must ensure:

- Paragraphs are used
- There are links between and within paragraphs although these may be ineffective at times
- There are (at least) attempts at referencing

- Word choice and grammar do not seriously undermine the meaning and comprehensibility of the argument
- Word choice and grammar are generally appropriate to an academic text

These are pass/ fail criteria. So irrespective of marks awarded elsewhere, if you do not meet these criteria you will fail overall.

6. **Submission requirements**

   Please submit your work via Gradescope by the deadline given. You should submit a jupyter notebook with all your code, text and results included in a single document. You are recommended to "reset the kernel" and "Run all cells" again before you submit.

7. **Academic misconduct and plagiarism**

   - Leeds students are part of an academic community that shares ideas and develops new ones.
   - You need to learn how to work with others, how to interpret and present other people's ideas, and how to produce your own independent academic work. It is essential that you can distinguish between other people's work and your own, and correctly acknowledge other people's work.
   - All students new to the University are expected to complete an online Academic Integrity tutorial and test, and all Leeds students should ensure that they are aware of the principles of Academic integrity.
   - When you submit work for assessment it is expected that it will meet the University's academic integrity standards.
   - If you do not understand what these standards are, or how they apply to your work, then please ask the module teaching staff for further guidance.

   By submitting this assignment you are confirming that the work is a true expression of your own work and ideas and that you have given credit to others where their work has contributed to yours.

8. **Assessment/marking criteria grid**

   The final assessment will be marked out of 50 according the following rubric.

   Algorithm implementation (10 marks)

Marks	Description
9-10	Algorithm(s) implemented accurately and efficiently. Professional quality code (Uniform formatting, unit tests where appropriate). No efficiency problems. Informative comments.
7-8	Algorithm(s) implemented with no errors. Some efficiency problems. Helpful comments throughout.
6-7	Algorithm(s) implemented with no errors. Some helpful comments.
5-6	Algorithm(s) implemented with minor errors. Some comments.
0-4	Serious issues with code implementation resulting in inaccurate results.

Presentation of results (15 marks)

Marks	Description
13-15	Results in a variety of appropriate formats (i.e. tables, plots, etc). Results and extensive additional useful information shown. Plots and tables labelled accurately.
10-12	Results in a variety of appropriate formats (i.e. tables, plots, etc). Results and additional useful information shown. Plots and tables labelled accurately.
7-9	Results in a variety of appropriate formats (i.e. tables, plots, etc). Results accurately shown.
4-6	Attempts are carefully formatting results suitable for technical audience. Results accurately shown.
0-3	Basic or very limited results shared.

Analysis of results (20 marks)

Marks	Description
17-20	Critical explanations using additional computational experiments and making reference to appropriate external literature covering all methods
13-16	Critical explanations using additional computational experiments covering all methods
9-12	Descriptive explanations using further analysis of suggested computational experiments covering all methods
5-8	Descriptive explanations based purely on suggested experiments covering all methods
0-4	No or very limited results explained

Writing (5 marks)

Marks	Description
5	Outstanding structure and clarity of writing, all in a suitable language. No errors.
4	Clear structure and writing in suitable language. Some minor errors.
3	Well structured with mostly clear writing in suitable language. Some errors.
2	Structure could have been improved. Some text required careful reading. Language not appropriate for technical report.

Marks	Description
1	Poor presentation and structure with unclear or confusion descriptions. Many errors.

A External library reference

The source code for `solvers.py`:

```
from typing import Callable, List, Tuple, TypeVar

import numpy as np

# Butcher tables for each of the methods used
TABLEAU = {
    "Heun": (
        np.array([[0.0, 0.0], [1.0, 0.0]]),
        np.array([0.5, 0.5]),
        np.array([0.0, 1.0]),
    ),
    "Ralston": (
        np.array([[0.0, 0.0], [2 / 3, 0.0]]),
        np.array([0.25, 0.75]),
        np.array([0.0, 2 / 3]),
    ),
    "Van der Houwen": (
        np.array([[0.0, 0.0, 0.0], [1 / 2, 0.0, 0.0], [0.0,
           0.75, 0.0]]),
        np.array([2 / 9, 1 / 3, 4 / 9]),
        np.array([0.0, 1 / 2, 3 / 4]),
    ),
    "SSPRK3": (
        np.array([[0.0, 0.0, 0.0], [1.0, 0.0, 0.0], [0.25,
           0.25, 0.0]]),
        np.array([1 / 6, 1 / 6, 2 / 3]),
        np.array([0.0, 1.0, 1 / 2]),
    ),
    "Runge-Kutta": (
        np.array(
            [
                [0.0, 0.0, 0.0, 0.0],
```

```
                [0.5, 0.0, 0.0, 0.0],
                [0.0, 0.5, 0.0, 0.0],
                [0.0, 0.0, 1.0, 0.0],
            ]
        ),
        np.array([1 / 6, 1 / 3, 1 / 3, 1 / 6]),
        np.array([0.0, 0.5, 0.5, 1.0]),
    ),
    "3/8-rule": (
        np.array(
            [
                [0.0, 0.0, 0.0, 0.0],
                [1 / 3, 0.0, 0.0, 0.0],
                [-1 / 3, 1.0, 0.0, 0.0],
                [1.0, -1.0, 1.0, 0.0],
            ]
        ),
        np.array([1 / 8, 3 / 8, 3 / 8, 1 / 8]),
        np.array([0.0, 1 / 3, 2 / 3, 1]),
    ),
    "Ralston-4": (
        np.array(
            [
                [0.0, 0.0, 0.0, 0.0],
                [0.4, 0.0, 0.0, 0.0],
                [0.29697761, 0.15875964, 0.0, 0.0],
                [0.21810040, -3.05096516, 3.83286476, 0.0],
            ]
        ),
        np.array([0.17476028, -0.55148066, 1.20553560,
           0.17118478]),
        np.array([0.0, 0.4, 0.45573725, 1.0]),
    ),
}

# types for y variable in solver
y_type = TypeVar("y_type", np.ndarray, np.double)


def solver(
    rhs: Callable[[np.double, y_type], y_type],
    y0: y_type,
```

```
    t0: np.double,
    dt: np.double,
    T: np.double,
    method: str,
) -> Tuple[List[np.double], List[y_type]]:
    """
    Solve the differential equation(s).


    Solve the differential equation specified by

    y'(t) = rhs(t, y) subject to y(t_0) = y_0.

    The problem is solved numerical using METHOD from t0 to T
        using a time step dt.

    Parameters
    ----------

    rhs
            A function describing the right hand side of the
               differential equation(s)
    y0
            The starting value of y
    t0
            The starting value of t
    dt
            The time step
    T
            The final or stopping time
    method
            The method used to advance to solver. method
               should be one of:
            Heun, Ralston, Van De Houwen, SSPRK3, Runge-Kutta
               , 3/8-rule, Ralston-4

    Returns
    -------

    t
            The time points where the solution was found
    y
```

```
            The estimate of the solution at each time point
    """

    # set initial data into solution arrays
    t_out = [t0]
    y_out = [y0]

    # extract method helpers
    matrix, weights, nodes = TABLEAU[method]
    s = len(weights)
    k: List[y_type | None] = [None for _ in range(s)]

    # count steps
    timesteps = int(T / dt)

    # time loop
    for step in range(timesteps):
        # build k's
        for i in range(s):
            temp = sum(matrix[i, j] * k[j] for j in range(i))
            k[i] = rhs(t_out[-1] + dt * nodes[i], y_out[-1] +
                dt * temp)

        y_update = sum([k[i] * weights[i] for i in range(s)])

        y_new = y_out[-1] + dt * y_update
        t_new = t_out[-1] + dt

        t_out.append(t_new)
        y_out.append(y_new)

    return t_out, y_out


def example_code_1():
    """
    Example code for single differential equation

    The problem is y'(t) = y subject to y(0) = 1.0.

    The problem is solved with dt = 0.1 until T = 1.0 using
       Heun's method
```

```
    """

    def rhs1(t: np.double, y: np.double) -> np.double:
        return -y

    t, y = solver(rhs1, 1.0, 0.0, 0.1, 1.0, "Heun")


def example_code_2():
    """
    example code for system of differential equations

    The problem is (x'(t), y'(t)) = (-y(t), x(t)) subject to
       (x(0), y(0)) = (1.0, 0.0)

    The problem is solved with dt = 0.1 until T = 1.0 using
       the Runge-Kutta method
    """

    def rhs2(t: np.double, y: np.ndarray) -> np.ndarray:
        return np.array([-y[1], y[0]])

    t, y = solver(rhs2, np.array([1.0, 0.0]), 0.0, 0.1, 1.0,
       "Runge-Kutta")


if __name__ == "__main__":
    for method, (matrix, weights, nodes) in TABLEAU.items():
        # test methods are explicit
        np.testing.assert_almost_equal(np.tril(matrix),
           matrix)
        # test methods are consistent
        np.testing.assert_almost_equal(sum(weights), 1.0)
        # test dimensions match
        n, m = matrix.shape
        assert n == m
        assert n == len(weights)
        assert n == len(nodes)

    example_code_1()
    example_code_2()
```