University of Nottingham
School of Mathematical Sciences

Submission Date: Tuesday 11th January 2022, 5:00pm (GMT) **Assessed Coursework 2**

The following questions are to be used for the coursework assessment in the module MATH4063.

A single zip or tar file containing your answers to the questions below and the code you used to obtain these answers should be submitted electronically via the MATH4063 Moodle page before the deadline at the top of this page. You should follow the instructions on the accompanying Coursework Submission template which is also provided on Moodle. Since this work is assessed, your submission must be entirely your own work (see the University's policy on Academic Misconduct).

The style and efficiency of your programs is important. A barely-passing solution will include attempts to write programs which include some of the correct elements of a solution. A borderline distinction solution will include working code that neatly and efficiently implements the relevant algorithms, and that shows evidence of testing.

An indication is given of the weighting of each question by means of a figure enclosed by square brackets, e.g. [12]. All non-integer calculations should be done in double precision.

Background Material

Systems of Ordinary Differential Equations

Cauchy problems, also known as Initial Value Problems (IVPs), consist of finding solutions to a system of Ordinary Differential Equations (ODEs), given suitable initial conditions. We will be concerned with the numerical approximation of the solution to the problem

$$\dot{\boldsymbol{u}}(t) = \boldsymbol{f}(t, \boldsymbol{u}(t)) \qquad \text{for } t \in [t^0, T], \tag{1}$$

$$\boldsymbol{u}(t^0) = \boldsymbol{u}^0, \tag{2}$$

where the dot denotes differentiation with respect to t, $\boldsymbol{f}$ is a sufficiently well-behaved function that maps $[t^0, T) \times \mathbb{R}^d$ to $\mathbb{R}^d$, the initial condition $\boldsymbol{u}^0 \in \mathbb{R}^d$ is a given vector, and the integer d is the dimension of the problem. We assume that $\boldsymbol{f}$ satisfies the Lipschitz condition

$$\|\boldsymbol{f}(t, \boldsymbol{w}) - \boldsymbol{f}(t, \boldsymbol{u})\| \leq \lambda \|\boldsymbol{w} - \boldsymbol{u}\| \qquad \text{for all } \boldsymbol{w}, \boldsymbol{u} \in \mathbb{R}^d,$$

where $\lambda > 0$ is a real constant independent of $\boldsymbol{w}$ and $\boldsymbol{u}$. This condition guarantees that the problem (1)–(2) possesses a unique solution. We seek an approximation to the solution $\boldsymbol{u}(t)$ of (1)–(2) at $N_t + 1$ evenly-spaced time points in the interval $[t^0, T]$, so we set

$$t^n = t^0 + nh \qquad \text{for } 0 < n \leq N_t \quad \text{where} \quad h = (T - t^0)/N_t\,.$$

The scalar h is referred to as the *time-step*. We use a subscript h to denote an approximation to $\boldsymbol{u}(t)$, in this case at the time points $\{t^n\}$,

$$\boldsymbol{u}_h(t^n) \approx \boldsymbol{u}(t^n), \qquad \text{for } 0 \leq n \leq N_t\,,$$

and we are interested in the behaviour of the error $\boldsymbol{e}_h^n = \boldsymbol{u}_h(t^n) - \boldsymbol{u}(t^n)$. We expect this error to decrease as the step size h tends to 0: the sequence of approximations $\{\boldsymbol{u}_h(t^n)\}$ will be generated

by a numerical method, which will be said to be convergent if

$$\lim_{h\to 0^+} \max_{n=0}^{N_t} \|\boldsymbol{e}_h^n\| \;=\; 0\,,$$

where $\|\cdot\|$ is a generic norm on $\mathbb{R}^d$. If the numerical method is convergent and

$$E(h) \;=\; \max_{n=0}^{N_t} \|\boldsymbol{e}_h^n\| \;=\; O(h^p)\,,$$

then it is said to be *an order p method*. The method may suffer from numerical instabilities if the value of h is too large, so the step size h must be set to a sufficiently small value during computations to get sensible approximations.

Schemes for Second-Order ODEs

Physical or engineering problems involving particle motion are often naturally written as a system of second-order ODEs of the form

$$\ddot{\boldsymbol{x}}(t) \;=\; \boldsymbol{f}(t,\boldsymbol{x}(t)) \qquad \text{for } \; t\in[t^0,T]\,, \tag{3}$$

$$\boldsymbol{x}(t^0) \;=\; \boldsymbol{x}^0\,, \qquad \boldsymbol{v}(t^0) \;=\; \dot{\boldsymbol{x}}(t^0) \;=\; \boldsymbol{v}^0\,, \tag{4}$$

in which $\boldsymbol{x}(t)$ represents a particle position and $\boldsymbol{v}(t)$ its velocity. It is possible to recast the problem as a system of first-order ODEs,

$$\dot{\boldsymbol{x}}(t) \;=\; \boldsymbol{v}(t)\,, \qquad \boldsymbol{x}(t^0) \;=\; \boldsymbol{x}^0\,, \tag{5}$$

$$\dot{\boldsymbol{v}}(t) \;=\; \boldsymbol{f}(t,\boldsymbol{x}(t))\,, \qquad \boldsymbol{v}(t^0) \;=\; \boldsymbol{v}^0\,, \tag{6}$$

on the interval $t\in[t^0,T]$, and then employ numerical methods for approximating first-order IVPs. Second-order systems of the type (3)–(4) may exhibit special symmetries that, ideally, would be preserved by the numerical scheme (*e.g.* reversibility, symplectic structure) so care is needed when devising an appropriate approximation. We will consider the following three methods, using N_t+1 evenly-spaced points in the interval $[t^0,T]$, starting from initial conditions of the form given in (5)–(6).

- The **forward Euler method** seeks an approximation to the solution $\boldsymbol{x}(t)$ of (3)–(4) on the interval $[t^0,T]$ by applying the iteration

$$\boldsymbol{x}_h(t^{n+1}) \;=\; \boldsymbol{x}_h(t^n) + h\,\boldsymbol{v}_h(t^n)\,, \tag{7}$$

$$\boldsymbol{v}_h(t^{n+1}) \;=\; \boldsymbol{v}_h(t^n) + h\,\boldsymbol{f}(t^n,\boldsymbol{x}_h(t^n))\,, \tag{8}$$

  for $0\le n<N_t$. This is an order 1 method.

- The **symplectic Euler method** seeks an approximation to the solution $\boldsymbol{x}(t)$ of (3)–(4) on the interval $[t^0,T]$ by applying the iteration

$$\boldsymbol{v}_h(t^{n+1}) \;=\; \boldsymbol{v}_h(t^n) + h\,\boldsymbol{f}(t^n,\boldsymbol{x}_h(t^n))\,, \tag{9}$$

$$\boldsymbol{x}_h(t^{n+1}) \;=\; \boldsymbol{x}_h(t^n) + h\,\boldsymbol{v}_h(t^{n+1})\,, \tag{10}$$

  for $0\le n<N_t$. This is also an order 1 method.

- The **Störmer-Verlet method** seeks an approximation to the solution $\boldsymbol{x}(t)$ of (3)–(4) on the interval $[t^0,T]$ by applying the iteration

$$\boldsymbol{x}_h(t^{n+1}) \;=\; \boldsymbol{x}_h(t^n) + h\,\boldsymbol{v}_h(t^n) + \frac{h^2}{2}\boldsymbol{f}(t^n,\boldsymbol{x}_h(t^n))\,, \tag{11}$$

$$\boldsymbol{v}_h(t^{n+1}) \;=\; \boldsymbol{v}_h(t^n) + \frac{h}{2}\left[\boldsymbol{f}(t^n,\boldsymbol{x}_h(t^n)) + \boldsymbol{f}(t^{n+1},\boldsymbol{x}_h(t^{n+1}))\right]\,, \tag{12}$$

  for $0\le n<N_t$. This is an order 2 method.

Coursework Questions

In `Templates/` you will find a set of folders, one for each question. The folders contain a small amount of code (`.hpp` and `.cpp` files) as well as empty files, which you must edit for the coursework. You can use any software you want to produce the plots requested below.

You must keep the folder structure and all file names as they are in the templates: the folder `Q1` in your submission, for instance, should be self-contained, and should include all the code necessary to compile and reproduce your results for Question 1. The template folders may also serve as a checklist for your submission. As part of your submission, you may also add files to the folders (for instance new classes, output files, plotting routines, etc.): if you do so, then write a brief `README.txt` file, containing a short description of each new file. When you attempt Question 2, make a new folder and put all the files necessary to produce your results in it; if needed, copy some files from `Q1` to `Q2`, etc.

This coursework requires you to implement algorithms for approximating second-order initial value problems of the form (3)–(4) in an object-oriented manner, then use them to approximate a range of second-order systems.

Familiarise yourself with the classes `Vector` and `ODEInterface` which have been provided in the folder `Templates/`. The class `Vector` is a modification of the class that is used in Unit 10 on Iterative Linear Solvers. The class `ODEInterface` is an abstract class that encapsulates an interface to problems of type (5)–(6).

1. In this question, you will use the forward Euler method to approximate a second-order IVP representing the position of a particle undergoing simple harmonic motion in one dimension,

$$\ddot{x}(t) = -a^2 x \qquad \text{for } t \in [0, T] \quad \text{where} \quad T > 0\,, \tag{13}$$

$$x(0) = 0\,, \qquad v(0) = \dot{x}(0) = a\,. \tag{14}$$

   The solution to this IVP is $x(t) = \sin(at)$.

   (a) Write an abstract class `AbstractODESolver` which contains the following members:

   - Protected variables for initial and final times

```
    double mFinalTime;
    double mInitialTime;
```

   - A protected pointer for the ODE system under consideration

```
    ODEInterface* mpODESystem;
```

   - A protected variable for the time-step size h

```
    double mStepSize;
```

   - A pure virtual public method

```
    virtual void Solve() = 0;
```

   - Any other member that you choose to implement. Your choices and your ability to design this class according to object-orientation design principles will be assessed.

   [15]

(b) Write a class `OscillatorODE` derived from `ODEInterface` which:

- Overrides the virtual method `ComputeF` in order to evaluate the right-hand side of (6).
- Overrides the virtual method `ComputeAnalyticSolution` in order to compute the exact solution to (3)–(4) (which is known for this problem).

[5]

(c) Write a class `ForwardEulerSolver`, derived from `AbstractODESolver`, with the following members:

- A public constructor

```
ForwardEulerSolver(ODEInterface& anODESystem,
    const double initialState,
    const double initialVelocity,
    const double initialTime,
    const double finalTime,
    const double stepSize,
    const std::string outputFileName="output.dat",
    const int saveGap = 1,
    const int printGap = 1);
```

  in which `initialState` is $x(t^0)$ and `initialVelocity` is $v(t^0)$.

- A public solution method

```
void Solve();
```

  which computes $\{x_h(t^n)\}$ and $\{v_h(t^n)\}$ using the forward Euler method for a first-order system of IVPs of the form (5)–(6), saves **selected elements** of the sequences $\{t^n\}$, $\{x_h(t^n)\}$, $\{v_h(t^n)\}$ in a file, and prints on screen an initial header and **selected elements** of the sequences $\{t^n\}$, $\{x_h(t^n)\}$, $\{v_h(t^n)\}$. The method should save to file every `saveGap` iterations and print on screen every `printGap` iterations.

- Any other member that you choose to implement. Your choices and your ability to design this class according to object-orientation design principles will be assessed.

[15]

(d) Write and execute a main `Driver.cpp` file which:

i. Approximates the IVP (13)–(14) for $a = 1.5$, $T = 30$, using the forward Euler method with $h = 0.01$, and outputs the solution to a file.

   Use your output to plot the approximate solution $x_h(t)$ against the approximate velocity $v_h(t)$ for $t \in [0, 30]$. Comment briefly on your results.

ii. Approximates the IVP (13)–(14) with $a = 1.5$, $T = 1$ using the forward Euler method with various values of h, computes the corresponding errors $E(h)$ and saves the sequences $\{h_k\}$, $\{E(h_k)\}$ to a file.

   Use your output to plot $\log E(h)$ as a function of $\log h$. Include in your report the values of h and $E(h)$ that you used to produce the plot and a brief explanation of why your results demonstrate that $E(h) = O(h)$.

Your choices for computing these errors and presenting this evidence will be assessed.

[5]

2. (a) Modify `ForwardEulerSolver` to create a new class `SymplecticEulerSolver`, also derived from `AbstractODESolver`, which computes $\{x_h(t^n)\}$ and $\{v_h(t^n)\}$ using the symplectic Euler method for a first-order system of IVPs of the form (5)–(6).

[5]

(b) Modify `ForwardEulerSolver` to create a new class `StoermerVerletSolver`, also derived from `AbstractODESolver`, which computes $\{x_h(t^n)\}$ and $\{v_h(t^n)\}$ using the Störmer-Verlet method for a first-order system of IVPs of the form (5)–(6).

[5]

(c) Write and execute a main `Driver.cpp` file which:

i. Approximates the IVP (13)–(14) for $a = 1.5$, $T = 30$, using the symplectic Euler and Störmer-Verlet methods with $h = 0.01$, and outputs the solutions to two files.

Use your output to plot the approximate solutions $x_h(t)$ against the approximate velocities $v_h(t)$ for $t \in [0, 30]$. Comment briefly on how the results compare with each other and with the results obtained for the forward Euler method.

ii. Approximates the IVP (13)–(14) with $a = 1.5$, $T = 1$ using the symplectic Euler and Störmer-Verlet methods with various values of h, computes the corresponding errors $E(h)$ and saves the sequences $\{h_k\}$, $\{E(h_k)\}$ to two files.

Use your output to plot $\log E(h)$ as a function of $\log h$ for each method. Include in your report the values of h and $E(h)$ that you used to produce the plots and a brief explanation of why your results demonstrate that the methods are order 1 and 2, respectively.

[10]

3. In this question you will modify the classes you have developed in Questions 1 and 2 to approximate a second-order IVP representing the gravitational effects of a fixed body on a free body in three dimensions,

$$\ddot{\boldsymbol{x}}(t) \;=\; \frac{G\,m_p(\boldsymbol{x}_p - \boldsymbol{x})}{|\boldsymbol{x}_p - \boldsymbol{x}|^3} \qquad \text{for} \;\; t \in [t^0, T] \quad \text{where} \quad T > t^0\,, \tag{15}$$

$$\boldsymbol{x}(t^0) \;=\; \boldsymbol{x}^0\,, \qquad \boldsymbol{v}(t^0) \;=\; \dot{\boldsymbol{x}}(t^0) \;=\; \boldsymbol{v}^0\,, \tag{16}$$

in which $G = 6.674 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}}$ is the universal gravitational constant, m_p and $\boldsymbol{x}_p \in \mathbb{R}^3$ are the mass and the position of the centre of mass of the fixed body, and $\boldsymbol{x} \in \mathbb{R}^3$ is the position of the centre of mass of the moving body. All bodies are assumed to be spherical.

This system has a family of solutions which represent circular orbits of the moving body around the fixed body, for which the speed $|\boldsymbol{v}_o|$ and period T_o of the orbit are given by

$$|\boldsymbol{v}_o|^2 \;=\; \frac{G\,m_p}{r}\,, \qquad {T_o}^2 \;=\; \frac{4\pi^2 r^3}{G\,m_p}\,, \tag{17}$$

in which r is the radius of the orbit (the distance between the centres of mass).

(a) Modify the abstract class `AbstractODESolver` and the derived classes for the methods, `ForwardEulerSolver`, `SymplecticEulerSolver` and `StoermerVerletSolver` so that the state $\boldsymbol{x}_h(t^n)$ and velocity $\boldsymbol{v}_h(t^n)$ are now stored in objects of type `Vector`.

For example, the constructor of the class `ForwardEulerSolver` will now take the form

```
ForwardEulerSolver(ODEInterface& anODESystem,
    const Vector& initialState,
    const Vector& initialVelocity,
    const double initialTime,
    const double finalTime,
    const double stepSize,
    const std::string outputFileName="output.dat",
    const int saveGap = 1,
    const int printGap = 1);
```

and the solution method will have to save and print values for $\boldsymbol{x}_h(t^n), \boldsymbol{v}_h(t^n) \in \mathbb{R}^3$. [5]

(b)
- Write a class `OrbitODE` derived from `ODEInterface` which overrides the virtual method `ComputeF` in order to evaluate the right-hand side of (15). `ODEInterface` will also need modifying now that $\boldsymbol{x}_h(t^n)$ is stored as type `Vector`.
- Write a method `DetectCollision` which checks whether the spherical bodies have collided during a time-step and, if there has been a collision, exits the code and reports the time and the state of the system at the end of the time-step during which the collision took place.

  You should decide where this method should be defined in your class structure and provide a reason why you have made this choice.

[5]

(c) Write and execute a main `Driver.cpp` file which:

i. Approximates the orbit of the moon around the earth using the forward Euler, symplectic Euler and Störmer-Verlet methods.

   You should assume that the position of the earth is fixed, the moon travels in a circular orbit with radius $r_o = 3.844 \times 10^8\,\mathrm{m}$, the earth has mass $5.972 \times 10^{24}\,\mathrm{kg}$ and radius $6.378 \times 10^6\,\mathrm{m}$, and the moon has mass $7.342 \times 10^{22}\,\mathrm{kg}$ and radius $1.737 \times 10^6\,\mathrm{m}$. Initialise your computation with $\boldsymbol{x}(0) = (r_o, 0, 0)^T$, $\boldsymbol{v}(0) = (0, |\boldsymbol{v}_o|, 0)^T$.

   Include in your report the approximate state of the system after one period ($T = t^0 + T_o$) for each of the three methods, obtained using 10000 time-steps. Using these results and your results from Questions 1 and 2, order the three numerical methods from best to worst in their ability to approximate second-order IVPs of the type investigated in this coursework. Explain why you have put the methods in this order.

ii. Approximates the time it would take the moon to collide with the earth if the computation was initialised with $\boldsymbol{x}(0) = (r_o, 0, 0)^T$, $\boldsymbol{v}(0) = (0, 0, 0)^T$.

   You should choose the best of the three methods you have implemented for this task and provide your answer to the nearest minute. Explain briefly what you have done to ensure that you have the answer to the appropriate level of accuracy *without finding an exact mathematical solution to the problem*.

   What difficulties do you encounter if you try to compute the time to collision if you assume that the bodies are point masses, *i.e.* they have zero radius?

iii. Approximates the trajectory of the moving body when $\boldsymbol{v}(0) = (0, k|\boldsymbol{v}_o|, 0)^T$ for a range of values of $k \in [0, 2]$, using the method you chose in part ii.

Describe how the trajectories change as k is increased in this range. You should provide plots of trajectories for 4 different values of k to help illustrate how they change and explain why you have chosen these 4 values.

[10]

4. In this question you will modify the classes you have developed in Questions 1, 2 and 3 to approximate the N-body problem, given by the system of second-order IVPs

$$\ddot{\boldsymbol{x}}_i(t) = \sum_{j \neq i}^{N} \frac{G\, m_j (\boldsymbol{x}_j - \boldsymbol{x}_i)}{|\boldsymbol{x}_j - \boldsymbol{x}_i|^3} \qquad \text{for } t \in [t^0, T] \qquad \text{where } T > t^0\,, \tag{18}$$

$$\boldsymbol{x}_i(t^0) = \boldsymbol{x}_i^0\,, \qquad \boldsymbol{v}_i(t^0) = \dot{\boldsymbol{x}}_i(t^0) = \boldsymbol{v}_i^0\,, \tag{19}$$

for each body $i = 1, \ldots, N$. In this equation m_j is the mass of body j and $\boldsymbol{x}_j \in \mathbb{R}^3$ is its position. The remaining parameters and variables are defined in Question 3.

(a)
- Write a class `NBodyODE` derived from `ODEInterface` which overrides the virtual method `ComputeF` in order to evaluate the right-hand side of (18).
- Modify your method `DetectCollision` so that it checks whether any of the spherical bodies have collided during a time-step and, if there has been a collision, exits the code and reports the time and the state of the system at the end of the time-step during which the collision took place.

[10]

(b) Write and execute a main `Driver.cpp` file which:

i. Approximates the orbit of the moon around the earth using the best of the three numerical methods you have implemented, as selected in Question 3.

You should use the same initial state as in Question 3(c)i. but the earth should now be allowed to move in response to the gravitational effects of the moon. Provide a plot of the trajectories of the two moving bodies and describe how the motion differs from that seen in Question 3. Include in your report the approximate state of the system after one period ($T = t^0 + T_o$), obtained using 10000 time-steps.

ii. Approximates the trajectories in a system with at least 5 moving bodies, again using the best of the three numerical methods, as selected in Question 3.

You are free to choose the system yourself, but you must choose it so that it provides you with additional evidence that your implementation of the algorithm is correct for multiple bodies. It must, therefore, be a situation in which you know something about how the system should behave and you can confirm that your numerical results reproduce that behaviour. It does not have to be a problem for which the exact solution is known.

Provide a plot of the trajectories of the moving bodies and use it to explain why the test case you have chosen helps to confirm the correctness of your implementation. You should make sure that you define the test case you choose in your report in a way which would allow another user to set up and run the same problem.

iii. Approximates a sequence of problems with $N = 2, 4, 8, 16, 32, 64$, bodies and computes the cpu times for each problem

You are free to choose how to initialise the system in each case, but you should describe the initial states in your report in a way which would allow another user to set up and run the same problem. Each test should be run for 10^5 time-steps and you should minimise printing to the screen or saving to file during these tests.

Provide the cpu times for each of your runs and describe how the runtime depends on the number of bodies. Use these runtimes to estimate the length of time it would take for your code to compute 10^5 time-steps with 2^{37} bodies (approximately the number of stars in our galaxy).

[10]

The output requested in Questions 1d, 2c, 3c, 4b should be included in your submission, in the format provided by the solution template file. Please be selective in the output you present: none of the questions should require more than 100 lines of code output to be presented (most need significantly fewer than that).