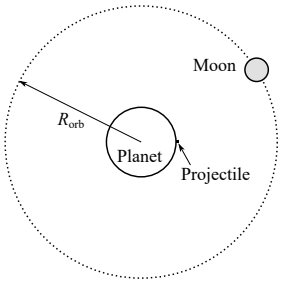
PHYS4070 - Due date: Tuesday 12/03/2025, 5:00pm

Project 1 - Simple ODE and N-body problem

Lecturer: Ben Roberts, Rm 6-427, b.roberts@uq.edu.au

A Numerical solution of a simple n-body problem

Consider a system consisting of a planet and moon, interacting via gravity. We will numerically model the dynamics of a projectile launched from the planet to the moon using fourth-order Runge-Kutta (RK4) integration, with a goal of eventually getting the projectile into a stable orbit around the moon.



	Mass	Radius
Planet	1	1
Moon	0.1	0.25
Projectile	$\ll 1$	$\ll 1$

We will work in units where the planet's mass m_{pl} and radius R_{pl} are both set to 1, and the gravitational constant $G = 1$. The moon orbits around the planet at a constant radius of $R_{\text{orb}} = 19\,R_{\text{pl}}$. Our co-ordinate system will be centred on the planet. Our system of equations has the form:

$$\ddot{\mathbf{r}}_i = \sum_{j \neq i} \frac{m_j}{|\mathbf{r}_{ij}|^2} \hat{\mathbf{r}}_{ij}, \tag{1}$$

where $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$, and $\mathbf{r}_i = (x_i, y_i)$ (i.e. we assume that all the dynamics are restricted to the xy-plane). For simplicity, we will **assume that the planet is fixed in place, so its acceleration is zero for all times**. We can cast this into the form of an ODE:

$$\partial_t \begin{pmatrix} \boldsymbol{r}_i(t) \\ \dot{\boldsymbol{r}}_i(t) \end{pmatrix} = f(\boldsymbol{r}_i(t)) = \begin{pmatrix} \boldsymbol{v}_i(t) \\ \ddot{\boldsymbol{r}}_i(t) \end{pmatrix} \tag{2}$$

which is in a form that can be solved using numerical techniques.

It will be helpful to remember Kepler's law:

$$T^2 = (2\pi)^2 \frac{R^3}{GM}$$

B Tasks

B.1 Two-body approximation:

We want to consider a projectile, launched from the surface of the planet towards the moon at some (purely radial) velocity v_0. Start by assuming that the moon does not influence the trajectory of the projectile.

1. Analytically calculate the initial velocity you would need to launch the projectile from the surface of the planet, such that it has zero velocity exactly as it reaches the surface of the moon (you can consider the moon to be fixed in place here). Hint: energy conservation.

Now, use the RK4 method to simulate the trajectory of both the projectile, and the moon, accounting for Earth's gravity only (use any sensible initial condition for the moon). Start your simulation with the projectile on the surface of the planet, and launch it radially at your calculated velocity, v_0.

2. How long (in our dimensionless units) does the particle take to get to the surface of the moon in your simulation?

3. Based on the time taken for the projectile to reach the moon's surface, what initial condition, $\{\mathbf{r}_{\text{moon}}, \dot{\mathbf{r}}_{\text{moon}}\}$, should you set for the moon such that it passes by the projectile at exactly the time that the projectile's velocity reaches zero? Demonstrate numerically (with a plot) that this works.

B.2 Three-body problem:

We will now generalise the simulation to include the gravitational interaction between the moon and the projectile. The goal will be to perform a simplistic 'orbital transfer', such that the projectile ends up in orbit around the moon. This will require the projectile to be instantaneously 'kicked' once it becomes close enough to the moon. This kick models thrust produced by the projectile (e.g. a spacecraft) as it is being manoeuvred into orbit around the moon. This will require re-writing your ODE solver to account for the three-body interactions.

1. Write out the equations of motion with this new moon–projectile interaction included. State any assumptions that you have made.

2. Build on your code from Part B.1 to include the moon–projectile gravitational interaction. Confirm that your simulation is giving reasonable results, and re-draw the plot you made in B.1.3. **Important:** make sure you update your whole system of equations *simultaneously.*

3. Plot the distance $r_{\mathrm{mp}}(t)$ between the moon and the projectile. At what time does this separation first drop to a value of $r_{\mathrm{mp}} = 0.5R_{\mathrm{pl}}$ (i.e. twice the radius of the moon)?

4. Analytically determine both the magnitude and direction of a 'kick' velocity $\Delta\mathbf{v}$ that would need to be instantaneously applied to the projectile, $\mathbf{v}_{\mathrm{proj}} \to \mathbf{v}'_{\mathrm{proj}} = \mathbf{v}_{\mathrm{proj}} + \Delta\mathbf{v}$, such that it enters a circular orbit around the moon at its current radius r_{mp}. Your answer should be expressed in terms of (i) the radial and angular separation of the moon and projectile (r_{mp} and $\theta_{\mathrm{mp}} = \arctan[(y_{\mathrm{moon}} - y_{\mathrm{proj}})/(x_{\mathrm{moon}} - x_{\mathrm{proj}})]$, respectively), (ii) the current (pre-kick) projectile velocity $\mathbf{v}_{\mathrm{proj}}$, and (iii) the current velocity of the moon $\mathbf{v}_{\mathrm{moon}}$. It might help to draw a diagram.

5. In your code, add in this instantaneous 'kick' velocity to your projectile at the time found above such that its motion is converted into a circular orbit around the moon at a radius of $0.5R_{\mathrm{pl}}$ for all later times.

   Run the code for a whole orbit of the moon around the central planet and plot the trajectories of both the moon and the particle in the planet-centred coordinates. Also plot the projectile's motion in moon-centred coordinates, and check how circular its trajectory is after the kick.

C Submission

Submit your report and your source code via blackboard. Preferably, upload a single zipped file, including:

- Your report (as a pdf)
- Your source code. You may organise the files however you wish – for readability, you may want to split your functions across several files, but it's up to you.
  - Please, only submit source files – do not submit compiled executable or large output data files.
- A bash script or Makefile that compiles the code.
- Any scripts you used to generate the plots, using the output from your code.

Important: your code should be self-contained. I should not need to modify your source code in any way to generate the answers to the assignment (modifying input options is fine, so long as they are read in by your code, e.g., from the command-line or from a text file, and it doesn't need to be recompiled).

Report format

Your report should contain a short introduction outlining the problem and the main goals, as well as a short conclusion summarising the results and how you obtained them. Make sure to include:

- All the results, plots and answers the above questions ask you to produce, including any important steps from your working, and any observations.
- Each section should contain a brief description and explanation of everything you've done to obtain your results/data/figures. You don't need code snippets, but you can include them if you like.
- Interpretation and analysis of your results.
- Keep it fairly short: absolute maximum of 5 pages

D Marking guide

- Marked out of 10.
- 6 marks are for the report/questions/plots
  - This is 50% for correctness of the result, and 50% for explanation/discussion, demonstrating insight and understanding.
- 4 marks are for the code, including correctness and code quality
  - Factors impacting quality: is the code easy to read/follow? Is it well structured? Are variables named appropriately? Is repetition avoided? etc.
  - This assignment only: 1 mark is given for using at least 1 C++ feature (e.g., classes or structs, operator overloading, std::vector/std::array etc.).

E Hints

- Break each task up into chunks: use functions! One function should have one job. Every job should be a function.
- For example, you should have a *function* that steps the system forwards using Runge-Kutta, and *another function* that calculates the gravitational acceleration on each body, etc. etc.
- If you're having trouble getting the Runge-Kutta to work, try at first with the much simpler Euler's method. If you write your code well, it will be very easy to simply replace this function with the RK version later.
- Don't forget to make use of the examples given in the lectures/tutorials etc.
- When plotting the trajectories: you will want to set your x and y axes to equal scales, otherwise your circular orbit might look elliptical. For gnuplot: '`set size ratio -1`'.
  For python/matplotlib: '`ax.set_aspect('equal')`'
- You can make use very simple structs/classes to greatly simplify your code. For example:

```
// 'using' just saves typing
using Vector = std::array<double, 2>;

// Defines a 'particle' object: position, velocity, mass
struct Particle {
  Vector r;
  Vector v;
  double mass;
};

// Gravitational acceleration of particle 1, due to particle 2
Vector a_gravity(const Particle &p1, const Particle &p2) {
  // You'll need to define the operator overloads for the following:
  // Displacement vector:
  Vector rij = p2.r - p1.r;
  double r2 = rij * rij;
  // unit vector:
  Vector uij = (1.0 / std::sqrt(r_sqr)) * rij;
  return (p2.mass / r_sqr) * uij;
}
```