Final Project

This project involves writing a package to solve an ODE system in the standard format, verifying the code, then using it to make movies. Before you start, please read the posted files on Python classes. When you're ready to make a movie, look at the posted movie demo file. *Confession.* I don't understand Python graphics and particularly animation very well. The demo I posted works for me, but the movie player crashes a lot. I would be grateful if someone who understands it better could explain it to me.

This project is in part about the process of developing a piece of scientific computing software. In particular, there should be a range of tests. The problem should not be too closely integrated into the solution software to allow for easier testing problems for validation. That's even if the overall project has a specific target application. Please upload code and whatever movie/movies you make. Part of the grade will be on the quality of the code and movies using criteria such as thoughtfulness of the scales, symbols, labels, etc (for movies) and modularity, clarity and ease of use of the software. Chapter 8 of the book Principles of Scientific Computing has the necessary background on ODE solvers. I had to give my NYUid (not Courant ID) and password to access this.

Principles of Scientific Computing
https://cs.nyu.edu/courses/spring09/G22.2112-001/book/book.pdf

Basic solver

Write a basic solver for the initial value problem

$$\dot{x} = f(x)\ ,\quad x(0) = x_0\ ,\quad x(t) \in \mathbb{R}^m\ . \tag{1}$$

(Below, d will be the space dimension, either 2 or 4, and n will be the number of planets.) This should have the option of using the first order forward Euler method or the four stage fourth order explicit Runge Kutta method (often called "the" Runge Kutta method). The solver should take as input an object that evaluates $f(x)$ for a given x, the initial condition x_0, the time step Δt, and the final time T. For order of accuracy checking, you might add the option of specifying Δt and the number of time steps, but this should not be used for the big runs that make the movie or movies.

Validate your code using the $m = 2$ model problem

$$\dot{x} = \frac{\pi}{\sqrt{x_1^2 + x_2^2}} \begin{pmatrix} -x_2 \\ x_1 \end{pmatrix}\ ,\quad x_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}\ ,\quad x(t) = \begin{pmatrix} \cos(\pi t) \\ \sin(\pi t) \end{pmatrix}\ . \tag{2}$$

This problem has some features of a good testing problem

- It is simple and has a simple explicit solution.

- It is not too simple. For example, a $d = 1$ model problem might miss indexing bugs in the software.
- It is non-linear. It is possible (I have done it) to get the Runge Kutta formulas wrong in a way that leads to 4-th order accuracy for linear problems, but not for non-linear ones.

Verify that both the Euler and the Runge Kutta options give the correct answer for $T = 2$ in the limit $\Delta t \to 0$.

Order of accuracy verification

Compute the error for a sequence Δt, $\frac{1}{2}\Delta t$, etc. Do the convergence analysis to verify that the Euler version is first order and the Runge Kutta is fourth order. Use$Tt = 2$ or $T = 4$ or some larger value. You may find that the clean verification for Runge Kutta is harder because the error is so small that roundoff spoils the computed asymptotic error expansions. It is possible that a larger T makes the Runge Kutta verification easier because the error is larger, maybe.

For the n body simulations below, it is helpful to have a qualitative idea what the errors from Euler's method look like. For this, make a simulation up to a long time such as $T = 20$ or $T = 40$ and make a 2D plot of the trajectory (one plot for each method?). You should see the Euler trajectory spiral out while the Runge Kutta solution follows the circle better, and for a longer time.

Adaptive Δt

Build on top of the ODE solver a routine that knows the order of accuracy but not the solution and tries to find Δt to achieve a specified error. This code should have as input ϵ and a first guess at Δt. It should do runs with Δt and $\frac{1}{2}\Delta t$ to estimate the first term (after the true solution) in the error expansion. This allows you to estimate the error:

$$r^{(\Delta t)} \approx x^{(\Delta t)}(T) - x(t) \ .$$

Stop if this estimated error suggests that

$$\left\| r^{(\Delta t)} \right\| \leq \epsilon \ . \tag{3}$$

Otherwise replace Δt with $\frac{1}{2}\Delta t$ and repeat. When you have found a Δt that satisfies the accuracy criterion (3), return the best estimate of the solution, which is $x^{(\Delta t)}(T) + r^{(\Delta t)}$. Test this using the test problem (2), for which you know the actual answer. Does this adaptive Δt solver produce solutions of accuracy ϵ? Is the solution more or less accurate? Try for Euler and Runge Kutta. Things to be careful of

- The simple procedure may fail if ϵ is too small because of roundoff error. The code should have a maximum number of Δt reductions before it prints an error message and gives up.

- It might be that $x^{(\frac{1}{2}\Delta x)} - x^{(\Delta x)}$ is smaller than the target accuracy ϵ but the asymptotic error expansion does not work because of roundoff. What should the code do in that case?
- What will/should the code do if Δt or T is so large that the asymptotic error expansion is not a good approximation (we say the expansion is "invalid", but that's an inappropriate word for this situaton.)

Planets, the n body problem

The n body problem is the ODE system that describes the motion of n "planets" (objects) under gravitational attraction. Suppose there are planets at locations r_1 and r_2 in $\mathbb{R}^2$ or $\mathbb{R}^3$. The force on the planet at r_1 coming from the planet at r_2 given by Newton's law of gravity:

$$F_{12} = g m_1 m_2 \frac{r_2 - r_1}{\|r_3 - r_1\|^3} \ .$$

The masses of the planets are m_1 and m_2 respectively. The overall *gravitational constant* is g. It is a dimensional constant, but we will set $g = 1$ for simplicity. The force F_{12} pulls the r_1 planet toward the r_2 planet, which accounts for the sign $r_2 - r_1$ rather than $r_1 - r_2$. The strength of the force is $g m_1 m_2 \|r_2 - r_1\|^{-2}$. This is Newton's *inverse square* law. Planets move according to Newton's law of motion

$$F_{\text{tot}} = ma \ .$$

Here, F_{tot} is total force on a planet, which is the sum of the forces from the other planets. On the right is m, the mass x_2 of the planet, and $a = \ddot{r}$, which is the accelleration. Altogether,

$$m_j \ddot{r}_j = \sum_{k \neq j} g m_j m_k \frac{r_k - r_j}{\|r_k - r_j\|^3} \ .$$

Note that m_j cancels from both sides. It will take some coding to put these equations into the standard ODE form You should do (at least in the beginning) the *restrictted* n body problem which means $d = 2$ instead of the physically relevant $d = 3$. There are phenomena that happen in the restricted or the full problem, and others that only happen for $d = 3$. For $d = 2$ and n planets, ODE system (1) will have $m = 4n$.

The two body ($n = 2$) problem has periodic elliptical trajectories if the initial velocities are small enough. See how well the Euler and Runge Kutta solvers reproduce this. The signature of periodicity is that if you run for a long time, you trace out the same ellipse many times. The solver does this only approximately, depending on Δt. If you take $r_1(0) = 0$ and $r_2(0) = (1, 0)$, make sure the initial velocities are not on the x axis, or you will see a line segment rather than an ellipse. Does the adaptive Δt finder work on this problem?

You should be able to imitate a star centered system by taking m_1 to be many orders of magnitude (powers of 10) larger than the other masses, and by taking $\dot{r}_1(0) = 0$. The solvers should not break and you should not need a very small Δt.

The $n = 3$ body problem (or $n > 3$) does not have simple behavior. Experiment with various situations to see what interesting movies you can make. If there is a central star and if the orbits are nearly circular and distinct, then the orbits change slowly. After a while a planet can be "ejected" by other planets. Astronomers believe there are lots of ejected (rogue) planets from various planet systems "out there" but we can't see them because they don't shine.