Numerical Methods II, Spring 2023
Assignment III: Planetary Orbits in a Solar System

Aleksandar Donev
Courant Institute, NYU, donev@courant.nyu.edu

Feb 1 2023
Due: **5pm Tuesday 2/?? 2023**

The optional parts of this assignment do not carry any points; only do optional things if you have time and interest for your own benefit.

1 [90 points] Newton's Laws of Motion

In this homework you will solve a system of ODEs modeling the motion of a collection of N planets around a star. The masses of the planets will be denoted with m_i and their positions with $\boldsymbol{r}_i(t)$, $i = 1, \ldots, N$. We will assume that the mass of the star m_0 is much larger than the masses of the planets, so we can assume that the star remains fixed at the origin, $\boldsymbol{r}_0(t) = \boldsymbol{0}$. Here you can assume that all of the planetary orbits are co-planar, $\boldsymbol{r}_i = [x_i(t), y_i(t)] \in \mathbb{R}^2$, although it is [**optional but encouraged**] recommended that you write your code so that the number of dimensions (2 or 3) can be an input parameter. That way your code can be used to study the impact of a comet coming out of the plane of the solar system without any changes.

Newton's laws of motion for the planets take the form of a system of *second order* ODEs,

$$\frac{d^2\boldsymbol{r}_i}{dt^2} = \boldsymbol{F}_i(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N) = \sum_{j=0, j\neq i}^{N} Gm_j \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{\|\boldsymbol{r}_j - \boldsymbol{r}_i\|_2^3}, \quad i = 1, \ldots, N.$$

For this assignment you can set the value of the gravitational constant $G = 1$, although good programming practice suggests that you should keep it is a (perhaps global) variable in the code instead of hard-wiring the unit value.

You will write your own code to solve the ODEs using the Bogacki–Shampine embedded RK3 method (BS-RK23) used in the MATLAB ode23 integrator, following the description in the Wikipedia page (and my notes) https://en.wikipedia.org/wiki/Bogacki-Shampine_method
This explicit one-step method is third order accurate, but also has an embedded second order estimator for adaptive error control. Put effort into writing a code that can integrate *any* first-order system of ODEs (like ode23) and not just the specific ODEs for gravitational systems. That is the rhs of the ODE should be input to the code, not hard-wired in the code. We usually write solvers assuming that the system of ODEs is first order, unless we are using some special integrator for 2nd order PDEs, so convert the system of ODEs to a first-order system first.

[**optional**] If you are familiar with symplectic 2nd order integrators you are welcome to try them and compare to ode23, see (optional) section 1.3 below.

1.1 [70pts] Single planet

Consider now the case $N = 1$, and denote the position of the single planet with $\boldsymbol{r}(t)$ with length $\|\boldsymbol{r}\|_2 = r$. It is well known since Kepler that the orbit of a planet around a massive star is an ellipse with the star being one if its foci, with the period of the orbit given by the formula

$$T = \frac{2\pi}{\omega} = (2\pi)\, a\sqrt{\frac{a}{Gm_0}},$$

where a is the major semi-axes of the ellipse, and ω is the angular frequency. The major semi-axes is given by

$$a = -\frac{Gm_0}{1}\frac{}{2E},$$

where the energy E (which is a constant of the motion) is the sum of the gravitational (potential) and kinetic energy:

$$E = -\frac{Gm_0}{r} + \frac{v^2}{2}.$$

Note that here we set the mass of the planet to unity because it cancels out in the end and only m_0 matters for a single planet. Assume that the initial position of the planet is $\boldsymbol{r}(0) = [r(0), 0]$ and set the initial velocity $\boldsymbol{v} = d\boldsymbol{r}/dt = [0, v(0)]$, which ensures that the initial position is either the furthest or the closest point to the star on the orbit (i.e., it is a vertex of the ellipse).

1.1.1 [30 pts] Circular orbit

For a circular orbit, $\boldsymbol{r} = r\,[\cos\omega t, \sin\omega t]$, the initial speed is $v(0) = \omega r(0)$, and the centripetal and gravitational forces are balanced,

$$r\omega^2 = \frac{Gm_0}{r^2}.$$

Use the BS-RK23 method to integrate a circular orbit over a period of revolution, using a fixed time step size $\Delta t = T/M$ where M is the number of time steps. While for simplicity in this part you should set $r = 1$ and $\omega = 1$, write your code so that these parameters can be changed easily. Observe that the only dimensionless number that matters is $\Delta t/T$ and not the value of Δt on its own, so that changing the values of the parameters does not require new simulations.

[10pts] Confirm that the BS-RK23 is third-order accurate as a way to test your implementation.

[20pts] Find the largest possible time step size Δt that ensures that the maximum error in the x and y position of the planet is no larger than $10^{-2}r$, i.e., one percent relative error or two digits of accuracy (first solution). Then compute the orbit with time step $\Delta t/2$ (second solution). Use Richardson extrapolation to obtain a more accurate solution (third solution) as a function of time. For all three solutions (least accurate, more accurate, and most accurate), plot the error

$$e^k = \left\| \boldsymbol{R}^k - \boldsymbol{r}\left(k\Delta t\right) \right\|,$$

where $\boldsymbol{R}^k$ is the numerical approximation and $\boldsymbol{r}\left(t\right)$ is the true solution, as a function of time $t = k\Delta t$ using both a linear and a logarithmic scale for the vertical axes, and comment on your observations (indicate which norm you used). How many digits of accuracy did you get with Richardson extrapolation? (Observe that you could not predict/estimate this using error estimates so you are only able to answer this question because we know the exact solution of the ODEs).

1.1.2 [40 pts] Adaptive integration

[20pts] Write an adaptive BS-RK3 integrator that ensures that each component of the position at time T is accurate to an absolute error of $10^{-2}r(0)$. Start the orbit with a velocity 1, 2, 4 and 8 times smaller than that required to get a circular orbit, so that you get increasingly more eccentric orbits. Integrate the orbit over one period T, and compute the error in the position after that one orbit (the planet should have returned to its initial position). Check that the adaptive time step control is doing something reasonable by plotting $x(t)$ for all points along the integration.

[20pts] Confirm whether the adaptive error control worked and the target error was accomplished (recall that error control is likely to be pessimistic and reduce the time step size too much). Plot the time step size as a function of time (including rejected ones, as explained in class), and comment on what you observe and whether it makes sense to you or not. Try error tolerance of $10^{-3}r(0)$ and see what that does — does the error tolerance in the input directly correspond to the actual error you get?

[**Optional**] To judge whether the adaptive control was worth it we would need to compare the total number of time steps taken to integrate the orbit to achieve a given error tolerance with a constant versus an adaptive time step size. In the case of a single planet we know the correct answer at the end of one period so you can do this comparison and see what you get.

1.2 [20 pts] Fun with multiple planets

This part is for you to play and have fun but also learn how to make scientific animations (movies). Consider the case $N > 1$, say $N = 5$, called the N-body problem. Play around with your adaptive integrator (if your adaptive strategy failed to work, just use the built-in *ode23*) and see if you can construct an example where something interesting happens, and report what you came up with. Make a movie showing off your example. If you can produce a standalone movie file (say AVI or MPEG) that would be great — submit that movie with your homework (*must not exceed 50MB*). If not, at least submit a script that can produce the movie in real time.

Note that because the planets interact with each other via gravitational forces, in general one cannot say much analytically about systems with more than three bodies. However, if the planets are sufficiently far from each other the gravity of the star will dominate and each planet will orbit around the star in an elliptical orbit (but that sounds a bit boring!). You can read more on Wikipedia under "Stability of the Solar System."

1.3 [optional] Long-time integration

Special classes of ODE solvers have been developed for integrating Newton's laws of motion. In particular, *symplectic* methods are supposed to be much better for integrating the equations over *long* periods of time. The simplest second-order symplectic method is the so-called Verlet integrator, based on a direct centered second order finite difference for $d^2\boldsymbol{r}/dt^2$,

$$\frac{\boldsymbol{r}_i^{k-1} - 2\boldsymbol{r}_i^k + \boldsymbol{r}_i^{k+1}}{\Delta t^2} = \boldsymbol{F}_i^k = \boldsymbol{F}_i\left(\boldsymbol{r}_1^k, \ldots, \boldsymbol{r}_N^k\right).$$

While in practice this method is used as a one-step method with position and velocity as two independent variables, here we have written it as a multistep method for integrating directly the second-order equations.

Write a Verlet integrator and use it integrate a circular and a somewhat elliptical orbit over many periods (define "many" as appropriate based on what you observe), using the time step size you found in part 1.1.1 but also trying larger time step sizes to see how robust the integrator is for larger time step sizes. For this part, you can generate the two starting values for the multistep method using the exact solution. Plot the orbits and see if you see any qualitative features. Compare the accuracy you get for the circular orbit to what you get using the BS-RK23 method with the same time step size, and comment on your observations.