Project 3: A model of the solar system

Joshua Bradt
(Dated: April 1, 2016)

Three algorithms for solving ordinary differential equations are discussed and applied to the problem of tracking the planets of the Solar System. The Verlet and fourth-order Runge-Kutta methods are shown to produce reasonably precise results for the two-body Earth-Sun system, but only the Runge-Kutta method behaves well when all eight planets are included.

I. INTRODUCTION

The solar system is a simple, physically meaningful system to use when comparing different techniques for solving differential equations. Using Newtonian gravity (and ignoring relativistic effects), the planets are held in orbit around the sun by the inverse square law force

$$\mathbf{F} = \frac{Gm_1m_2}{r^3}\mathbf{r}$$

where the constant $G = 6.67 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}} = 2.959 \times 10^{-4}\,\mathrm{au^3\,m_\odot^{-1}\,d^{-2}}$. [1] Here, the vector $\mathbf{r}$ points from one body to the other. For N bodies, this generalizes to give the following force on body k:

$$\mathbf{F}_k = \sum_{i \neq k} \frac{Gm_im_k}{r_{ik}^3}\mathbf{r}_{ik}$$

Naturally there are N such equations for the N bodies in the system.

In Cartesian coordinates, these can be rewritten as $3N$ coupled equations:

$$\frac{d^2x_k}{dt^2} = \sum_{i \neq k} \frac{Gm_i}{r_{ik}^3}(x_i - x_k)$$
$$\frac{d^2y_k}{dt^2} = \sum_{i \neq k} \frac{Gm_i}{r_{ik}^3}(y_i - y_k)$$
$$\frac{d^2z_k}{dt^2} = \sum_{i \neq k} \frac{Gm_i}{r_{ik}^3}(z_i - z_k)$$

where

$$r_{ik} = \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2 + (z_i - z_k)}.$$

The motion of the planets can then be simulated given appropriate initial conditions and an algorithm for solving these equations.

II. NUMERICAL ALGORITHMS

A. Euler's method

Perhaps the simplest method for solving a set of differential equations is Euler's method. This simple algorithm uses the first derivatives to step from one point to the next.

Given an initial position $\mathbf{r}_i$ and velocity $\mathbf{v}_i$ at time t_i, the position and the velocity at the next step are calculated as [2]

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \frac{\mathbf{F}(\mathbf{r}_i, t_i)}{m_i}\Delta t \tag{1}$$
$$\mathbf{r}_{i+1} = \mathbf{r}_i + \mathbf{v}_i\Delta t. \tag{2}$$

A possible implementation of this algorithm is shown in pseudocode in Algorithm 1.

This algorithm is very simple to implement, but as a first-order method, its $O(\Delta t)$ error can severely limit the precision of the results. This can be helped by choosing a small time step, but this increases computation time.

Algorithm 1 Euler method for position and velocity

```
function EULER(r, v, h)
    a ← findAcceleration(r)
    v' ← v + ha
    r' ← r + hv
    return (r', v')
end function
```

B. Velocity Verlet algorithm

Another method for solving the equations can be found by beginning with Taylor expansions for the position and velocity: [2]

$$\mathbf{r}(t+h) = \mathbf{r}(t) + h\dot{\mathbf{r}}(t) + \frac{h^2}{2}\ddot{\mathbf{r}}(t) + O(h^3) \tag{3}$$
$$\mathbf{v}(t+h) = \mathbf{v}(t) + h\dot{\mathbf{v}}(t) + \frac{h^2}{2}\ddot{\mathbf{v}}(t) + O(h^3). \tag{4}$$

To second order, the derivative of the second equation is

$$\dot{\mathbf{v}}(t+h) = \dot{\mathbf{v}}(t) + h\ddot{\mathbf{v}}(t) + O(h^2)$$

which can be rearranged to find

$$\frac{h^2}{2}\ddot{\mathbf{v}}(t) = \frac{h}{2}[\dot{\mathbf{v}}(t+h) - \dot{\mathbf{v}}(t)] + O(h^3).$$

This can then be plugged into (4) to find

$$\mathbf{v}(t+h) = \mathbf{v}(t) + \frac{h}{2}[\dot{\mathbf{v}}(t+h) + \dot{\mathbf{v}}(t)] + O(h^3).$$

Finally, note that $\mathbf{v}(t) = \dot{\mathbf{r}}(t)$ and $\mathbf{F}(t) = \dot{\mathbf{v}}(t) = \ddot{\mathbf{r}}(t)$, so (3) and (4) can be rewritten (in discretized form) as

$$\mathbf{r}_{i+1} = \mathbf{r}_i + h\mathbf{v}_i + \frac{h^2}{2}\frac{\mathbf{F}(\mathbf{r}_i, t_i)}{m} + O(h^3) \tag{5}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \frac{h}{2}\left(\frac{\mathbf{F}(\mathbf{r}_{i+1}, t_{i+1})}{m} + \frac{\mathbf{F}(\mathbf{r}_i, t_i)}{m}\right) + O(h^3) \tag{6}$$

These two equations can then be evaluated in the order written at each iteration to find the new position and velocity for each body. [2] This is shown using pseudocode in Algorithm 2.

Algorithm 2 Verlet method for position and velocity

function VERLET($\mathbf{r}$, $\mathbf{v}$, h)
  $\mathbf{a} \leftarrow$ findAcceleration($\mathbf{r}$)
  $\mathbf{r}' \leftarrow \mathbf{r} + h\mathbf{v} + \frac{1}{2}h^2\mathbf{a}$
  $\mathbf{a}' \leftarrow$ findAcceleration($\mathbf{r}'$)
  $\mathbf{v}' \leftarrow \mathbf{v} + \frac{1}{2}h(\mathbf{a} + \mathbf{a}')$
  return $(\mathbf{r}', \mathbf{v}')$
end function

C. Runge-Kutta method

The final method that will be considered here is the fourth-order Runge-Kutta method, or RK4. This method of numerical integration is similar to Simpson's rule, which states that [3]

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

In this case, the value of the function being integrated is known at the starting point but unknown at the midpoint and the end of the step. At these locations, the function is approximated using Euler's method. This leads to the set of equations [2]

$$k_1 = f(x_i, t_i) \tag{7}$$

$$k_2 = f(x_i + \tfrac{h}{2}k_1, t_i + \tfrac{h}{2}) \tag{8}$$

$$k_3 = f(x_i + \tfrac{h}{2}k_2, t_i + \tfrac{h}{2}) \tag{9}$$

$$k_4 = f(x_i + hk_3, t_i + h) \tag{10}$$

$$x_{i+1} = x_i + \tfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5). \tag{11}$$

One clear advantage of this method is that the error scales as h^5, making this a much more precise method than either of the previous two. [2]

An implementation of this method is shown in Algorithm 3.

Algorithm 3 RK4 method for position and velocity

function RK4($\mathbf{r}$, $\mathbf{v}$, h)
  $\mathbf{k}_1^v \leftarrow$ findAcceleration($\mathbf{r}$)
  $\mathbf{k}_1^r \leftarrow \mathbf{v}$
  $\mathbf{k}_2^v \leftarrow$ findAcceleration($\mathbf{r} + \frac{h}{2}\mathbf{k}_1^r$)
  $\mathbf{k}_2^r \leftarrow \mathbf{v} + \frac{h}{2}\mathbf{k}_1^v$
  $\mathbf{k}_3^v \leftarrow$ findAcceleration($\mathbf{r} + \frac{h}{2}\mathbf{k}_2^r$)
  $\mathbf{k}_3^r \leftarrow \mathbf{v} + \frac{h}{2}\mathbf{k}_2^v$
  $\mathbf{k}_4^v \leftarrow$ findAcceleration($\mathbf{r} + h\mathbf{k}_3^r$)
  $\mathbf{k}_4^r \leftarrow \mathbf{v} + h\mathbf{k}_3^v$
  $\mathbf{v}' \leftarrow \mathbf{v} + \frac{h}{6}(\mathbf{k}_1^v + 2\mathbf{k}_2^v + 2\mathbf{k}_3^v + \mathbf{k}_4^v)$
  $\mathbf{r}' \leftarrow \mathbf{r} + \frac{h}{6}(\mathbf{k}_1^r + 2\mathbf{k}_2^r + 2\mathbf{k}_3^r + \mathbf{k}_4^r)$
  return $(\mathbf{r}', \mathbf{v}')$
end function

III. RESULTS

The algorithms described above were implemented in a C++ program [4] to track the motion of the planets. This code was run for a variety of test cases, which will be described in the following sections. Unless stated otherwise, planetary masses were taken from Wolfram Alpha [1] and initial position and velocity vectors were taken from the NASA Jet Propulsion Laboratory's online HORIZONS tool [5]. The position and velocity vectors were calculated relative to the Solar System's barycenter and for the date March 25, 2016, at 00:00:00 TDB.

A. Earth-Sun system

First, the code was tested considering only the motion of the Earth around the Sun. In this case, the orbit of the Earth was assumed to be circular, and the Sun was fixed in place at the origin of the two-dimensional coordinate system.

The initial position of the Earth was set to the vector $\mathbf{r} = (1.0, 0)\,\text{au}$ since 1 au is defined as the approximate distance between the Earth and the Sun. The initial velocity of the Earth can then be found by equating the centripetal force to the gravitational force to find

$$v = \sqrt{\frac{Gm_\odot}{r}} = 1.7202 \times 10^{-2}\,\text{au/d}$$

in the tangential direction, which is y in this case. [6] One result of running with these parameters is shown in Figure 1.

One test of the quality of this calculation is to check that the Earth's kinetic and potential energies and its angular momentum are constants. These must all be constant since for a circular orbit, the Earth-Sun distance and the Earth's velocity must remain constant. For the case shown in Figure 1, where the integration was done using the RK4 algorithm with a step size of 1.0 d, the maximum deviation of the Earth's kinetic energy from its initial value was 0.012 %, and the maximum deviation of the Earth-Sun distance from its initial value was

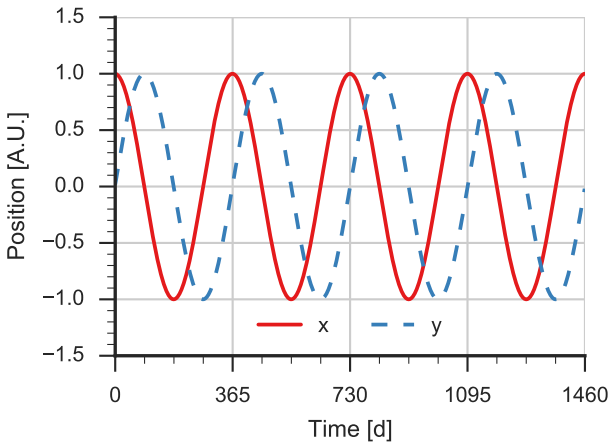


FIG. 1. Earth's position over 4 years assuming a circular orbit. This was calculated using the RK4 integrator with a time step of 1.0 d.

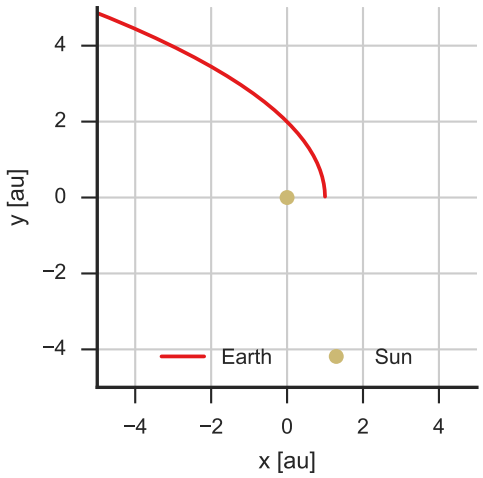


FIG. 2. Trajectory of the Earth given an initial tangential velocity equal to its escape velocity. The trajectory appears to be parabolic, as expected. [6] The step size was 1.0 d, and the integrator was RK4.

0.0060 %. This implies that the kinetic energy, potential energy, and angular momentum were all constant.

Due to its very small error of $O(h^5)$, RK4 will generally give good results for a reasonable step size, as shown above. On the other hand, Euler's method, using the same step size of 1.0 d, produces a maximum kinetic energy deviation of 1.76 % and a maximum deviation in the Earth-Sun distance of 0.89 %. Even with a step size of 0.01 d, these two deviations are larger than the RK4 values at 0.024 % and 0.012 %. This confirms that the choice of algorithm is very important when precise results are needed.

B. Escape velocity

Another simple test is to plot the behavior of a planet at its escape velocity. The escape velocity is the velocity at which the planet's total energy is zero [6], or where its kinetic and potential energy are equal. This happens for the Earth when

$$v = \sqrt{\frac{2Gm_\odot}{r}} = 2.4327 \times 10^{-2}\,\text{au/d} \tag{12}$$

in the tangential direction. The results of running the program with this velocity are shown in Figure 2.

C. Earth-Sun-Jupiter system

To look at the stability of the Earth-Sun system, we can add an additional planet. The planet with the largest ability to perturb Earth's orbit is Jupiter, the most massive planet in the solar system at $m_J = 9.5458 \times 10^{-4} m_\odot$ [1].

As a first approximation, we will place Jupiter in a circular orbit at 5.20 au, its mean distance from the Sun [2]. Its initial position vector is $\mathbf{r} = (5.20, 0)$ au and its initial velocity vector is $\mathbf{v} = (0, 7.543 \times 10^{-3})$ au/d.

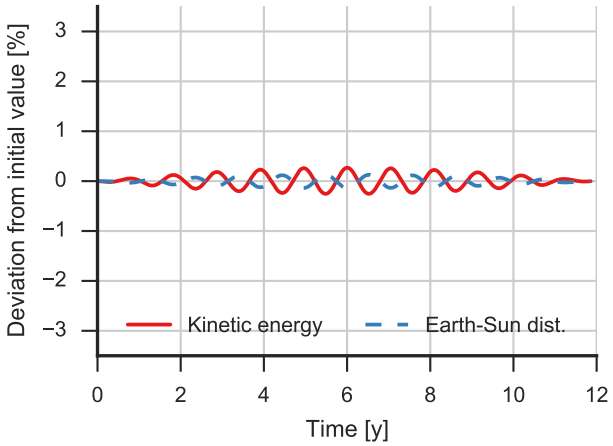
The addition of Jupiter to the system influences the Earth's orbit and deforms it from the expected circular shape. This effect is greater if the mass of Jupiter is increased, as shown in Figure 3. The bottom panel of Figure 3 shows the trajectory the Earth would follow if the mass of Jupiter were $9.5458 \times 10^{-1} m_\odot$, or $1000 m_J$. The orbit in this case is severely deformed, and the trajectory eventually diverges after the Earth passes too close to the origin, causing problems with numerical precision.

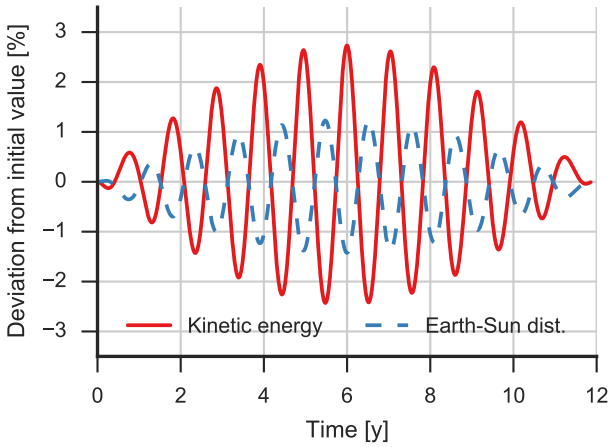
D. Full solar system

Finally, we can consider the full set of eight planets orbiting around the barycenter of the Solar System. The calculation is then extended into three spatial dimensions, and the Sun is no longer fixed at the origin of the coordinate system, but is allowed to move. The trajectories produced by this simulation are shown in Figure 4.

IV. CONCLUSION

Of the three integration methods described in this paper, the RK4 method consistently produced the best results. This is due to its error of $O(h^5)$, which is two orders of magnitude better than the Verlet algorithm and four orders better than Euler's method. That being said, the RK4 method is more computationally expensive than either of the other methods, requiring four calls to the





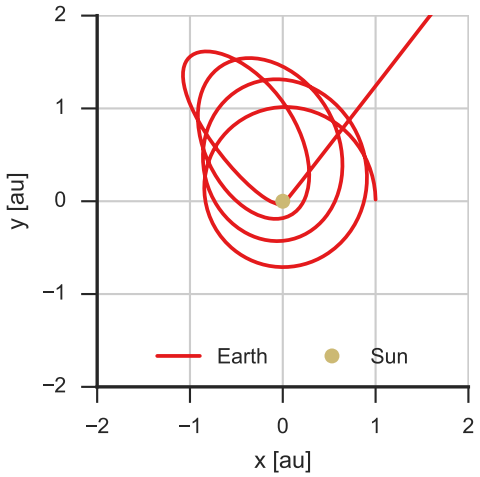


FIG. 3. The effects of Jupiter on the Earth's orbit. Top: deviations in the Earth's kinetic energy and distance from the Sun for a normal-mass Jupiter. Middle: same as top, but with the mass of Jupiter multiplied by 10. Bottom: The trajectory of the Earth if the mass of Jupiter were 1000 times larger than normal. All of these were calculated with RK4 and a step size of 1.0 d.

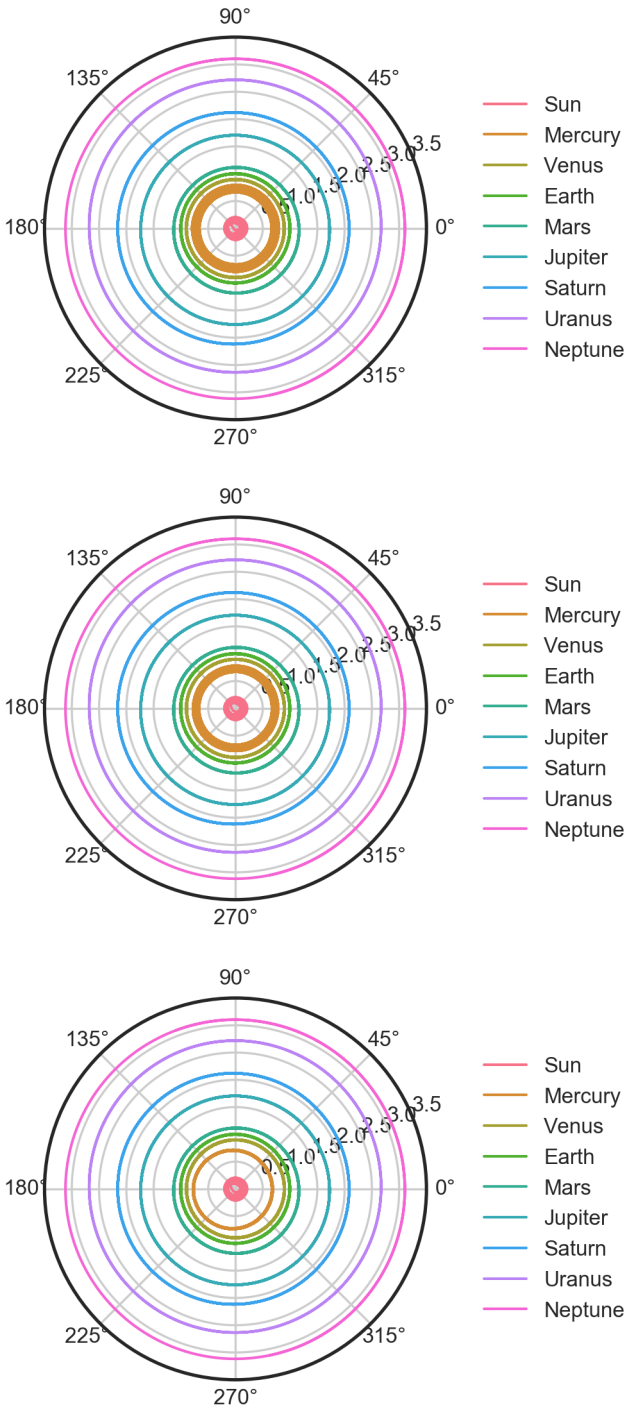


FIG. 4. Orbits of Solar System bodies about the Solar System's barycenter, using the Euler (top), Verlet (middle) and RK4 (bottom) algorithms. The step size was 1.0 d in each case, and the simulation ran for 91554 simulated days, which is longer than the orbital period of all of the planets. [1] The radii are plotted as $\sqrt[3]{r}$ for better visibility of the inner planets. The orbits of the innermost planets (and especially Mercury) are unstable for the Euler and Verlet methods, but more stable for RK4.

function that calculates the acceleration of the body being tracked. This suggests that the Verlet method with a reasonably small step size might be an acceptable choice if calculating the acceleration is particularly expensive. However, if calculating each step is relatively inexpensive, then RK4 appears to be the best choice.

[1] Wolfram—Alpha, "Wolfram Alpha," http://www.wolframalpha.com.

[2] M. Hjorth-Jensen, "Computational Physics MSU," (2016), https://github.com/CompPhysics/ComputationalPhysicsMSU.

[3] J. Stewart, *Calculus: Early Transcendentals*, 6th ed. (Cengage Learning, Belmont, CA, 2008).

[4] J. Bradt, "Comp-phys repository," (2016), https://github.com/jbradt/comp-phys.

[5] "NASA JPL HORIZONS System," http://ssd.jpl.nasa.gov.

[6] H. Goldstein, C. Poole, and J. Safko, *Classical Mechanics*, 3rd ed. (Addison Wesley, San Francisco, CA, 2002).