NUMERICALLY SOLVING THE EQUATIONS OF PLANETARY MOTION WITH AN ADAPTIVE RUNGE-KUTTA METHOD

CALEB FROELICH

Abstract. By utilizing Newton's Law, the system of differential equations that describe planetary trajectories in our solar system is derived. Put into standard form, the problem is then solved numerically using an adaptive Runge-Kutta method implemented from scratch. Relevant mathematical theory underlying the basics of the numerical analysis necessary to implement the model is presented. Comparisons to MATLAB's built-in solver ode45 showed that the models built from scratch outperform MATLAB's models in terms of speed, function evaluations and computational efficiency. Results from this analysis and modeling of complex equations will benefit the development of other challenging numerical analysis applications and problems.

1. Introduction

Throughout the evolution of modern science, men yearned to understand the motion of the planets. This yearning, this driving force, led these men to study the cosmos and the results immensely shaped mathematical and physical thought. Among the notable revolutionary discoveries include the laws of planetary motion by Johannes Kepler in 1600, which describe the motion of planets around the sun. Isaac Newton and his introduction of the laws of motion and gravitation in 1687 confirmed Kepler's laws and described the underlying cause of planetary motion. Using only the laws of motion, the theory of universal gravity, and the calculus that he invented to solve the problem at hand, Newton could derive Kepler's three laws. Kepler discovered the laws of planetary motion, but Newton understood them. [2]

Kepler's observations and Newton's mathematics have long been celebrated by mathematicians and physicists. Solving the equations of planetary motion has been the subject of numerous articles and textbooks. However, in this paper we'll deviate from the classical solution technique and use the power of computing in the form of numerical analysis and a couple adaptive Runge-Kutta methods. In order to accomplish this goal, we first start by deriving the system of differential equations that defines the equations of motion using Newton's theory.

2. Deriving the Equations of Planetary Motion

2.1. Newton's Law of Gravitation.

Newton's great contribution to the advancement of science was his theory of universal gravitation. It states that every point mass attracts every other point mass by a force acting along the line intersecting the two points. That force is proportional to the product of the two masses, and inversely proportional to the square of the distance between them. In mathematical terms, the force between the point masses can be expressed as:

$$F = G\frac{mM}{r^2},$$

where M is the mass of one object, m the mass of the other object, r the distance between the two objects, and G is the gravitational constant, $G \approx 6.67 \times 10^{-11} m^3/(kg \cdot sec^2)$. Using this law, the governing differential equations that describe the motions of the orbits can be easily derived, as long as a couple of approximations are made. First, we assume that the masses of the planets can be approximated as point masses. This is reasonable due to the vast distances between bodies in the solar system.

The second assumption we make is that the force of gravity propagates instantaneously. This is the assumption that Newton inherently made, however, Albert Einstein showed by his theory of relativity that gravity is due to curvature of space-time. Newton's law is still used as an excellent approximation of the effects of gravity in most situations as long as the objects studied are not extremely massive and the velocities are small compared to the speed of light. Even though the planets are quite massive, we'll assume that Newton's law still holds.

2.2. **2-Body System.** With these assumptions in hand, consider two objects of mass m_1 and m_2 located in three-dimensional space as shown in figure 1.

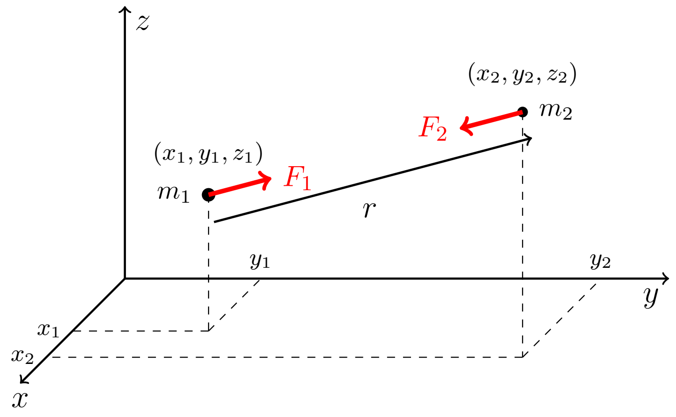


FIGURE 1. Simplified Model of 2 Planets

If the distance between m_1 and m_2 is r, then in accordance with Newton's law, the force acting on each of the masses is

$$\vec{F_1} = G\frac{m_1 m_2}{r^2}(\hat{\mathbf{r}})$$
$$\vec{F_2} = G\frac{m_1 m_2}{r^2}(-\hat{\mathbf{r}}),$$

where $r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$ and $\hat{\mathbf{r}}$ is the unit vector pointing from m_1 to m_2. Considering just mass m_1, the respective components of the force F, are

$$F_{1x} = F_1 \frac{x_2 - x_1}{r}$$
$$F_{1y} = F_1 \frac{y_2 - y_1}{r}$$
$$F_{1z} = F_1 \frac{z_2 - z_1}{r}.$$

In accordance with Newton's second law of dynamics ($F = ma$), we have the following set of differential equations:

$$F_{1x} = m_1 \frac{d^2x}{dt^2}$$
$$F_{1y} = m_1 \frac{d^2y}{dt^2}$$
$$F_{1z} = m_1 \frac{d^2z}{dt^2}$$

Combining the equations, we get

$$\begin{aligned}\frac{d^2x}{dt^2} &= G \cdot m_2 \, \frac{x_2 - x_1}{r^3}\\ \frac{d^2y}{dt^2} &= G \cdot m_2 \, \frac{y_2 - y_1}{r^3}\\ \frac{d^2z}{dt^2} &= G \cdot m_2 \, \frac{z_2 - z_1}{r^3}\end{aligned}$$

Scaling this up to an N-bodied system and indexing the planets by i, we can describe the motion of planet j with mass m_j using the set of following differential equations.

$$\begin{aligned}\frac{d^2x_j}{dt^2} &= \sum_{i=1,\, i\neq j}^{N} G \cdot m_i \, \frac{x_i - x_j}{r_{i,\,j}^3} & \vec{X}(t_0) &= \vec{X}_0, \quad \vec{X}'(t_0) = \vec{X}_0'\\ \frac{d^2y_j}{dt^2} &= \sum_{i=1,\, i\neq j}^{N} G \cdot m_i \, \frac{y_i - y_j}{r_{i,\,j}^3} & \vec{Y}(t_0) &= \vec{Y}_0, \quad \vec{Y}'(t_0) = \vec{Y}_0'\\ \frac{d^2z_j}{dt^2} &= \sum_{i=1,\, i\neq j}^{N} G \cdot m_i \, \frac{z_i - z_j}{r_{i,\,j}^3} & \vec{Z}(t_0) &= \vec{Z}_0, \quad \vec{Z}'(t_0) = \vec{Z}_0'\end{aligned}$$

where $r_{i,j}$ is the distance between planets i and j. Since this is a second order system, two initial conditions must be specified for each ODE. Here $\vec{X}$, $\vec{Y}$, and $\vec{Z}$ are vectors of length N.

2.3. **Initial Value Problem.** Converting the problem to a system of first order ODE's, we end up with the following initial value problem (IVP). For each planet, we have six variables, the positions in the respective coordinates (x, y, and z) and their velocities ($\dot{x}, \dot{y}$, and $\dot{z}$).

$$\vec{Y} = [\, x, y, z, \dot{x}, \dot{y}, \dot{z} \,] = \left[\, x, y, z, \frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt} \,\right]$$

Thus, the system of ODE's, for a single planet, is represented by

$$\frac{d\vec{Y}}{dt} = \frac{d}{dt}\begin{bmatrix} x\\ y\\ z\\ \dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix} \dot{x}\\ \dot{y}\\ \dot{z}\\ \sum_{i=1,\, i\neq j}^{N} Gm_j \, \frac{x_j - x_i}{r_{i,\,j}^3}\\ \sum_{i=1,\, i\neq j}^{N} Gm_j \, \frac{y_j - y_i}{r_{i,\,j}^3}\\ \sum_{i=1,\, i\neq j}^{N} Gm_j \, \frac{z_j - z_i}{r_{i,\,j}^3}\end{bmatrix}$$

These are the equations for only one planet. When we have N planets, $\vec{Y}$ takes the form

$$\begin{aligned}\vec{Y} = [\, & x_1, x_2, \cdots, x_N,\, y_1, y_2, \cdots, y_N,\, z_1, z_2, \cdots, z_N,\\ & \dot{x_1}, \dot{x_2}, \cdots, \dot{x_N},\, \dot{y_1}, \dot{y_2}, \cdots, \dot{y_N},\, \dot{z_1}, \dot{z_2}, \cdots, \dot{z_N} \,].\end{aligned}$$

3. Numerical Approach

3.1. **Basics of Numerically Solving ODEs.** Most numerical solutions for solving ordinary differential equations come in the form of a table of values, giving the value of the independent variable(s) for specific values of the dependent variable(s). One of the simplest numerical methods for solving ODEs is Euler's method.

To illustrate Euler's method, consider the first order ODE $y' = f(t, y)$, with $y(t_0) = y_0$. When solving this using Euler's method, we take a small step of length h in the direction of the derivative of y, forming a tangent line approximation of our unknown function. We can continue the linear extrapolation based on our initial condition and the derivative of y. The extrapolation is accurate for times not too far in the future, but the estimate eventually breaks down as we move further away from our initial condition. The difference between the true solution and the approximation is called the truncation error. The local truncation error is the error induced in one step of the numerical approximation. Note that the smaller the step-size, the smaller the local truncation error.

The Euler method forms a first-order method, which means that the local truncation error, E_{LT}, is proportional to the square of the step size, i.e., $E_{LT} \simeq O(h^2)$. The truncation error, E_T, which comes from the accumulated local truncation errors is proportional to the step size, i.e., $E_T \simeq O(h)$. Higher order methods decrease the truncation error, E_T. For an $O(h^n)$ method, decreasing the step-size h by a factor of two (2) decreases the error by a factor of 2^n.

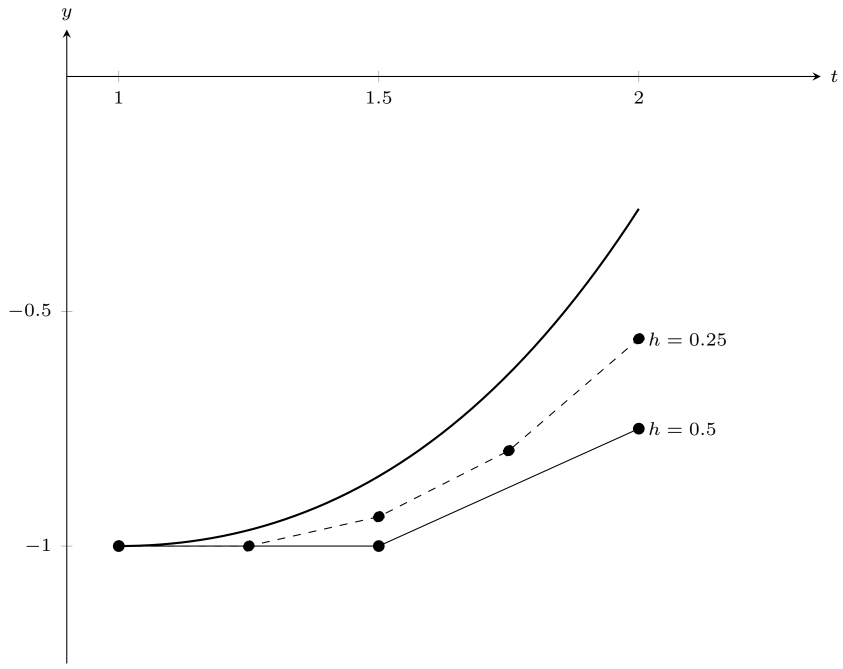


Figure 2. Iteratively solving an ODE using Euler's Method [4]

A good numerical method is required to accurately and quickly calculate the orbits of the solar system. Simple methods (e.g., Euler) need a very small time step to remain stable, thus, a large amount of computing time is necessary. Therefore, a more elaborate method is necessary to increase accuracy and to reduce calculation time.

3.2. The Runge-Kutta Family.

3.2. The Runge-Kutta Family. The Runge-Kutta (RK) family is a more accurate way of approximating the value of the solution at the next time step. The general form of a RK method is shown in equation (3.1).

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i, \tag{3.1}$$

where

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + c_2 h, y_n + h(a_{21} k_1)), \\ k_3 &= f(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2)), \\ &\vdots \\ k_s &= f(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \cdots + a_{s,s-1} k_{s-1})). \end{aligned}$$

Unlike Euler's method, which jumps from solution to solution like a leapfrog, the RK method feels its way forward by evaluating the solution at other points, and after sniffing out the solution, returns again, and only then it makes a full step, using the information gained in those smaller steps.

How these steps are made can be fine-tuned via the coefficients, a_{ij}, b_i, and c_i. The matrix a_{ij} is known as the RK matrix and the b_i's and c_i's are respectively known as weights and nodes. The choice of these parameters is not unique, thus, it is standard practice to organize these values in a *Butcher tableau* which makes it easier to keep track of the values and specify a particular RK method. [1]

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & & \ddots & & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \end{array}$$

3.3. Adaptive Runge-Kutta Methods. From our knowledge of numerical methods we know that;

- Smaller time steps produce smaller errors, but are more computationally expensive.
- Higher order approximations give better numerical solutions, but require more function evaluations.
- The local truncation error varies with a constant step-size, as it depends on the properties of our differential equation and its derivatives.

Thus, a good ODE integrator would demonstrate adaptive control over its progress, frequently making changes in the stepsize, h. The purpose of controlling the stepsize is to achieve a predetermined accuracy in the solution with minimal computation. As described in Press's "Numerical Recipes: The Art of Scientific Computing", an adaptive method allows for *"many small steps to tiptoe through treacherous terrain, but a few great strides to speed through smooth uninteresting countryside."* [6]

Implementation of an adaptive method requires returning an estimate of the local truncation error at each time step. While this adds to the computational burden, the reward is quite worth it as gains in efficiency can sometimes surpass factors of 10, 100 or more. [1] The classical approach is to combine a higher-order method with a lower-order method. Then by subtracting the solution obtained using each method, an estimate of the truncation error can be obtained.

Consider the Runge-Kutta-Fehlberg (RKF) method, which combines a 4th order RK method (3.2), with a 5th order RK method (3.3). [3]

$$y(t+h) = y(t) + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5 \tag{3.2}$$
$$\hat{y}(t+h) = y(t) + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6 \tag{3.3}$$

where

$$\begin{aligned}
k_1 &= hf(t,x)\\
k_2 &= hf\left(t+\frac{1}{4}h, x+\frac{1}{4}k_1\right)\\
k_3 &= hf\left(t+\frac{3}{8}h, x+\frac{3}{32}k_1+\frac{9}{32}k_2\right)\\
k_4 &= hf\left(t+\frac{12}{13}h, x+\frac{1932}{2197}k_1-\frac{7200}{2197}k_2+\frac{7296}{2197}k_3\right)\\
k_5 &= hf\left(t+h, x+\frac{439}{216}k_1-8k_2+\frac{3680}{513}k_3-\frac{845}{4104}k_4\right)\\
k_6 &= hf\left(t+\frac{1}{2}h, x-\frac{8}{27}k_1+2k_2-\frac{3544}{2565}k_3-\frac{1859}{4104}k_4-\frac{11}{40}k_5\right)
\end{aligned}$$

By using two methods, we can obtain an estimate of the local truncation error:

$$|\hat{y}(t+h) - y(t+h)| = \left|\frac{1}{360}k_1 - \frac{128}{4275}k_3 - \frac{2197}{75240}k_4 + \frac{1}{50}k_5 + \frac{2}{55}k_6\right| \tag{3.4}$$

Notice that nodes of both methods are the same, we have just added one more row in the RK matrix and changed the weights. This greatly improves performance as the function evaluations for the two methods, with the exception of the added term (k_6) in the 5th order RK method, are the same. This can be visualized using the following Butcher Tableau.

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & & \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & \\
\hline
 & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55}\\
 & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5} & 0
\end{array}$$

The Runge-Kutta-Felhberg method is a straightforward and fairly popular method used. Due to it's simplicity and the availability of resource material, we chose to build an adaptive RK model using RKF45 from scratch.

4. Numerical Model

4.1. **Model Verification using ODE45.** Before writing and implementing our own adaptive numerical solver, we needed to verify the IVP model of the solar system. We used Matlab's built in "ode45" solver which is based around a different combination of a 4th and 5th order Runge-Kutta, the Dormand–Prince method. Simulating the planetary orbits over 165 earth years (or one orbital period of Neptune) with an error tolerance of 1E-12 gave good results; however, the computational demand of 101.84s and $2,808,703$ function evaluations left plenty of room for improvement. Figure (3) shows a plot of the solution.

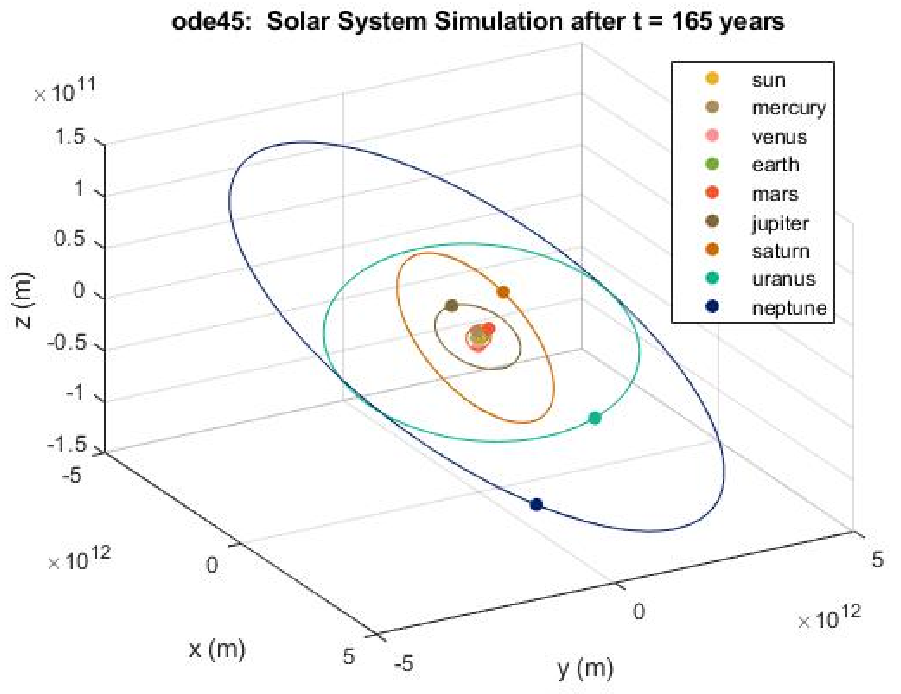
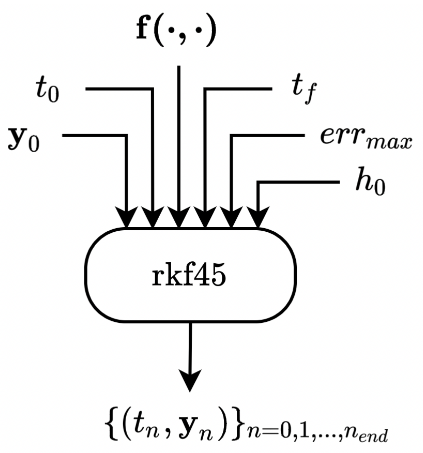


Figure 3. Planetary Orbit Simulation using ODE45 across 165 Earth Years

4.2. **Model Construction.** The structure of our rkf45 model is seen in the following block diagram. The inputs and outputs are similar to MATLAB's "ode45" function in order to allow for comparison between the models, without having to restructure the IVP.


$\mathbf{f}(\cdot,\cdot)$
t_0
t_f
$\mathbf{y}_0$
err_{max}
h_0
rkf45
$\{(t_n, \mathbf{y}_n)\}_{n=0,1,\ldots,n_{end}}$

The inputs are the function $\mathbf{f}(\cdot,\cdot)$, the starting point t_0, the initial value $\mathbf{y}_0$, intial step-size h_0, the end-point t_f, and the error tolerance $err_{max} > 0$ so that the numerical error is within the Euclidean norm of err_{max}. The output is the computed solution sequence at the points $t_0 < t_1 < \cdots < t_f$, which, of course, are not equispaced.

The basic flow of our implementation of RKF's method can be seen in the pseudocode below.

```
Given, $\frac{dY}{dt}, [t_0, t_f], Y_0, h_0, err_{max}$ :
choose $n_{max}, n_{attempts}$
choose MIN_STEP_SCALE, MAX_STEP_SCALE
set $h = h_0, t_i = t_0, k = 0, Y_i = Y_0,$ scale $= 1$
while $(k < n_{max}$ and $t_i < t_f)$ :
    for $j = 1 : n_{attempts}$
        compute RKF4, RKF5, and $err = |\text{RKF4} - \text{RKF5}|$
        compute scale factor based on $err$ and $err_{max}$
        $note : MIN\_STEP\_SCALE < scale < MAX\_STEP\_SCALE$
        if err $\approx 0$ :
            break, set $Y_i =$ RKF5, scale = MAX_STEP_SCALE
            else if $err < err_{max} \cdot \text{norm}(Y_i)$ :
                break, set $Y_i =$ RKF5, use computed scale factor
            else:
                set $h = \text{scale} \cdot h$ and try again
        end for
        save $Y_i$ as a column in solution $Y,$ and $t_i$ to $t$.
        set $h = \text{scale} \cdot h$ and perform next iteration
```

With each iteration, we compute the approximate solutions using the 4th and 5th order RK coefficients. These solutions are utilized to calculate the approximate error. Then by using our estimate of the local truncation error and the desired error tolerance, err_{max}, we compute a new scale factor that determines whether we should increase or decrease our previous step-size. We update our scale, step-size, and solution accordingly.

To determine the scale factor, we use equation (4.1).

$$\text{scale} = c \left| \frac{\epsilon \cdot \text{norm}(Y_i)}{\delta(t_f - t_0)} \right|^k \tag{4.1}$$

where

$\epsilon =$ desired error tolerance.
$\delta =$ estimate of the error using the two methods.
$c =$ error coefficient.
$k =$ error exponent.

The scale is strongly influenced by the the desired error tolerance, ϵ, and our estimate of the error, δ. Since δ is in the denominator, as our estimated error decreases, we take larger steps. The error tolerance, ϵ, is in the numerator, thus, with a smaller error tolerance, we will shorten the step-size to ensure that we can meet the intended accuracy. The error coefficient, c, and the error exponent, k, can be fine-tuned to achieve optimal performance. [5]

4.3. **Model Results.** Our numerical solver (rkf45), coded on MATLAB, implemented the RKF method described in the numerical approach section of this paper. A simulation of 165 Earth years (or one orbital period of Neptune) for all eight planets of the solar system is shown in figure (4). Figure (5) shows the inner four (4) planets of the solar system simulated across three (3) Earth years (one orbital period of Mars).

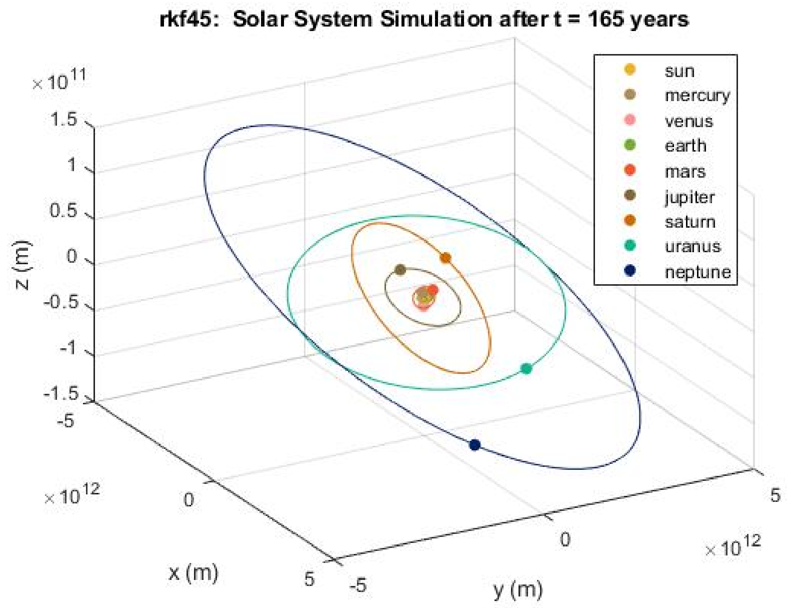


FIGURE 4. Planetary Orbit Simulation using RKF45 across 165 Earth Years

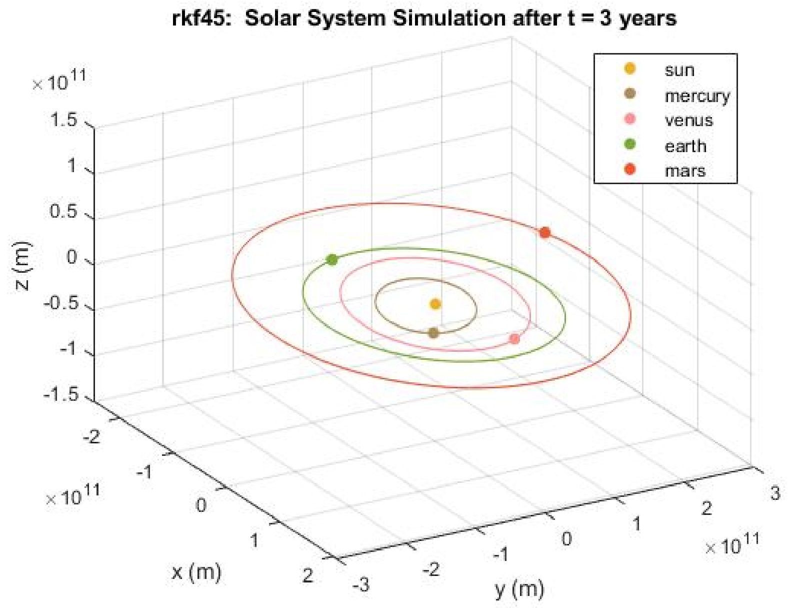


FIGURE 5. Planetary Orbit Simulation using RKF45 across 3 Earth Years

Overall, our rkf45 model seems to be slightly outperforming MATLAB's ode45 model using the Dormand-Prince method. By setting the error tolerance of the models to the same value (1E-12), the time, number of iterations, and function evaluations were recorded for ten (10) iterations for simulating 165 earth years of orbit (see table 1). The rkf45 function described above takes less than half the time of MATLAB's ode45, yet still takes around the same number of steps to meet the error tolerance. For comparison, the solution as obtained using the ode113 method is also listed in (table 1). MATLAB's ode113 significantly outperforms both rkf45 and ode45, as expected since it is a much higher order method (up to the 13th order).

5. Higher Order Methods

5.1. **Verner's Methods.** The method presented above, along with some of the other Runge–Kutta pairs presented in Fehlberg's original paper, have recently been criticized for a lack of computational robustness, especially when the differential equation being solved is approximately equal to a quadrature problem. In addition, Fehlberg's methods were intended to utilize the lower order method to produce output values and use the higher order method to collect error estimates. This is incredibly inefficient and as many numerical analysts argue "*it is wasteful to propagate a low order approximation when a higher order approximation is available.*" [1]

While not as well-known, the methods of Verner overcome the issue of computational robustness inherent in many of the Fehlberg methods. Additionally, Verner suggests that the higher-order method be utilized as an the approximate output in order to minimize the truncation error. The Butcher tableau for Verner's 5-6 order method is shown below.

$$\begin{array}{c|cccccccc}
0 & & & & & & & & \\
\frac{1}{18} & \frac{1}{18} & & & & & & & \\
\frac{1}{6} & -\frac{1}{12} & \frac{1}{4} & & & & & & \\
\frac{2}{9} & -\frac{2}{81} & \frac{4}{27} & \frac{8}{81} & & & & & \\
\frac{2}{3} & \frac{40}{33} & -\frac{4}{11} & -\frac{56}{11} & \frac{54}{11} & & & & \\
1 & -\frac{369}{73} & \frac{72}{73} & \frac{5380}{219} & -\frac{12285}{584} & \frac{2695}{1752} & & & \\
\frac{8}{9} & -\frac{8716}{891} & \frac{656}{297} & \frac{39520}{891} & -\frac{416}{11} & \frac{52}{27} & 0 & & \\
1 & \frac{3015}{256} & -\frac{9}{4} & -\frac{4219}{78} & \frac{5985}{125} & -\frac{539}{384} & 0 & \frac{693}{3328} & \\
\hline
 & \frac{3}{80} & 0 & \frac{4}{25} & \frac{243}{1120} & \frac{77}{160} & \frac{73}{700} & 0 & 0 \\
 & \frac{57}{640} & 0 & -\frac{16}{65} & \frac{1377}{2240} & \frac{121}{320} & 0 & \frac{891}{8320} & \frac{2}{35}
\end{array}$$

Using similar pseudocode as that presented in section (4.2), we construct a model utilizing Verner's coefficients. The 6th order approximation is propagated and the difference between approximations is used to form an estimate of the error.

The results from utilizing Verner's coefficients showed a slight improvement over the Runge-Kutta-Fehlberg coefficients. All three basic statistics (time, steps, and function evaluations) that were used to determine model performance improved relative to the rkf45 adaptive model.

Model Comparison

model	time (s)	steps	feval
MATLAB's ode45 model	98.5700	467602	2808703
MATLAB's ode113 model	10.8890	115757	233290
rkf45 adaptive model	36.7779	177413	1064472
vnr56 adaptive model	34.6094	124316	994520

Table 1. Model Performance Comparison

6. Conclusions

Combining Newton's theory and the power of Runge-Kutta numerical integration, the equations of planetary motion were solved using several methods. The constructed models streamlined the design when compared to commercial solutions available on MATLAB. The outcome of this modeling delivers more efficient code specifically designed to solve the equations of planetary motion. The results of the methods indicate potential as the time, steps, and function evaluations were reduced while still satisfying our error criterion.

Despite the success of our initial models, future research would carefully analyze the key differences between each of the methods presented in this paper. This research would delve into determining the stability of each of the methods and further analyze the differences between the methods by solving well-know test problems (like the Chemakzo problem) and evaluating performance.

The dynamical properties of the solar system are the focus of much applied research and of continuing interest to a variety of scientific disciples. As computing and numerical tools advance, it is critical that we push the limits of this type of modeling. The work presented in this paper transcends the basic modeling of the planetary orbits. The ability to develop technology to solve complex, challenging problems, like the equations of planetary motion and the techniques derived, set the stage for other numerical analysis applications and problems.

Acknowledgements

The author would like to thank Dr. Ross Magi for his wonderful guidance throughout this project, help with editing, and for the *LaTeX* advice and code that he provided.

References

[1] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. 3rd edition, 2016.

[2] Wu-Yi Hsiang and Elder Straume. Revisiting the mathematical synthesis of the laws of Kepler and Galileo leading to Newton's law of universal gravitation. August 2014.

[3] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, 2nd edition, 2008.

[4] Ross Magi. fig_20_01_euler2.tex, 2020.

[5] John Mathews and Kurtis Fink. *Numerical Methods Using Matlab*. Prentice-Hall Inc., 4 edition, 2004.

[6] William Press and Saul Teukolsy. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, January 2007.

Department of Mathematics, Walla Walla University, College Place, WA 99324