THE EQUATIONS OF PLANETARY MOTION AND THEIR NUMERICAL SOLUTION

Jonathan Njeunje, Dinuka Sewwandi de Silva

January 27, 2021

Abstract

Each day we ask ourselves questions about the big universe and how this great "mechanics" function in such an incredible stability over the centuries. Some great minds of Earth's history, such as Isaac Newton worked on the theory of gravitation presented in the Principia, this stood to be a major contribution in answering this question. By utilizing the theory we will focus in setting up the differential equations that describe planetary trajectories in our solar system, linearizing these equations and providing their solution with the help of numerical method implemented from scratch.

1 Introduction

In our solar system, all the planets have an elliptic trajectory around the sun and the sun is also non-static and describes a motion about a reference origin, like portrayed on the figure below.

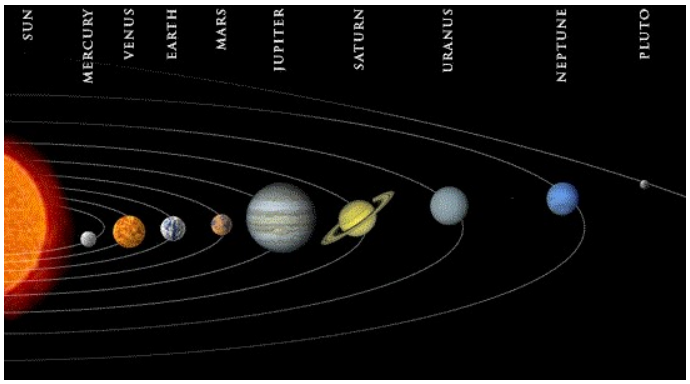


Figure 1: Solar System

Those elliptical motions (orbits) can best be figured out by understanding the general equation of planetary motion in the Cartesian co-ordinates system. Let us consider the following figure to best picture our scenario.

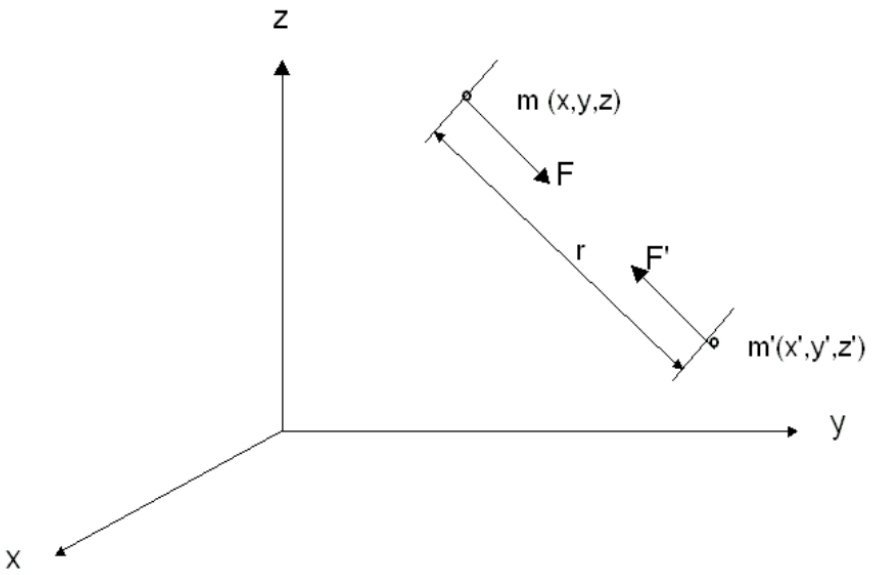


Figure 2: Representation of two bodies under gravitational attraction in Cartesian co-ordinates

In this project we are going to derive the equations of planetary motion based on the assumption that the masses of the planets can be approximated to point masses. This is reasonable cue to the vast distances between bodies in the solar system.

Then, according to the *Newton's law of gravitational force* acting on each of the masses we obtain the following.

$$\mathbf{F} = G\frac{mm'}{r^2}(\hat{\mathbf{r}})$$

$$\mathbf{F'} = G\frac{mm'}{r^2}(-\hat{\mathbf{r}})$$

The respective components F_x, F_y and F_z of the force $\mathbf{F}$, are:

$$F_x = F\frac{x' - x}{r};$$

$$F_y = F\frac{y' - y}{r};$$

$$F_z = F\frac{z' - z}{r}.$$

Furthermore, following *Newton's second law of dynamics* we obtain:

$$F_x = m\frac{d^2x}{dt^2};$$

$$F_y = m\frac{d^2y}{dt^2};$$

$$F_z = m\frac{d^2z}{dt^2}.$$

Combining both set of equations, we finally get:

$$\frac{d^2x}{dt^2} = Gm' \frac{x' - x}{r^3};$$
$$\frac{d^2y}{dt^2} = Gm' \frac{y' - y}{r^3};$$
$$\frac{d^2z}{dt^2} = Gm' \frac{z' - z}{r^3}.$$

Where $\mathbf{r}$, is given by: $$r = \sqrt{(x' - x)^2 + (y' - y)^2 + (z' - z)^2} \tag{1}$$

Where G is a gravitational force and m and M are masses of the given planet and Sun respectively, r is the distance between the planet and the Sun and F is the force.

2 IVP - Initial Value Problem

By using the general second order ordinary differential equation system of planetary motion we will now be able to discuss the dynamics of such a motion for all the planets in our solar system including the Sun. In order to reach this realization we designed a global system of equations, accounting for all the interactions of j bodies of masses $m_1, m_2..., m_j$ on a given body, α of mass m_α.

Then our system of equation is:

$$\frac{d^2x_\alpha}{dt^2} = \sum_{j=1;j\neq\alpha}^{nb} Gm_j \frac{x_j - x_\alpha}{(r_{j,\alpha})^3}$$

$$\frac{d^2y_\alpha}{dt^2} = \sum_{j=1;j\neq\alpha}^{nb} Gm_j \frac{y_j - y_\alpha}{(r_{j,\alpha})^3}$$

$$\frac{d^2z_\alpha}{dt^2} = \sum_{j=1;j\neq\alpha}^{nb} Gm_j \frac{z_j - z_\alpha}{(r_{j,\alpha})^3}$$

$$r_{j,\alpha} = \sqrt{(x_j - x_\alpha)^2 + (y_j - y_\alpha)^2 + (z_j - z_\alpha)^2}$$

where,
nb - Number of bodies which we consider
G - Constant of universal gravitation
m_j - Mass of body j
$r_{j,\alpha}$ - Distance between body j and body α
$x_\alpha, y_\alpha, z_\alpha$ - Cartesian coordinates of body α
x_j, y_j, z_j - Cartesian coordinates of body j

We can convert this system of equations into a standard initial value problem in the following way:

$$Y = \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \frac{dx}{dt} \\ \frac{dy}{dt} \\ \frac{dz}{dt} \end{bmatrix}$$

thus,

$$\frac{dY}{dt} = F(t,Y) \Rightarrow \frac{d}{dt}\begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \sum_{j=1;j\neq\alpha}^{nb} Gm_j \frac{x_j - x_\alpha}{(r_{j,\alpha})^3} \\ \sum_{j=1;j\neq\alpha}^{nb} Gm_j \frac{y_j - y_\alpha}{(r_{j,\alpha})^3} \\ \sum_{j=1;j\neq\alpha}^{nb} Gm_j \frac{z_j - z_\alpha}{(r_{j,\alpha})^3} \end{bmatrix}$$

where,
$G = 6.67E^{-11}m^3kg^1s^{-2}$
$nb :=$ number of bodies $= 10$

After converting our 2^{nd} order system of ODE's to a 1^{st} order system of ODE's, system of equations, we needed a set of initial values for our subsequent ODE Solver. This set of initial states is obtained from the JPL (Jet Propulsion Lab) ephemeris database using the HORIZION web interface. This comprised of the initial states positions (both x y z co-ordinates) and velocities (both x y z co-ordinates), masses and mean radius of all 10 major bodies involved in our solar system.

The following table summarizes this initial states corresponding to the solar system configuration on April 6th, 2018:

Table 1: Initial States from JPL HORIZONS System

		POSITION (m)			**VELOCITY (m/sec)**			**(kg)**	**(m)**
#	**BODY**	**PX**	**PY**	**PZ**	**VX**	**VY**	**VZ**	**MASS**	**RADIUS**
1	SUN	1.81899E+08	9.83630E+08	-1.58778E+07	-1.12474E+01	7.54876E+00	2.68723E-01	1.98854E+30	6.95500E+08
2	MERCURY	-5.67576E+10	-2.73592E+10	2.89173E+09	1.16497E+04	-4.14793E+04	-4.45952E+03	3.30200E+23	2.44000E+06
3	VENUS	4.28480E+10	1.00073E+11	-1.11872E+09	-3.22930E+04	1.36960E+04	2.05091E+03	4.86850E+24	6.05180E+06
4	EARTH	-1.43778E+11	-4.00067E+10	-1.38875E+07	7.65151E+03	-2.87514E+04	2.08354E+00	5.97219E+24	6.37101E+06
5	MARS	-1.14746E+11	-1.96294E+11	-1.32908E+09	2.18369E+04	-1.01132E+04	-7.47957E+02	6.41850E+23	3.38990E+06
6	JUPITER	-5.66899E+11	-5.77495E+11	1.50755E+10	9.16793E+03	-8.53244E+03	-1.69767E+02	1.89813E+27	6.99110E+07
7	SATURN	8.20513E+10	-1.50241E+12	2.28565E+10	9.11312E+03	4.96372E+02	-3.71643E+02	5.68319E+26	5.82320E+07
8	URANUS	2.62506E+12	1.40273E+12	-2.87982E+10	-3.25937E+03	5.68878E+03	6.32569E+01	8.68103E+25	2.53620E+07
9	NEPTUNE	4.30300E+12	-1.24223E+12	-7.35857E+10	1.47132E+03	5.25363E+03	-1.42701E+02	1.02410E+26	2.46240E+07
10	PLUTO	1.65554E+12	-4.73503E+12	2.77962E+10	5.24541E+03	6.38510E+02	-1.60709E+03	1.30700E+22	1.19500E+06

3 Numerical method description

Before diving into writing and implementing our own numerical method of ODE Solver we needed to verify our designed IVP model of the solar system by existing and certified ODE solvers such as "ode45" and "ode113". These two solvers amongst many others where chosen for particular reasons we will discuss in the next section.

After simulating our IVP with the above solvers, we obtained positive results (discussed in the next section) validating our designed IVP model for the solar system.

The next step was to implement from scratch our own ODE solver and apply the designed IVP to it. We made the choice of implementing an **Explicit Runge-Kutta method (ERK)**. But before we could used this numerical method, we first needed to define our RK-stages (ν), RK-nodes (c_i), RK-weights (b_i) and RK-matrix ($A = [a_{ij}]$). And, verify its order of convergence, exactness and stability.

The general equations of an ERK is:

$$y_{n+1} = y_n + h \sum_{j=1}^{\nu} b_j f(t_n + c_j h, \xi_j)$$

where

$\xi_1 = y_n$
$\xi_2 = y_n + h a_{2,1} f(t_n, \xi_1)$
$\xi_3 = y_n + h a_{3,1} f(t_n, \xi_1) + h a_{3,2} f(t_n + c_2 h, \xi_2)$
$\vdots$
$\xi_i = y_n + h \sum_{j=1}^{i-1} a_{i,j} f(t_n + c_j h, \xi_j), \quad i = 1, \ldots, \nu$

and h is the step size of our time span. According to our designed IVP, ξ_i will be a vector of $6 * nb = 6 * 10 = 60$ elements. Similarly, y_{n+1} also will be a vector of 60 elements.

3.1 Definition of our RK method

The chosen ERK for our implementation has the following parameters:

$$\nu = 4$$
$$c = \begin{bmatrix} 0 & .5 & .5 & 1 \end{bmatrix}$$
$$b = \begin{bmatrix} 1/6 & 1/3 & 1/3 & 1/6 \end{bmatrix}$$
$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ .5 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This method is identified as an Explicit Runge-Kutta method due to its lower triangular matrix, A. Additionally, to the above definition, the ξ_i's are as follows:

$\xi_1 = y_n$
$\xi_2 = y_n + .5hf(t_n, \xi_1) = y_n + .5hf(t_n, y_n)$
$\xi_3 = y_n + .5hf(t_n + c_2 h, \xi_2) = y_n + .5hf\Big(t_n + .5h, y_n + .5hf(t_n, y_n)\Big)$
$\xi_4 = y_n + hf(t_n + c_3 h, \xi_3) = y_n + hf\Big(t_n + .5h, y_n + .5hf\Big(t_n + .5h, y_n + .5hf(t_n, y_n)\Big)\Big)$

By applying these ξ_i's on the general ERK formula, we can predict order of convergence of this method.

3.2 Order of convergence of the chosen RK

The chosen ERK method defined in the previous subsection is of stage, $\nu = 4$ and thus of order 4. This claim can further be verified by applying the ERK on the following IVP; applied under different values of the step size and interpreting the resultant graphs.

IVP: $y'(t) = -y;\ \ y_0 = 1$ when $t = 0$

The ERK will by applied with step sizes: $h = .1/2^k,$ with $k = 1, 2, 3, 4.$

The exact solution for this IVP is known to be: $y(t) = e^{-t}$

After running the written MatLab codes for this text, the following graphs were obtained:

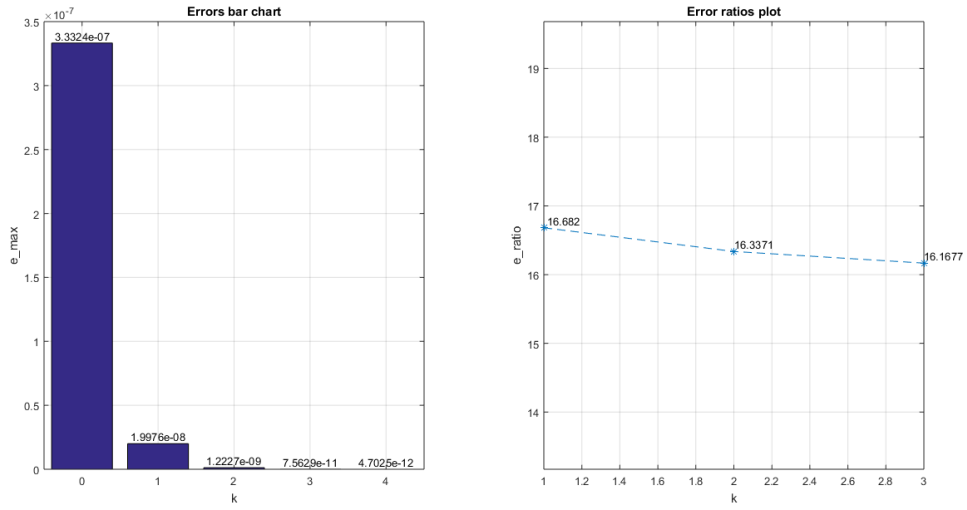


Figure 3: Error/Error-ratio test

By inspecting the error-ratio plot on the right of the above figure we observe a trend:

$$\text{error-ratio} \to 16 = 2^4 = 2^p$$

This interpretation let us conclude that the order of convergence, $p = 4$.

According to the definition of exactness, a given method is exact for the polynomials of degree less than or equal to d , where d is the degree of the exactness and usually it is equal to the order of the method. Therefore, the chosen ERK method is exact for the polynomials of degree less than or equal to 4. Because, 4 is the order of convergence of this ERK method.

Additionally, it can be interpreted from the errors bar chart that the ERK method of oder 4 is exact, as the error is in a close neighborhood of zero.

3.3 verification of stability of the chosen RK

The stability of RK method can be define according to the general equation of RK and the following equations. Then apply RK method for the following IVP.

$$y' = \lambda y$$

$$y(t_0) = y_0$$

Now consider the equations for the ξ's. In general its given by:

$$\xi_k = y_n + h\lambda \sum_{i=1}^{k} a_{k,i} f(t_n + c_i h, \xi_i)$$

In this equation, $\sum_{i=1}^{k} a_{k,i} f(t_n + c_i h, \xi_i)$ is a dot product of the k^{th} row of the matrix A and a vector of ξ. Let's define some vectors as follows,

$$\xi = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_\nu \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_\nu \end{bmatrix}, \quad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Then, the system of equations for ξ's can be represented as;

$$\xi = y_n.\mathbf{1} + h\lambda A\xi \Rightarrow \xi = (I - h\lambda A)^{-1}.\mathbf{1}.y_n$$

By using this ξ in the general equation of RK, we can be obtain,

$$y_n = (1 + zb^T(I - zA)^{-1}\mathbf{1})^n y_0 \quad where \;\; z = h\lambda$$

Then the stability domain of RK can be find when $|r(z)| < 1$ where $r(z) = 1 + zb^T(I - zA)^{-1}\mathbf{1}$.

By using above condition, can be find the stability domain of the chosen ERK, which is mentioned in subsection (3.1). Therefore,

$$I - zA = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -.5z & 1 & 0 & 0 \\ 0 & -.5z & 1 & 0 \\ 0 & 0 & -z & 1 \end{bmatrix} \text{ and } (I - zA)^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ .5z & 1 & 0 & 0 \\ .25z^2 & .5z & 1 & 0 \\ .25z^3 & .5z^2 & z & 1 \end{bmatrix}$$

$$b^T(I - zA)^{-1}\mathbf{1} = \begin{bmatrix} 1/6 & 1/3 & 1/3 & 1/6 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ .5z & 1 & 0 & 0 \\ .25z^2 & .5z & 1 & 0 \\ .25z^3 & .5z^2 & z & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = 1 + \frac{z}{2} + \frac{z^2}{6} + \frac{z^3}{24}$$

$$\Rightarrow r(z) = 1 + z\Big(1 + \frac{z}{2} + \frac{z^2}{6} + \frac{z^3}{24}\Big) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}$$

By using Mathematica codes for $r(z)$, the following stability (shaded) region was obtained:

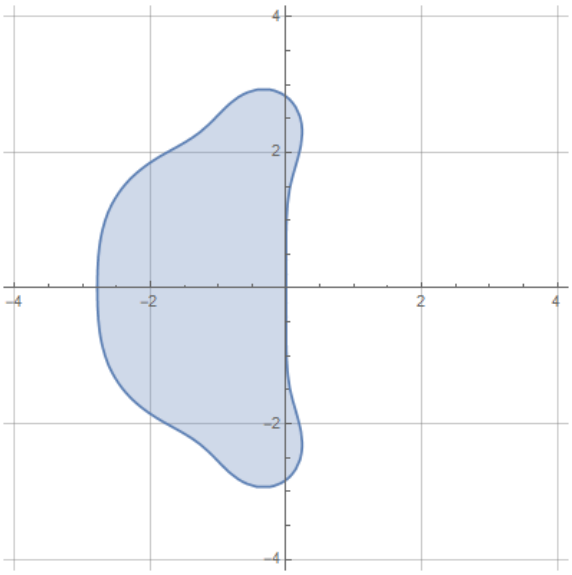


Figure 4: Stability domain

According to this stability region it can be concluded that, the chosen ERK method is not A-stable. Because, for a method to be A-stable it must include $\mathbb{C}^-$ in its stability region. Despite the fact that an A-stability wasn't reached, necessary stability domain could be achieved. This achieved stability will be useful the next section.

4 Numerical Results and Discussion

After a repeated set of simulations we made some adjustments regarding the number of planets considered and the step size to use with the different ODE Solvers.

The adjustment made on the number of planet consider, dealt with the non-consideration of planet Pluto. This adjustment was made in order to reduce the computational intensity of our code and enhanced faster calculations.

The second adjustment made on the step size, mainly had to constrain the step size to a value of 1. This adjustment was done to obtain better stability with our set of non-stiff ODE Solvers. Mainly, due to the fact that our IVP is a Stiff IVP, and, thereby necessitated (for more appropriate circumstances) a stiff ODE Solver with an A-stability region.

Nonetheless, with the appropriate adjustments, the following satisfactory results were obtained and interpreted as follows:

4.1 *Build-in ode45*

The first ODE used was the ode45. This build-in MatLab ODE Solver was chosen to test the IVP model and make sure it is designed correctly. The result obtained, Fig 5, showed great instability of this ODE Solver due to the stiffness of the IVP. And, the added adjustment couldn't lead to better results.

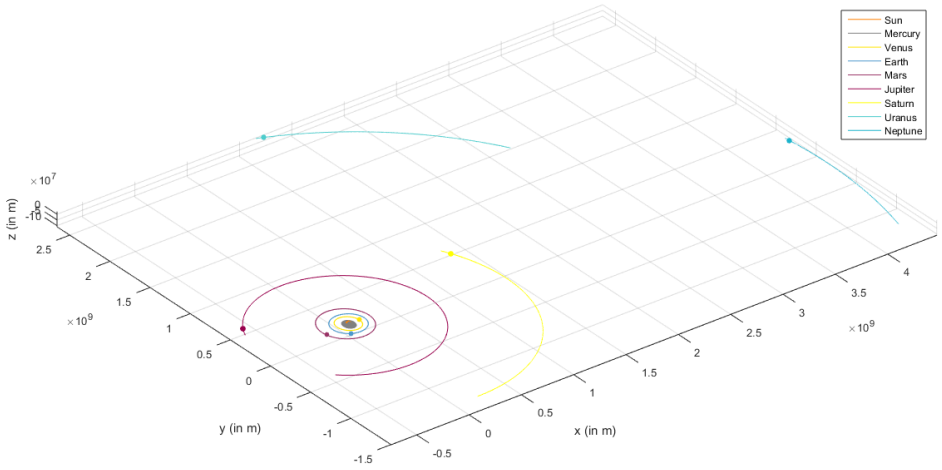


Figure 5: Solar System Simulation - ode45

In the next figure, Fig 6, we can observe the instability as the parameters of Mercury tend to decay much faster than that of the other planets. This behavior generated an error caused by the lack of better accuracy.

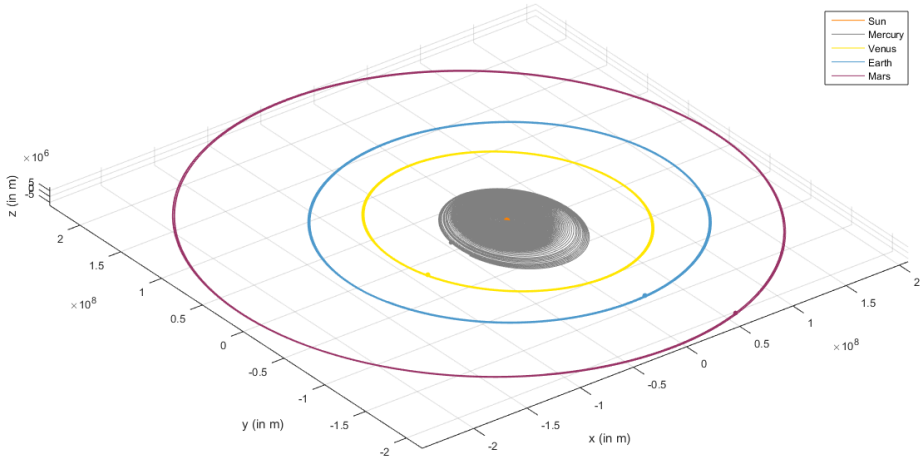
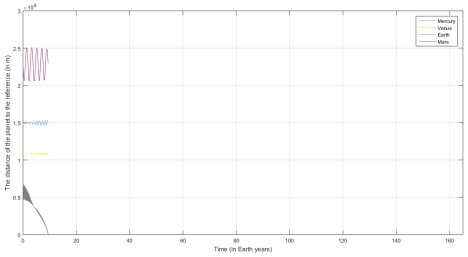
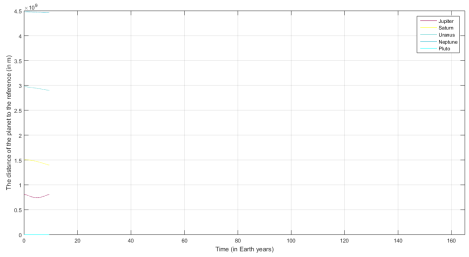


Figure 6: Solar System Simulation - ode45 - Inner planets

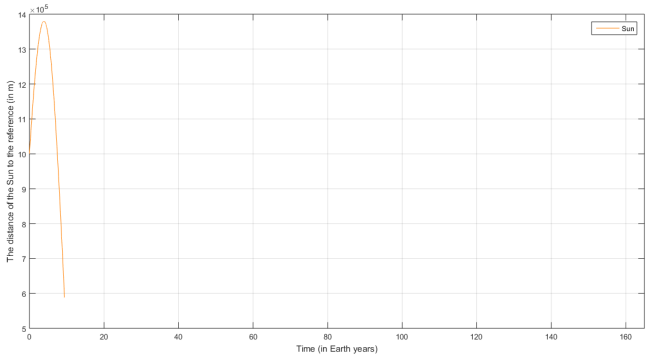
This ODE Solver couldn't complete the solution and crashed. The following figures show the plot of the distance from the different bodies to the reference point which in this case is not the sun but the origin.



(a) Distance from origin - ode45 - Inner planet



(b) Distance from origin - ode45 - Outer planet



(c) Distance from origin - ode45 - Sun

Figure 7: Distance from origin

4.2 *Build-in ode113*

By observing from the previous method that the modeled IVP had stringent error tolerances, it was necessary to retry the test of the IVP with higher accuracy ODE Solver as ode113. The ode113 is also a MatLab build-in Solver. The figure, Fig 8, depicts the result obtained.

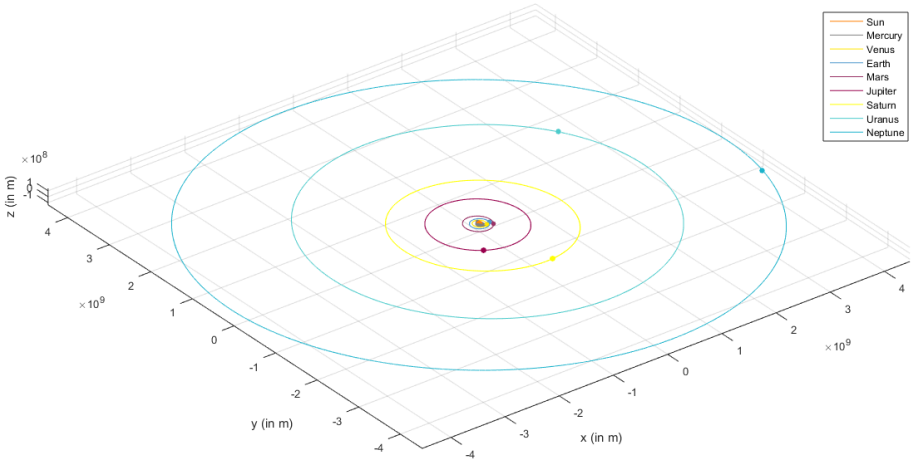


Figure 8: Solar System Simulation - ode113

This result had better stability and calculated a complete solution for a time span corresponding to the period of revolution of the planet Neptune (165 earth years). The result lead to a validation of the IVP model.

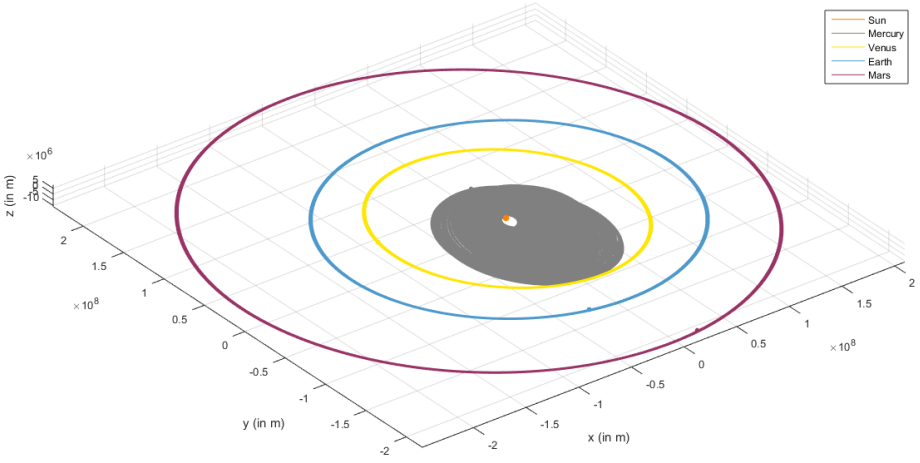
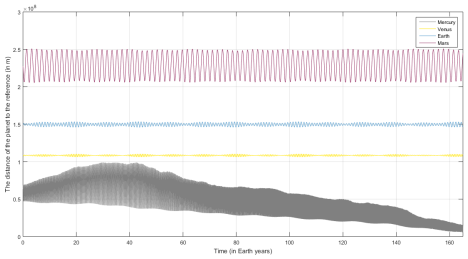
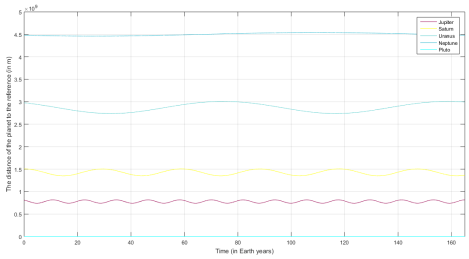


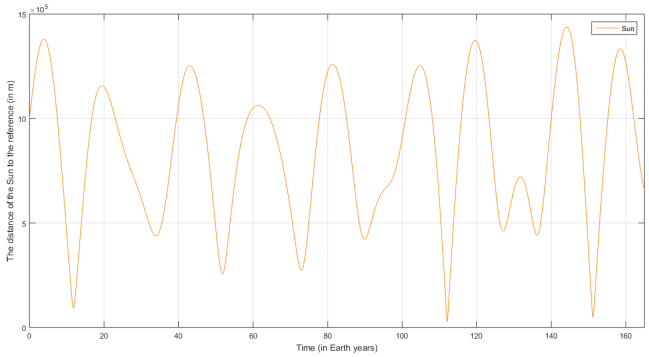
Figure 9: Solar System Simulation - ode113 - Inner planets



(a) Distance from origin - ode113 - Inner planet



(b) Distance from origin - ode113 - Outer planet



(c) Distance from origin - ode113 - Sun

Figure 10: Distance from origin

4.3 *Implemented ERK of order 4, ode652*

The final step was to build from scratch an ODE Solver to attain better (or similar) results as those obtained with ode113.

The ODE Solver, ode652, coded on MatLab implemented the ERK of order 4 earlier described in the Numerical method section of this paper. This implementation achieved better stability compared to ode113 and ode45. A complete simulation of a time span equivalent to the period of the planet Neptune was obtained. Fig 11, shows this result and Fig 12, shows a bigger scale of the inner planets of the solar system.

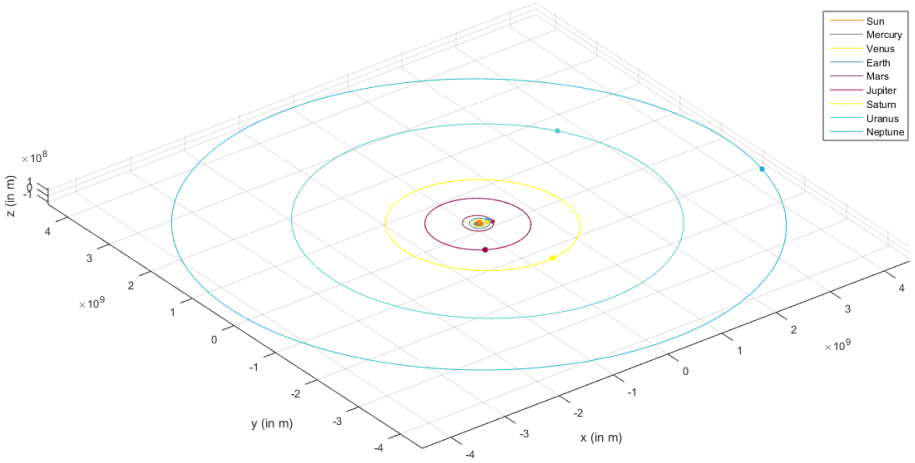


Figure 11: Solar System Simulation - ode652

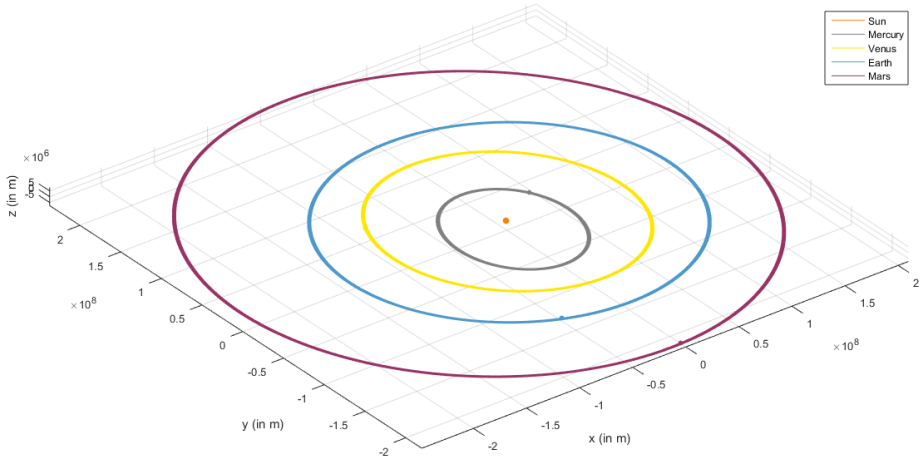
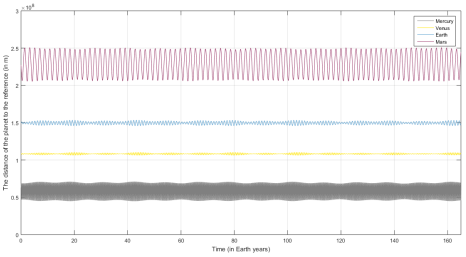
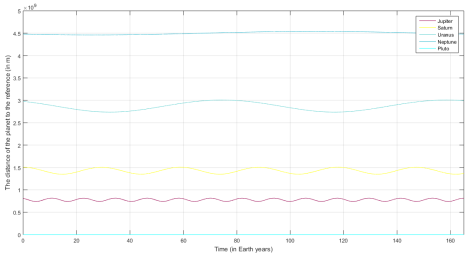


Figure 12: Solar System Simulation - ode652 - Inner planets

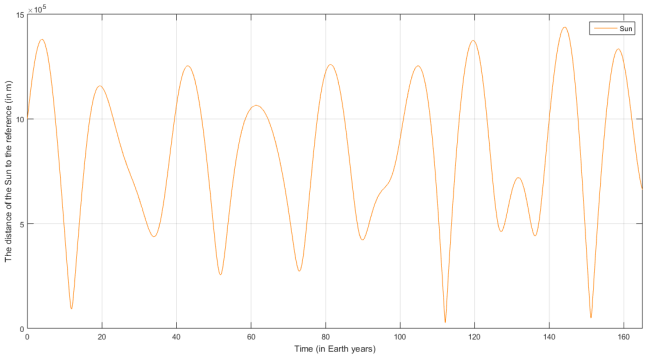
The above figure and of the distances from the reference point below, clearly shows a more stable revolution of Mercury. This result is obtained by a solution made with better accuracy compared to those of ode113 and ode45.



(a) Distance from origin - ode652 - Inner planet



(b) Distance from origin - ode652 - Outer planet



(c) Distance from origin - ode652 - Sun

Figure 13: Distance from origin

With this implementation the stability domain portrait by the chosen ERK of order 4, showed to sufficient (under certain adjustments) for the obtained solution.

5 Conclusion

As a summary, the implemented method discussed in this paper is a non A-stable method. Nonetheless, this method had a stability region necessary and sufficient for our implementation. The modeled IVP mentioned under section 2 was analyzed to be stiff due to the existence of some its parameters who tend to grow faster than others (Mercury's). Thereby, higher stability needed to be reached for satisfactory result.

After, attempting solutions of the IVP with build-in MatLab ODE Solvers, we realized a very high error dependency of the IVP model. Due to this condition, the choice of implementing the ERK of order 4 was made appropriate because of the better accuracy/stability it provided under certain adjustments.

Finally, we were able to implement a method, ode652, with much more better stability than certain build-in MatLab methods, ode45 and ode113 of quite similar non A-stability behavior, to solve the modeled IVP of solar system dynamics.

References

[1] ARIEH ISERLES, University of Cambridge, (1992), A First Course in the Numerical Analysis of Differential Equations

[2] NAZA, (2018), HORIZONS Web-Interface,
https://ssd.jpl.nasa.gov/horizons.cgi#top

[3] InfoPlease, (2018), Basic Planetary Data,
https://www.infoplease.com/science-health/solar-system/basic-planetary-data

[4] MATLAB, (2018), Choose an ODE Solver - MATLAB and Simulink,
https://www.mathworks.com/help/matlab/math/choose-an-ode-solver.html

[5] Wikipedia, (25 October 2017), List of Runge–Kutta methods,
https://en.wikipedia.org/wiki/List_of_Runge%E2%80%93Kutta_methods

[6] Guido Kanschat, (May 2, 2018), Numerical Analysis of Ordinary Differential Equations,
https://en.wikipedia.org/wiki/List_of_Runge%E2%80%93Kutta_methods

[7] Kyriacos Papadatos, (Unknown), THE EQUATIONS OF PLANETARY MOTION AND THEIR SOLUTION,
http://gsjournal.net/Science-Journals/Research%20Papers-Astrophysics/Download/3763

Appendices

```
%% FUNCTION: ode652 ************************************************
    function [T,Y] = ode652(ODE,TSPAN,Y_INIT)
        % Definition of the RK-Method parameters
        A_matrix = [0 0 0 0; .5 0 0 0; 0 .5 0 0; 0 0 1 0];
        b_weights = [1/6 1/3 1/3 1/6];
        c_nodes = [0 .5 .5 1];

        nu = length(c_nodes); %Det of RK-Method's # of stages

        STEP = TSPAN(2)-TSPAN(1); % Def of the step size
        T = TSPAN'; % Def of the time span.
        N = length(T); %Det of the # of steps.
        M = length(Y_INIT); % The number of values for a given step.
        Y = zeros(M,N); % Initialisation of the vector of Y.
        Y(:,1) = Y_INIT; % Def of the Initial value.

        %% RK-Method
        Xi = zeros(M,nu);%Initialisation of the Xi's.
        for n = 1:N-1

            So = zeros(M,1); %Init of the outter sum for each Y.
            for j = 1:nu

                Si = zeros(M,1); %init of the inner sum for each Xi.
                for i = 1:j-1
                    Si = Si + A_matrix(j,i)*ODE(T(n)+c_nodes(i)*STEP,Xi
                        (:,i)); %Det of the inner sum.
                end;

                Xi(:,j) = Y(:,n)+STEP*Si; %Determination of Xi's

                So = So + b_weights(j)*ODE(T(n)+c_nodes(j)*STEP,Xi(:,j)
                    ); %Det of the outter sum.

            end;

            Y(:,n+1) = Y(:,n)+STEP*So; %Det of the Approximation of the
                 next y by RK-method
        end;

        Y = Y';
    end
```