Student Name: Mina Gerges
ID: 70040441

Computational Physics_ PHYS624

A Numerical Solution for Planetary Motion Using the RK4 Method

Table of Contents

A Numerical Solution for Planetary Motion Using the RK4 Method

1. Abstract

This report intends to elaborate the use of the Runge-Kutta 4th order (RK4) numerical scheme in planetary motion simulation. The RK4 is one of the powerful numerical methods used to solve ordinary differential equations and would therefore find applicability in modeling the trajectory of celestial bodies under the influence of central forces, often making it popular in astrophysics. The paper brings forth certain mathematical derivation models for planetary motion and the implementation of the RK4 scheme.

2. Introduction

It is well known that Planetary motion follows the law of universal gravitation alongside with the newton's second law of motion, which states that every mass attracts every other mass with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between their centers. The motion equations for a planet can be derived out of this and yield a system of ordinary differential equations (ODEs) describing the position and velocity of the planet over time.

In this report, The study intends to project the RK4 method for simulating planetary motion and assess how effective it actually is. The investigation will be done through C++ numerically with checking the conservation of both total energy and angular momentum. Lastly, the numerical value of both eccentricity and th eperiod will be calculated too.

3. Theoritical Framework:

The motion of a planet under a central gravitational force is described by Newton's law of gravitation and the equations of motion for a planet in a two-dimensional space can be expressed as follows:

3.1 Main Equations of the system:

the equation represents the Gravitational force between two objects

$$F = -G\frac{mM}{r^2} \qquad (1)$$

where:

- (G) is the gravitational constant.
- (m) the mass of the planet
- (M) is the mass of the central body (e.g., the Sun),
- (r =$\sqrt{x^2 + y^2}$) is the distance from the central body.

This force could be expressed in terms of Newton's second law with the acceleration as the second derivative of the position:

$$F = m.\frac{d^2\vec{r}}{dt^2} = -G\frac{mM}{r^3}\vec{r} \qquad (2)$$

which leads to a second order differential equation in space, for a kind of simplicity we will consider the motion in 2-dimensional space (x, y).

3.2 Equation of motion in the cartesian coordinate

Acceleration in x-coordinate $\frac{d^2x}{dt^2} = -G\frac{M}{r^3}x$ **(3)**

Acceleration in y-coordinate. $\frac{d^2y}{dt^2} = -G\frac{M}{r^3}y$ **(4)**

Distance for the central body $r = \sqrt{x^2 + y^2}$ **(5)**

In order to proceed with the numerical solution we need to Decomposing the main equation "2 nd ODE" into a system of two "1st ODE" equations for each coordinate to represent the position and velocity:

$\frac{dx}{dt} = vx$ Velocity in x-coordinate **(6)**

$\frac{dy}{dt} = vy$ Velocity in y-coordinate **(7)**

$\frac{dvx}{dt} = -G\frac{M}{r^3}x$ Acceleration in x-coordinate **(8)**

$\frac{dvy}{dt} = -G\frac{M}{r^3}y$ Acceleration in x-coordinate **(9)**

where (v_x) and (v_y) are the components of the planet's velocity.

3.3 Energy and angular momentum

This motion as it is considered a motion in a central field is charterized by the conservation of the total energy (E) and angular momentum (L) as well , therefore, we ca caluclate both using the computed values of x,y,vx and vy as follows:

$E = \frac{1}{2} m v^2 - G\frac{mM}{r}$

K.E = $\frac{1}{2} m (vx^2 + vy^2)$

P.E $= -G\frac{mM}{\sqrt{x^2+y^2}}$

Energy = K.E + P.E **(10)**

$\boldsymbol{L = m(x\,vy - y\,vx)}$ **(11)**

3.4 calculating the eccentricity and the period

The eccennetricity is calculated in the context of semi major axis, where the script loops over all calculated r and selectes the minimum and maximum for the central body

a = (r_max + r_min) / 2

then the escript calculates the eccentricity in terms of {r_min, r_max, a}as follows:

e = (r_max - r_min) / 2 a

4. Methodology

4.1 The initial conditions of the problem:

The initial conditions to the physical system is known to be as follows:

- Mass of the star (M = 1.989 * 10^{30}) kg (mass of the Sun)
- Initial position ((x_0, y_0) = (1.496 *10^{11}, 0)) m
- Initial velocity ($(v_{\{x0\}}, v_{\{y0\}})$ = (0, v)) m/s (orbital speed of Earth)

Since the distance: 1 AU = 1.496*10^{11} m and the Time: 3.15*10^7 s, one could simplify the computational way will transform the uints of distances in terms of AU and time in terms of years, therfore, our initial conditions becomes :

GM = $4\pi^2$ gravitational constant (AU^3 / year^2)

r0 = 1.0 average Earth-Sun distance in AU

m = 1.0 mass of Earth in Earth masses

v0 = $\sqrt{\frac{GM}{r0}}$ ideal circular orbit velocity in AU/year

x = r0, y = 0 vx = 0, vy = v

vy is defined in our problem at different velocities (v = 0.8, 1.1, 1.2, 1.3, 1.4 v0.)

At this point the problem is converted into an initial value ODE that has system of 1st ODE with initial conditions that allows us to proceed with the next step in RK4.

4.2 RK4 Method Implementation

4.2.1 Definition of the technique

The RK4 technique is a numerical method to solve ODEs by approximating the solution at many points in time. The method calculates derivatives at multiple points for each time step in order to have more precision.
The steps include:
1. Calculate the intermediary slopes \(k_1, k_2, k_3, k_4\).
2. Combine these slopes to compute the next value of the variable.
The general form of RK4 slopes is as follows:

$$k1 = h \cdot f(t_n, y_n)$$
$$k2 = h \cdot f\left(t_n + \frac{h}{2}, y_n + \frac{hk1}{2}\right)$$
$$k3 = h \cdot f\left(t_n + \frac{h}{2}, y_n + \frac{hk2}{2}\right)$$
$$k4 = h \cdot f(t_n + h, y_n + hk3)$$

4.2.3 Implementation

The programming language used here is C++ as we can implement the RK4 method to simulate the planetary motion. Since we have four equations in the system one needs to apply those calculations for K(1,2,3&4) for each eqaution :

K1 terms

```
k1x = dx_dt(vx) * dt
k1y = dy_dt(vy) * dt
k1vx = dvx_dt(x, y) * dt
k1vy = dvy_dt(x, y) * dt
```

k2 terms

```
k2x = dx_dt(vx + 0.5 * k1vx) * dt
k2y = dy_dt(vy + 0.5 * k1vy) * dt
k2vx = dvx_dt(x + 0.5 * k1x, y + 0.5 * k1y) * dt
k2vy = dvy_dt(x + 0.5 * k1x, y + 0.5 * k1y) * dt
```

k3 terms

```
k3x = dx_dt(vx + 0.5 * k2vx) * dt
k3y = dy_dt(vy + 0.5 * k2vy) * dt
k3vx = dvx_dt(x + 0.5 * k2x, y + 0.5 * k2y) * dt
k3vy = dvy_dt(x + 0.5 * k2x, y + 0.5 * k2y) * dt
```

k4 terms

```
k4x = dx_dt(vx + k3vx) * dt
k4y = dy_dt(vy + k3vy) * dt
k4vx = dvx_dt(x + k3x, y + k3y) * dt
k4vy = dvy_dt(x + k3x, y + k3y) * dt
```

One can easily get the following for equations for position and velocity;

```
x = (k1x + 2 * k2x + 2 * k3x + k4x) / 6.0
y = (k1y + 2 * k2y + 2 * k3y + k4y) / 6.0
vx = (k1vx + 2 * k2vx + 2 * k3vx + k4vx) / 6.0
vy = (k1vy + 2 * k2vy + 2 * k3vy + k4vy) / 6.0
```

Repeat the RK4 steps for each time increment until the desired simulation time is reached through an iterative way.

4.2.2 Script structure:

The code would involve defining the system of equations, initializing the parameters, and iterating through time steps to update the position and velocity of the planet.

The sturcture in the C++ script is organized in terms of steps as the following:

1. Defining a function for the x positon
2. Defining a function for the y position
3. Defining a function for the acceklecration in x- cooridnate.
4. Defining a function for the acceklecration in y- cooridnate.
5. Defining a function to perform the RK4 sheme.
6. Defining a function to simulate the orbit and that include the following:
    a. The vy as (ratio*v0)
    b. The initial conditions
    c. Time settings of the simulation. “Had been set to 3 years with a tsep of 1/365.25”
    d. Orbit parameters as r_min, r_max, period (T), semi-major axis(a) , Total Energy (E) and angular momentum(L).
    e. A loop to calculate the positionr at every step and track the value r_min and r_max.
    f. Calculating the total energy and angular momentum at each timestep.
    g. Calculate the eccentricity and the period
        eccentricity = (r_max - r_min) / (r_max + r_min)
        period = 2 * M_PI * sqrt(a * a * a / GM)
7. Saving data to a “.txt” file for each initial velocity.

5. Benchmarking

5.1. Eccentricities and Periods calculated for different celestial objects:

This table illustrates the Eccentricities and Periods calculated for different celestial objects as Mercury and Venus.

Mercury(d(AU)= 0.387 , mass= 0.055)				
Data for velocity ratio 0.8	**Data for velocity ratio 1.0**	**Data for velocity ratio 1.05**	**Data for velocity ratio 1.2**	**Data for velocity ratio 1.4**
Final r_min: 0.182117647061	**Final r_min: 0.387000000000**	Final r_min: 0.387000000000	Final r_min: 0.387000000000	Final r_min: 0.387000000000
Final r_max: 0.387000000000	**Final r_max: 0.387000000000**	Final r_max: 0.475395543175	Final r_max: 0.995142857142	Final r_max: 9.818441869254
Eccentricity: 0.359999999996	**Eccentricity: 0.000000000000**	Eccentricity: 0.102500000000	Eccentricity: 0.440000000000	Eccentricity: 0.924158109966
Period: 0.151795216170 years	**Period: 0.240750084112 years**	Period: 0.283148823634 years	Period: 0.574492557770 years	Period: 11.526639919005 years
Venus (d(AU)= 0.723, mass= 0.815)				
Data for velocity ratio 0.8	**Data for velocity ratio 1.0**	**Data for velocity ratio 1.05**	**Data for velocity ratio 1.2**	**Data for velocity ratio 1.4**
Final r_min: 0.340235294151	**Final r_min: 0.723000000000**	Final r_min: 0.723000000000	Final r_min: 0.723000000000	Final r_min: 0.723000000000
Final r_max: 0.723000000000	**Final r_max: 0.723000000000**	Final r_max: 0.888142061277	Final r_max: 1.859142857065	Final r_max: 10.227438135382
Eccentricity: 0.359999999957	**Eccentricity: 0.000000000000**	Eccentricity: 0.102499999998	Eccentricity: 0.439999999983	Eccentricity: 0.867950489093
Period: 0.387613668628 years	**Period: 0.614762610281 years**	Period: 0.723029072063 years	Period: 1.466984095515 years	Period: 12.811566950851 years

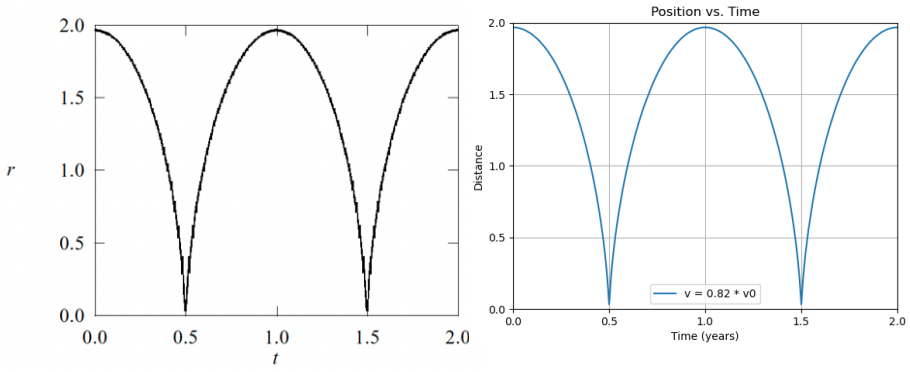
The second column reprsents the ideal velocity of the planet and it shows the condition of the circular orbit with approximately **r_min = r_max** yeilds an **eccentricity of zero**, moreover, the calculated period match the theoritical model introduced in NASA's fact sheet("Planetary Fact Sheet," n.d.) with a period 0.240749 years for Mercury and a Period 0.614763 years for Venus.

5.2. Benchmarking the position $\vec{r}$ to (Pang, 2006) figure 8.1 using the same initial conditions

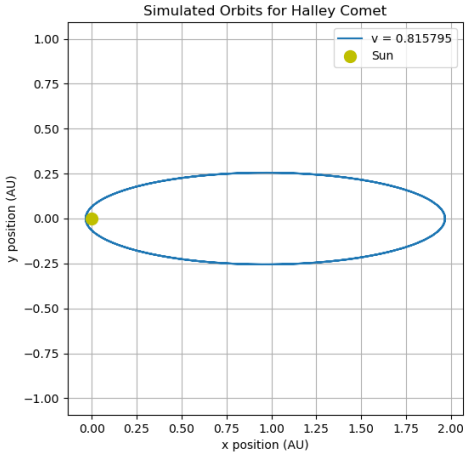
x[0] = 1.966843 y[0] = 0 r[0] = x[0] vx[0] = 0 vy[0] = 0.815795

Plot 1:Fig. 8.1 to the left the distance between Halley's comet and the Sun. The period of the comet is used as the unit of time, and the semimajor axis of the orbit is used as the unit of distance.(Pang, 2006).
Plot2: The right panel shows the distance as a function of time for the same conditions mention above
The calcualted parameters are **r_min** and **r_max** are benched to the textbook example

- Final r_min: 0.033159647870
- Final r_max: 1.966843000000
- Eccentricity: 0.966840396031



Plot3: the last plot in the line shows the orbit with its parameters and the sun at the origin.

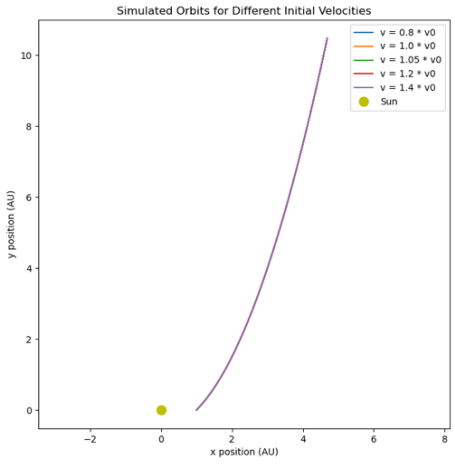


5.3 Escape velocity

Setting vx = vy = v0, which leads to the escape velocity case where orbits are open and hyperbolic.
In a hyperbolic orbit:

- The eccentricity e>1
- The trajectory takes the shape of a hyperbola.
- The object approaches the central body, reaches a minimum distance and then moves away, never to return.

Plot4: Presents the case when the planet is at the escape velocity level.



6.Results and Discussion

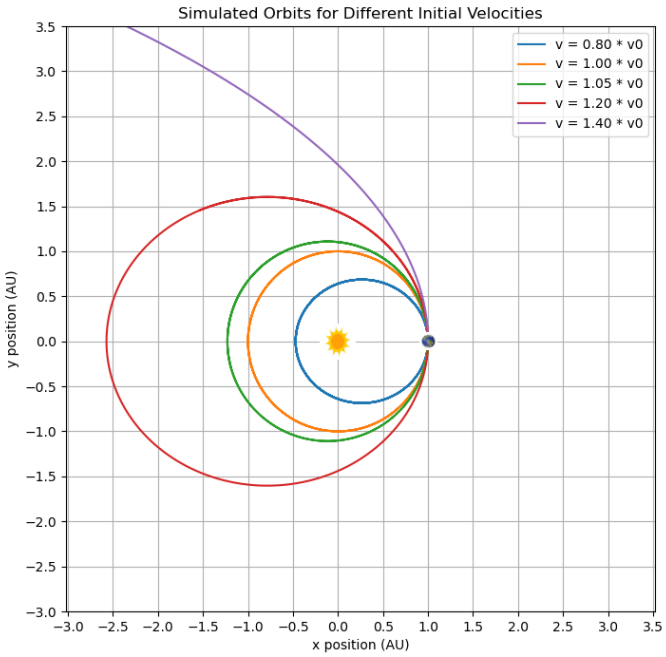
6.1 Trajectory Analysis

The RK4 simulation of trajectory: a planet under a constant gravitational field by numerical integration. The graphs produced from the output data confirm Keplerian elliptical orbits where they are consistent with Kepler's First Law as the path in which planets orbit is elliptical, keeping the central body at one of its foci.

Plot5: presents the Earth's orbits with different initial velocities in 2D- Plane

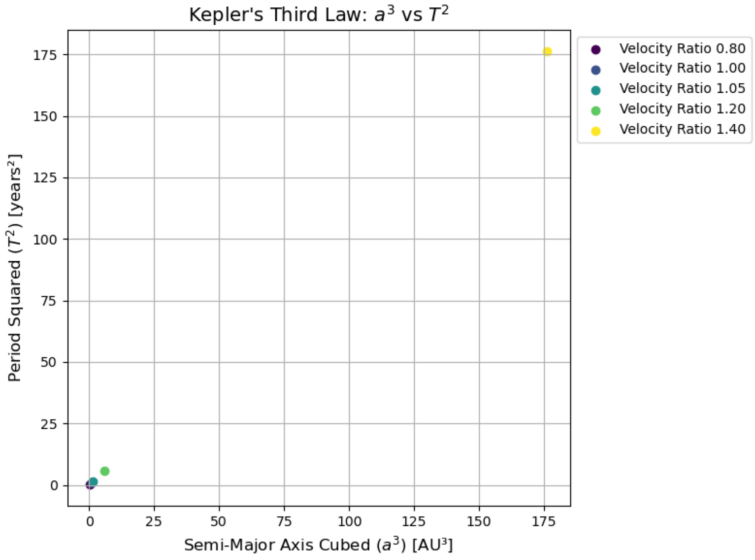
With the analysis of the plot, one can see that Earth's position is located at 1AU, whish is the initial condition for simulating the orbit with th sun located at the origin(0,0) illustrating a perfect circular orbit.

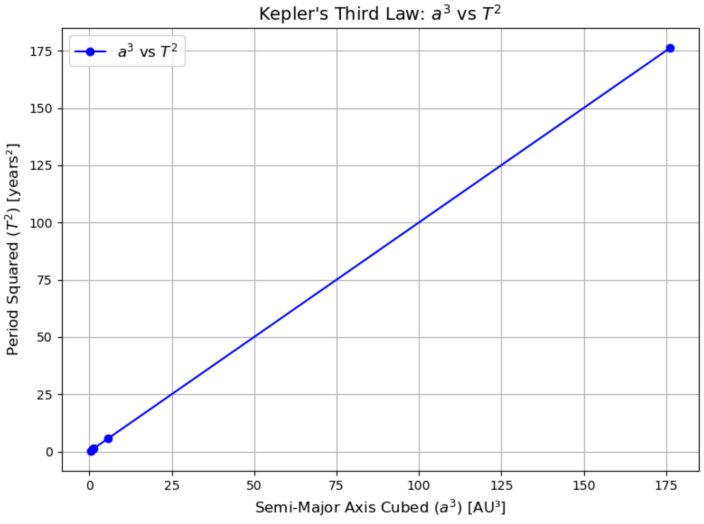
A Crucial Observations Initially at velocity ratios neared to 1.0, orbits are nearly circular, matching expectations from theory. Altered initial velocity ratios create a vast array of orbital types (e.g., travelling in an eccentric orbit or even a hyperbolic trajectory for higher ratios).



Sample result form the program in C++
At v = v0
Final r_min: 1.000000000000
Final r_max: 1.000000000000
Eccentricity: 0.000000000000
Period: 1.000000000000 years

Plot6: Verifying Kepler's Third Law
The plot presents thhe relation between the squre of period as a function of semi-major axis





6.2 Energy & Angular Momentum Conservation

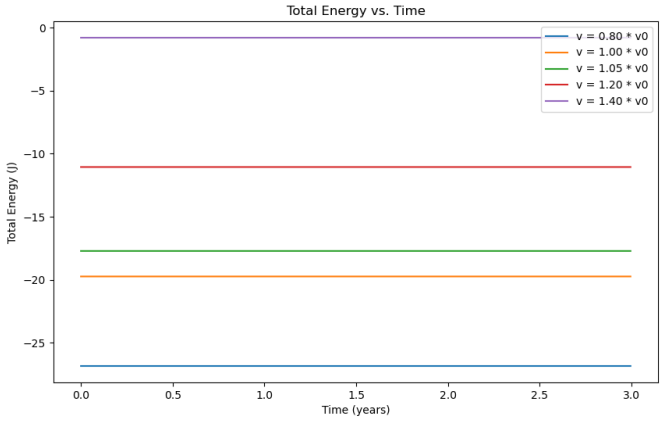
The computed vlaues of {x ,y, vx, vy} had been utilized to calulcate the kinetic and potentisal Energies and the angular momentum as well to check the conservation of both.

Table 1: illustrates a sample of the produced data at {v = 0.8v0}

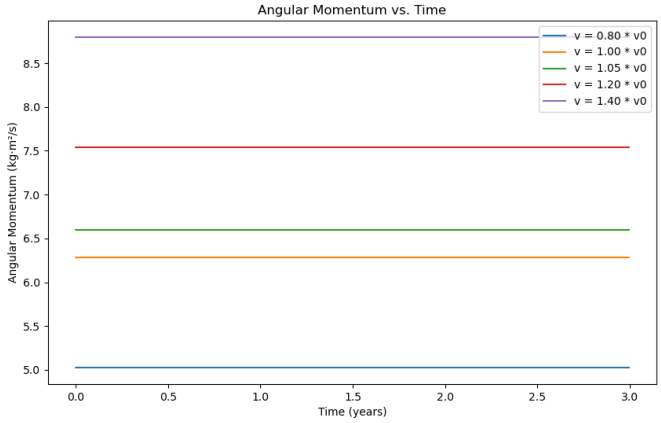
# time(s)	x(AU)	y(AU)	vx	vy	r(AU)	K.E(J)	P.E(J)	E(J)	L
0	1	0	0	5.02655	1	12.6331	-39.4784	-26.8453	5.02655
0.00273785	0.999852	0.0137613	-0.108086	5.0258	0.999947	12.6352	-39.4805	-26.8453	5.02655
0.0054757	0.999408	0.0275184	-0.216175	5.02357	0.999787	12.6415	-39.4868	-26.8453	5.02655
2.85248	-0.45032	-0.159949	2.62874	-10.2284	0.477884	55.7655	-82.6109	-26.8453	5.02655

The data shows and confirms a fixed value of both the total energy and angular momentum which is the case with the theory as total energy and angular momentum are conserved in central force field(Goldstein et al., 2014)

Plot7 : presents the total energy for the orbit at different initial velocities as a function of time.



Plot8 : presents the Angular momentum at different initial velocities as a function of time.

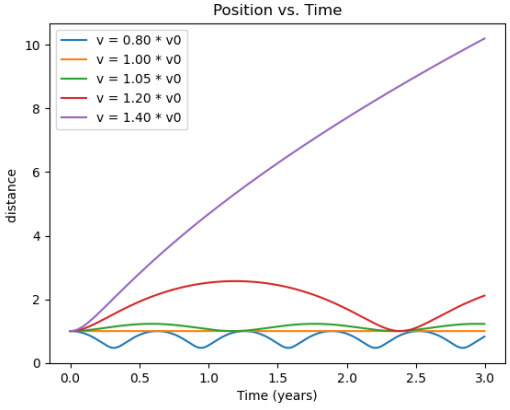


Note:

The two plot shows that energy and angular momentum are consevered which agreed with the theoritical model.

Plot9: presents the position $\vec{r}$ of the planet as a function of time at at different initial velocities.

For the case of v =v0 , one can observe that the position is perfectly-presented as r_min = r_max leading to an eccentriciry of zero.



7. Future work

Implementation of the velocity version of the Verlet algorithm which is reasonable in practice; it is usually accurate enough because the corresponding physical quantities are rescaled as mentioned in (Pang, 2006), for example, the temperature of the system in most molecular dynamics simulations.

8. Conclusion

The RK4 method provides an effective numerical approach to simulate planetary motion under gravitational forces. By discretizing the equations of motion and iteratively updating the state of the system, we can obtain accurate trajectories for celestial bodies. This method is widely used in astrophysics and can be extended to more complex systems involving multiple bodies and additional forces as stellar wind effect or two body problems.

9. References

Goldstein, H., Safko, J.L., Poole, C.P., 2014. Classical mechanics, Third, Pearson new international edition. ed. Pearson, Harlow, England.

Pang, T., 2006. An introduction to computational physics, 2nd ed. ed. Cambridge University Press, Cambridge ; New York.

Planetary Fact Sheet [WWW Document], n.d. URL https://nssdc.gsfc.nasa.gov/planetary/factsheet/ (accessed 11.25.24).

10. Appendices

10.1 The script

```
#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>
#include <string>
#include <iomanip>
using namespace std;

// Constants
const double GM = 4 * M_PI * M_PI;  // gravitational constant (AU^3 / year^2)
const double r0 = 1.0;              // average Earth-Sun distance in AU
const double m = 1.0;               // mass of Jupiter in Earth masses
const double v0 = sqrt(GM / r0);    // ideal circular orbit velocity in AU/year

// Differential equations written in first-order form
double dx_dt(double vx) {
    return vx;  // dx/dt = vx
}

double dy_dt(double vy) {
    return vy;  // dy/dt = vy
}

double dvx_dt(double x, double y) {
    double r = sqrt(x * x + y * y);
    return -GM * x / pow(r, 3);  // dvx/dt = -GMx / r^3
}

double dvy_dt(double x, double y) {
    double r = sqrt(x * x + y * y);
    return -GM * y / pow(r, 3);  // dvy/dt = -GMy / r^3
}
// Runge-Kutta 4th-order step function
void rk4_step(double &x, double &y, double &vx, double &vy, double dt) {
    // k1 terms
    double k1x = dx_dt(vx) * dt;
    double k1y = dy_dt(vy) * dt;
    double k1vx = dvx_dt(x, y) * dt;
    double k1vy = dvy_dt(x, y) * dt;
    // k2 terms
    double k2x = dx_dt(vx + 0.5 * k1vx) * dt;
    double k2y = dy_dt(vy + 0.5 * k1vy) * dt;
    double k2vx = dvx_dt(x + 0.5 * k1x, y + 0.5 * k1y) * dt;
    double k2vy = dvy_dt(x + 0.5 * k1x, y + 0.5 * k1y) * dt;
    // k3 terms
    double k3x = dx_dt(vx + 0.5 * k2vx) * dt;
    double k3y = dy_dt(vy + 0.5 * k2vy) * dt;
    double k3vx = dvx_dt(x + 0.5 * k2x, y + 0.5 * k2y) * dt;
    double k3vy = dvy_dt(x + 0.5 * k2x, y + 0.5 * k2y) * dt;
    // k4 terms
    double k4x = dx_dt(vx + k3vx) * dt;
    double k4y = dy_dt(vy + k3vy) * dt;
    double k4vx = dvx_dt(x + k3x, y + k3y) * dt;
    double k4vy = dvy_dt(x + k3x, y + k3y) * dt;

    // Update position and velocity
    x += (k1x + 2 * k2x + 2 * k3x + k4x) / 6.0;
    y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6.0;
    vx += (k1vx + 2 * k2vx + 2 * k3vx + k4vx) / 6.0;
    vy += (k1vy + 2 * k2vy + 2 * k3vy + k4vy) / 6.0;
}
// Main function
void simulate_orbit(double velocity_ratio) {
    double v = velocity_ratio * v0;

    // Initial position and velocity
    double x = r0, y = 0;
    double vx = 0, vy = v ;

    // Time settings
    double dt = 1.0 / (365.25*100) ;  // time step
    double total_time = 3.0;          // 3 years
    int steps = total_time / dt;

    double r_min = r0, r_max = r0;
    double total_energy = 0, angular_momentum = 0;
    double a = 0, period = 0;
```

```
    // Open a file to store results
    string filename = "orbit_data_" + to_string(velocity_ratio) + ".txt";
    ofstream outfile(filename);
    if (!outfile.is_open()) {
        cerr << "Error opening file: " << filename << endl;
        return;
    }
    // Header for the output file
    outfile << "# time(s) x(AU) y(AU) vx vy r(AU) kinetic_energy(J) potential_energy(J) total_energy(J) angular_momentum\n";

    // Run simulation
    for (int i = 0; i < steps; ++i) {
        // Calculate the distance r
        double r = sqrt(x * x + y * y);
        // Track min/max radius for eccentricity calculation
        if (r < r_min) r_min = r;
        if (r > r_max) r_max = r;

        // Calculate energy and angular momentum
        double kinetic_energy = 0.5 * m * (vx * vx + vy * vy);
        double potential_energy = -(GM * m) / r;
        total_energy = kinetic_energy + potential_energy;
        angular_momentum = m * (x * vy - y * vx);

        // Save to file
        outfile << i * dt << " " << x << " " << y << " " << vx << " " << vy << " " << r << " "
                << kinetic_energy << " " << potential_energy << " "
                << total_energy << " " << angular_momentum << endl;

        // Perform one RK4 integration step
        rk4_step(x, y, vx, vy, dt);
    }
    // Eccentricity
    a = (r_min + r_max) / 2;  // Semi-major axis
    double eccentricity = (r_max - r_min) / 2*a;
    // Period
    period = 2 * M_PI * sqrt(a * a * a / GM);
    outfile.close();
    cout << fixed << setprecision(12) << endl;
    cout << "Final r_min: " << r_min << endl;
    cout << "Final r_max: " << r_max << endl;
    cout << "Eccentricity: " << eccentricity << endl;
    cout << "Period: " << period << " years" << endl;
    cout << "Data for velocity ratio " << velocity_ratio << " saved to " << filename << endl;
}
int main() {
    vector<double> velocity_ratios;
    velocity_ratios.push_back(0.8);
    velocity_ratios.push_back(1.0);
    velocity_ratios.push_back(1.05);
    velocity_ratios.push_back(1.2);
    velocity_ratios.push_back(1.4);
    // Loop over velocity ratios
    for (size_t i = 0; i < velocity_ratios.size(); ++i) {
        double ratio = velocity_ratios[i];
        simulate_orbit(ratio);
    }
    return 0;
}
```

10.2 code results

This shows the results of running the code and saving results to a txt file for each initial velocity

Final r_min: 0.470588235332
Final r_max: 1.000000000000
Eccentricity: 0.359999999965
Period: 0.630509504224 years
Data for velocity ratio 0.800000000000 saved to orbit_data_0.800000.txt

Final r_min: 1.000000000000
Final r_max: 1.000000000000
Eccentricity: 0.000000000000
Period: 1.000000000000 years
Data for velocity ratio 1.000000000000 saved to orbit_data_1.000000.txt

Final r_min: 1.000000000000
Final r_max: 1.228412256234
Eccentricity: 0.102499999987
Period: 1.176111006033 years
Data for velocity ratio 1.050000000000 saved to orbit_data_1.050000.txt

Final r_min: 1.000000000000
Final r_max: 2.571428571420
Eccentricity: 0.439999999999
Period: 2.386261088495 years
Data for velocity ratio 1.200000000000 saved to orbit_data_1.200000.txt

Final r_min: 1.000000000000
Final r_max: 10.210902303181
Eccentricity: 0.821602227375
Period: 13.271373090849 years
Data for velocity ratio 1.400000000000 saved to orbit_data_1.400000.txt

Benchmarking results:

Final r_min: 0.033159647870
Final r_max: 1.966843000000
Eccentricity: 0.966840396031
Data for velocity ratio 0.815795000000 saved to orbit_data_0.815795.txt

10.3 Results in the terminal snippet

```
(base) minagerges@APPLEs-MBP ~ % cd Downloads
(base) minagerges@APPLEs-MBP Downloads % g++ Ecc\&period.cpp
(base) minagerges@APPLEs-MBP Downloads % ./a.out

Final r_min: 0.470588235332
Final r_max: 1.000000000000
Eccentricity: 0.359999999965
Period: 0.630509504224 years
Data for velocity ratio 0.800000000000 saved to orbit_data_0.800000.txt

Final r_min: 1.000000000000
Final r_max: 1.000000000000
Eccentricity: 0.000000000000
Period: 1.000000000000 years
Data for velocity ratio 1.000000000000 saved to orbit_data_1.000000.txt

Final r_min: 1.000000000000
Final r_max: 1.228412256234
Eccentricity: 0.102499999987
Period: 1.176111006033 years
Data for velocity ratio 1.050000000000 saved to orbit_data_1.050000.txt

Final r_min: 1.000000000000
Final r_max: 2.571428571420
Eccentricity: 0.439999999999
Period: 2.386261088495 years
Data for velocity ratio 1.200000000000 saved to orbit_data_1.200000.txt

Final r_min: 1.000000000000
Final r_max: 10.210902303181
Eccentricity: 0.821602227375
Period: 13.271373090849 years
Data for velocity ratio 1.400000000000 saved to orbit_data_1.400000.txt
```