Supervised by: Przemysław Korohoda, Dr. Eng
Date of submission: DD/MM/YYYY

Numerical Methods for Solving Differential Equations in Two-Body Problem

Computation Techniques Final Project

Kamil Chaj

Computational Techniques
Electronics and Telecommunications
AGH University of Science and Technology, Kraków, Poland
2023/2024

Abstract

Two-body problem discuss predicting orbit of two bodies viewed as point masses. The problem assumes that the two objects interact only with one another, and all other bodies are ignored. Goal of this project is to implement and test different methods of numerically solving ordinary differential equations in case of two-body problem. Methods we are going to implement and compare are matrix system of first order differential equations, Euler method, Runge-Kutta method and build-in ODE solvers in MATLAB.

(brief summery of results ...)

Contents

Chapter 1

Two Body Problem

1.1 Assumptions

For the problem to be not too complicated we can take few assumptions that will simplify solution but maintain overall correct results.

- in system there exists only two bodies
- bodies have uniform mass distributions and are perfectly symmetrical

1.2 Two-body system

In the system we define two bodies with masses m_1, m_2 and positions $\vec{r_1}$, $\vec{r_2}$, where distance between bodies is represented by $\vec{r_0}$ which can be described by following equation.

$$\vec{r_0} = \vec{r_2} - \vec{r_1} \tag{1.1}$$

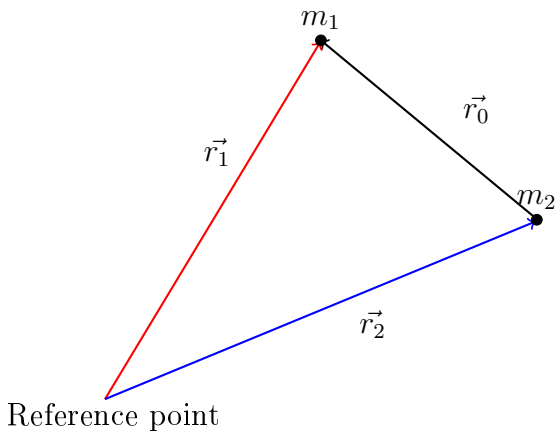

Figure 1.1: two-body system

1.3 Derivation of differential equation

Lets start derivation with defining forces $\vec{F_{12}}$ and $\vec{F_{21}}$ and relation between these forces coming from our assumption that there are only two bodies and therefore only two forces which net force is zero.

$$m_1 \frac{d^2\vec{r_1}}{dt^2} = \vec{F_{12}} \tag{1.2}$$

$$m_2 \frac{d^2\vec{r_2}}{dt^2} = \vec{F_{21}} \tag{1.3}$$

$$\vec{F_{12}} = -\vec{F_{21}} = \vec{F} \tag{1.4}$$

After combining above equations we get system of differential equation describing behavior of any two-body system.

$$\begin{cases} m_1 \frac{d^2\vec{r_1}}{dt^2} = \vec{F} \\ m_2 \frac{d^2\vec{r_2}}{dt^2} = -\vec{F} \end{cases} \tag{1.5}$$

1.4 System of two planets

In our case we want to investigate two-body system where force $\vec{F}$ is described by Newton's law of universal gravitation (1.6)

$$\vec{F} = G\frac{m_1 m_2}{r_0^2}\hat{r_0}, \qquad \hat{r_0} = \frac{\vec{r_0}}{r_0} \tag{1.6}$$

By substituting $\vec{F}$ (1.6) and $\vec{r_0}$ (1.1) in equation (1.12) we get final system of nonlinear differential equations.

$$\begin{cases} \frac{d^2\vec{r_1}}{dt^2} = \frac{Gm_2}{||\vec{r_2} - \vec{r_1}||^3}(\vec{r_2} - \vec{r_1}) \\ \frac{d^2\vec{r_2}}{dt^2} = \frac{Gm_1}{||\vec{r_2} - \vec{r_1}||^3}(\vec{r_1} - \vec{r_2}) \end{cases} \tag{1.7}$$

1.5 Initial conditions

For second order differential equation we need to specify position and velocity at some time, usually at time = 0, to find particular solution. In our system we have two second order equation therefore we need two initial position and velocity vectors which gives us 12 scalar quantities if we consider three dimensional case.

$$\begin{bmatrix} \vec{r_1}(t_0) \\ \vec{r_2}(t_0) \\ \vec{v_1}(t_0) \\ \vec{v_2}(t_0) \end{bmatrix} = \begin{bmatrix} x_1(t_0) & y_1(t_0) & z_1(t_0) \\ x_2(t_0) & y_2(t_0) & z_2(t_0) \\ v_{x1}(t_0) & v_{y1}(t_0) & v_{z1}(t_0) \\ v_{x1}(t_0) & v_{y1}(t_0) & v_{z1}(t_0) \end{bmatrix}, \qquad t_0 = 0 \tag{1.8}$$

1.5.1 Earth-Moon system

First we need to assign values for all parameters in our equations. G parameter is Newtonian constant of gravitation, note that this constant dictates what unites we have to use later

$$G = 6.6743 \cdot 10^{-11} m^3 kg^{-1} s^{-2} \tag{1.9}$$

Mass of the Earth

$$m_1 = 5.97 \cdot 10^{24} kg$$

Mass of the Moon

$$m_2 = 7.34 \cdot 10^{22} kg$$

And initial conditions for the system, $v_{\vec{moon}}$ is perpendicular to position vector, v_{earth} is velocity of Earth and Moon around Sun

$$v_{moon} = 1.022 \cdot 10^3 \frac{m}{s}, \qquad v_{earth} = 29.78 \cdot 10^3 \frac{m}{s}, \qquad r_0 = 3.844 \cdot 10^5 km$$

$$\begin{bmatrix} \vec{v_1}(0) \\ \vec{v_2}(0) \\ \vec{r_1}(0) \\ \vec{r_2}(0) \end{bmatrix} = \begin{bmatrix} 0 & v_{moon} & v_{earth} \\ 0 & 0 & v_{earth} \\ 0 & 0 & 0 \\ r_0 & 0 & 0 \end{bmatrix} \tag{1.10}$$

1.5.2 Equal masses system

Gravitation constant G stays the same as in previous case (1.9). Both masses are equal and for this example we will use 2 Marses.

$$m_1 = m_2 = 0.641 \cdot 10^{24} kg$$

Because it is purely hypothetical example we can set any initial values.

$$v_1 = 1 \cdot 10^3 \frac{m}{s}, \qquad v_2 = 1 \cdot 10^3 \frac{m}{s}, \qquad r_0 = 1 \cdot 10^8 m$$

$$\begin{bmatrix} \vec{v_1}(0) \\ \vec{v_2}(0) \\ \vec{r_1}(0) \\ \vec{r_2}(0) \end{bmatrix} = \begin{bmatrix} 0 & v_1 & 0 \\ 0 & v_2 & 0 \\ 0 & 0 & 0 \\ r_0 & 0 & 0 \end{bmatrix} \tag{1.11}$$

1.5.3 Arbitrary system

IDK TBD

For this system we will start with general equation for two-body problem (1.12) and specify Force with different expression which is linear. Force is proportional to distance between bodies

$$\begin{cases} \frac{d^2\vec{r_1}}{dt^2} = \frac{\vec{r_2} - \vec{r_1}}{m_1} \\ \frac{d^2\vec{r_2}}{dt^2} = \frac{\vec{r_2} + \vec{r_1}}{m_2} \end{cases} \tag{1.12}$$

$$m_1 = 10kg$$

$$m_2 = 10kg$$

$$v_1 = 1\frac{m}{s}, \qquad v_2 = 1\frac{m}{s}, \qquad r_0 = 10m$$

$$\begin{bmatrix} \vec{v_1}(0) \\ \vec{v_2}(0) \\ \vec{r_1}(0) \\ \vec{r_2}(0) \end{bmatrix} = \begin{bmatrix} 0 & v_1 & 0 \\ 0 & -v_2 & 0 \\ -0.5r_0 & 0 & 0 \\ 0.5r_0 & 0 & 0 \end{bmatrix} \tag{1.13}$$

Chapter 2

Numerical method for differential equations

In numerical approximations of differential equation we replace infinitesimal change d with finite change Δ and iteratively calculate small pieces of the curve.

$$\frac{d\vec{v}}{dt} = f(\vec{r}, t) \implies \frac{\Delta\vec{v}}{\Delta t} = f(\vec{r}, t_n) \tag{2.1}$$

$$\begin{aligned} t_n &= t_0 + n\Delta t \\ t_n &= t_{n-1} + \Delta t \end{aligned} \tag{2.2}$$

By multiplying equation (2.1) by Δt we get change of a function in time interval Δt

$$\Delta\vec{v} = f(\vec{r}, t_n)\Delta t \tag{2.3}$$

Truncation error is cause by using finite time step and comes from the definition of limit. Truncation error can be used for adjusting time step in adaptive methods

$$\frac{\Delta\vec{v_{n+1}} - \Delta\vec{v_n}}{\Delta t} = \text{Truncation error} \tag{2.4}$$

2.1 Linear example

IDK TBD

Linear differential equations can be solved using method of undetermined coefficients

2.2 Euler method

This is the simplest method for solving differential equations, where next value is sum of current value and change evaluated at the beginning of Δt

$$\begin{aligned} \vec{r_{n+1}} &= \vec{r_n} + \Delta\vec{r} \\ &= \vec{r_n} + f(\vec{r_n}, t_0 + n\Delta t)\Delta t \end{aligned} \tag{2.5}$$

For higher order equations we simply need to convert n-th order equation to system of n first order equations

$$\begin{bmatrix} \vec{r_{n+1}} \\ \vec{v_{n+1}} \end{bmatrix} = \begin{bmatrix} r_n + v_n\Delta t \\ v_n + f(\vec{r_n}, t)\Delta t \end{bmatrix} \tag{2.6}$$

2.3 4th order Runge-Kutta method

Runge-Kutta method is similar to Euler method but instead of evaluating change of a function once per iteration, in case of 4th order method, it evaluates function 4 times per iteration, and takes weighted average of these evaluations.

$$\begin{aligned} k_1 &= f(\vec{r_n}, t_0 + n\Delta t) \\ k_2 &= f\left(\vec{r_n} + k_1\frac{\Delta t}{2}, t_0 + \Delta t(n + \frac{1}{2})\right) \\ k_3 &= f\left(\vec{r_n} + k_2\frac{\Delta t}{2}, t_0 + \Delta t(n + \frac{1}{2})\right) \\ k_4 &= f\left(\vec{r_n} + k_3\Delta t, t_0 + \Delta t(n + 1)\right) \end{aligned} \tag{2.7}$$

Times at which each evaluation is calculated can be adjusted but the most common is to evaluated function once at beginning and end the interval and twice in the middle of the interval but for second evaluation taking result of the first as an input

$$\vec{v_{n+1}} = \vec{v_n} + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.8}$$

Evaluation made in the middle of the interval have usually greater weight.

$$\begin{bmatrix} \vec{r_{n+1}} \\ \vec{v_{n+1}} \end{bmatrix} = \begin{bmatrix} r_n + v_n\Delta t \\ v_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{bmatrix} \tag{2.9}$$

2.4 Adaptive Runge-Kutta method

In adaptive Runge-Kutta method after each iteration program check if truncation does not exceed set limit, and if it does, evaluation is rejected and the same evaluation is made again with new time step calculated based on maximal and current error until truncation is within set error limit.

For 4th order method new time step is calculated using following formula

$$\Delta\tau = \Delta t\left(\frac{\varepsilon}{e}\right)^{\frac{1}{5}} \tag{2.10}$$

$\Delta\tau$ - new time step, e - truncation error, ε - tolerance

2.5 Matlab ODE45 solver

Chapter 3

Results

3.1 COOL PLOTS

3.2 Accumulation error

3.3 Computation time

3.4 Conclusions

Appendix A

Sources

1. Planetary Fact Sheets - NASA
2. Numerical Methods for Engineers - Jeffrey Chasnov
3. Fundamentals of Orbital Mechanics - Alfonso Gonzalez
4. Wikipedia
   (a) Two-body problem
   (b) Newton's law of universal gravitation
   (c) Runge-Kutta methods
   (d) Truncation error
5. Fundamentals of Physics Extended 10th edition - Halliday & Resnick

Appendix B

Code

Only core functions listed, all code is available on Github repository

Listing B.1: base_ode.m

```
function dydt = base_ode(t, r, m_1, m_2, G)
    r_0 = r(10:12) - r(7:9);
    abs_r_0 = norm(r_0);
    vx1 = G .* m_2 .* r_0(1) ./ abs_r_0 .^3;
    vy1 = G .* m_2 .* r_0(2) ./ abs_r_0 .^3;
    vz1 = G .* m_2 .* r_0(3) ./ abs_r_0 .^3;
    vx2 = - G .* m_1 .* r_0(1) ./ abs_r_0 .^3;
    vy2 = - G .* m_1 .* r_0(2) ./ abs_r_0 .^3;
    vz2 = - G .* m_1 .* r_0(3) ./ abs_r_0 .^3;
    dydt(7:12) = r(1:6);
    dydt(1:6) = [vx1, vy1, vz1, vx2, vy2, vz2];
    dydt = dydt';
end
```

Listing B.2: euler.m

```
function r = euler(func, tspan, h, initial_conditions, m_1, m_2, G)
    r = zeros(round(tspan(2)/h), 12);
    r(1, :) = initial_conditions(:);
    for k = 2:length(r)
        r(k, :) = r(k-1, :) + h .* func(0, r(k-1, :), m_1, m_2, G)';
    end
end
```

Listing B.3: RK4.m

```
function r = RK4(func, tspan, h, initial_conditions, m_1, m_2, G)
    r = zeros(round(tspan(2)/h), 12);
    r(1, :) = initial_conditions(:);
    for k = 2:length(r)
        k_1 = func(0, r(k-1, :), m_1, m_2, G)';
        k_2 = func(0, r(k-1, :) + h .* k_1 ./2 , m_1, m_2, G)';
        k_3 = func(0, r(k-1, :) + h .* k_2 ./2 , m_1, m_2, G)';
        k_4 = func(0, r(k-1, :) + h .* k_3, m_1, m_2, G)';
        r(k, :) = r(k-1, :) + h ./ 6 .* (k_1 + 2.*k_2 + 2.*k_3 + k_4);
    end
end
```

Listing B.4: Adaptive_RK.m

```
function [t, r] = Adaptive_RK(func, tspan, max_err, initial_conditions, m_1↩
    , m_2, G)
    k = 1; t(k) = 0; h = 1e18;
```

```
    r(1, :) = initial_conditions(:);
    while t < tspan(2)
        k = k + 1;
        while 1
            k_1 = func(0, r(k-1, :), m_1, m_2, G)';
            k_2 = func(0, r(k-1, :) + h .* k_1 ./2 , m_1, m_2, G)';
            k_3 = func(0, r(k-1, :) + h .* k_2 ./2 , m_1, m_2, G)';
            k_4 = func(0, r(k-1, :) + h .* k_3, m_1, m_2, G)';
            r(k, :) = r(k-1, :) + h ./ 6 .* (k_1 + 2.*k_2 + 2.*k_3 + k_4);

            % e = max(abs(r(k, :) - r(k - 1, :)));
            e = max(abs(r(k, 7:12) - r(k - 1, 7:12)));
            h = 0.9 .* h .* (max_err./e).^(1/5);
            if e < max_err
                break;
            end
        end

        t(k) = t(k-1) + h;
    end
end
```