Comparison of Numerical Ordinary Differential Equation Solvers as Applied to the Double Pendulum System

Tom Stephen

November 20, 2020

1 Introduction

There are a great number of problems in the study of dynamics and chaos for which there is no analytical solution to describe the motion, however we may formulate a differential equation (or system of differential equations) to express the situation. For these problems, we may use numerical methods to obtain an approximate solution instead.

One such case is the 'double-pendulum' system, in which one pendulum is affixed to the end of another. Using Lagrangian mechanics, we may determine the rate at which the rotational velocity of each pendulum changes with time, but we can go no further analytically. This report aims to derive the equations of motion for the double pendulum, and compare different numerical methods to solve them.

Although this project focusses on a single system, the conclusions may be applied to a range of systems, including those where the analytical solution is possible to derive, but needlessly difficult.

2 Theory

2.1 Double Pendulum

Let us first carefully define the double pendulum system.

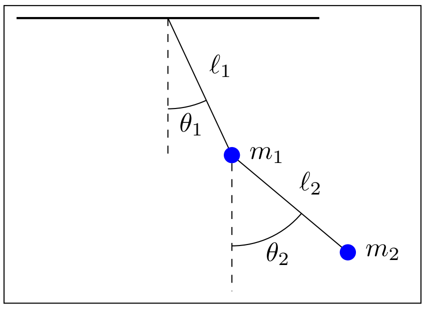


Figure 1: Diagram of Double Pendulum System

So ℓ_n represents the length of each 'arm', m_n represents the mass of each bob, and θ_n represents the angle of each arm as measured counter-clockwise from vertically down. Finally, denote positive y as being vertically upwards, and positive x as being positive horizontally to the right.

Before we discuss the dynamics of the system, we must make a few assumptions.

1. We assume that there are no external forces aside from gravity. Thus, we discount air resistance, friction in the joints, etc.
2. We assume the two arms of the double pendulum cannot collide, and that the arms are rigid.
3. The arms are massless, so the entire mass is located within m_1 and m_2.

We may express the position of each of the bobs in terms of the arm lengths and angles,

$$
\begin{aligned}
x_1 &= \ell_1 \sin\theta_1 & y_1 &= -\ell_1 \cos\theta_1 \\
x_2 &= \ell_1 \sin\theta_1 + \ell_2 \sin\theta_2 & y_2 &= -\ell_1 \cos\theta_1 - \ell_2 \cos\theta_2
\end{aligned}
$$

Differentiating with respect to time,

$$
\begin{aligned}
\dot{x}_1 &= \ell_1 \dot{\theta}_1 \cos\theta_1 & \dot{y}_1 &= \ell_1 \dot{\theta}_1 \sin\theta_1 \\
\dot{x}_2 &= \ell_1 \dot{\theta}_1 \cos\theta_1 + \ell_2 \dot{\theta}_2 \cos\theta_2 & \dot{y}_2 &= \ell_1 \dot{\theta}_1 \cos\theta_1 + \ell_2 \dot{\theta}_2 \sin\theta_2
\end{aligned}
$$

The Lagrangian is the difference from the total kinetic energy, T, of the system to the total potential energy, V, of the system[5],

$$L = T - V$$

First, the kinetic energy,

$$
\begin{aligned}
T &= T_1 + T_2 \\
&= \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 \\
&= \frac{1}{2} m_1 \left(\dot{x}_1^2 + \dot{y}_1^2\right) + \frac{1}{2} m_2 \left(\dot{x}_2^2 + \dot{y}_2^2\right) \\
&= \frac{1}{2} m_1 \ell_1^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 \left(\ell_1^2 \dot{\theta}_1^2 + \ell_2^2 \dot{\theta}_2^2 + 2\ell_1 \ell_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2)\right) \\
&= \frac{1}{2}(m_1 + m_2)\ell_1^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 \ell_2^2 \dot{\theta}_2^2 + m_2 \ell_1 \ell_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2)
\end{aligned}
$$

and potential energy,

$$
\begin{aligned}
V &= V_1 + V_2 \\
&= m_1 g y_1 + m_2 g y_2 \\
&= -m_1 g \ell_1 \cos\theta_1 - m_2 g \left(\ell_1 \cos\theta_1 + \ell_2 \cos\theta_2\right) \\
&= -\left(m_1 + m_2\right) g \ell_1 \cos\theta_1 - m_2 g \ell_2 \cos\theta_2
\end{aligned}
$$

The Lagrangian is then

$$L = \frac{1}{2}(m_1 + m_2)\ell_1^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 \ell_2^2 \dot{\theta}_2^2 + m_2 \ell_1 \ell_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2) + (m_1 + m_2) g \ell_1 \cos\theta_1 + m_2 g \ell_2 \cos\theta_2$$

The momenta are then defined as [1]

$$
\begin{aligned}
p_{\theta_1} &= \frac{\partial L}{\partial \dot{\theta}_1} = \left(m_1 + m_2\right) \ell_1^2 \dot{\theta}_1 + m_2 \ell_1 \ell_2 \dot{\theta}_2 \cos(\theta_1 - \theta_2) \\
p_{\theta_2} &= \frac{\partial L}{\partial \dot{\theta}_2} = m_2 \ell_2^2 \dot{\theta}_2 + m_2 \ell_1 \ell_2 \dot{\theta}_1 \cos(\theta_1 - \theta_2)
\end{aligned}
$$

Now, we may determine the equations of motion using the Euler-Lagrange equations, [1]

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) - \frac{\partial L}{\partial \theta_i} = 0 \implies \frac{dp_{\theta_i}}{dt} - \frac{\partial L}{\partial \theta_i} = 0 \qquad i = 1, 2$$

Computing the relevant derivatives and substituting in, we are left with

$$
\begin{aligned}
\left(m_1 + m_2\right) \ell_1 \ddot{\theta}_1 + m_2 \ell_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) + m_2 \ell_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) + \left(m_1 + m_2\right) g \sin\theta_1 &= 0 \\
l_2 \ddot{\theta}_2 + \ell_1 \ddot{\theta}_1 \cos(\theta_1 - \theta_2) - \ell_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + g \sin\theta_2 &= 0
\end{aligned}
$$

The above form a coupled system of second-order, non-linear, differential equation. Rearranging for second derivatives of θ,

$$
\begin{aligned}
\ddot{\theta}_1 + h_1(\theta_1, \theta_2)\, \ddot{\theta}_2 &= f_1\left(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2\right) \\
\ddot{\theta}_2 + h_2(\theta_1, \theta_2)\, \ddot{\theta}_1 &= f_2\left(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2\right)
\end{aligned}
$$

where

$$\begin{aligned}
h_1(\theta_1,\,\theta_2) &= \frac{\ell_2}{\ell_1}\left(\frac{m_2}{m_1+m_2}\right)\cos\left(\theta_1-\theta_2\right)\\
h_2(\theta_1,\,\theta_2) &= \frac{\ell_1}{\ell_2}\cos\left(\theta_1-\theta_2\right)\\
f_1\left(\theta_1,\,\theta_2,\,\dot\theta_1,\,\dot\theta_2\right) &= -\frac{\ell_2}{\ell_1}\left(\frac{m_2}{m_1+m_2}\right)\dot\theta_2^2\sin\left(\theta_1-\theta_2\right)-\frac{g}{\ell_1}\sin\theta_1\\
f_2\left(\theta_1,\,\theta_2,\,\dot\theta_1,\,\dot\theta_2\right) &= \frac{\ell_1}{\ell_2}\dot\theta_1^2\sin\left(\theta_1-\theta_2\right)-\frac{g}{\ell_2}\sin\theta_2
\end{aligned}$$

Returning to the system of ODEs, we may also express it in matrix form,[1]

$$\begin{pmatrix}1 & h_1\\ h_2 & 1\end{pmatrix}\begin{pmatrix}\ddot\theta_1\\ \ddot\theta_2\end{pmatrix}=\begin{pmatrix}f_1\\ f_2\end{pmatrix}$$

To solve for the second derivatives of θ, we compute the inverse of the coefficient matrix,

$$\begin{pmatrix}1 & h_1\\ h_2 & 1\end{pmatrix}^{-1}=\frac{1}{\det}\begin{pmatrix}1 & -h_1\\ -h_2 & 1\end{pmatrix}=\frac{1}{1-h_1h_2}\begin{pmatrix}1 & -h_1\\ -h_2 & 1\end{pmatrix}$$

and so

$$\begin{pmatrix}\ddot\theta_1\\ \ddot\theta_2\end{pmatrix}=\frac{1}{1-h_1h_2}\begin{pmatrix}1 & -h_1\\ -h_2 & 1\end{pmatrix}\begin{pmatrix}f_1\\ f_2\end{pmatrix}=\frac{1}{1-h_1h_2}\begin{pmatrix}f_1-h_1f_2\\ -h_2f_1+f_2\end{pmatrix}$$

Finally, let $\omega_i=\dot\theta_i$, and

$$\begin{aligned}
g_1 &= \frac{f_1-h_1f_2}{1-h_1h_2}\\
g_2 &= \frac{-h_2f_1+f_2}{1-h_1h_2}
\end{aligned}$$

Now, we have everything necessary to express the double pendulum system as a system of coupled first order differential equations,

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\begin{pmatrix}\theta_1\\ \theta_2\\ \omega_1\\ \omega_2\end{pmatrix}=\begin{pmatrix}\omega_1\\ \omega_2\\ g_1\left(\theta_1,\,\theta_2,\,\omega_1,\,\omega_2\right)\\ g_2\left(\theta_1,\,\theta_2,\,\omega_1,\,\omega_2\right)\end{pmatrix}\tag{1}$$

This, along with a set of initial conditions, is sufficient to solve the double pendulum system.

2.2 Ordinary Different Equation Solvers

To solve this ODE, we shall use four different methods:

1. ode45, an implementation of the Runge-Kutta 4-5 method included with Matlab. It is often the first recommended ODE solver to use in Matlab[2];

2. ode15s, a common alternative to ode45, more suitable for stiff systems. It is suggested to be used when ode45 fails or is inefficient[2];

3. ode113, a variable-step, variable-order solver[2], also included in Matlab. It computes the first 13 numerical derivatives of the differential equations, and uses the first 12 (13th is used to evaluate error in prediction) to compute the solution and extrapolate forwards in time to verify if the solution is suitable [4];

4. Euler's method, a more simple method than the previous three, and as such is an obvious choice for easy of implementation.

Euler's method is defined as[3], given initial conditions y_0, and some time step $\mathrm{d}t$, the successive solution is given by

$$y_n = y_{n-1} + y' \,\mathrm{d}t$$

The Euler's method is implemented in listing 2.

To evaluate how well each method solves the ODE, we shall look at five separate sets of initial conditions, and evaluate the total energy over time. As this is a closed system, the total energy of the system should stay constant. This is further discussed below.

2.3 Analysis

There is no analytical solution to the double-pendulum system. That is, it is not possible to write $\theta_1(t) = \dots$. As there is no analytical solution, we cannot compare the numerical solutions over time to some analytical result. Instead, we may investigate the energy of the system. As we are not considering friction, etc, and there are no driving forces, the total energy of the system (that is, the sum of potential and kinetic of both masses) should remain constant with time. If the energy increases or decreases for some method, there is clearly some issue with applying said method to this system.

Of course, some scenarios of the system (henceforth just referred to as corresponding to some initial condition, IC) will have a greater total energy than either (consider a pendulum starting at rest and one starting with a large velocity). To compare between situations, we normalise the energy by defining an 'Energy Error Factor', or EEF:

$$\text{EEF} = \frac{|\text{current energy} - \text{intial energy}|}{\text{initial energy}}$$

So, for an ideal numerical ODE solver, the EEF will stay at zero as time progresses - a larger EEF implies a less suitable solver.

3 Result

3.1 Model in Code

Before we may solve numerically the system of differential equations presented in equation (1), we must represent them in code. This is done using a Matlab function,

Listing 1: System of ODEs in Matlab

```matlab
function dy = doublePendulum(t, y, mass, len, g)
    % ODE to describe double pendulum system
    % returns instantaneous derivative of y

    % read constants in dev-friendly way
    theta1 = y(1);
    theta2 = y(2);
    omega1 = y(3);
    omega2 = y(4);

    % define functions
    h1 = @(theta1,theta2) (len(2)/len(1))*(mass(2)/sum(mass))*cos(theta1-theta2);
    h2 = @(theta1,theta2) (len(1)/len(2))*cos(theta1-theta2);

    f1 = @(theta1,theta2,omega2) -(len(2)/len(1)) * (mass(2)/sum(mass)) * omega2^2 * sin(theta1-theta2) - (g/len(1))*sin(theta1);
    f2 = @(theta1,theta2,omega1) (len(1)/len(2)) * omega1^2 * sin(theta1 - theta2) - (g/len(2))*sin(theta2);

    g1 = @(theta1,theta2,omega1,omega2) (f1(theta1,theta2,omega2) - h1(theta1,theta2) * f2(theta1,theta2,omega1)) / (1 - h1(theta1,theta2) * h2(theta1,theta2));
    g2 = @(theta1,theta2,omega1,omega2) (-h2(theta1,theta2) * f1(theta1,theta2,omega2) + f2(theta1,theta2,omega1)) / (1 - h1(theta1,theta2) * h2(theta1,theta2));

    % create dy vector to return
    dy = zeros(4, 1);
    dy(1) = omega1;
    dy(2) = omega2;
    dy(3) = g1(theta1,theta2,omega1,omega2);
    dy(4) = g2(theta1,theta2,omega1,omega2);
end
```

The bulk of the function is a direct translation of the functions defined in the Theory section. The function returns dy, which is the time derivative of the angular position-velocity vector. This function, along with the time span and initial conditions are sufficient for each of the numerical ODE-solvers to work.

Euler's method is not included in Matlab, so a custom function has been written.

Listing 2: Euler's Method Function

```matlab
function [t, y] = eulerTimeIndep(s, tspan, IC, mass, len, g)
    y(1,:) = IC;
    t = tspan;

    for i = 2:length(tspan)
        dy = s(t(i), y(i-1,:));
        y(i,:) = y(i-1,:) + dy.' .* (t(i) - t(i-1));
    end
end
```

3.2 Initial Conditions

The initial conditions that we shall investigate are,

(a) "Close to Rest", in which the initial angles are very small and the pendulum is at rest,

(b) "Horizontal", in which the pendulum is at rest, horizontally along the x-axis,

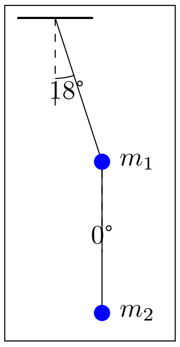
(c) "Vertical inverted", in which the pendulum is at rest, upside down,

(d) "High Energy, Moving", in which the pendulum starts in the same position as "(c) Vertical inverted", but with a significant rotational velocity, and

(e) "Folded horizontal", where one arm is positioned horizontally to the right, and the other horizontally to the left.

The code to produce the below plots is attached as an appendix.

3.2.1 (a) Close to Rest



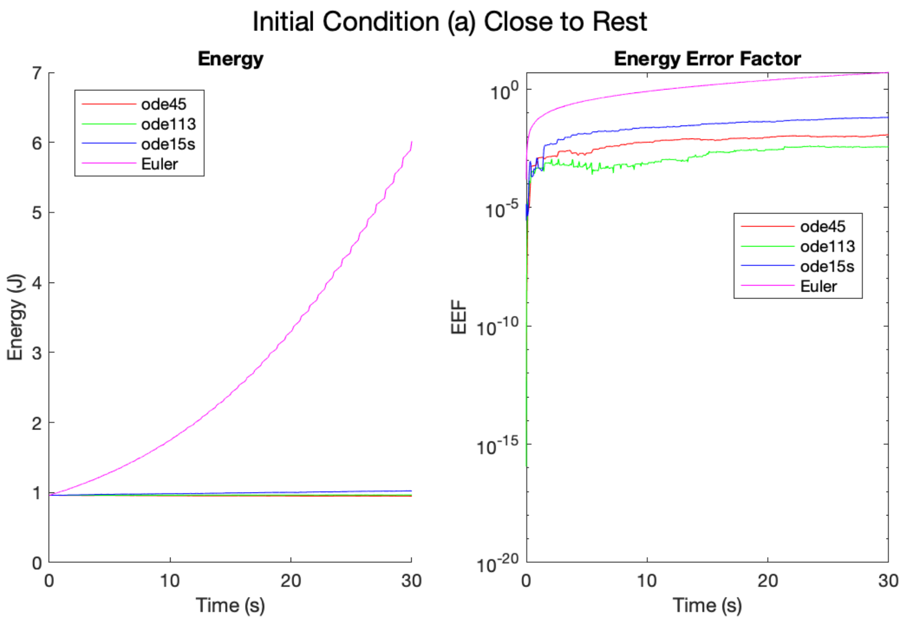
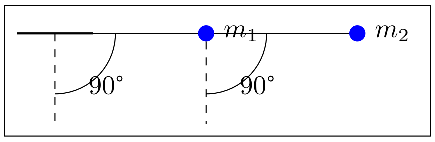


Figure 2: Plot of energy over time, and EEF over time, for IC (a)

This is designed to be a low energy system, and as such any good ODE solver should successfully solve the system well. All of the methods, except Euler, has a close-to-constant energy over the 30 second time span. Of these, ode113 performed the best. Euler's method, however, predicted the energy to be increasing exponentially - clearly this is not the case. It is likely that the small energy state settled into small oscillations, which Euler's method would overshoot with each time step. Decreasing the time step would improve this result, but this would greatly increase the run-time with Euler's method, already longer than each of the other methods.

3.2.2 (b) Horiztonal



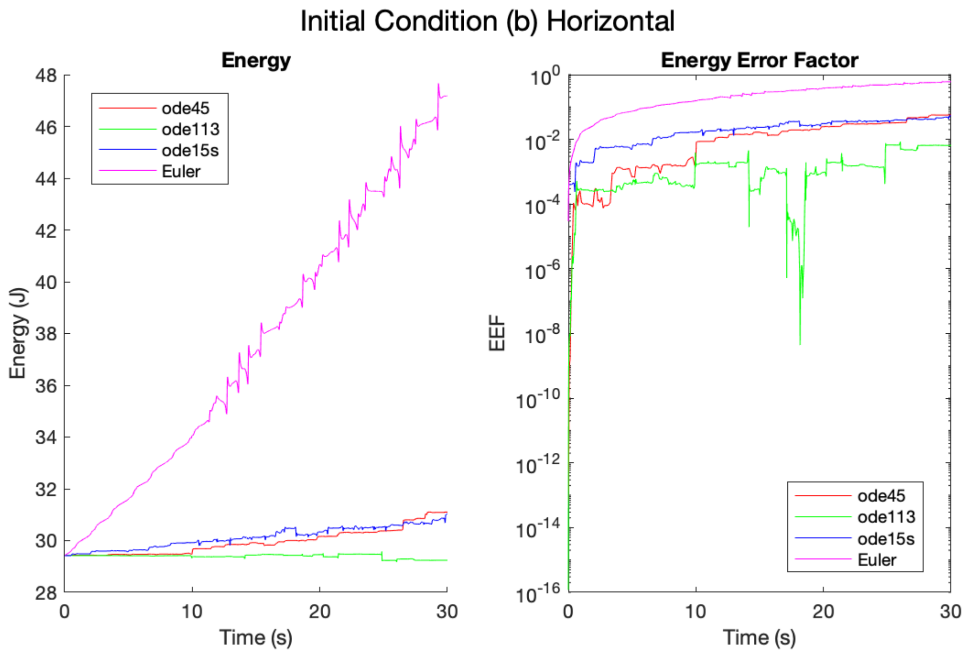
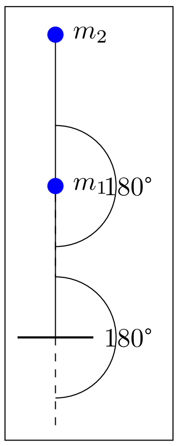


Figure 3: Plot of energy over time, and EEF over time, for IC (b)

This is a higher-energy system than in part (a), however the motion is not intially chaotic (as the pendulum falls). After passing the y-axis, however, ℓ_2 begins to flip over ℓ_1, and this corresponds to a jump in energy in each of the methods are 10s. As before, Euler's method predicts a significant linear increase in energy, until the flip occurs in which the increase becomes 'noisey'. This suggests that Euler's method is not appropriate for faster velocities, and particularly has issues when there are sudden changes in the solution. Again, ode113 performed the best.

3.2.3 (c) Vertical Inverted



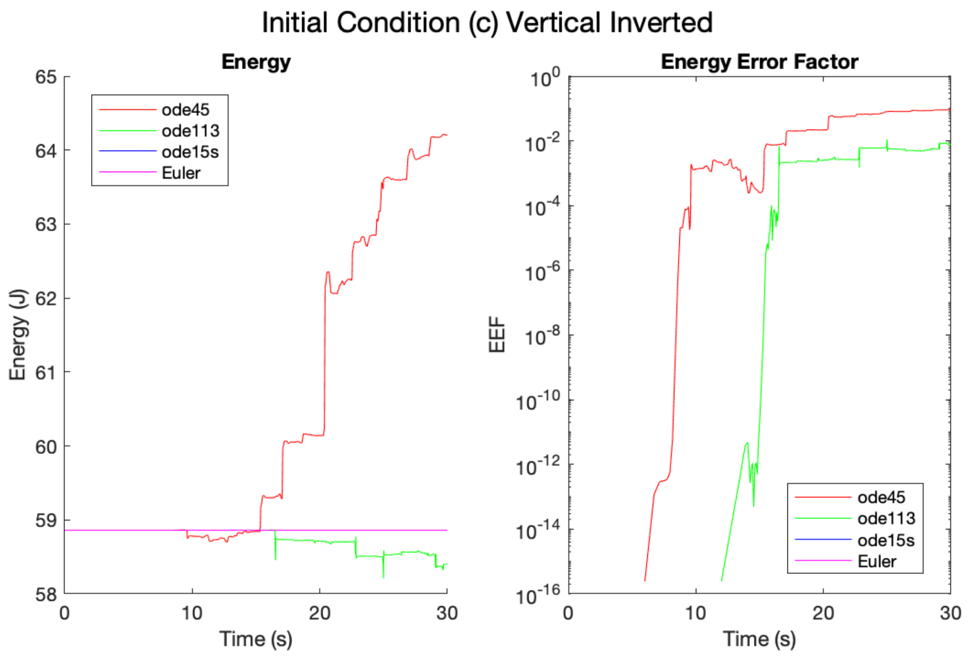
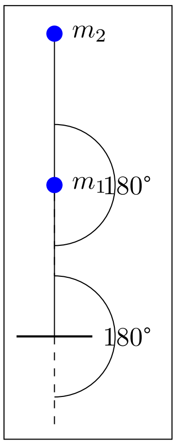


Figure 4: Plot of energy over time, and EEF over time, for IC (c)

This solution shows some particularly interesting results. In the model, there is no lateral force, and so the pendulum should stay balanced and not collapsed. However, the forces on the pendulum are a trigonometric function of the angles, and we are not completing symbolic equations, so $\cos(\pi) \approx -1 \neq -1$, for example. This arrises from not being able to store numbers, such as π, exactly in memory. For ode45 and ode113, numerical errors eventually add to the point in which the pendulum collapses, in a fairly chaotic state - the pendulum falls, and then the lower mass ‘snaps’ back upwards, causing a rapid change in the solution, leading to a significant change in the energy. ode15s and Euler’s method predict a perfectly constant energy as these solutions find that the pendulum should stay constant - likely due to the time step being fixed arbitrarily small such that numerical errors in forces are too small in comparison to the time step to perturb the pendulum.

3.2.4 (d) High Energy, Moving



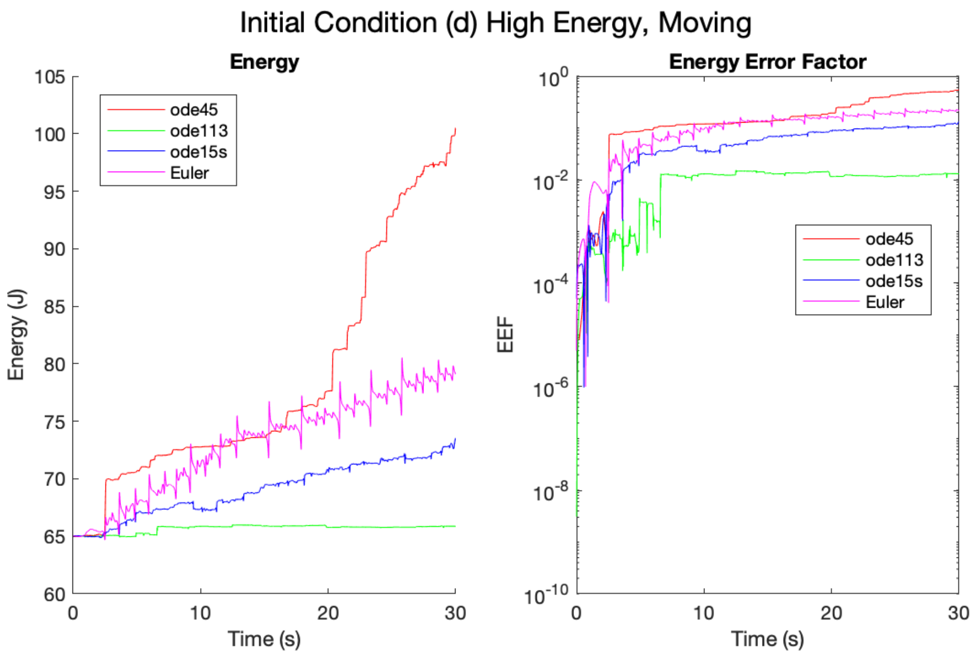
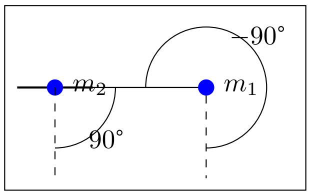


Figure 5: Plot of energy over time, and EEF over time, for IC (d)

Each method has difficulties in coming up with a realistic solution for these initial conditions - the high energy corresponds to a chaotic solution, and all solvers (except ode113) struggle to solve it, particularly ode45. Before the pendulum collapses and the motion changes suddenly, all solvers predict a constant energy.

3.2.5 (e) Folded Horiztonal



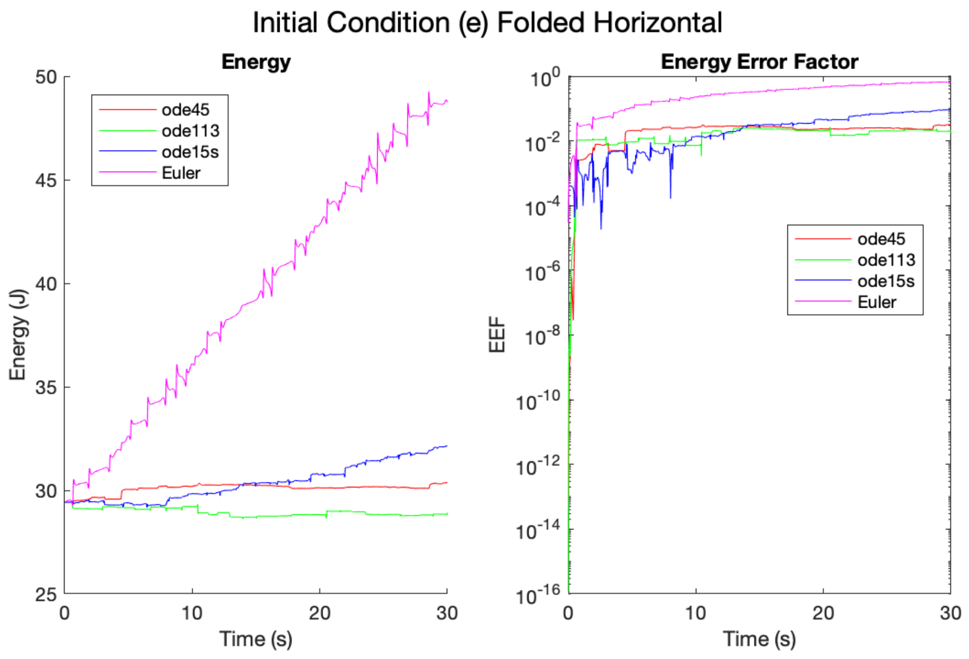


Figure 6: Plot of energy over time, and EEF over time, for IC (e)

This situation provides similar results to IC's (c), in which one of the masses is 'snapped' across when then tension acts opposite its motion. This leads to a jump in the energy predicted by ode45, and Euler's method. Euler's method again finds the energy to be increasing roughly linearly.

4 Discussion

Common across all initial conditions is that Euler's method, with an arbitrarily small step size, is not suitable for solving ODEs corresponding to chaotic systems. Also, although ode45 performed slightly better than ode15s at small energies, it quickly becomes unsuitable for high energies. ode113 was consistently the best method in terms of keeping the energy constant with time, which implies that it is providing a convincing solution to the system.

It is possible that, for a much smaller time step, Euler's method would improve. However, with the current time step, it is running considerably slower than the other methods, and decreasing the step size would increase the run-time - it is simply too inefficient. A large part of the efficiency in ode45 stems from the adaptive time-step part of the Runge-Kutta method. This, however, is choosing time-steps that are too large to accurately model the double pendulum situation as the true solution is often not smooth. This causes ode45 to overshoot the solution, and then over correct, similar to what we should expect in a stiff-system. ode15s is especially suited for stiff-systems, which explains why it performs better than ode45 for high energy scenarios.

ode113 extrapolates it's solution forwards in time using many numerical derivatives, and so takes into account the rotational acceleration, jerk, and so forth, so rapid changes in angular position and velocity have less of an effect. This leads to ode113 having comparable results for low energy systems, and significantly better results for high energy systems - it is more resilient for chaotic systems such as this. This does however come with a sacrifice in performance - computing so many derivatives increases run-time, and it is not possible to evaluate whether or not it is necessary to calculate the derivatives without actually computing them.

Considering each of the initial conditions, and the above discussion, we propose that, assuming processing time/power/memory are not a significant concern, ode113 is the best method for solving the double pendulum system.

To extend the rigour and improve this report, there are a few key improvements and extensions that may be considered:

1. When choosing an ODE solver, it is possible that memory usage and run-time are valid concerns. Thus, an investigation into the time-dependance of each method could be considered, in which each method solves the same ODE with very slightly different initial conditions many times, and then the average time for each method is calculated. Then, one could propose another metric for measuring performance, as a function of EEF after, for example, 100 seconds and the time to generate the solution. Then, the best solution in terms of accuracy - time could be chosen.

2. An investigation into Euler's method dependance on step size, with a 3D scatter plot of step size against time against accuracy. Perhaps there exists a method to calculate the ideal step size for Euler's method given the ODE for some situation.

3. Applying these models to different systems of ODEs! The double pendulum system was chosen purely for the reasons discussed in the introduction, and because it is interesting. There many other interesting, similar ODEs that these solvers could easily be applied to. Choosing an ODE with an analytical solution would allow for a much better analysis of how the analytical and numerical solutions compare.

5 Conclusion

This project aimed to compare different numerical system of ordinary differential equation solvers as applied to the double pendulum system, to determine which is most suitable. There does not exist an analytical solution, so instead the energy at each time step was calculated and graphed, as it should be constant for a closed system. Across multiple sets of initial conditions, Euler's method was shown to work poorly, and ode45, often the first-recommended solver in Matlab, produced poor results for high-energy systems. ode133 was consistently the best solver. To improve this investigation further, a different system of ODEs should be considered, perhaps one with an analytical solution to aid in evaluating the accuracy of each solver.

6 Self-Evaluation

I believe that I have taken care in choosing a system and sets of initial conditions such that I may critically evaluate the performance of each ODE solver, and I believe this was successful to show the strengths and weaknesses of different methods. As there is no analytical solution, I proposed an original, alternative metric to evaluate how well accurate each solution was - rather than just, say, plotting positions over time and discussing that.

In saying this, this report is by no means perfect, as discussed in the end of the discussion. More in-depth theory regarding each of the methods would also help to critically evaluate exactly why each solver performed well or poorly. Perhaps looking up the source code in Matlab for each of the methods to identify limiting factors in their implementation, and a more in-depth review of the papers that each of these methods has been defined from.

For these reasons, I believe this report deserves a medium-high 6, or perhaps a low 7.

7 Appendix

Listing 3: Calculate error factor for instant in time

```matlab
function eef = errorFactor(energy)
    % computes "Energy Error Factor" (EEF)
    % eef = |E_current - E_initial| / E_initial

    init = energy(1);
    eef = abs(energy - init) / init;
end
```

Listing 4: Calculate energy given current conditions

```matlab
function [KE, PE] = pendulumEnergy(y, mass, len, g)
    % KE = 1/2 * (mass(1) * (y(3) * len(1))^2 + mass(2) * (y(4) * len
       (2))^2);

    % height1 = len(1) * sin(y(1));
    % height2 = len(2) * sin(y(2)) + height1;
    % PE = g * (mass(1) * height1 + mass(2) * height2);

    KEa = 1/2 * (sum(mass)) * len(1)^2 * y(3)^2;
    KEb = 1/2 * mass(2) * len(2)^2 * y(4)^2;
    KEc = mass(2) * len(1) * len(2) * y(3) * y(4) * cos(y(1) - y(2));
    KE = KEa + KEb + KEc;

    PE1 = (1-cos(y(1))) * len(1) * mass(1) * g;
    PE2 = ((1-cos(y(1))) * len(1) + (1-cos(y(2))) * len(2)) * mass(2) *
        g;
    PE = PE1 + PE2;
end
```

Listing 5: Calculate energy given vector of conditions

```matlab
function [KE, PE, TE] = pendulumEnergyVec(yy, mass, len, g)
    KE = [];
    PE = [];
    for i = 1:length(yy)
        y = yy(:,i);
        [KE_ind, PE_ind] = pendulumEnergy(y, mass, len, g);
        KE(i) = KE_ind; PE(i) = PE_ind;
    end
    TE = KE + PE;
end
```

Listing 6: Main Simulation

```
% === Double Pendulum Simulation ===
% Comparison of ODE solver methods for determining solution
% to double pendulum system for range of initial conditions.

% Prepare,
clear all; close all;

% Define constants,
g = 9.81;
mass = [1, 1];
len = [1, 1];
tspan = [0 30];
tvec = linspace(0, 30, 10000);

% Parameterize function,
s = @(t,y) doublePendulum(t, y, mass, len, g);

% Initial conditions,
ICs = {
    [pi/10, 0, 0, 0],
    [pi/2, pi/2, 0, 0],
    [pi, pi, 0, 0],
    [pi, pi, pi/2, pi/2],
    [pi/2, -pi/2, 0, 0],
};
titles = {"(a) Close to Rest", "(b) Horizontal", "(c) Vertical Inverted
   ", "(d) High Energy, Moving", "(e) Folded Horizontal"};
fileTitle = {"a", "b", "c", "d", "e"};

% Loop over ICs,
for i = 1:length(ICs)
    IC = ICs{i};

    % Generate solutions,
    % ode45
    sol_ode45 = ode45(s, tspan, IC);
    [KE, PE, TE_ode45] = pendulumEnergyVec(sol_ode45.y, mass, len, g);
    eef_ode45 = errorFactor(TE_ode45);

    % ode113
    sol_ode113 = ode113(s, tspan, IC);
    [KE, PE, TE_ode113] = pendulumEnergyVec(sol_ode113.y, mass, len, g)
       ;
    eef_ode113 = errorFactor(TE_ode113);

    % ode15s
    sol_ode15s = ode15s(s, tspan, IC);
    [KE, PE, TE_ode15s] = pendulumEnergyVec(sol_ode15s.y, mass, len, g)
       ;
    eef_ode15s = errorFactor(TE_ode15s);

    % euler
    [time, sol_euler] = eulerTimeIndep(s, tvec, IC);
    sol_euler = sol_euler';
    [KE, PE, TE_euler] = pendulumEnergyVec(sol_euler, mass, len, g);
    eef_euler = errorFactor(TE_euler);

    % Plot energy over time
    figure
    subplot(1,2,1)
    hold on
    plot(sol_ode45.x, TE_ode45, "r")
    plot(sol_ode113.x, TE_ode113, "g")
    plot(sol_ode15s.x, TE_ode15s, "b")
    plot(tvec, TE_euler, "m")
    legend(["ode45", "ode113", "ode15s", "Euler"],'Location', 'Best')
    title("Energy");
    xlabel("Time (s)")
    ylabel("Energy (J)")

    % Plot error factors
    subplot(1,2,2)
    semilogy(sol_ode45.x, eef_ode45, "r")
    hold on
    semilogy(sol_ode113.x, eef_ode113, "g")
    semilogy(sol_ode15s.x, eef_ode15s, "b")
    semilogy(tvec, eef_euler, "m")
    legend(["ode45", "ode113", "ode15s", "Euler"],'Location', 'Best')
    title("Energy Error Factor");
    xlabel("Time (s)")
    ylabel("EEF")

    set(gcf,'Position',[100 100 600 350])

    sgtitle("Initial Condition " + titles{i})
    filePath = pwd + "/E_EEF_plots/" + fileTitle{i} + ".png";
    saveas(gcf, filePath)
end
```

References

[1] Erwin Kreyszig. *Advanced Engineering Mathematics*. 6th. Don Ford, 1988.

[2] Mathworks. *Ordinary Differential Equations*. 2020. URL: `https://www.mathworks.com/help/matlab/ordinary-differential-equations.html`.

[3] Timothy Sauer. *Numerical Analysis*. 2nd. George Mason University, 2012.

[4] Lawrence Shampine and Mark Reichelt. *THE MATLAB ODE SUITE*.

[5] Ziliang Zhou and Chad Whiteman. “Motion of Double Pendulum”. In: *Nonlinear Analysis: Theory, Methods and Applications* 26.7 (1996), pp. 1177–1191.