Project 4 - Double Pendulum

Chiarotto Elisa

October 10, 2022

1 Abstract

The double pendulum is a simple system composed by two masses, yet it can lead to a chaotic behaviour if given enough energy and time, therefore the result can be strongly influenced by small fluctuation in the initial condition and also by the integration method of choice.

2 Equations

Let us consider a double pendulum, namely a system composed by two masses (m_1 and m_2), connected by two rigid, massless wires of length L_1 and L_2, as shown in the figure below.

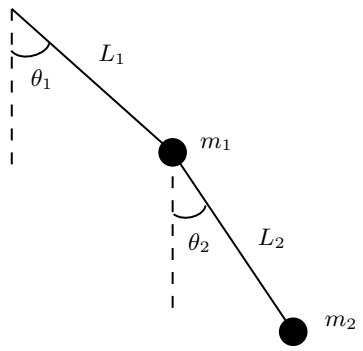


Figure 1: Double pendulum diagram

The positions of the masses can be found using:

$$\begin{cases} x_1 = +L_1 \sin\theta_1 \\ y_1 = -L_1 \cos\theta_1 \end{cases} \quad \text{and} \quad \begin{cases} x_2 = x_1 + L_2 \sin\theta_2 \\ y_2 = y_1 - L_2 \cos\theta_2 \end{cases} \tag{1}$$

The equations of motion are:

$$\frac{d^2\theta_1}{dt^2} = \frac{-g(2m_1+m_2)\sin\theta_1 - gm_2\sin(\theta_1-2\theta_2) - 2m_2\sin(\theta_1-\theta_2)\big(\dot\theta_2^2 L_2 + \dot\theta_1^2 L_1\cos(\theta_1-\theta_2)\big)}{L_1\big(2m_1+m_2-m_2\cos(2\theta_1-2\theta_2)\big)}$$

$$\frac{d^2\theta_2}{dt^2} = \frac{2\sin(\theta_1-\theta_2)\big(\dot\theta_1^2 L_1(m_1+m_2) + g(m_1+m_2)\cos\theta_1 + \dot\theta_2^2 L_2 m_2\cos(\theta_1-\theta_2)\big)}{L_2\big(2m_1+m_2-m_2\cos(2\theta_1-2\theta_2)\big)} \tag{2}$$

The energy of the system is:

$$E = \frac{1}{2}(m_1+m_2)v_1^2 + \frac{1}{2}m_2v_2^2 + m_2v_1v_2\cos(\theta_1-\theta_2) + g\big[m_1(L_1+L_2+y_1) + m_2(L_1+L_2+y_2)\big] \tag{3}$$

where $v_1 = L_1\dot{\theta}_1$ and $v_2 = L_2\dot{\theta}_2$.

For the following arguments let us set $m_1 = m_2 = m$ and $L_1 = L_2 = L$.
For simplicity I will consider any quantity in the International System of Units.

2.1 Small angles hypotesis and analitic solution

It is possible to calculate the analytic solution in the small angles hypothesis: $\cos\theta \approx 1 - \frac{1}{2}\theta^2$, which means that for $\theta < 38°$ I have less then 1% relative error.

The differential equations become: $\ddot{\theta}_1 = g\theta_2 - 2\frac{g}{L}\theta_1$ and $\ddot{\theta}_2 = 2g(\theta_1 - \theta_2)$, solving this leads to the following equations:

$$\begin{cases} \theta_1(t) = \frac{1}{\sqrt{2}}(\Theta_+ \cos(\omega_+ t) + \Theta_- \cos(\omega_- t)) \\ \theta_2(t) = \Theta_+ \cos(\omega_+ t) - \Theta_- \cos(\omega_- t) \end{cases} \quad \text{where} \quad \begin{cases} \omega_+ = \sqrt{\frac{g}{L}(2-\sqrt{2})} \\ \omega_- = \sqrt{\frac{g}{L}(2+\sqrt{2})} \end{cases} \quad \text{and} \quad \begin{cases} \Theta_+ = \frac{1}{2}(\sqrt{2}\theta_{1,0} + \theta_{2,0}) \\ \Theta_- = \frac{1}{2}(\sqrt{2}\theta_{1,0} - \theta_{2,0} \end{cases} \tag{4}$$

2.2 Compatibily with the analytic solution

The compatibility with the analytic solution can be studied by plotting the error $\theta - \theta_{sol}$ in the small angles regime, where the analytic solution is known.
For this study, I used $\theta_1 = 0,\ \theta_2 = 0.001,\ \dot{\theta}_1 = \dot{\theta}_2 = 0$ and $dt = 0.001$.

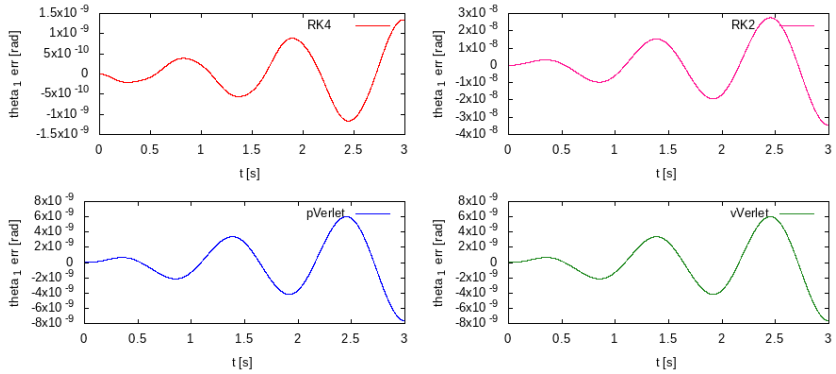


Figure 2: evolution of the error of θ_1 over time

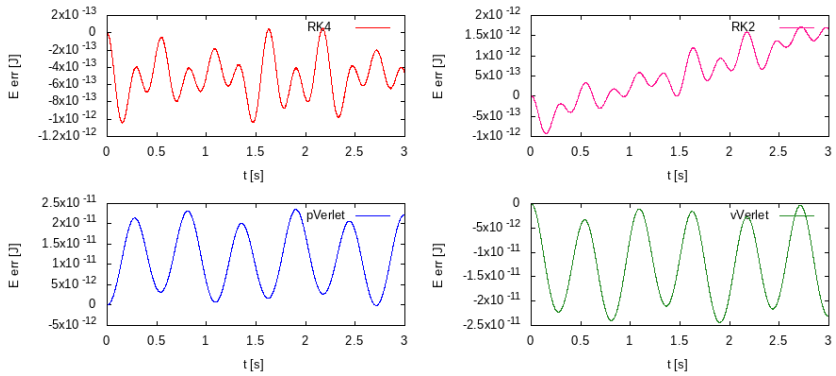


Figure 3: evolution of the error of E over time

3 Methods

Different methods have been used in order to compare the results: Runge-Kutta at the 2nd and 4th order, position Verlet and velocity Verlet.

3.1 Brief description of RK4 (main method)

Let the increment be $h = \delta t$ let's define the vectors containing the variables:

$$\begin{aligned} Y &= (\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) \\ R &= (\dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1, \ddot{\theta}_2) \end{aligned} \tag{5}$$

The latter is the differentiable equation (dYdT in the code below). The Runge-Kutta (RK) is a method based on the Taylor expansion, RK4 can be summarized as follow:

$$\begin{aligned} k_1 &= R(t_n, Y_n) \\ k_2 &= R(t_n + \frac{h}{2}, Y_n + \frac{h}{2}k_1) \\ k_3 &= R(t_n + \frac{h}{2}, Y_n + \frac{h}{2}k_2) \\ k_4 &= R(t_n + h, Y_n + hk_3) \\ Y_{n+1} &= Y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{6}$$

This method has a local accuracy of $O(h^5)$, which correspond to a global accuracy of $O(h^4)$.

3.2 Brief description of RK2

Likewise Runge-Kutta at the second order is described as:

$$\begin{aligned} k_1 &= R(t_n, Y_n) \\ k_2 &= R(t_n + \frac{h}{2}, Y_n + \frac{h}{2}k_1) \\ Y_{n+1} &= Y_n + hk_2 \end{aligned} \tag{7}$$

This method has a local accuracy of $O(h^3)$, which correspond to a global accuracy of $O(h^2)$.

3.3 Brief description of velocity-Verlet

Let the increment be $h = \delta t$ and let's define the vectors containing the variables:

$$\begin{aligned} X &= (\theta_1, \theta_2) \\ V &= (\dot{\theta}_1, \dot{\theta}_2) \\ A &= (\ddot{\theta}_1, \ddot{\theta}_2) \end{aligned} \tag{8}$$

$A(X)$ is the acceleration (acc in the code below), then the velocity-Verlet (vV) can be summarized as follow:

$$\begin{aligned} V_{n+\frac{1}{2}} &= V_n + \frac{h}{2}A(X_n) \\ X_{n+1} &= X_n + hV_{n+\frac{1}{2}} \\ V_n &= V_{n+\frac{1}{2}} + \frac{h}{2}A(X_{n+1}) \end{aligned} \tag{9}$$

This method has a local accuracy of $O(h^3)$, which correspond to a global accuracy of $O(h^2)$.

It should be noted that the both position and velocity Verlet work only in the small angles regime when the differentiable equation (Eq. 2) only depends on θ and not on $\dot{\theta}$.

3.4 Brief description of position-Verlet

Likewise the position-Verlet (pV) can be summarized as follow:

$$\begin{aligned} X_{n+\frac{1}{2}} &= X_n + \frac{h}{2} V_n \\ V_{n+1} &= V_n + hA(X_{n+\frac{1}{2}}) \\ X_n &= X_{n+\frac{1}{2}} + \frac{h}{2} V_{n+1} \end{aligned} \tag{10}$$

This method has a local accuracy of $O(h^3)$, which correspond to a global accuracy of $O(h^2)$.

3.5 Convergence of the methods

It is possible to find the error in the small angles regime, where the analytic solution is known, therefore it is possible to study the convergence of the algorithms.
For this study I used $\theta_1 = 0,\ \theta_2 = 0.001,\ \dot{\theta}_1 = 0,\ \dot{\theta}_2 = 0$ and $dt = 0.001$.

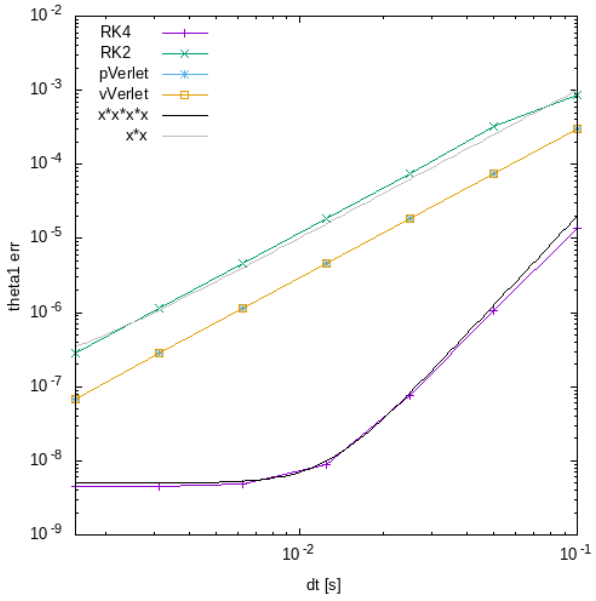


Figure 4: convergence of the algorithms

In the second figure beside the algorithms, a function of dt^4 and dt^2 are also being plotted and they have the same slope as respectively RK4 and RK2 (as well as pV and vV), this proves that the algorithms converge as expected.

3.6 Conservation of the energy

Keeping in mind that the fluctuations of the energy increase as the step δt decreases, for $\delta t = 0.001,\ \theta_1 = \theta_2 = \frac{\pi}{4},\ \dot{\theta}_1 = \dot{\theta}_2 = 0$, the following result is obtained and the energy is in general not conserved.

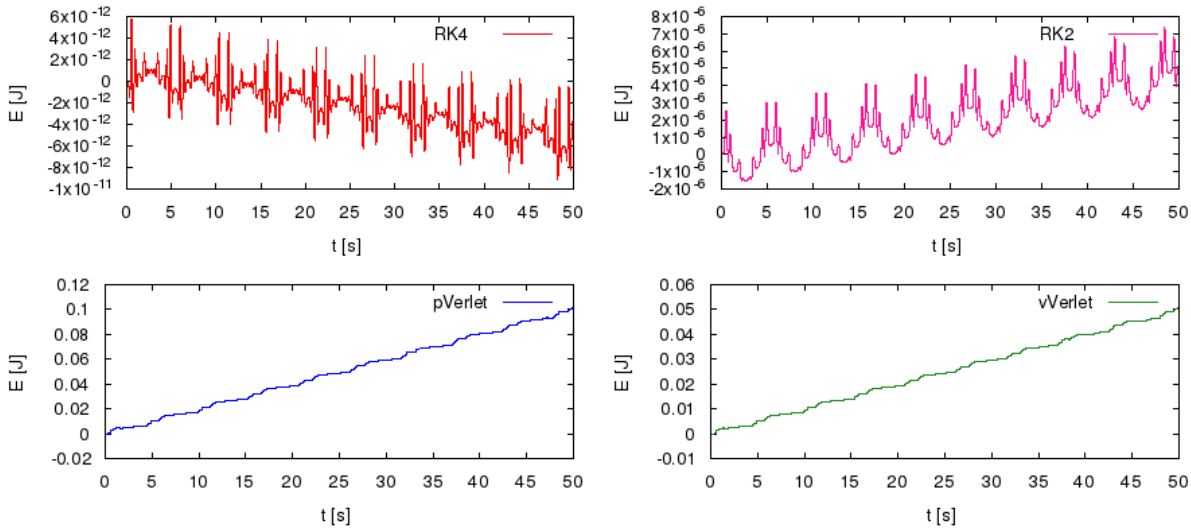


Figure 5: evolution of $E(t) - E(0)$ over time

The RK4 method is the best, which is to be expected since its global accuracy is $O(dt^4)$.
The vV and pV are the worst, which is also to be expected since their global accuracy is $O(dt^2)$ and there is an additional cumulative error due to the approximation. In the example above ($\theta = \frac{\pi}{4}$) the relative cumulative error is $\approx 2\%$.

4 Behaviour and initial conditions

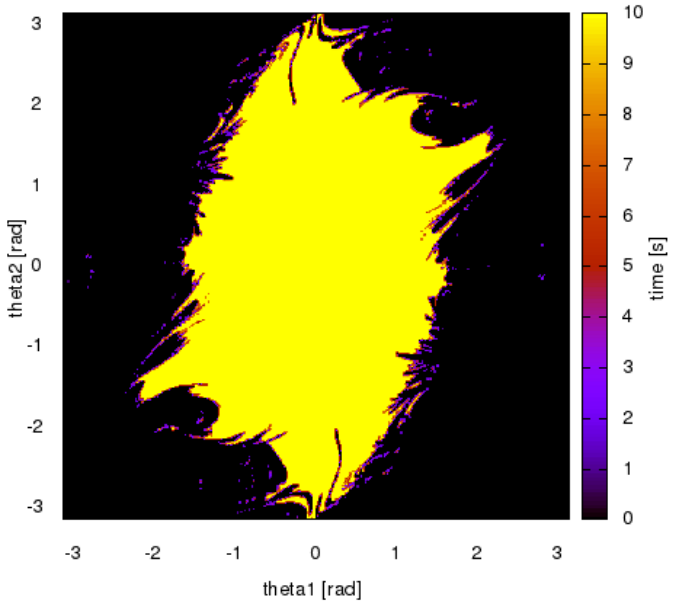
4.1 Chaoticness

For the study of the chaoticness I used RK4 and $\delta t = 0.001$, adding a fluctuation of $\delta\theta = 0.01$ radiants, which correspond to a fluctuation of $\delta\theta \approx 0.573$ degrees.

To set the tolerance I studied the difference of trajectory ($\Delta\theta$) between the two cases (with and without fluctuation) for few examples in which I considered $\theta_1 = \theta_2 = \theta$, $\dot{\theta}_1 = \dot{\theta}_2 = 0$ where $\theta = \left\{\frac{\pi}{4}, \frac{\pi}{3}, \frac{2}{5}\pi, \frac{\pi}{2}, \frac{3}{4}\pi\right\}$.

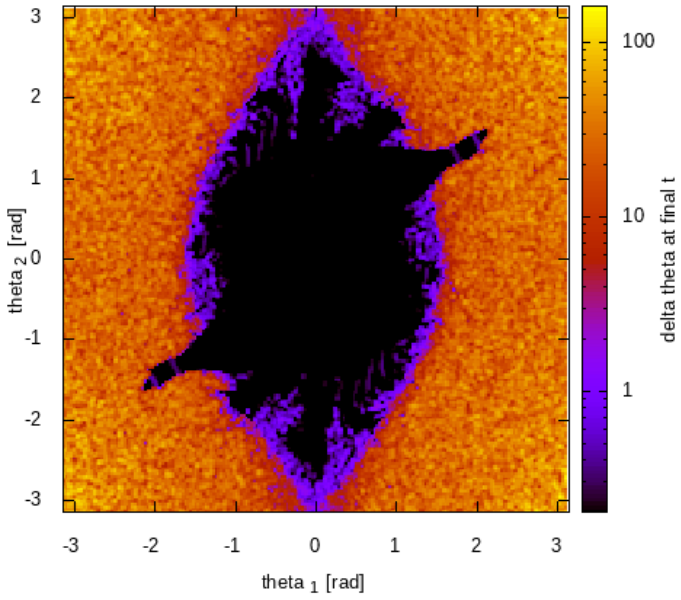
For $\theta < \frac{\pi}{2}$ the $\Delta\theta$s for both m_1, m_2 oscillate but remain roughly of an order of magnitude smaller or at most the same order of the initial value.
On the contrary, for $\theta > \frac{\pi}{2}$ the $\Delta\theta$s become roughly of an order of magnitude bigger or at most the same order of the initial value.
The larger range of values of $\Delta\theta$ is a symptom of the divergence of the trajectory, therefore it gives a sort of measurement of the chaoticness, since it is expected for the trajectories to diverge in a chaotic system (see Lyapunov exponent for the mathematical definition).

After this preliminary study I explored the different combination of $\theta_1, \theta_2 \in [0, 2\pi)$, maintaining $\dot{\theta}_1 = \dot{\theta}_2 = 0$.
Exploring 500 values for both the angles and setting $fluc = 0.01$, $tol = 0.5$, the following image is obtained, where the color maps the first time when the difference of either one of the angles ($\Delta\theta$) is bigger then the threshold (tol).



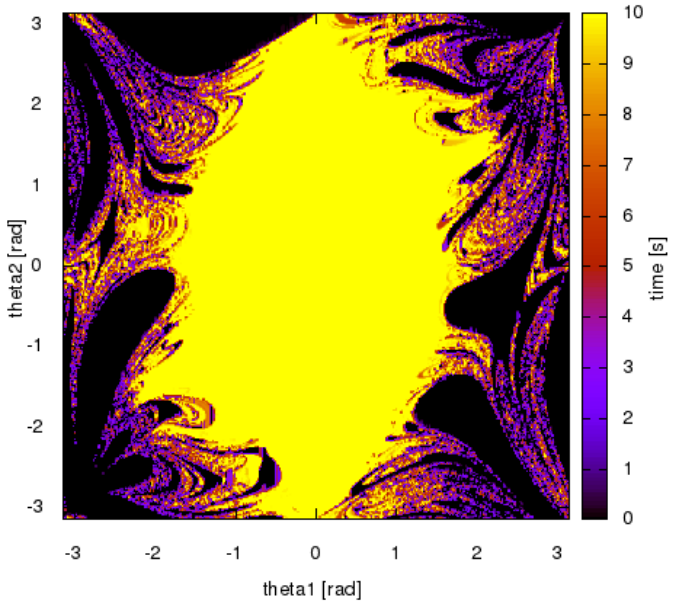
This graph delineate the non-choatich region (yellow) rather clearly but doesn't give much of a measure of the size of the fluctuation.
Therefore I studied $|\theta_2 - \theta_2^{fluc}|$ for 200 values for both the angles:

To generate the last figure the angles have not been normalized to stay in the range $[0, 2\pi)$, this is to prevent the distortion of the distances (for example: $\theta = \pi$, $\theta^{fluc} = 3\pi$ when normalized have null distance, but have followed different trajectories).



4.2 Flip of the pendulum

The flip occurs when either one of the two masses reach the maximum height.
Exploring 500 values for both the angles the following image is obtained, where the color maps the time when the first flip occurs, in the central region (yellow) this doesn't happen, incidentally this region overlaps with the non-chaotic region shown before.



5 Results

The best method to study the double pendulum, among the ones I've used, is Runge-Kutta of 4^{th} order since it has the smallest error and reaches a plateau in precision for $dt < 0.001$.

The system has a region in which the trajectory doesn't diverge and no flip occur, this region is encapsulated within:

$$\begin{cases} x < +\frac{\pi}{2}\cos\frac{y}{2} \\ x > -\frac{\pi}{2}\cos\frac{y}{2} \end{cases} \tag{11}$$

6 Code

```
#include <iostream>
#include <cmath>
#include <fstream>

#include "ode.h"

using namespace std;

void dYdt (double, double *, double *);
void acc_func(double *, double *, double *);
void solution(double , double *, double *);


void energy_coord(double , double , double , double , double &, double &, double &,
    double &, double &);

void NormAng(double &);

#define SESSION 4
// 0 = general study
// 1 = small angles
// 2 = convergence_angle
// 3 = flip and chaos
// 4 = chaos-bis (fluctuation size)

static double g_L1=1., g_m1=1., g_L2=1., g_m2=1.; // length and mass of the pendulum

// ------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------

int main (){
  double g=9.81, L1=g_L1, m1=g_m1, L2=g_L2, m2=g_m2;

  int k=0, N=1000, n_var=2, neq=2*n_var; // deq=2=degree of differential equations, var=
    varing variables, assuming deq is the same for each variable
  double Ysol[neq], Y[neq], X[n_var], V[n_var]; // X = variabiables, V = dX/dt

  double dt=0.001, t, E0, E, E_sol; // N=number of steps to be done, E=energy
  double x1, x2, y1, y2, v1, v2; // position in space (cartesian coord) and angular speed
     times length of the rod
  double x1_sol, x2_sol, y1_sol, y2_sol, v1_sol, v2_sol;



  // ------------------------------------------------------------------------------------
  // ------------------------------------------------------------------------------------
  // ------------------------------------------------------------------------------------
  #if SESSION == 0 // ------------------------- GENERAL STUDY: -------------------------
  // ------------------------------------------------------------------------------------

  double Y0[neq]={M_PI*0.25, M_PI*0.25, 0., 0.}; // Y0={theta1, theta2, v_theta1,
    v_theta2} -> initial values (border condition), R=dY/dt
  // NB: thetas are calculated from the vertical, if theta1=theta2=2*pi/4, then the
    centre and the masses form a line parallel to the ground at time t=0

  N=50*N;

  ofstream fdata;          // declare Output stream class to operate on files
  fdata.open("dati.dat"); // open output file

  cout<<"\t ##### SESSION 0 - general study #####\n"<<endl;

  // RK methods ----------------------
  for (int j=1; j<=2; j++){
    for (int i=0; i<neq; i++) Y[i]=Y0[i]; // Y must contein initial values when you start
     integrating
```

```
62      t=0; // starting time
63      for (int i=0; i<N; i++){
64
65        if (j==1) {
66          RK4Step (t, Y, dYdt, dt, neq);
67        } else {
68          RK2Step (t, Y, dYdt, dt, neq);
69        }
70        t+=dt;
71
72        for (int l=0; l<n_var; l++) NormAng(Y[l]); // this seems useless, but if I plot the
       graph I want thetas to stay in [ -pi, +pi ]
73
74        energy_coord(Y[0], Y[1], Y[2], Y[3], x1, x2, y1, y2, E);
75        if (i==0) E0=E;
76
77        // t, theta, v_theta, error, x=sin(theta), y=cos(theta)
78        fdata<< t <<" "<< Y[0] <<" "<< Y[1] <<" "<< Y[2] <<" "<< Y[3] <<" "<< E-E0 <<" "
79          << x1 <<" "<< y1 <<" "<< x2 <<" "<< y2 << endl;
80
81      }
82      cout<<"RK"<<j*2<<" method completed"<<endl;
83      fdata<<"\n"<<endl;
84    }
85
86
87
88    // Verlet methods ----------------------
89    for (int j=1; j<=2; j++){
90      for (int i=0; i<n_var; i++) {
91        X[i]=Y0[i]; // X,V must contein initial values when you start integrating
92        V[i]=Y0[i+n_var];
93      }
94      t=0; // starting time
95
96      for(int i=0; i<N; i++){
97
98        if (j==1) {
99          p_Verlet (X, V, acc_func, dt, n_var);
100       } else {
101         v_Verlet (X, V, acc_func, dt, n_var);
102       }
103       t+=dt;
104
105       for (int l=0; l<n_var; l++) NormAng(X[l]); // this seems useless, but if I plot the
       graph I want thetas to stay in [ -pi, +pi ]
106
107       energy_coord(X[0], X[1], V[0], V[1], x1, x2, y1, y2, E);
108       if (i==0) E0=E;
109
110       // t, theta, v_theta, error, x=sin(theta), y=cos(theta)
111       fdata<< t <<" "<< X[0] <<" "<< X[1] <<" "<< V[0] <<" "<< V[1] <<" "<< E-E0 <<" "
112         << x1 <<" "<< y1 <<" "<< x2 <<" "<< y2 << endl;
113
114     }
115     if (j==1) {
116       cout<<"p_Verlet method completed"<<endl;
117     } else {
118       cout<<"v_Verlet method completed"<<endl;
119     }
120     fdata<<"\n"<<endl;
121   }
122
123   fdata.close();
124
125
126
127   // ------------------------------------------------------------------------------
128   // ------------------------------------------------------------------------------
```

```
// ------------------------------------------------------------------------------------
#elif SESSION == 1 // ------------------------ SMALL ANGLES: ------------------------
// ------------------------------------------------------------------------------------

double Y0[neq]={0., 0.001, 0., 0.};

m2=m1; // solution with both equal to simplify
L2=L1;

N=3*N;

ofstream fdata;         // declare Output stream class to operate on files
fdata.open("dati_small.dat"); // open output file


cout<<"\t ##### SESSION 1 - small angles #####\n"<<endl;

// RK methods ----------------------
for (int j=1; j<=2; j++){

  for (int i=0; i<neq; i++) Y[i]=Y0[i]; // Y must contein initial values when you start integrating
  t=0; // starting time
  for (int i=0; i<N; i++){

    if (j==1) {
      RK4Step (t, Y, dYdt, dt, neq);
    } else {
      RK2Step (t, Y, dYdt, dt, neq);
    }
    t+=dt;
    solution(t, Y0, Ysol);

    for (int l=0; l<n_var; l++) NormAng(Y[l]); // this seems useless, but if I plot the graph I want thetas to stay in [ -pi, +pi ]

    energy_coord(Y[0], Y[1], Y[2], Y[3], x1, x2, y1, y2, E);
    if (i==0) E0=E;
    energy_coord(Ysol[0], Ysol[1], Ysol[2], Ysol[3], x1_sol, x2_sol, y1_sol, y2_sol, E_sol);

    fdata<< t <<" "<< dt <<" "
      << E-E_sol <<" "<< Y[0]-Ysol[0] <<" "<< Y[1]-Ysol[1] <<" "<< Y[2]-Ysol[2] <<" "<< Y[3]-Ysol[3] << " "
      << E <<" "<< E_sol <<" "<< Y[0] <<" "<< Ysol[0] <<" "<< Y[1] <<" "<< Ysol[1] <<" "<< Y[2] <<" "<< Ysol[2] <<" "<< Y[3] <<" "<< Ysol[3] << endl;
      // the last line is useful to check the sovrapposion in a qualitative/intuitive way
  }

  fdata<<"\n"<<endl;

}



// Verlet methods ----------------------
for (int j=1; j<=3; j++){

  for (int i=0; i<n_var; i++) {
    X[i]=Y0[i]; // X,V must contein initial values when you start integrating
    V[i]=Y0[i+n_var];
  }
  t=0; // starting time

  for(int i=0; i<N; i++){

    if (j==1) {
      p_Verlet (X, V, acc_func, dt, n_var);
```

```
192       } else {
193         v_Verlet (X, V, acc_func, dt, n_var);
194       }
195       t+=dt;
196       solution(t, Y0, Ysol);
197 
198       for (int l=0; l<n_var; l++) NormAng(X[l]); // this seems useless, but if I plot the
      graph I want thetas to stay in [ -pi, +pi ]
199 
200       energy_coord(X[0], X[1], V[0], V[1], x1, x2, y1, y2, E);
201       if (i==0) E0=E;
202       energy_coord(Ysol[0], Ysol[1], Ysol[2], Ysol[3], x1_sol, x2_sol, y1_sol, y2_sol,
    E_sol);
203 
204       fdata<< t <<" "<< dt <<" "
205         << E-E_sol <<" "<< X[0]-Ysol[0] <<" "<< X[1]-Ysol[1] <<" "<< V[0]-Ysol[2] <<" "<<
     V[1]-Ysol[3] << " "
206         << E <<" "<< E_sol <<" "<< X[0] <<" "<< Ysol[0] <<" "<< X[1] <<" "<< Ysol[1] <<"
    "<< V[0] <<" "<< Ysol[2] <<" "<< V[1] <<" "<< Ysol[3] << endl;
207         // the last line is useful to check the sovrapposion in a qualitative/intuitive
    way
208     }
209     fdata<<"\n"<<endl;
210 
211   }
212 
213   fdata.close();
214 
215 
216 
217   // ------------------------------------------------------------------------------------
218   // ------------------------------------------------------------------------------------
219   // ------------------------------------------------------------------------------------
220   #elif SESSION == 2 // ------------------------- CONVERGENCE: -------------------------
221   // ------------------------------------------------------------------------------------
222 
223   double Y0[neq]={0., 0.001, 0., 0.};
224 
225   int N_min=100, N_max=10000, N_step=2;
226   double t_max=10.;
227 
228   m2=m1; // solution with both equal to simplify
229   L2=L1;
230 
231 
232   ofstream fdata;         // declare Output stream class to operate on files
233   fdata.open("dati_conv.dat"); // open output file
234 
235 
236   cout<<"\t ##### SESSION 2 - convergence #####\n"<<endl;
237 
238   // RK methods ----------------------
239   for (int j=1; j<=2; j++){
240 
241     for (int N=N_min; N<N_max; N=N*N_step){
242 
243       dt = t_max/(double)N;
244       if (j==1) cout<<" --- dt="<<dt<<endl; // printing dt only the first time
245 
246       for (int i=0; i<neq; i++) Y[i]=Y0[i]; // Y must contein initial values when you
    start integrating
247       t=0; // starting time
248 
249       for (int i=0; i<N; i++){
250 
251         if (j==1) {
252           RK4Step (t, Y, dYdt, dt, neq);
253         } else {
254           RK2Step (t, Y, dYdt, dt, neq);
```

```
        }
        t+=dt;

        for (int l=0; l<n_var; l++) NormAng(Y[l]); // this seems useless, but if I plot the graph I want thetas to stay in [ -pi, +pi ]
      }

      solution(t, Y0, Ysol);

      energy_coord(Y[0], Y[1], Y[2], Y[3], x1, x2, y1, y2, E);
      energy_coord(Ysol[0], Ysol[1], Ysol[2], Ysol[3], x1_sol, x2_sol, y1_sol, y2_sol, E_sol);

      fdata<< t <<" "<< dt <<" "<< fabs(E-E_sol) <<" "
        << fabs(Y[0]-Ysol[0]) <<" "<< fabs(Y[1]-Ysol[1]) <<" "<< fabs(Y[2]-Ysol[2]) <<" "
    << fabs(Y[3]-Ysol[3]) << endl;

    }
    fdata<<"\n"<<endl;

  }



  // Verlet methods ----------------------
  for (int j=1; j<=3; j++){

    for (int N=N_min; N<N_max; N=N*N_step){

      dt = t_max/(double)N;

      for (int i=0; i<n_var; i++) {
        X[i]=Y0[i]; // X,V must contein initial values when you start integrating
        V[i]=Y0[i+n_var];
      }
      t=0; // starting time

      for(int i=0; i<N; i++){

        if (j==1) {
          p_Verlet (X, V, acc_func, dt, n_var);
        } else {
          v_Verlet (X, V, acc_func, dt, n_var);
        }
        t+=dt;

        for (int l=0; l<n_var; l++) NormAng(X[l]); // this seems useless, but if I plot the graph I want thetas to stay in [ -pi, +pi ]
      }

      solution(t, Y0, Ysol);

      energy_coord(X[0], X[1], V[0], V[1], x1, x2, y1, y2, E);
      energy_coord(Ysol[0], Ysol[1], Ysol[2], Ysol[3], x1_sol, x2_sol, y1_sol, y2_sol, E_sol);

      fdata<< t <<" "<< dt <<" "<< fabs(E-E_sol) <<" "
        << fabs(X[0]-Ysol[0]) <<" "<< fabs(X[1]-Ysol[1]) <<" "<< fabs(V[0]-Ysol[2]) <<" "
    << fabs(V[1]-Ysol[3]) << endl;

    }
    fdata<<"\n"<<endl;

  }

  fdata.close();
```

```
// ------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------
#elif SESSION == 3 // ----------------------- FLIP AND CHAOS: ------------------------
// ------------------------------------------------------------------------------------

int N_ang = 500 ; // 2 pi / N_ang = step for theta 1 and theta 2
double fluc = 0.01, tol = 0.5;
double theta_prec[2]={0., 0.}; // angle of the previous interaction

bool flip = false, ok_flip = false; // control variables for the flip data
bool rand = false, ok_rand = false; // control variables for the flip data

double Ya[neq], Ya0[neq]={0., 0., 0., 0.}; // pendulum of reference
double Yb[neq], Yb0[neq]={0., 0., 0., 0.}; // pendulum with fluctuation

N=10*N;

ofstream fdata_flip;         // declare Output stream class to operate on files
fdata_flip.open("dati_flip.dat"); // open output file

ofstream fdata_rand;         // declare Output stream class to operate on files
fdata_rand.open("dati_rand.dat"); // open output file


cout<<"\t ##### SESSION 3 - flip and chaos #####\n"<<endl;

// loop over theta1 and theta2 ----------------------
for (int a=0; a<N_ang; a++){
  Ya0[0] = (2*(double)a/(double)N_ang-1)*M_PI; // starts from -pi and ends in +pi
  Yb0[0] = Ya0[0] + fluc ;

  for (int b=0; b<N_ang; b++){
    Ya0[1] = (2*(double)b/(double)N_ang-1)*M_PI;
    Yb0[1] = Ya0[1] + fluc ;
    ok_flip = false;
    ok_rand = false;


    // ------------------------------------------------------------------------------

    // RK4 method ----------------------
    for (int i=0; i<neq; i++) { // Y must contein initial values when you start
  integrating
      Ya[i]=Ya0[i];
      Yb[i]=Yb0[i];
    }
    t=0; // starting time
    k=0; // no flip or divergence

    // ----------------------------------------

    for (int i=0; i<N; i++){
      theta_prec[0] = Ya[0];
      theta_prec[1] = Ya[1];

      RK4Step (t, Ya, dYdt, dt, neq);
      RK4Step (t, Yb, dYdt, dt, neq);
      t+=dt;

      // check if the flip occurred
      if ( fabs(Ya[0]-M_PI) < 1. or fabs(Ya[1]-M_PI) < 1. ) { // only checking when the
   angle is close to pi
        if ( (Ya[0]-M_PI)*(theta_prec[0]-M_PI) < 0. or (Ya[1]-M_PI)*(theta_prec[1]-M_PI
  ) < 0. ) { // one of mass flip
        //if ( (Ya[1]-M_PI)*(theta_prec[1]-M_PI) < 0. ) { // checking only when m2
  flips, alternative
          flip = true;
          k++;
```

```
        }
      }

      // check if it's diverging
      if ( fabs(Ya[0]-Yb[0])>tol or fabs(Ya[1]-Yb[1])>tol ) {
        rand = true;
        k++;
      }

      for (int l=0; l<n_var; l++) NormAng(Ya[l]); // this seems useless, but if I plot the graph I want thetas to stay in [ -pi, +pi ]
      for (int l=0; l<n_var; l++) NormAng(Yb[l]);

      // flip save ----------------------
      if (not ok_flip and flip) { // saves the first flip only
        fdata_flip << t <<" "<< Ya0[0] <<" "<< Ya0[1] << endl; // t, theta
        flip = false;
        ok_flip = true;
      }
      if (not ok_flip and i==N-1) fdata_flip << t <<" "<< Ya0[0] <<" "<< Ya0[1] << endl; // flip didn't occur

      // rand save ----------------------
      if (not ok_rand and rand) {
        fdata_rand << t <<" "<< Ya0[0] <<" "<< Ya0[1] << endl; // t, theta
        rand = false;
        ok_rand = true;
      }
      if (not ok_rand and i==N-1) fdata_rand << t <<" "<< Ya0[0] <<" "<< Ya0[1] << endl;

      // if both flip and divergence happened already, then skip to the next iteration to save time
      if (k==2) continue;

    }

    // ------------------------------------------------------------------------------
  }
  fdata_flip<<endl;
  fdata_rand<<endl;

}

fdata_flip.close();
fdata_rand.close();




// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------
// ------------------------------------------------------------------------------
#elif SESSION == 4 // ---------------------------- CHAOS: ----------------------------
// ------------------------------------------------------------------------------

int N_ang = 200, N_tol=200; // 2 pi / N_ang = step for theta 1 and theta 2
double fluc = 0.01, delta_mean[n_var]={0.,0.};


double Ya[neq], Ya0[neq]={0., 0., 0., 0.}; // pendulum of reference
double Yb[neq], Yb0[neq]={0., 0., 0., 0.}; // pendulum with fluctuation

N=25*N;

ofstream fdata_chaos;         // declare Output stream class to operate on files
fdata_chaos.open("dati_chaos.dat"); // open output file


```

```
448   cout<<"\t ##### SESSION 4 - chaos (fluctuation size) #####\n"<<endl;
449
450   // loop over theta1 and theta2 ----------------------
451   for (int a=0; a<N_ang; a++){
452     Ya0[0] = (2*(double)a/(double)N_ang-1)*M_PI; // starts from -pi and ends in +pi
453     Yb0[0] = Ya0[0] + fluc ;
454
455     for (int b=0; b<N_ang; b++){
456       Ya0[1] = (2*(double)b/(double)N_ang-1)*M_PI;
457       Yb0[1] = Ya0[1] + fluc ;
458
459
460       // ------------------------------------------------------------------------------
461
462       // RK4 method ----------------------
463       for (int i=0; i<neq; i++) { // Y must contein initial values when you start
      integrating
464         Ya[i]=Ya0[i];
465         Yb[i]=Yb0[i];
466       }
467       t=0; // starting time
468       k=0; // no flip or divergence
469
470       // -----------------------------------------
471
472       for (int i=0; i<N; i++){
473         RK4Step (t, Ya, dYdt, dt, neq);
474         RK4Step (t, Yb, dYdt, dt, neq);
475         t+=dt;
476
477         // check if it's diverging
478         if (i>N-N_tol) { //
479           delta_mean[0] += fabs(Ya[0]-Yb[0]);
480           delta_mean[1] += fabs(Ya[1]-Yb[1]);
481         }
482       }
483
484       delta_mean[0] /= N_tol*M_PI;
485       delta_mean[1] /= N_tol*M_PI;
486
487       //fdata_chaos << delta_mean[0]+delta_mean[1])*0.5 <<" "<< Ya0[0] <<" "<< Ya0[1] <<
      endl;
488       fdata_chaos << Ya0[0] <<" "<< Ya0[1]  <<" "
489       << delta_mean[0] <<" "<< delta_mean[1] <<" "<< (delta_mean[0]+delta_mean[1])*0.5 <<
       " "
490       << Ya[0] << " " << Ya[1] << " " << Ya[2] << " " << Ya[3] << " "
491       << Yb[0] << " " << Yb[1] << " " << Yb[2] << " " << Yb[3] << " " << endl;
492       // ------------------------------------------------------------------------------
493     }
494     fdata_chaos<<endl;
495   }
496   fdata_chaos.close();
497
498
499   #endif
500 }
501
502
503
504
505
506
507
508
509
510 // ------------------------------------------------------------------------------
511 // ------------------------------------------------------------------------------
512 // ------------------------------------------------------------------------------
513
```

```
514
515 // differential function for Runge-Kutta
516 void dYdt (double t, double *Y, double *R){
517   // Compute the right-hand side of the ODE dy/dt = -t*y
518   // Y contiene x0, y0, vx0, vy0
519   double theta1 = Y[0], theta2 = Y[1]; // coord (angle)
520   double d_theta1 = Y[2], d_theta2= Y[3]; // speed
521   double g=9.81, L1=g_L1, m1=g_m1, L2=g_L2, m2=g_m2;
522
523   R[0]=d_theta1;
524   R[1]=d_theta2;
525
526   R[2] = ( // dd_theta1 (acc)
527     -g*(2*m1+m2)*sin(theta1)
528     -m2*g*sin(theta1-2*theta2)
529     -2*sin(theta1-theta2)*m2*(
530       d_theta2*d_theta2 *L2
531       +d_theta1*d_theta1 *L1*cos(theta1-theta2)
532     ))/(
533       L1*(2*m1+m2-m2*cos( 2*theta1-2*theta2))
534     );
535
536   R[3] = ( // dd_theta2
537     2*sin(theta1-theta2)*(
538       d_theta1*d_theta1 *L1*(m1+m2)
539       +g*(m1+m2)*cos(theta1)
540       + d_theta2*d_theta2*L2*m2*cos(theta1-theta2)
541     ))/(
542       L2*(2*m1+m2-m2*cos( 2*theta1-2*theta2))
543     );
544
545 }
546
547
548 // ------------------------------------------------------------------------------------
549 // differential function for Verlet
550 void acc_func(double *X, double *V, double *a){
551   double theta1 = X[0], theta2 = X[1]; // coord (angle)
552   double d_theta1 = V[0], d_theta2= V[1]; // speed
553   double g=9.81, L1=g_L1, m1=g_m1, L2=g_L2, m2=g_m2;
554
555   a[0] = ( // dd_theta1 (acc)
556     -g*(2*m1+m2)*sin(theta1)
557     -m2*g*sin(theta1-2*theta2)
558     -2*sin(theta1-theta2)*m2*(
559       d_theta2*d_theta2 *L2
560       +d_theta1*d_theta1 *L1*cos(theta1-theta2)
561     ))/(
562       L1*(2*m1+m2-m2*cos( 2*theta1-2*theta2))
563     );
564
565   a[1] = ( // dd_theta2
566     2*sin(theta1-theta2)*(
567       d_theta1*d_theta1 *L1*(m1+m2)
568       +g*(m1+m2)*cos(theta1)
569       + d_theta2*d_theta2*L2*m2*cos(theta1-theta2)
570     ))/(
571       L2*(2*m1+m2-m2*cos( 2*theta1-2*theta2))
572     );
573 }
574
575
576 // ------------------------------------------------------------------------------------
577 // analitic solution in small angles regime (supposing L1=L2 and m1=m2)
578 void solution(double t, double *Y0, double *Ysol){
579
580   double g=9.81, L1=g_L1, m1=g_m1, L2=g_L2, m2=g_m2;
581
582   double omega1 = sqrt((2-sqrt(2))*g/L1);
```

```
583   double omega2 = sqrt((2+sqrt(2))*g/L1);
584
585   double A1 = (sqrt(2)*Y0[0] + Y0[1])*0.5;
586   double A2 = (sqrt(2)*Y0[0] - Y0[1])*0.5;
587
588   Ysol[0] =  ( A1*cos(omega1*t) + A2*cos(omega2*t) )/sqrt(2);
589   Ysol[1] =    A1*cos(omega1*t) - A2*cos(omega2*t);
590
591   Ysol[2] = - ( omega1*A1*sin(omega1*t) + omega2*A2*sin(omega2*t) )/sqrt(2);
592   Ysol[3] = - ( omega1*A1*sin(omega1*t) - omega2*A2*sin(omega2*t) );
593
594 }
595
596
597 // ------------------------------------------------------------------------------------
598 // calculate the energy and the cartesian coordinates
599 // recurrent operation: this will help with readability and lower the chance of error
600 void energy_coord(double theta1, double theta2, double omega1, double omega2, double &x1,
        double &x2, double &y1, double &y2, double &E){
601   double g=9.81, L1=g_L1, m1=g_m1, L2=L1, m2=m1;
602   double v1, v2;
603
604   x1 =  L1*sin(theta1);
605   y1 = -L1*cos(theta1);
606   x2 = x1 +L2*sin(theta2);
607   y2 = y1 -L2*cos(theta2);
608
609   v1 = L1*omega1;
610   v2 = L2*omega2;
611
612   E = 0.5*((m1+m2)*v1*v1 + m2*v2*v2) +m2*cos(theta1-theta2)*v1*v2 +g*(m1*y1 + m2*y2);
613 }
614
615
616 // ------------------------------------------------------------------------------------
617 // normalize angles so they stay in [-pi, +pi]
618 void NormAng(double &ang){
619   if (ang> M_PI) ang-= 2*M_PI;
620   if (ang<-M_PI) ang+= 2*M_PI;
621 }
```