Doppio pendolo

Carola Maria Caivano

October 26, 2022

Abstract

In questo progetto si è andati a studiare il comportamento dinamico di un pendolo doppio, cercando di risolvere le sue equazioni del moto attraverso alcuni metodi dell'analisi numerica. Il sistema è risultato sensibile sia alle condizioni iniziali che al metodo di risoluzione scelto.

1 Introduzione

Il doppio pendolo è un sistema costituito da due pendoli attaccati uno all'estremità dell'altro. Possiamo descrivere questo sistema come un sistema caotico, dal momento che il suo comportamento dinamico è molto sensibile a piccole variazioni delle condizioni iniziali. Come conseguenza il moto di un doppio pendolo è molto difficile da prevedere.
Per studiare il comportamento del doppio pendolo, si è cercata una soluzione approssimata delle equazioni differenziali ordinarie, che ne descrivono il moto, utilizzando quattro diversi metodi: il metodo di Eulero, Runge Kutta del quarto ordine, position-Verlet e velocity-Verlet.

2 Teoria

2.1 Equazioni del moto

Se consideriamo il nostro doppio pendolo esso è formato da due pendoli, costituiti da due masse m_1 e m_2 attaccate a due asticelle l_1 e l_2, che nel nostro studio abbiamo considerato prive di massa.

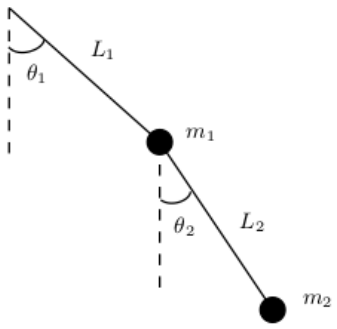

Figure 1: doppio pendolo

La posizione delle due masse è data da:

$$\begin{cases} x_1 = l_1 \sin\theta_1 \\ x_2 = x_1 + l_2 \sin\theta_2 \\ y_1 = -l_1 \cos\theta_1 \\ y_2 = y_1 - l_2 \cos\theta_2 \end{cases} . \tag{1}$$

Gli angoli θ_1 e θ_2, che variano in funzione del tempo, rappresentano gli angoli che le due asticelle del doppio pendolo formano con la verticale.
Si possono calcolare le velocità delle masse: $u_1 = \frac{\partial x_1}{\partial t} = \dot{\theta_1} l_1 \cos(\theta_1), v_1 = \frac{\partial y_1}{\partial t} = \dot{\theta_1} l_1 \sin(\theta_1), u_2 = \frac{\partial x_2}{\partial t} = \dot{\theta_2} l_2 \cos(\theta_2) + u_1, v_2 = \frac{\partial y_2}{\partial t} = \dot{\theta_2} l_2 \sin(\theta_2) + v_1$, da cui si può calcolare l'energia cinetica e l'energia potenziale (dove l'energia potenziale V=0 è associata alla configurazione $(\theta_1, \theta_2) = (0, 0)$)

$$\begin{cases} T_1 = \frac{1}{2} m_1 (u_1^2 + v_1^2) \\ T_2 = \frac{1}{2} m_2 (u_2^2 + v_2^2) \\ V_1 = m_1 g y_1 \\ V_2 = m_2 g y_2, \end{cases} \tag{2}$$

da cui si ottiene:

$$\begin{cases} T = T_1 + T_2 = \frac{1}{2} m_1 l_1^2 \dot{\theta_1}^2 + \frac{1}{2} \left(m_2 l_1^2 \dot{\theta_1}^2 + l_2^2 \dot{\theta_2}^2 + 2 l_1 l_2 m_2 \dot{\theta_1} \dot{\theta_2} \cos(\theta_1 - \theta_2) \right) \\ V = V_1 + V_2 = -m_1 g l_1 \cos(\theta_1) - m_2 g l_1 \cos(\theta_1) - m_2 g l_2 \cos(\theta_2) \end{cases} . \tag{3}$$

Si può così calcolare la Lagrangiana come

$$\mathcal{L} = T - V, \tag{4}$$

e attraverso le equazioni di Eulero-Lagrange

$$\frac{\partial \mathcal{L}}{\partial \theta_k} - \frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{\theta_k}} = 0 \qquad (k = 1, 2), \tag{5}$$

si ottengono le equazioni del moto:

$$\begin{cases} \ddot{\theta}_1 = -\dfrac{(2m_1 + m_2) g \sin(\theta_1) + m_2 g \sin(\theta_1 - 2\theta_2) + 2 m_2 \sin(\theta_1 - \theta_2) \left[l_2 \dot{\theta}_2^2 + l_1 \dot{\theta}_1^2 \cos(\theta_1 - \theta_2) \right]}{l_1 \left[2m_1 + m_2 - m_2 \cos(2(\theta_1 - \theta_2)) \right]} \\ \ddot{\theta}_2 = \dfrac{2 \sin(\theta_1 - \theta_2) \left[l_1 (m_1 + m_2) \dot{\theta}_1^2 + g(m_1 + m_2) \cos\theta_1) + m_2 l_2 \dot{\theta}_2^2 cos(\theta_1 - \theta_2) \right]}{l_2 \left[2m_1 + m_2 - m_2 \cos(2(\theta_1 - \theta_2)) \right]} \end{cases} . \tag{6}$$

Questo è un sistema di equazioni differenziali ordinarie al secondo ordine, che può essere ridotto ad un sistema al primo ordine:

$$\begin{cases} \dot{\theta_1} = \omega_1 \\ \dot{\omega_1} = \dfrac{-g(2m_1 + m_2) sin\theta_1 - m_2 g \sin(\theta_1 - 2\theta_2) - 2 sin(\theta_1 - \theta_2) m_2 (\omega_2^2 l_2 + \omega_1^2 l_1 \cos(\theta_- \theta_2))}{l1(2m_1 + m_2 - m_2 cos(2\theta_1 - 2\theta_2))} \\ \dot{\theta_2} = \omega_2 \\ \dot{\omega_2} = \dfrac{2 \sin(\theta_1 - \theta_2)(\omega_1^2 l_1 (m_1 + m_2) + g(m_1 + m_2) \cos\theta_1 + \omega_2^2 l_2 m_2 \cos(\theta_1 - \theta_2))}{l_2 (2m_1 + m_2 - m_2 \cos(2\theta_1 - 2\theta_2))}. \end{cases} \tag{7}$$

2.2 Analisi per piccoli angoli

Per piccoli angoli è possibile calcolare la soluzione analitica. Linearizzando la lagrangiana otteniamo le seguenti equazioni del moto:

$$\begin{cases} \ddot{\theta}_1 = g\theta_2 - 2\frac{g}{L}\theta_1 \\ \ddot{\theta}_2 = 2g(\theta_1 - \theta_2) \end{cases} . \tag{8}$$

Risolvendole si ottengono le seguenti equazioni:

$$\begin{cases} \theta_1(t) = \frac{1}{\sqrt{2}} (\Theta_+ \cos(\omega_+ t) + \Theta_- \cos(\omega_- t)) \\ \theta_2(t) = \Omega_+ \cos(\omega_+ t) - \Theta_- \cos(\omega_- t)) \end{cases} \text{, dove } \begin{cases} \omega_+ = \sqrt{\frac{g}{L}(2 - \sqrt{2})} \\ \omega_- = \sqrt{\frac{g}{L}(2 + \sqrt{2})} \end{cases} \text{ e } \begin{cases} \Theta_+ = \frac{1}{2}(\sqrt{2}\theta_{1,0} + \theta_{2,0}) \\ \Theta_- = \frac{1}{2}(\sqrt{2}\theta_{1,0} - \theta_{2,0}) \end{cases}$$

3 Metodi

Dal momento che esiste una soluzione analitica per le equazioni del moto solo per piccoli angoli, si sono andati ad utilizzare diversi metodi per risolverle numericamente, in modo da verificare i risultati ottenuti (i diversi metodi dovrebbero restituirci soluzioni simili) e in modo da individuare il metodo che meglio descrive il nostro sistema.

3.1 Metodo di Eulero

Il primo metodo utilizzato è il metodo di Eulero, che partendo da una funzione iniziale $y(t)$ ci permette di calcolare:

$$y(t + \Delta t) = y(t) + \dot{y}(t)\Delta t, \tag{9}$$

con Δt che rappresenta un piccolo incremento del tempo.
Questo metodo ha un errore di troncamento che è $O(h^2)$, quindi è un metodo del primo ordine.

3.2 RK4

Il secondo metodo utilizzato è il metodo di Runge-Kutta del quarto ordine, che valuta la derivata della funzione quattro volte per ogni step Δt.
Possiamo riscrivere il nostro sistema come:

$$\frac{dY}{dt} = R(t, Y), \tag{10}$$

dove Y contiene le variabili $(\theta_1, \theta_2, \omega_1, \omega_1)$ e $\frac{dY}{dt}$ è il vettore che contiene le derivate temporali $(\dot{\theta_1}, \dot{\theta_2}, \dot{\omega_1}, \dot{\omega_2})$.
Considerando la condizione iniziale $Y(0) = Y_0$ e l'intervallo $R \supset [0, t_e]$ (il dominio di integrazione), si divide l'intervallo in N steps larghi h.
Nei metodi di Runge-Kutta si va a calcolare Y_{n+1} usando Y_n. In paticolare nel metodo del quarto ordine si ha che:

$$\begin{aligned} Y_{n+1} &= Y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= R(t_n, Y_n) \\ k_2 &= R(t_n + \tfrac{h}{2}, Y_n + \tfrac{h}{2}k_1) \\ k_3 &= R(t_n + \tfrac{h}{2}, Y_n + \tfrac{h}{2}k_2) \\ k_4 &= R(t_n + h, Y_n + hk_3). \end{aligned} \tag{11}$$

Questo metodo ha un errore di troncamento che è $O(h^5)$, quindi è un metodo del quarto ordine.

3.3 Velocity-Verlet

Definiamo l'incremento come $h = \delta t$ e i vettori contenenti le variabili come:

$$\begin{aligned} X &= (\theta_1, \theta_2) \\ V &= (\dot{\theta_1}, \dot{\theta_2}) \\ A &= (\ddot{\theta_1}, \ddot{\theta_2}), \end{aligned} \tag{12}$$

dove A(x) è l'accelerazione.
Nel metodo velocity-Verlet si ha che:

$$\begin{aligned} V_{n+\frac{1}{2}} &= V_n + \tfrac{h}{2}A(X_n) \\ X_{n+1} &= X_n + hV_{n+\frac{1}{2}} \\ V_n &= V_{n+\frac{1}{2}} + \tfrac{h}{2}A(X_{n+1}) \end{aligned} \quad . \tag{13}$$

Questo metodo ha un errore di troncamento che è $O(h^3)$, quindi è un metodo del secondo ordine.

3.4 Position-Verlet

In position-Verlet si ha che:

$$\begin{aligned} X_{n+\frac{1}{2}} &= X_n + \tfrac{h}{2} V_n \\ V_{n+1} &= V_n + hA(X_{n+\frac{1}{2}}) \, . \\ X_n &= X_{n+\frac{1}{2}} + \tfrac{h}{2} A(V_{n+1}) \end{aligned} \tag{14}$$

Questo metodo ha un errore di troncamento che è $O(h^3)$, quindi è un metodo del secondo ordine.
Sia velocity-Verlet, che position-Verlet funzionano solo per piccoli angoli, quando (Eq. 6) dipende solo da θ e non da $\dot{\theta}$.

3.5 Studio di compatibilità con la soluzione

Si vuole studiare l'accuratezza degli algoritmi, per farlo si studia l'errore sull'angolo θ_1 andando a variare t e mantenendo dt costante. Si indica la soluzione dei diversi metodi come $\tilde{\theta_1}$ e quella analitica come θ_1, l'errore è definito come $\Delta\theta = \tilde{\theta_1} - \theta_1$.

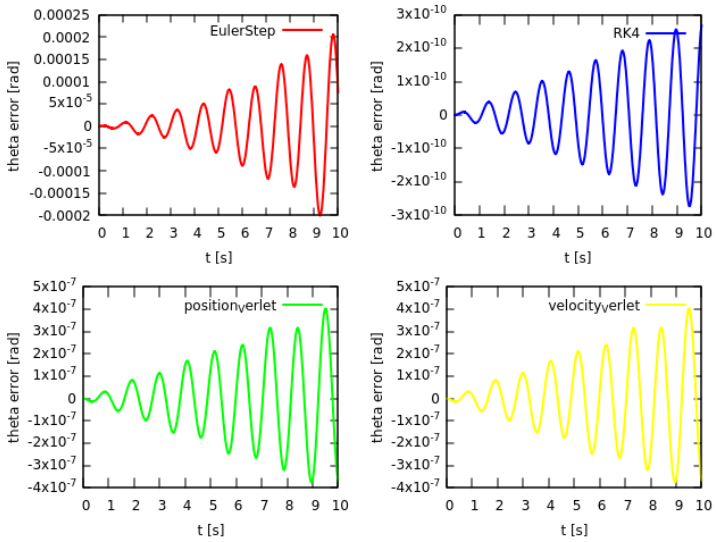

Figure 2: Condizioni iniziali: $\theta_1 = 0\ rad$, $\omega_1 = 0\ rad/s, \theta_2 = 10^{-5}\ rad$, $\omega_2 = 0\ rad/s$

Dopo 10^3 iterazioni l'errore $\Delta\theta$ per RK4 è dell'ordine di 10^{-10}, mentre per position Verlet e velocity Verlet è dell'ordine di 10^{-7}, possiamo quindi dire che questi algoritmi descrivono in modo accurato il sistema per angoli piccoli.

3.6 Studio della convergenza

Per studiare la converegenza dei nostri algoritmi si sono andati a testare i modelli su un sistema più elementare, quello del pendolo semplice. Nel caso di approssimazione per piccoli angoli, l'equazione del moto che deve essere risolta è $\ddot{\theta} = -\frac{g}{L}\sin(\theta)$ e la soluzione analitica è $\theta(t) = \theta_0 \cos(\sqrt{\frac{g}{L}}t)$.
Il seguente plot si ottiene dividendo ad ogni iterazione $dt \to \frac{dt}{2}$ e definendo l'errore su θ come $\Delta\theta = |\tilde{\theta} - \theta|$, dove $\tilde{\theta}$ è la soluzione analitica.
Per tale studio si è andati ad utilizzare come condizioni iniziali $\theta = 0.08$ e $\dot{\theta_1} = 0$.

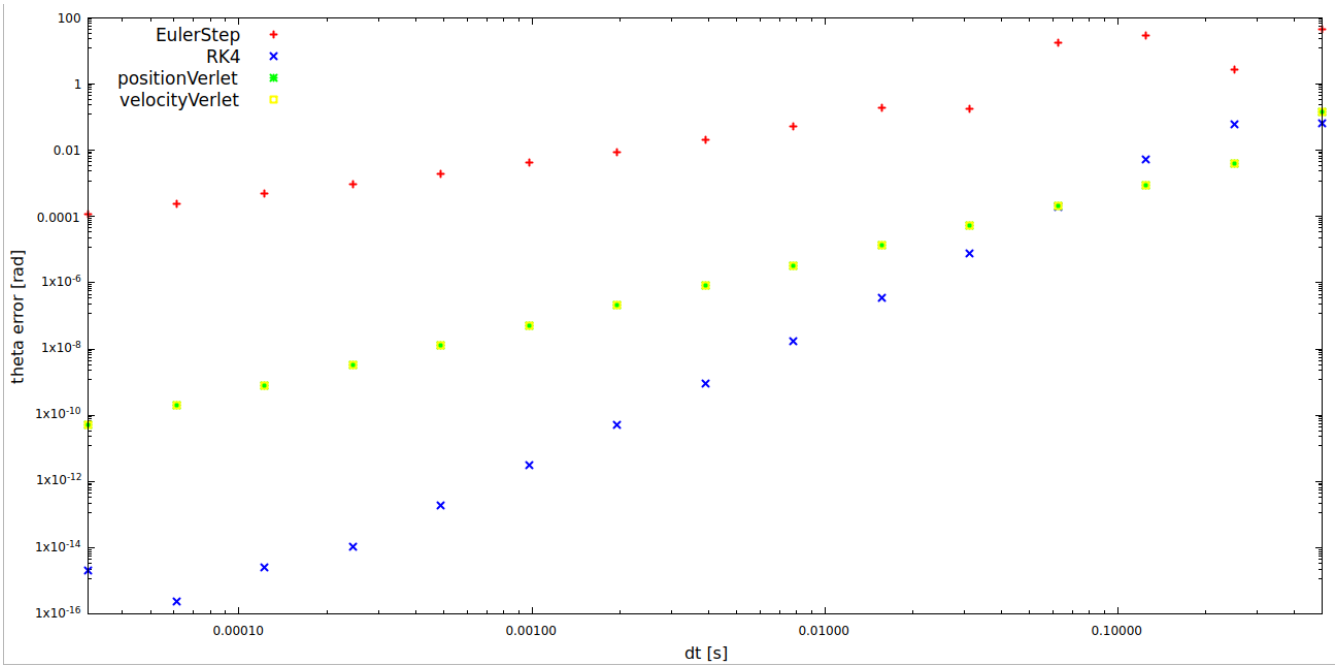

Figure 3: Convergenza degli algoritmi in scala logaritmica

Per verificare la convergenza si è andati a calcolare il coefficiente angolare in scala logaritmica per ognuno dei metodi. Per RK4 si è ottenuto un coefficiente di ~ 4.1, possiamo dire che il nostro algoritmo implenta un metodo al quarto ordine. Per Eulero si è ottenuto un coefficiente di ~ 1.16, l'algoritmo implementa un metodo al primo ordine. Per Position Verlet e Velocity Verlet si è ottenuto un coefficiente angolare di ~ 1.96, gli algoritmi implementano un metodo al secondo ordine.

4 Risultati

4.1 Energia

Si è andati a studiare come l'energia totale varia nel tempo. L'energia del sistema è:

$$E = \frac{1}{2}(m_1 + m_2)v_1^2 + \frac{1}{2}m_2v_2^2 + m_2v_1v_2\cos(\theta_1 - \theta_2) + g[m_1(l_1 + l_2 + y_1) + m_2(l_1 + l_2 + y_2)], \qquad (15)$$

dove $v_1 = l_1\omega_1$ e $v_2 = l_2\omega_2$. Dal momento che abbiamo una forza conservativa che agisce sul nostro sistema, la forza di gravità, e il sistema non è smorzato, l'energia dovrebbe rimanere costante.
Per quanto riguarda i metodi di Verlet, essi dovrebbero funzionare correttamente solo nel caso in cui la nostra forza dipenda dalla velocità. Si è visto precedentemente come in un regime non caotico non sia presente una differenza sostanziale tra le traiettorie di questi metodi e gli altri, si potrebbero quindi interpretare queste piccole differenze come fluttuazioni dovute all'utilizzo di un differente metodo.
Il fatto che questo metodo non sia corretto per descrivere il nostro sistema è però ben visibile dal grafico sull'energia, dove si presentano fluttuazioni molto ampie rispetto a RK4.

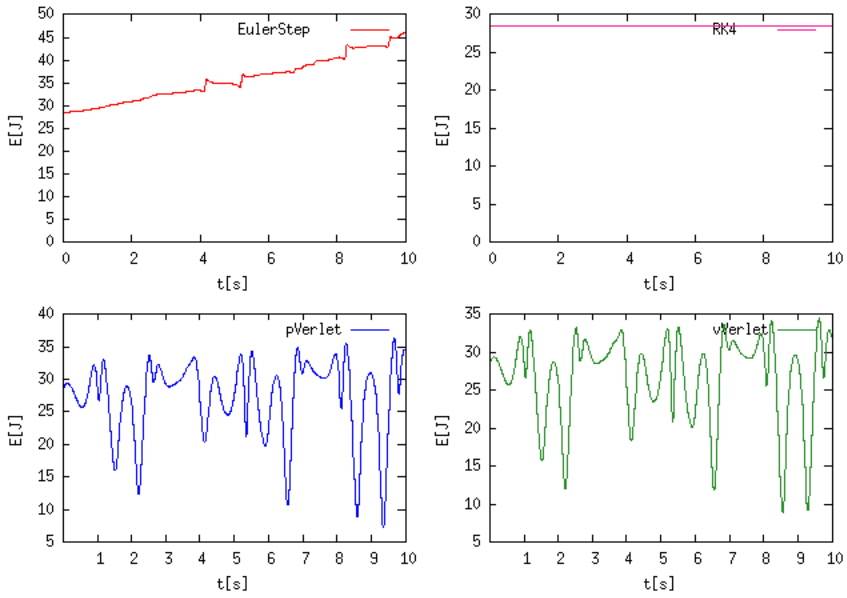

Figure 4: Condizioni iniziali: $\theta_1 = \theta_2 = \pi/6\ rad$, $\omega_1 = \omega_2 = 0\ rad/s$, $m_1 = m_2 = 1\ kg$ e $l_1 = l_2 = 1.0\ m$

Si è andati a fare un'analisi quantitativa dell'errore sull'energia per il metodo RK4:

$$\Delta E = \frac{E(t) - E_{in}}{E_{in}}, \tag{16}$$

dove E_{in} è l'energia iniziale del sistema ed $E(t)$ è quella calcolata al tempo t.
Andando a studiare l'andamento di ΔE al variare di t, si ottiene il seguente grafico:

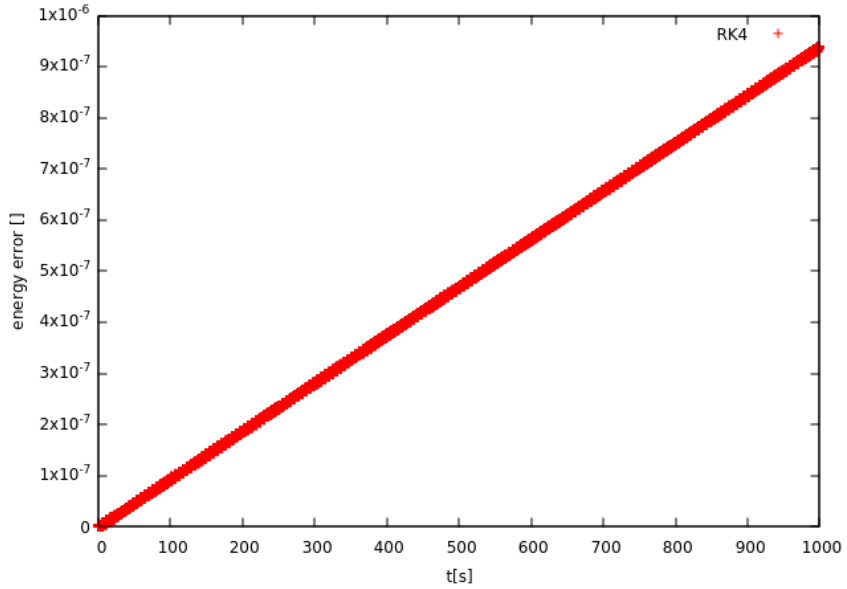

Figure 5: Condizioni iniziali: $\theta_1 = 0, \theta_2 = 10^{-5}$, $\omega_1 = \omega_2 = 0\ rad/s$, $m_1 = m_2 = 1\ kg$ e $l_1 = l_2 = 1.0\ m$. dt=0.01

L'errore relativo aumenta con il procedere dell'integrazione, ma dato che dopo 10^6 passi l'errore sull'energia ha un valore ancora piccolo ($\Delta E \sim 10^{-6}$), possiamo dire che per RK4 l'energia è conservata. Inoltre dato il piccolo errore ottenuto per l'energia, si vede che dt=0.01 è un buono step, quindi verrà utilizzato da qui in avanti.

4.2 Studio del comportamento caotico

È possibile studiare il comportamento caotico del sistema guardando come piccoli cambiamenti sulle condizioni iniziali influenzano il moto.
Si è visto che se entrambi gli angoli sono $\theta_1, \theta_2 < \pi/2$ e $\omega_1 = \omega_2 = 0$, il sistema tende a non essere caotico. Si è andati a studiare la traiettoria nello spazio delle fasi per tre diverse condizioni iniziali usando RK4, $\theta_1 = \theta_2 = \theta$ dove $\theta = \{\frac{\pi}{2}, \frac{\pi}{2} + 0.01, \frac{\pi}{2} - 0.01\}$. Si vede come all'inizio le curve si sovrappongono e dopo poco iniziano a distaccarsi, piccole deviazioni sulle condizioni iniziali sono esponenzialmente amplificate con il trascorrere del tempo.
Si sono andate ad osservare le traiettorie utilizzando come step dt=0.01 e dt=0.001 e sono state ottenute le stesse soluzioni. Le nostre soluzioni non dipendono quindi da dt.

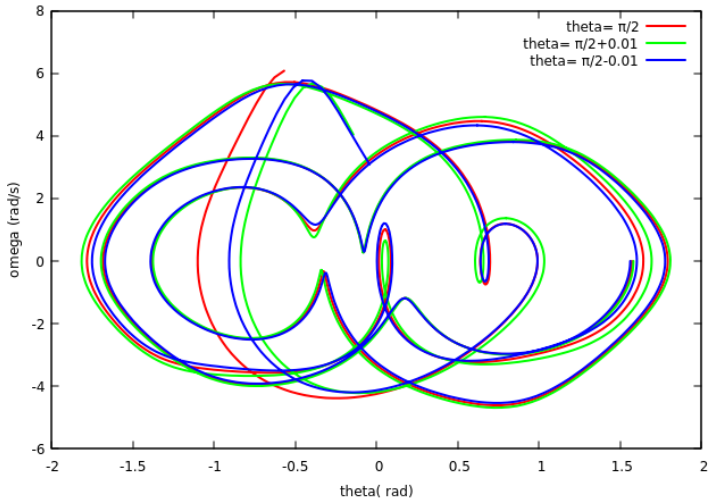

Figure 6: dt=0.01

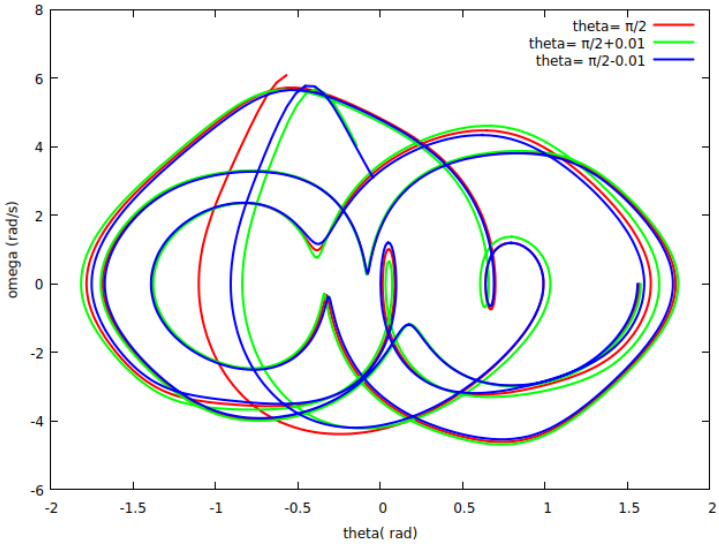

Figure 7: dt=0.001

4.3 Flip

Il pendolo si capovolge quando l'angolo arriva a π e prosegue, ovvero se $(\theta_2^n - \pi) \cdot (\theta_2^{n+1} - \pi) < 0$.
Si è andati a studiare il fenomeno assegnando agli angoli valori compresi nell'intervallo $[-\pi, \pi]$. Ogni volta che il pendolo si è capovolto si è andati a salvare t_{flip} (tempo di flip).
Di seguito si riporta il grafico con i risultati ottenuti, dove il tempo di flip si trova su una griglia, dove il giallo indica il caso in cui il pendolo non si è capovolto.

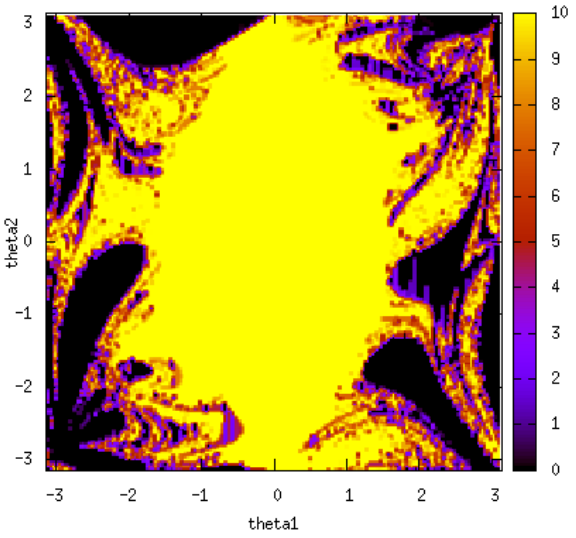

Figure 8: Condizioni iniziali $(\theta_1, \theta_2, 0, 0)$, dove l'unità di misura per gli angoli è in radianti

5 Conclusioni

Tra tutti i metodi utilizzati, Runge-Kutta del quarto ordine è risultato essere il migliore per integrare le equazioni del moto del doppio pendolo, dal momento che è l'unico che mantiene l'energia costante.
Dallo studio delle traiettorie nello spazio delle fasi e del tempo di flip, si è visto come il doppio pendolo sia un sistema caotico.

6 Codice

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


#define l1 1.
#define l2 1.
#define g 9.81
#define m1 1.
#define m2 1.
#define eta g/l1

using namespace std;

void EulerStep (double, double *, void(*)(double, double *, double*), double, int);
void RK4Step(double, double*, void(*RHSFunc)(double, double*, double*), double, int);
void position_Verlet(double *, double *, void (*)(double*, double*,double*), double, int);
void velocity_Verlet(double *, double *,void (*)(double *, double *, double *), double, int);

void dYdt(double, double *, double *);
void accel(double *, double *, double*);
void solution(double , double *, double *);
void NormAng(double &);

int main(){
  double t, dt=0.01;
  int n_var=2, neq=2*n_var;
  double x1, x2, y1, y2, v1, v2;    //position in space and angular speed
  double E, E_in, N=1000;
  double Y[neq], Y0[neq]={0.0001,0.,0.,0.}; // initial values
  double X[n_var], V[n_var];
  double a1,a2,b1,b2;
  double theta1_true, theta2_true; //analytic solutions

  a1=(Y0[0]*sqrt(2)+Y0[1])/(2*sqrt(2));
  a2=(Y0[0]*sqrt(2)-Y0[1])/(2*sqrt(2));

  b1=sqrt(eta)*sqrt(2-sqrt(2));
  b2=sqrt(eta)*sqrt(2+sqrt(2));

  ofstream fdata, data, data_ang;
  fdata.open("doublependulum.dat");
  data.open("convergence.dat");
  data_ang.open("angle_error.dat");

  //initial values
  Y[0]=Y0[0];         //theta1
  Y[1]=Y0[1];         //theta2
  Y[2]=Y0[2];         //omega1
  Y[3]=Y0[3];         //omega2

   //-----------------Euler-method-----------------------------------
  t=0.;
  E=0.;


  fdata << t << "  " << Y[0] << "  " << Y[1]<<"  " << Y[2] << "  " << Y[3]<<" "<<E<<" "<<
    x1<<" "<<y1<<" "<<x2<<" "<<y2<<" "<<endl;

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

    EulerStep (t, Y, dYdt, dt, neq);
    t+=dt;

    x1=l1*sin(Y[0]);
```

```
    x2=x1+l2*sin(Y[1]);
    y1=-l1*cos(Y[0]);
    y2=y1-l2*cos(Y[1]);

    E = 0.5*((m1+m2)*l1*Y[2]*l1*Y[2] + m2*l2*Y[3]*l2*Y[3]) +m2*( cos(Y[0])*cos(Y[1]) +
    sin(Y[0])*sin(Y[1]) )*
    l1*Y[2]*l2*Y[3]   +g*( m1*(l1+l2+y1) + m2*(l1+l2+y2) );

    theta1_true=a1*cos(b1*t)+a2*cos(b2*t);
    theta2_true=sqrt(2)*(a1*cos(b1*t)-a2*cos(b2*t));

    data_ang<<t<<" "<<Y[0]-theta1_true<<endl;
    fdata << t << "  " << Y[0] << "  " << Y[1]<<"  " << Y[2] << "  " << Y[3]<<" "<<E<<" "
    <<x1<<" "<<y1<<" "<<x2<<" "<<y2<<" "<<endl;
  }

  fdata<<"\n"<<endl;
  fdata<<"\n"<<endl;
  data_ang<<"\n"<<endl;


  //--------------RK4--method---------------------------
  //initial values
  Y[0]=Y0[0];         //theta1
  Y[1]=Y0[1];         //theta2
  Y[2]=Y0[2];         //omega1
  Y[3]=Y0[3];         //omega2
  t=0.;
  E=0.;
  x1=l1*sin(Y[0]);
  x2=x1+l2*sin(Y[1]);
  y1=-l1*cos(Y[0]);
  y2=y1-l2*cos(Y[1]);
  E_in = 0.5*((m1+m2)*l1*Y[2]*l1*Y[2] + m2*l2*Y[3]*l2*Y[3]) +m2*( cos(Y[0])*cos(Y[1]) +
    sin(Y[0])*sin(Y[1]) )*
  l1*Y[2]*l2*Y[3]   +g*( m1*(l1+l2+y1) + m2*(l1+l2+y2) );

  fdata << t << "  " << Y[0] << "  " << Y[1]<<"  " << Y[2] << "  " << Y[3]<<" "<<E<<" "<<
    x1<<" "<<y1<<" "<<x2<<" "<<y2<<" "<<endl;

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

    RK4Step (t, Y, dYdt, dt, neq);
    t+=dt;

    x1=l1*sin(Y[0]);
    x2=x1+l2*sin(Y[1]);
    y1=-l1*cos(Y[0]);
    y2=y1-l2*cos(Y[1]);

    E = 0.5*((m1+m2)*l1*Y[2]*l1*Y[2] + m2*l2*Y[3]*l2*Y[3]) +m2*( cos(Y[0])*cos(Y[1]) +
    sin(Y[0])*sin(Y[1]) )*
    l1*Y[2]*l2*Y[3]   +g*( m1*(l1+l2+y1) + m2*(l1+l2+y2) );

    theta1_true=a1*cos(b1*t)+a2*cos(b2*t);
    theta2_true=sqrt(2)*(a1*cos(b1*t)-a2*cos(b2*t));

    data_ang<<t<<" "<<Y[0]-theta1_true<<endl;
    data<< t << "  " << abs(E-E_in)/E_in << "  " <<endl;
    fdata << t << "  " << Y[0] << "  " << Y[1]<<"  " << Y[2] << "  " << Y[3]<<" "<<E<<" "
    <<x1<<" "<<y1<<" "<<x2<<" "<<y2<<" "<<endl;
  }

  fdata<<"\n"<<endl;
  fdata<<"\n"<<endl;
  data_ang<<"\n"<<endl;

//--------------position- Verlet-method---------------------------
  X[0]=Y0[0];   //initial values
```

```
128   X[1]=Y0[1];
129   V[0]=Y0[2];
130   V[1]=Y0[3];
131   t=0.;
132   E=0.;
133
134   for(int i=0; i<N; i++){
135
136     position_Verlet (X, V, accel, dt, n_var);
137     t+=dt;
138
139     x1=l1*sin(X[0]);
140     x2=x1+l2*sin(X[1]);
141     y1=-l1*cos(X[0]);
142     y2=y1-l2*cos(X[1]);
143
144     E = 0.5*((m1+m2)*l1*V[0]*l1*V[1] + m2*l2*V[1]*l2*V[1]) +m2*( cos(X[0])*cos(X[1]) +
      sin(X[0])*sin(X[1]) )*
145     l1*V[0]*l2*V[1]   +g*( m1*(l1+l2+y1) + m2*(l1+l2+y2) );
146
147     theta1_true=a1*cos(b1*t)+a2*cos(b2*t);
148     theta2_true=sqrt(2)*(a1*cos(b1*t)-a2*cos(b2*t));
149
150     data_ang<<t<<" "<<X[0]-theta1_true<<endl;
151     fdata<< t <<" "<< X[0] <<" "<< X[1] <<" "<< V[0] <<" "<< V[1] <<" "<< E <<" "<< x1 <<
      " "<< y1 <<" "<< x2 <<" "<< y2 << endl;
152   }
153   fdata<<"\n"<<endl;
154   fdata<<"\n"<<endl;
155   data_ang<<"\n"<<endl;
156
157   //----------------velocity-Verlet-method---------------------------
158   X[0]=Y0[0];   //initial values
159   X[1]=Y0[1];
160   V[0]=Y0[2];
161   V[1]=Y0[3];
162   t=0.;
163   E=0.;
164
165   for(int i=0; i<N; i++){
166
167     velocity_Verlet (X, V, accel, dt, n_var);
168     t+=dt;
169
170     x1=l1*sin(X[0]);
171     x2=x1+l2*sin(X[1]);
172     y1=-l1*cos(X[0]);
173     y2=y1-l2*cos(X[1]);
174
175     E = 0.5*((m1+m2)*l1*V[0]*l1*V[1] + m2*l2*V[1]*l2*V[1]) +m2*( cos(X[0])*cos(X[1]) +
      sin(X[0])*sin(X[1]) )*
176     l1*V[0]*l2*V[1]   +g*( m1*(l1+l2+y1) + m2*(l1+l2+y2) );
177
178     theta1_true=a1*cos(b1*t)+a2*cos(b2*t);
179     theta2_true=sqrt(2)*(a1*cos(b1*t)-a2*cos(b2*t));
180
181     data_ang<<t<<" "<<X[0]-theta1_true<<endl;
182     fdata<< t <<" "<< X[0] <<" "<< X[1] <<" "<< V[0] <<" "<< V[1] <<" "<< E <<" "<< x1 <<
      " "<< y1 <<" "<< x2 <<" "<< y2 << endl;
183   }
184   fdata<<"\n"<<endl;
185   fdata<<"\n"<<endl;
186
187
188   data.close();
189   fdata.close();
190   data_ang.close();
191
192   cout<<"programma eseguito correttamente"<<"\n"<<endl;
```

```
193
194
195 //-----------------------------------FLIP----------------------------
196   int na=10;
197   double k;
198   bool flip = false, ok_flip = false;
199   double theta_init[n_var];
200
201   ofstream data2;
202   data2.open("flip.dat");
203
204   t = 0;
205   int n = 0;
206
207
208   for (int c = 0; c < na; c++)
209   {
210     cout <<"iterazione: "<< c << endl;
211     cout<<endl;
212     for (int j=0; j<na; j++){
213       Y0[0] = (2*(double)j/(double)na-1)*M_PI; // starts from -pi and ends in +pi
214
215       for(int k=0; k<na; k++){
216       Y0[1] = (2*(double)k/(double)na-1)*M_PI; // starts from -pi and ends in +pi
217
218       ok_flip = false;
219
220       // RK methods ----------------------
221         for (int a=0; a<neq; a++) { //initial values
222           Y[a]=Y0[a];
223         }
224         t=0; // starting time
225         n=0; // no flip or divergenc
226
227         for (int i=0; i<N; i++){
228           theta_init[0]=Y[0];
229           theta_init[1]=Y[1];
230           t += dt;
231           RK4Step(t, Y, dYdt, dt, neq);
232
233           if ( fabs(Y[0]-M_PI) < 1. or fabs(Y[1]-M_PI) < 1. ) { // only checking when the
    angle is close to pi
234             if ( (Y[0]-M_PI)*(theta_init[0]-M_PI) < 0. or (Y[1]-M_PI)*(theta_init[1]-M_PI
    ) < 0. ) {
235               n++;
236               flip = true;
237             }
238           }
239
240           // flip save ----------------------
241         if (not ok_flip and flip) {
242           data2 << t <<" "<< Y0[0] <<" "<< Y0[1] << endl; // t, theta
243           flip = false;
244           ok_flip = true;
245         }
246         if (not ok_flip and i==N-1) data2 << t <<" "<< Y0[0] <<" "<< Y0[1] << endl;
247         }
248       }
249       data2 << endl;
250     }
251   }
252   data2.close();
253
254
255
256   return 0;
257 }
258
259
```

```
void dYdt(double t, double *Y, double *R)
{

  double theta1=Y[0]; //first angle
  double theta2=Y[1]; //second angle
  double omega1=Y[2]; //first speed
  double omega2=Y[3]; //second speed

  R[0]=omega1; //R=dY/dt
  R[1]=omega2;

  R[2]=(-g*(2*m1+m2)*sin(theta1)-m2*g*sin(theta1-2*theta2)
  -2*sin(theta1-theta2)*m2*(omega2*omega2*l2+omega1*omega1*l1*cos(theta1-theta2)))/(l1
    *(2*m1+m2-m2*cos(2*theta1-2*theta2)));
  R[3]=(2*sin(theta1-theta2)*(omega1*omega1*l1*(m1+m2)+g*(m1+m2)*cos(theta1)
  +omega2*omega2*l2*m2*cos(theta1-theta2)))/(l2*(2*m1+m2-m2*cos(2*theta1-2*theta2)));
}

void accel(double *X,double *V, double *a){
  double theta1=X[0]; //first angle
  double omega1=V[0]; //first speed
  double theta2=X[1]; //second angle
  double omega2=V[1]; //second speed

  a[0]=(-g*(2*m1+m2)*sin(theta1)-m2*g*sin(theta1-2*theta2)-2*sin(theta1-theta2)*m2*(
  omega2*omega2*l2+omega1*omega1*l1*cos(theta1-theta2)))/(l1*(2*m1+m2-m2*cos(2*theta1-2*
    theta2)));

  a[1]=(2*sin(theta1-theta2)*(omega1*omega1*l1*(m1+m2)+g*(m1+m2)*cos(theta1)
  +omega2*omega2*l2*m2*cos(theta1-theta2)))/(l2*(2*m1+m2-m2*cos(2*theta1-2*theta2)));
}

//--------------analitic-solution-small-angles----------
void solution(double t, double *Y0, double *Ysol){

  double omega1 = sqrt((2-sqrt(2))*g/l1);
  double omega2 = sqrt((2+sqrt(2))*g/l1);

  double a1 = (sqrt(2)*Y0[0] + Y0[1])*0.5;
  double a2 = (sqrt(2)*Y0[0] - Y0[1])*0.5;

  Ysol[0] =  ( a1*cos(omega1*t) + a2*cos(omega2*t) )/sqrt(2);
  Ysol[1] =    a1*cos(omega1*t) - a2*cos(omega2*t);

  Ysol[2] = - ( omega1*a1*sin(omega1*t) + omega2*a2*sin(omega2*t) )/sqrt(2);
  Ysol[3] = - ( omega1*a1*sin(omega1*t) - omega2*a2*sin(omega2*t) );

}

//------------------------RK4Step------------------------------
void RK4Step(double t,
             double *Y,
             void(*RHSFunc)(double, double*, double*),
             double dt, int neq)
{
  double Y1[neq], k1[neq], k2[neq], k3[neq], k4[neq];

  //k1
  RHSFunc(t,Y,k1);
  for(int i=0;i<neq;i++){
    Y1[i]=Y[i]+0.5*dt*k1[i];
  }

  //k2
  RHSFunc(t+0.5*dt,Y1,k2);
  for(int i=0;i<neq;i++){
    Y1[i]=Y[i]+0.5*dt*k2[i];
  }

```

```
  //k3
  RHSFunc(t+0.5*dt,Y1,k3);
  for(int i=0;i<neq;i++){
    Y1[i]=Y[i]+dt*k3[i];
  }

  //k4
  RHSFunc(t+0.5*dt,Y1,k4);
  for(int i=0;i<neq;i++){
    Y[i]+=(dt/6.)*(k1[i]+2.*k2[i]+2.*k3[i]+k4[i]);
  }
}

//-------------------------position Verlet------------------------------
void position_Verlet(double *x, double *v, void (*acc)(double *, double*,double*), double
     dt, int neq)
{
  int n;
  double a[neq];

  for(n=0; n<neq;n++){
    x[n]+=0.5*dt*v[n];
  }

  acc(x,v,a);

  for(int n=0;n<neq;n++){
    v[n]+=dt*a[n];
  }

  for(n=0;n<neq;n++){
    x[n]+=0.5*dt*v[n];
  }

}

//-------------------------velocity Verlet------------------------------
void velocity_Verlet(double *x, double *v,void (*acc)(double *, double *, double *),
    double dt,int neq)
{
  int n;
  double a[neq];

  acc(x,v,a);

  for(n=0;n<neq;n++) v[n]+=0.5*dt*a[n];

  for(n=0;n<neq;n++) x[n]+=dt*v[n];

  acc(x,v,a);

  for(n=0;n<neq;n++) v[n]+=0.5*dt*a[n];
}

//--------------------Euler-method---------------
void EulerStep (double t, double *Y, void(*RHSFunc)(double, double *, double*), double dt
    , int neq){
  double rhs[neq];

  RHSFunc(t,Y,rhs);
  for(int k=0;k<neq;k++){
    Y[k]+=dt*rhs[k];
  }
}

void NormAng(double &ang){
  if (ang> M_PI) ang-= 2*M_PI;
  if (ang<-M_PI) ang+= 2*M_PI;
}
```

6.1 Codice convergenza algoritmi

```
#include "ode.h"

#define g 9.81

#define M 1
#define L 1

#define pos_in .08
#define vel_in 0.

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


int main(){
  using namespace std;
  int n= 2;
  int N = 20;
  double h = .5;
  double Y[n];
  double t = 0;
  ofstream fwriter;
  int n_var=1;
  double X[n_var], V[n_var];

//---------------CONVERGENCE----------------
  int top = 15;

  double EulErr,RK4Err,posErr,velErr;
  double temp;
  double omega = sqrt(g/L);
  double err1,err2,err3,err4 = 0.;
  fdata.open("dp_conv.dat");
  for(int jj=0;jj<top;jj++){
    t = 0;
    Y[0] = pos_in;
    Y[1] = vel_in;
    for(int i = 0; i<N; i++){
      EulerStep(t,Y,dYdt,h,n);
      t += h;
    }
    temp = fabs(Y[0]-AnalyticalSolution(t));
    EulErr = fabs(temp - err1);
    err1 = temp;
    t = 0;
    Y[0] = pos_in;
    Y[1] = vel_in;
    for(int i = 0; i<N; i++){
      RK4Step(t,Y,dYdt,h,n);
      t += h;
    }
    temp = fabs(Y[0]-AnalyticalSolution(t));
    RK4Err = fabs(temp - err2);
    err2 = temp;
    t = 0;
    X[0] = pos_in;
    V[0] = vel_in;
    for(int i=0; i<N; i++){
      position_Verlet (X, V, acc_func, h, n_var);
      t+=h;
    }
    temp = fabs(X[0]-AnalyticalSolution(t));
    posErr = fabs(temp - err3);
    err3 = temp;
    t = 0;
    X[0] = pos_in;
```

```
68      V[0] = vel_in;
69      for(int i=0; i<N; i++){
70        velocity_Verlet (X, V, acc_func, h, n_var);
71        t+=h;
72      }
73      temp = fabs(X[0]-AnalyticalSolution(t));
74      velErr = fabs(temp - err4);
75      err4 = temp;
76      fdata << h << " " << EulErr << " "  << RK4Err << " "<<posErr<<" "<<velErr<<endl;
77
78      N *= 2;
79      h /= 2;
80
81    }
82
83    fdata.close();
84    return 0;
85  }
86
87  void dYdt(double t, double *Y, double *R){
88    double x = Y[0];
89    double v = Y[1];
90    R[0] = v;
91    R[1] = -(g/L)*sin(x);
92  }
93
94  void acc_func(double *theta_t, double *a)
95  {
96    double theta=theta_t[0];
97    a[0]=-sin(theta);
98  }
99
100 double AnalyticalSolution(double t){
101   double omega = sqrt(g/L);
102   return IVP_pos*cos(omega*t);
103 }
```