Pendolo doppio

Sara De Benedetti

Marzo 2024

1 Introduzione

Nelle pagine seguenti ci si propone lo studio del doppio pendolo, che è interessante in quanto è un sistema semplice con il quale si può studiare il caos; per alcuni range di condizioni iniziali, infatti, ha un comportamento caotico, ovvero scelte due condizioni iniziali vicine queste si allontanano esponenzialmente l'una dall'altra al passare del tempo. Per piccoli angoli e basse velocità iniziali il sistema ha un comportamento quasi-periodico. Questo sistema fisico è composto da due pendoli il secondo dei quali è attaccato all'estremità libera del primo, come si può vedere in figura 1. Per semplificare il problema è stato considerato un pendolo planare che consiste di due masse puntiformi m_1 e m_2 attaccate a due fili inestensibili e privi di massa lunghi l_1 e l_2, immerse in un campo gravitazionale e non soggette a forze d'attrito.

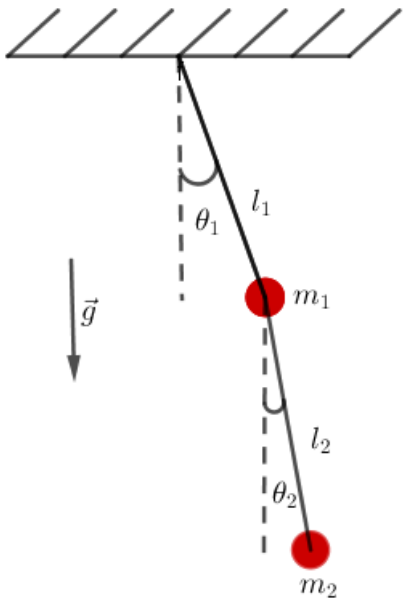

Figure 1: Doppio pendolo

Al fine di ricavare le equazioni del moto si scrivono le coordinate cartesiane della posizione delle due masse in funzione degli angoli, θ_1 e θ_2 compresi tra i bracci dei pendoli e la verticale (equazione 1).

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

Le velocità delle masse m_1 e m_2 si possono vedere in equazione 2.

$$\begin{cases} \dot{x}_1 = l_1\dot{\theta}_1 \cos\theta_1 \\ \dot{y}_1 = l_1\dot{\theta}_1 \sin\theta_1 \end{cases} \qquad \begin{cases} \dot{x}_2 = \dot{x}_1 + l_2\dot{\theta}_2 \cos\theta_2 \\ \dot{y}_2 = \dot{y}_1 + l_2\dot{\theta}_2 \sin\theta_2 \end{cases} \tag{2}$$

L'energia cinetica T del sistema è la somma delle energie cinetiche dei due pendoli (equazione 3).

$$T = \frac{m_1}{2}(\dot{x}_1^2 + \dot{y}_1^2) + \frac{m_2}{2}(\dot{x}_2^2 + \dot{y}_2^2) \tag{3}$$

L'energia potenziale si ottiene allo stesso modo:

$$U = m_1 g y_1 + m_2 g y_2 \tag{4}$$

Si può così ricavare la lagrangiana ($L = T - U$) che con le opportune sostituzioni è:

$$L = \frac{(m_1+m_2)}{2} l_1^2\dot{\theta}_1^2 + \frac{m_2}{2} l_2^2\dot{\theta}_2^2 + m_2 l_1 l_2 \dot{\theta}_1\dot{\theta}_2 cos(\theta_1 - \theta_2) + g(m_1+m_2) l_1 cos(\theta_1) + g m_2 l_2 cos\theta - 2 \tag{5}$$

A questo punto si possono applicare le equazioni di Eulero-Lagrange (6) al fine di ottenere le equazioni del moto (7).

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) - \frac{\partial L}{\partial \theta_i} = 0 \quad i = 1, 2 \tag{6}$$

$$\begin{cases} \dot{\theta}_1 = \omega_1 \\ \dot{\theta}_2 = \omega_2 \\ \dot{\omega}_1 = \dfrac{-g\,(2m_1 + m_2)\sin\theta_1 - g\,m_2\sin(\theta_1 - 2\theta_2) - 2m_2\sin(\theta_1 - \theta_2)(\dot{\theta}_2^2\, l_2 + \dot{\theta}_1^2\, l_1 cos(\theta_1 - \theta_2))}{l_1\,(2m_1 + m_2 - m_2\cos(2\theta_1 - 2\theta_2))} \\ \dot{\omega}_2 = \dfrac{2\sin(\theta_1 - \theta_2)(\dot{\theta}_1^2\, l_1(m_1 + m_2) + g\,(m_1 + m_2)\cos\theta_1 + \dot{\theta}_2^2\, l_2\, m_2\cos(\theta_1 - \theta_2))}{l_2\,(2m_1 + m_2 - m_2\cos(2\theta_1 - 2\theta_2))} \end{cases} \tag{7}$$

Queste equazioni differenziali non si possono risolvere analiticamente, quindi è necessario fare uso di metodi di integrazione numerica, ci sarà un approfondimento sulla scelta della tecnica utilizzata nel paragrafo 2.

Nella seguente trattazione si fanno le assunzioni: $m_1 = m_2 = 1$ e $l_1 = l_2 = 1$.

2 Scelta dell'algoritmo

Per compiere integrazione numerica di equazioni differenziali ordinarie sono possibili diversi metodi. Il metodo di Eulero è la tecnica più semplice, però, come si può vedere in figura 2 il metodo di Runge-Kutta del quarto ordine porta le performance migliori. L'algoritmo Runge-Kutta non conserva l'area simplettica come, invece, il sistema fisico fa in quanto hamiltoniano; altri metodi numerici come il Leapfrog Method o un Verlet permetterebbero di conservare l'area simplettica, e l'energia, ma le equazioni differenziali 7 non sono scritte in modo tale per cui queste tecniche possano essere implementate, le derivate delle velocità angolari dipendono dalle velocità angolari stesse perché nel sistema ci sono delle forze che dipendono dalla velocità. Nonostante il sistema fisico conservi l'energia meccanica il metodo d'integrazione scelto non la conserva; bisogna assicurarsi che le variazioni di energia siano abbastanza piccole da poter essere trascurate. In figura 2 si possono osservare le fluttuazioni, al variare del passo temporale dt scelto, dell'errore relativo $\sigma = \frac{|E_0 - E|}{E_0}$ in cui E_0 è l'energia ricavata a partire dalle condizioni iniziali e E è l'energia calcolata a tempi successivi (equazione 8).

$$E = \frac{m_1}{2} l_1^2\dot{\theta}_1^2 + \frac{m_2}{2}\left(l_1^2\dot{\theta}_1^2 + l_2^2\dot{\theta}_2^2 + 2l_1 l_2\dot{\theta}_1\dot{\theta}_2 cos(\theta_1 - \theta_2)\right) - (m_1 + m_2) g l_1 cos(\theta_1) - m_2 g l_2 cos\theta_2 \tag{8}$$

Dal confronto di questi due grafici (figura 3) si può notare che l'errore dell'energia per passo temporale dt più grande è maggiore. Quindi si è deciso di approfondire lo studio del problema andando a graficare la dipendenza dell'errore relativo dell'energia valutato dopo un grande numero di iterazioni (10^6) dall'intervallo dt, i risultati sono riportati in figura 4.

Per un passo dt pari a 0.01 si ottiene un errore relativo del 16% che non è trascurabile, quindi si è continuata la trattazione del sistema usando un intervallo $dt = 0.005$ che porta un errore dell'energia pari al 0.03%.

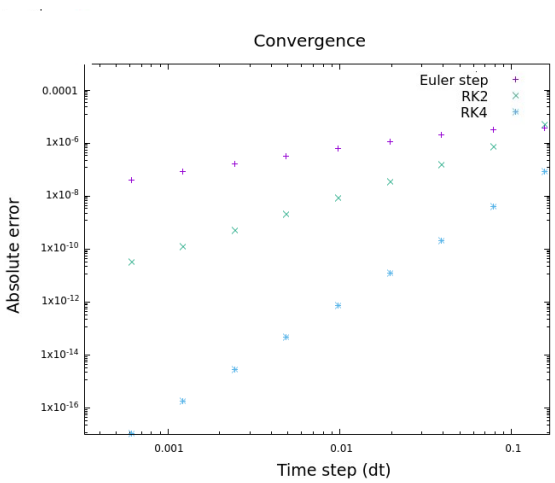

Figure 2: Grafico di convergenza dell'equazione differenziale $\frac{dx}{dt} = -tx$ con $x(0) = 1$ usando il metodo di Eulero e Runge-Kutta al secondo e quarto ordine.

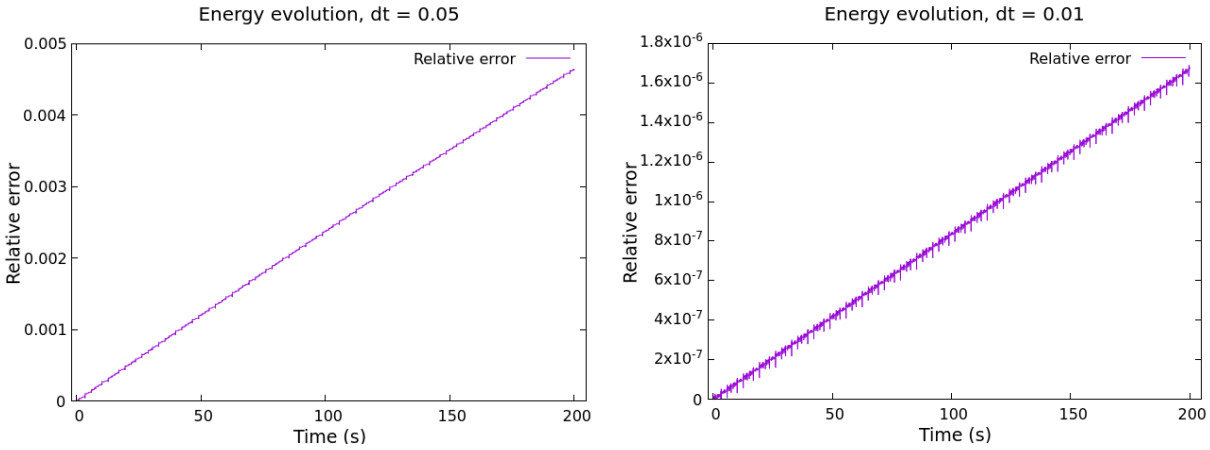

Figure 3: Errore relativo dell'energia al passare del tempo per passi temporali di $dt = 0.05\,s$ (sinistra) e $dt = 0.01\,s$ (destra) per condizioni iniziali di $\theta_1 = 0.9\,rad$, $\theta_2 = 0.3\,rad$, $\omega_1 = 0.2\,rad/s$ e $\omega_2 = 0\,rad/s$.

3 Dinamica del sistema

A questo punto si può studiare la dinamica del sistema. Si è deciso di osservare il sistema al passare di $6 \cdot 10^4$ iterazioni compiute con un passo di $dt = 0.005\,s$, questo alto numero di iterazioni permette di osservare l'evoluzione temporale del pendolo doppio. In primo luogo si è scelto di esaminare la dinamica per spostamento dalla condizione di equilibrio di angoli iniziali piccoli $\theta_1 = \theta_2 = 0.1\,rad$ e velocità angolari iniziali nulle. In queste condizioni il moto è quasi periodico come si vede in figura 3. La regolarità del moto che si osserva viene anche rispecchiata nello spazio delle fasi (figura 6), in cui le traiettorie giacciono su un toro. L'integrazione del sistema a tempi più lunghi permetterebbe lo studio dell'ergodicità del sistema, la densità delle traiettorie sullo spazio delle fasi e la dimensione di quest'ultimo. All'aumentare dello spostamento iniziale degli angoli dalla condizione di equilibrio e della velocità angolare delle masse il sistema mostra una transizione al caos; le traiettorie degli angoli perdono la periodicità (figura 3) e lo spazio delle fasi non presenta più un toro come attrattore (figura 3).

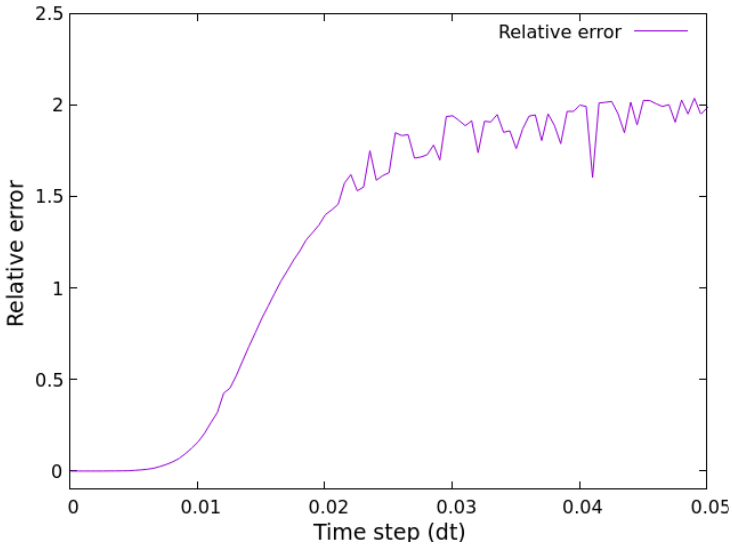

Figure 4: Errore relativo dell'energia in funzione del passo temporale dt

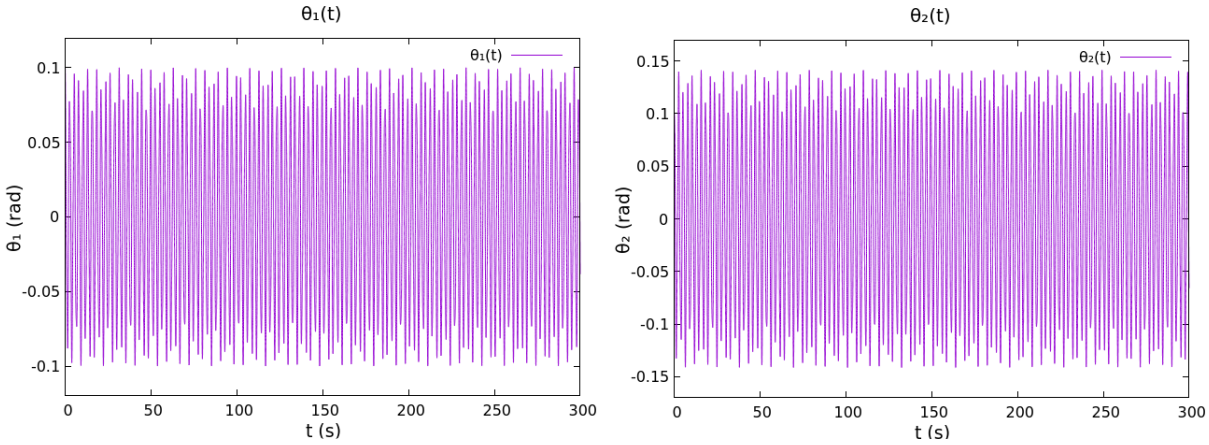

Figure 5: Angoli della massa 1 (θ_1) e della massa 2 (θ_2) al passare del tempo t presi con spostamento iniziale dalla condizione di equilibrio piccolo: $\theta_1 = \theta_2 = 0.1\,rad$ e $\omega_1 = \omega_2 = 0\,rad/s$

4 Giro completo del pendolo

In questa sezione ci si propone di trovare dopo quanto tempo le masse del pendolo compiono il primo giro intero. Affinché si possa considerare che una massa abbia fatto un giro completo la richiesta minima è che l'angolo del pendolo corrispondente alla massa sia pari a $k\pi$ rad con $k = \pm 1$. Quindi le energie potenziali minime per cui il pendolo più interno e quello esterno facciano un giro completo sono rispettivamente:

$$U_{min,1} = g(l_1(m_1 + m_2) - l_2 m_2)$$

$$U_{min,2} = g(l_2 m_2 - l_1(m_1 + m_2))$$

Quindi l'energia minima energia potenziale U affinché ci sia un giro è tale che $U \geq U_{min,i}$ con $i = 1, 2$, questa relazione si può scrivere esplicitamente come si vede in formula 9.

$$\begin{aligned} l_1(m_1 + m_2)(\cos\theta_1 - 1) + l_2 m_2(\cos\theta_2 + 1) \leq 0 \\ l_1(m_1 + m_2)(\cos\theta_1 + 1) + l_2 m_2(\cos\theta_2 - 1) \leq 0 \end{aligned} \tag{9}$$

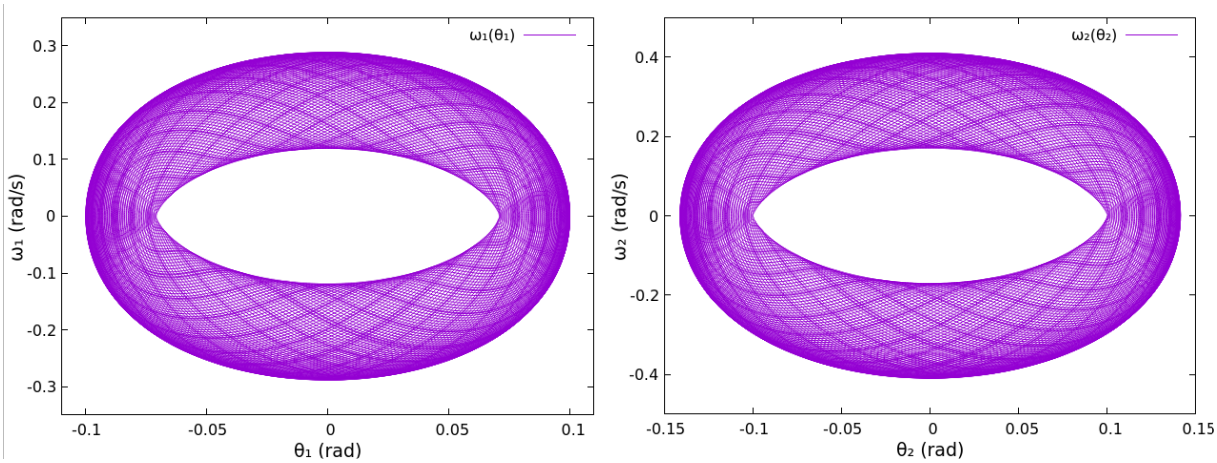

Figure 6: Velocità angolari ω delle due masse in funzione dell'angolo con spostamento iniziale dalla condizione di equilibrio piccolo: $\theta_1 = \theta_2 = 0.1\,rad$ e $\omega_1 = \omega_2 = 0\,rad/s$

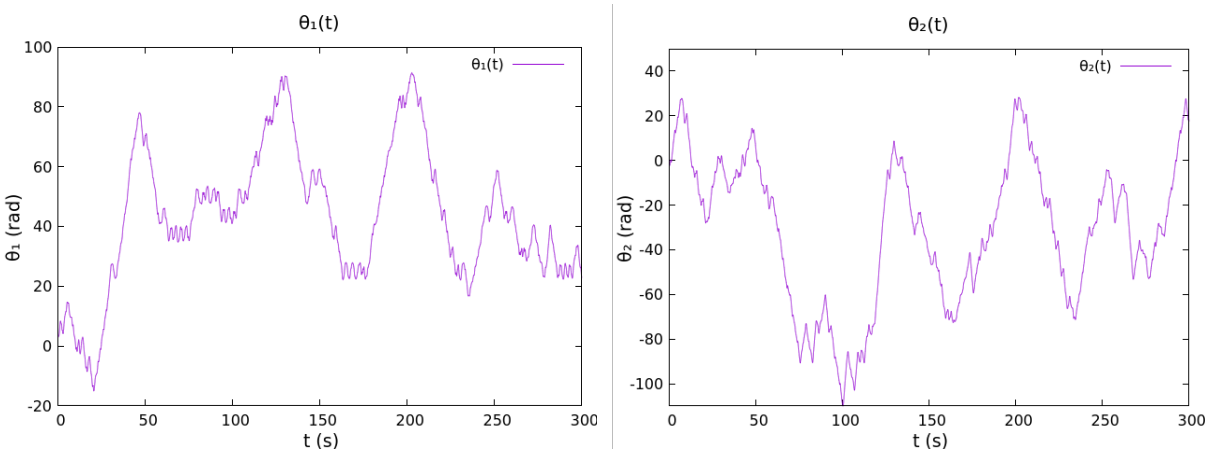

Figure 7: Angoli della massa 1 (θ_1) e della massa 2 (θ_2) al passare del tempo t presi con spostamento iniziale dalla condizione di equilibrio piccolo: $\theta_1 = 3.0\,rad$, $\theta_2 = -2.5\,rad$, $\omega_1 = 1.0\,rad/s$ e $\omega_2 = 0.0\,rad/s$

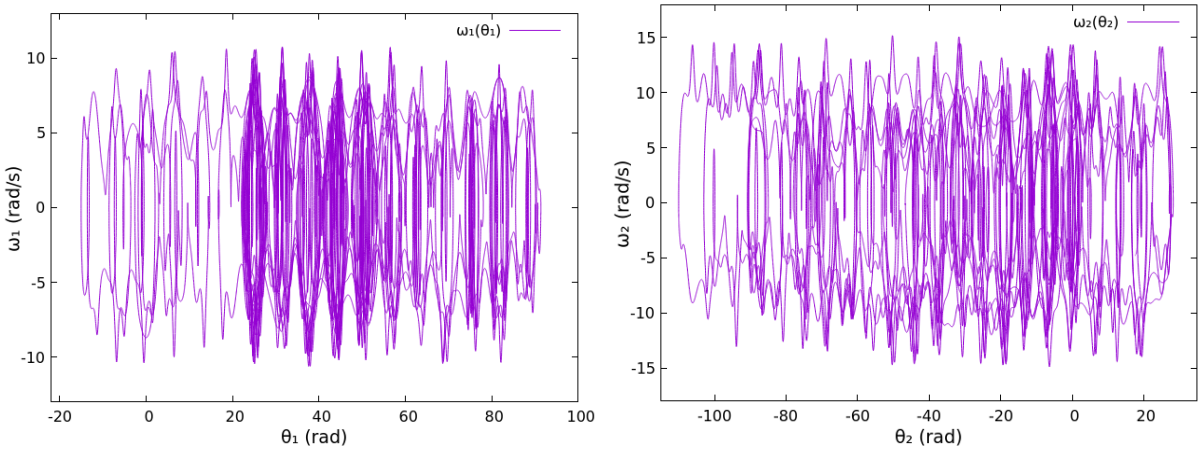

Figure 8: Velocità angolari ω delle due masse in funzione dell'angolo con spostamento iniziale dalla condizione di equilibrio piccolo: $\theta_1 = 3.0\,rad$, $\theta_2 = -2.5\,rad$, $\omega_1 = 1.0\,rad/s$ e $\omega_2 = 0.0\,rad/s$

Oppure si può vedere in formula 10 imponendo $m_1 = m_2 = 1$ kg e $l_1 = l_2 = 1$ m.

$$\begin{aligned} 2\cos\theta_1 + \cos\theta_2 &\leq -1 \\ 2\cos\theta_1 + \cos\theta_2 &\leq +1 \end{aligned} \tag{10}$$

Da queste disequazioni si può ricavare per quali coppie di θ_1 e θ_2 le due masse m_1 e m_2 compiono un giro completo, se $\omega_1 = \omega_2 = 0\,rad/s$, e si possono vedere in viola rispettivamente nelle figure 9 e 10.

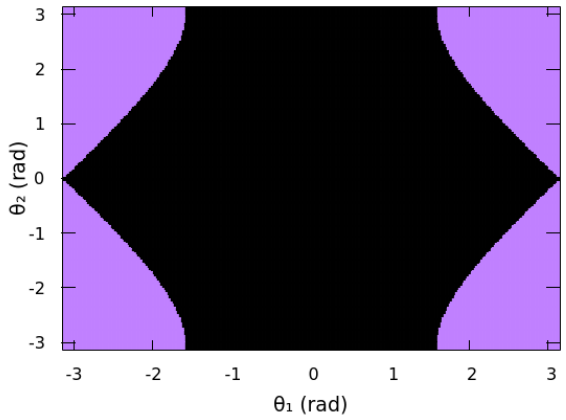

Figure 9: In viola le coppie di angoli θ_1 e θ_2 per cui la massa m_1 fa un giro completo

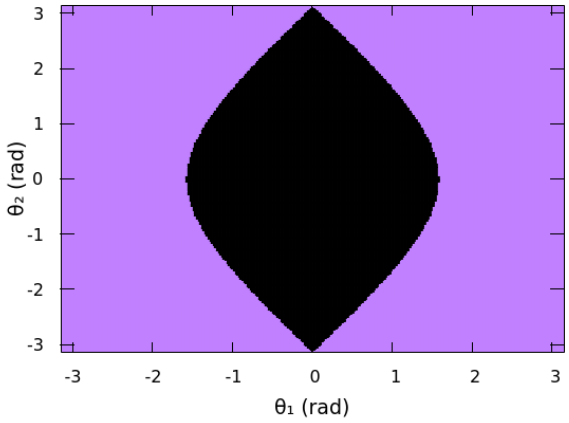

Figure 10: In viola le coppie di angoli θ_1 e θ_2 per cui la massa m_2 fa un giro completo

Si sono cercati di riprodurre questi risultati usando metodi numerici. Si è calcolato dopo quanto tempo le masse m_1 e m_2 compiono un giro completo prendendo come condizioni iniziali ogni angolo compreso tra -π e $+\pi\,rad$ con passi di $\Delta\theta = 0.5°$ e $\omega_1 = \omega_2 = 0\,rad/s$. Ogni set di condizioni iniziali ha avuto al massimo un tempo $t = 1000\,s$ di runtime con passi temporale pari a $dt = 0.005\,s$ e i risultati sono stati raccolti nei grafici 11 e 12, nei quali le zone nere rappresentano quei valori di angoli iniziali per cui i pendoli non compiono un giro completo. Invece per gli altri valori temporali misurati è stata definita una barra di colori: in giallo le coppie di θ_1 e θ_2 per cui il pendolo ci mette più tempo a girare e in viola quelle per cui ce ne mette di meno. Qualitativamente parlando gli spazi degli angoli in cui

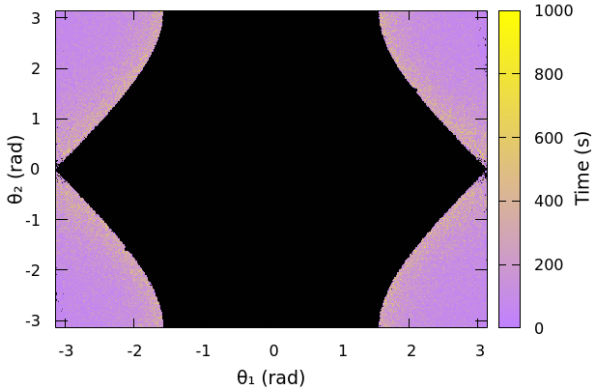

Figure 11: Tempo per cui la massa m_1 fa un giro completo in funzione delle coppie di angoli θ_1 e θ_2

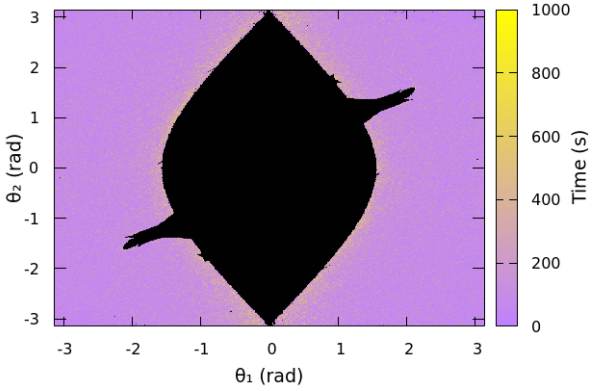

Figure 12: Tempo per cui la massa m_2 fa un giro completo in funzione delle coppie di angoli θ_1 e θ_2

si trova che i pendoli fanno un giro intero con metodi numerici corrisponde alle regioni teoriche trovate usando la disuguaglianza dell'energia potenziale (equazione 10). Per ottenere delle conclusioni più vicine alle previsioni toriche bisognerebbe integrare per periodi temporali più lunghi e usare una griglia sugli angoli più fine.

5 Esponenti di Lyapunov

In un sistema caotico due condizioni iniziali vicine tra loro (δl_0) si allontanano esponenzialmente l'una dall'altra $\delta l = \delta l_0\, e^{\lambda t}$, il coefficiente λ ricavato per un tempo t infinito è chiamato esponente di Lyapunov, che si può, quindi, definire come:

$$\lambda = \lim_{t\to\infty} \lim_{\delta l_0 \to 0} \frac{1}{t}\, log \left| \frac{\delta l}{\delta l_0} \right|$$

Questa definizione porta degli evidenti problemi dovuti al fatto che non si può calcolare numericamente un valore dopo un tempo infinito, quindi si è obbligati a dare una stima non molto accurata degli esponenti di Lyapunov. La misura è stata portata avanti nel seguente modo: si è iniziato il processo prendendo due condizioni iniziali con θ_1 e θ_2 distanti di $10^{-8}\, rad$ e lasciando evolvere il sistema dinamico per un tempo $t = 1000\, s$, a questo punto si è calcolata la distanza euclidea tra le due masse m_1 delle due diverse condizioni iniziali, con questa si ricava l'esponente di Lyapunov (λ_1) relativo a m_1. Analogamente si può fare per la seconda massa. Questa prima stima è fatta a tempi troppo brevi e quindi è ancora dipendente dalle condizioni iniziali, in linea teorica non dovrebbe esserlo per un set di condizioni iniziali che portano il sistema ad avere un comportamento caotico. Pertanto, dopo aver fatto questa prima misura, si è riavvicinata una delle due traiettorie all'altra, che, invece, rimane inalterata, nuovamente apportando una differenza tra i due θ_1 e θ_2 pari a $10^{-8}\, rad$. Si è compiuto questo processo cento volte e il valore finale dell'esponente di Lyapunov è stato ricavato tramite la media delle misure ripetute delle distanze δl (equazione 11).

$$\lambda = \frac{1}{t} log \left(\sum_{i=1}^{100} \frac{\delta l_i}{\delta l_{0,i}} \right) \tag{11}$$

Dalla definizione si può evincere che gli esponenti di Lyapunov sono positivi quando il sistema è in regime caotico, quindi si può fare una scansione delle condizioni iniziali al fine di capire per quali condizioni iniziali il sistema diventa caotico. Si è calcolato l'esponente di Lyapunov duecento volte tra 0 e π radianti per gli angoli, non è necessario studiare anche gli altri due quadranti che portano risultati analoghi per via della simmetria, e tra 0 e 2π rad/s per le velocità angolari delle due masse. I risultati si possono osservare nelle figure 13, 14, 15 e 16. Si può notare che per condizioni iniziali vicine allo zero gli esponenti di Lyapunov risultano essere negativi stando ad indicare che il sistema non è caotico, ci sono delle fluttuazioni dovute alle diverse approssimazioni fatte nel calcolo che portano gli esponenti di Lyapunov ad essere positivi anche se il sistema ancora non è caotico. In tutti e quattro i grafici c'è una discontinuità evidente per un certo valore del parametro di controllo, dopo questa discontinuità l'esponente di Lyapunov tende a stabilizzarsi e il sistema diventa caotico. Si è ottenuto che il sistema passa ad essere in regime caotico quando si considerano le seguenti condizioni iniziali: per θ_1 maggiore di 1.13 rad, per θ_2 maggiori di 1.57 rad, per ω_1 maggiori di 3.12 rad/s e per ω_2 maggiori di 5.13 rad/s.

6 Conclusioni

Il doppio pendolo, pur essendo un sistema piuttosto semplice, ha delle caratteristiche interessanti. Inoltre la soluzione analitica del problema non è possibile, quindi l'unico modo di esplorare il comportamento del sistema è tramite l'utilizzo di metodi numerici. L'algoritmo Runge-Kutta al quarto ordine, utilizzato per la trattazione, come abbiamo visto, porta dei problemi, per esempio non conserva l'energia del sistema. Cercare di minimizzare l'errore sull'energia anche dopo un grande numero di iterazioni ha portato a svolgere lo studio del pendolo doppio usando un passo temporale pari a $dt = 0.005\, s$. Una volta presa questa accortezza si è passati a studiare la dinamica del sistema e gli esponenti di Lyapunov che permettono di capire per quali condizioni iniziali il pendolo ha un comportamento caotico. Infine si è esaminato lo spazio degli angoli per cui i pendoli compiono un giro completo. Per continuare la ricerca si potrebbe ripetere lo studio con metodi numerici più raffinati, ricavare i diagrammi di biforcazione o studiare l'ergodicità del sistema.

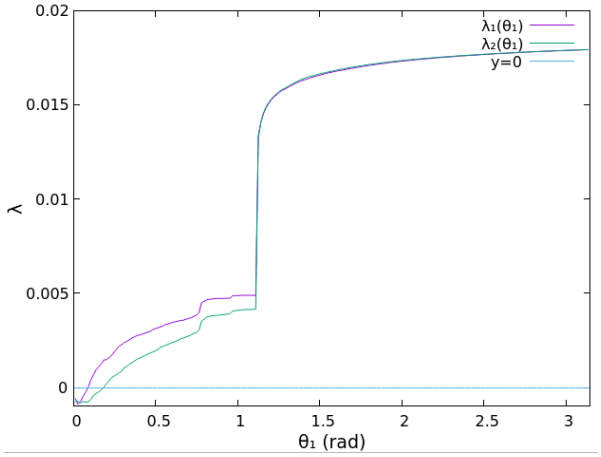

Figure 13: Esponenti di Lyapunov per m_1 e m_2 in funzione dell'angolo θ_1

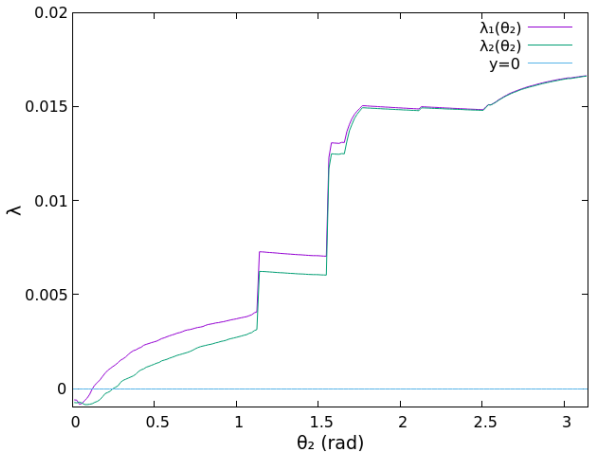

Figure 14: Esponenti di Lyapunov per m_1 e m_2 in funzione dell'angolo θ_2

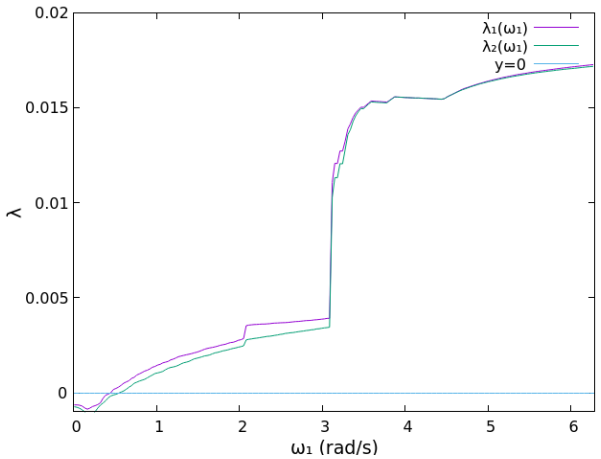

Figure 15: Esponenti di Lyapunov per m_1 e m_2 in funzione dell'angolo ω_1

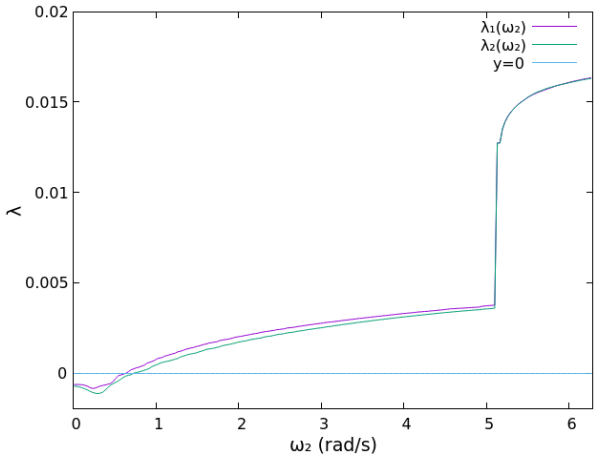

Figure 16: Esponenti di Lyapunov per m_1 e m_2 in funzione dell'angolo ω_2

7 Codice

```
1
2  #include <iostream>
3  #include<cmath>
4  #include <fstream>
5  using namespace std;
6
7  void dYdt(double, double*, double*);
8  void RK4(double , double *, double , int , void (*Func)
9          (double, double*, double*));
10
11 #define STAGE 5
12 #define PARAM 4
13
14 // Constants for mass, length, and gravity
15 double g_m = 1.; // m1 = m2 = m
16 double g_l = 1.; // l1 = l2 = l
17 double g_g = 9.8;
18
19
20
21 int main(){
22     double ti = 0., t = ti, tf; // Initial and final time
23     int neq = 4; // Number of equations
24     int m; // Number of iterations
25     double Y[neq], Y1[neq]; // Arrays to hold system variables
26
27 #if STAGE == 1
28     /* This stage wants to compare two different time steps
29     to see how the relative error of the total energy evolves */
30     //Fixed initial conditions
31     Y[0] = 0.9; Y[1] = 0.3; Y[2] = 0.2; Y[3] = 0.;
32     //Time steps
33     double dt1, dt2;
34     dt1 = 0.05; dt2 = 0.01;
35     //open two files to store data
36     ofstream file1("stage1_dt1.txt");
37     ofstream file2("stage1_dt2.txt");
38     tf = 200.; //End time for simulation
39     m = (tf - ti) / dt1; //number of iterations
40     double en_i, en;
41     // Initial energy
42     en_i = g_m * g_l * g_l * (Y[2] * Y[2] + 0.5 * Y[3] * Y[3] +
43         Y[2] * Y[3] * cos(Y[0] - Y[1])) - g_m * g_g * g_l *
44         (cos(Y[1]) + 2. * cos(Y[0]));
45     // Loop for first time step using Runge-Kutta method
46     for(int i = 0; i < m; i++){
47         RK4(t, Y, dt1, neq, dYdt);
48         t += dt1;
49         // Calculate energy
50         en = g_m * g_l * g_l * (Y[2] * Y[2] + 0.5 * Y[3] * Y[3] +
51              Y[2] * Y[3] * cos(Y[0] - Y[1])) - g_m * g_g * g_l *
52              (cos(Y[1]) + 2. * cos(Y[0]));
53         //Store energy relative error into file
54         file1 << t << "    " << fabs((en_i - en) / en_i) << endl;
55     }
56
57     // Reset system initial conditions and time for second time step
58     Y[0] = 0.9; Y[1] = 0.3; Y[2] = 0.2; Y[3] = 0.;
59     t = 0; //Reset time
60     m = (tf - ti) / dt2; //number of iterations
61     // Loop for second time step using Runge-Kutta method
62     for(int j = 0; j < m; j++){
63         RK4(t, Y, dt2, neq, dYdt);
64         t += dt2;
65         // Calculate energy and write to file
66         en = g_m * g_l * g_l * (Y[2] * Y[2] + 0.5 * Y[3] * Y[3] +
67              Y[2] * Y[3] * cos(Y[0] - Y[1])) - g_m * g_g * g_l *
```

```
            (cos(Y[1]) + 2. * cos(Y[0]));
        //Store energy relative error into file
        file2 << t << "   " << fabs((en_i - en) / en_i) << endl;
    }

    // Close files
    file1.close();
    file2.close();
#endif
#if STAGE == 2
/*This stage wants to show how relative energy at a fixed time t and
  fixed initial conditions varies with different time steps*/
  //Fixed initial conditions
    Y[0] = 3.; Y[1] = -2.5; Y[2] = 1.; Y[3] = 0.;
    //Time steps
    double  dt_min = 0.0001, dt = dt_min, dt_max = 0.05;
    int m = 1000000; //number of iterations
    int n = 100; //number of points stored into file
    ofstream file1("stage2_en_err(dt).txt");
    double en_i, en; //Total energy
    // Initial energy
    en_i = g_m * g_l * g_l * (Y[2] * Y[2] + 0.5 * Y[3] * Y[3] +
           Y[2] * Y[3] * cos(Y[0] - Y[1])) - g_m * g_g * g_l *
           (cos(Y[1]) + 2. * cos(Y[0]));
    // Loop to vary time step using Runge-Kutta method
    while(dt < dt_max){
        for(int i = 0; i < m; i++){
            RK4(t, Y, dt, neq, dYdt);
            t += dt;
        }
        // Calculate energy and write to file
        en = g_m * g_l * g_l * (Y[2] * Y[2] + 0.5 * Y[3] * Y[3] +
             Y[2] * Y[3] * cos(Y[0] - Y[1])) - g_m * g_g * g_l *
             (cos(Y[1]) + 2. * cos(Y[0]));
        //Store energy relative error into file
        file1 << dt << "   " << fabs((en_i - en) / en_i) << endl;
        t = 0.;
        // Reset initial conditions
        Y[0] = 3.; Y[1] = -2.5; Y[2] = 1.; Y[3] = 0.;
        dt += ((dt_max - dt_min) / n);
    }
    file1.close();
#endif

#if STAGE == 3
/*This stage wants to see how the dynamic of the double pendulum evolves
  when starting with small end big initial conditions */
    //Open file to store data into
    ofstream file1("moto_pendolo.txt");
    //Initial conditions
    Y[0] = 0.1; Y[1] = 0.1; Y[2] = 0.0; Y[3] = 0.;
    //Initial condition for chaotic motion
    Y1[0] = 3.0; Y1[1] = -2.5 ; Y1[2] = 1.; Y1[3] = 0.;
    //Time step
    double dt;
    dt = 0.005;
    // Loop for simulating the system motion using Runge-Kutta method
    for(double t = 0.; t < 300.; t += dt){
        RK4(t, Y, dt, neq, dYdt);
        RK4(t, Y1, dt, neq, dYdt);
        // Write data to file
        file1 << t << "   " << Y[0] << "   " << Y[1] << "   " << Y[2]
              << "   " << Y[3]  << "   " << Y1[0] << "   " << Y1[1]
              << "   " << Y1[2] << "   " << Y1[3] << endl;
    }
    file1.close();
#endif

#if STAGE == 4
```

```
    double err = 1.e-8; //Error to initial condition
    double x = 0.01; //Control parameter
    double x_end; //End of the simulation
    double dx; //Control parameter step
    double parm; //Indicates which parameter is the control one
    int N = 100; //Number of times the initial conditions are riconnetted
    int  n_points = 200; //Number of points stored in file
    #if PARAM == 1
        // Lyapunov exponent for the angle of the first mass
        ofstream file1("esp_lyapunov_theta1.txt");
        x_end = M_PI;
        parm = 0;
        dx = x_end / n_points;
    #endif
    #if PARAM == 2
        // Lyapunov exponentss for the angle of the second mass
        ofstream file1("esp_lyapunov_theta2.txt");
        x_end = M_PI;
        parm = 1;
        dx = x_end / n_points;
    #endif
    #if PARAM == 3
        // Lyapunov exponents for the angular velocity of the first mass
        ofstream file1("esp_lyapunov_omega1.txt");
        x_end = 2 * M_PI;
        parm = 2;
        dx = x_end / n_points;
    #endif
    #if PARAM == 4
        // Lyapunov exponents for the angular velocity of the second mass
        ofstream file1("esp_lyapunov_omega2.txt");
        x_end = 2 * M_PI;
        parm = 3;
        dx = x_end / n_points;
    #endif

    // Initial conditions
    for(int i = 0; i < neq; i++){
        if(i == parm) Y[i] = x;
        else Y[i] = 0.;

        if(i < 2) Y1[i] = Y[i] + err;
        else Y1[i] = Y[i];
    }
    double dt = 0.005; //Time step
    double t_end = 1000.; //Time at which the evolution of the system stops
    double esp_lyap1, esp_lyap2; //Lyapunov exponents
    double x1_0, x2_0, y1_0, y2_0, x1_1, x2_1, y1_1, y2_1; //Cartesian coordinates
    double d0_1 = 0., d0_2 = 0., d1_1 = 0., d1_2 = 0.; //Euclidian distances
    int n_step; //Number of iterations
    t = 0.;//Initial time
    n_step = t_end / dt;

    while(x < x_end){
        for(int i = 0; i < N; i++){
            // Cartesian coordinates for initial conditions
            x1_0 = g_l * sin(Y[0]);
            x2_0 = x1_0 + g_l * sin(Y[1]);
            y1_0 = -g_l * cos(Y[0]);
            y2_0 = y1_0 - g_l * cos(Y[1]);

            x1_1 = g_l * sin(Y1[0]);
            x2_1 = x1_1 + g_l * sin(Y1[1]);
            y1_1 = -g_l * cos(Y1[0]);
            y2_1 = y1_1 - g_l * cos(Y1[1]);

            // Euclidean distances for initial conditions
            d0_1 += sqrt((x1_0 - x1_1) * (x1_0 - x1_1) + (y1_0 - y1_1) * (y1_0 - y1_1));
            d0_2 += sqrt((x2_0 - x2_1) * (x2_0 - x2_1) + (y2_0 - y2_1) * (y2_0 - y2_1));
```

```
            //Loop for the evolution of the system using Runge-Kutta method
            for(int k = 0; k < n_step; k++){
                RK4(t, Y, dt, neq, dYdt);
                RK4(t, Y1, dt, neq, dYdt);
                t += dt;
            }

            // Cartesian coordinates
            x1_0 = g_l * sin(Y[0]);
            x2_0 = x1_0 + g_l * sin(Y[1]);
            y1_0 = -g_l * cos(Y[0]);
            y2_0 = y1_0 - g_l * cos(Y[1]);

            x1_1 = g_l * sin(Y1[0]);
            x2_1 = x1_1 + g_l * sin(Y1[1]);
            y1_1 = -g_l * cos(Y1[0]);
            y2_1 = y1_1 - g_l * cos(Y1[1]);

            // Euclidean distances
            d1_1 += sqrt((x1_0 - x1_1) * (x1_0 - x1_1) + (y1_0 - y1_1) * (y1_0 - y1_1));
            d1_2 += sqrt((x2_0 - x2_1) * (x2_0 - x2_1) + (y2_0 - y2_1) * (y2_0 - y2_1));

            // Making the conditions close again
            for(int i = 0; i < neq; i++){
                if(i < 2) Y1[i] = Y[i] + err;
                else Y1[i] = Y[i];
            }
            t = 0.;
        }
        //Lyapunov exponents
        esp_lyap1 = log(fabs(d1_1 / d0_1)) / t_end;
        esp_lyap2 = log(fabs(d1_2 / d0_2)) / t_end;
        //Store data into file
        file1 << x << "  " << esp_lyap1 << "    " << esp_lyap2 << endl;
        //Increase of the control parameter
        x += dx;
        // Initial conditions
        for(int i = 0; i < neq; i++){
            if(i == parm) Y[i] = x;
            else Y[i] = 0.;

            if(i < 2) Y1[i] = Y[i] + err;
            else Y1[i] = Y[i];
        }
    }
    //Close file
    file1.close();
#endif

#if STAGE == 5
/*This stage wants to see for which set of initial condition for
  the angels does the pendulum flip and after how much time*/
    //Open files
    ofstream file1("flip1.txt");
    ofstream file2("flip2.txt");
    double dt = 0.005; //Time step
    double t_stop = 1000.; //Time for which the system evolution stops
    double tol = 1.e-7; //Tollerance
    int stop; //Number of iteratins
    int dati = 720; //Number of points for each angle
    int flip1 = 0, flip2 = 0; //Number of flips
    double theta1_old = 0., theta2_old = 0.;//control parameters
    //Number of iterations
    stop = t_stop / dt;
    //Loop on theta 1
    for(int i = 0; i < dati; i++){
        //Loop on theta 2
        for(int j = 0; j < dati; j++){
            //Initial conditions
```

```
            Y[0] = i * (2 * M_PI / dati) - M_PI;
            Y[1] = j * (2 * M_PI / dati) - M_PI;
            Y[2] = 0.; Y[3] = 0.;
            t = 0.;
            //Loop for the evolution of the system using Runge-Kutta method
            for(int m = 0; m < stop; m++){
                RK4(t, Y, dt, neq, dYdt);
                //check whether the pendulum has flipped
                if(flip1 == 0 && sin(Y[0]) * sin(theta1_old) < 0.
                   && t > dt && cos (Y [0]) < -1. + tol){
                    //Store for what time and angles the pandulum has flipped
                    file1 << t << "   " << i * (2 * M_PI / dati) - M_PI
                          << "   " << j * (2 * M_PI / dati) - M_PI << endl;
                    flip1 = 1;
                }
                if(flip2 == 0 && sin(Y[1]) * sin(theta2_old) < 0.
                   && t > dt && cos (Y [1]) < -1. + tol){
                    //Store for what time and angles the pandulum has flipped
                    file2 << t << "   " << i * (2 * M_PI / dati) - M_PI
                          << "   " << j * (2 * M_PI / dati) - M_PI << endl;
                    flip2 = 1;
                }
                //Store the values of the angles
                if(flip1 == 0) theta1_old = Y[0];
                if(flip2 == 0) theta2_old = Y[1];
                //Break when both pendula have flipped
                if(flip1 == 1 && flip2 == 1) break;
                //Increase time
                t += dt;

            }
            //Store for which set of angels the pendulum hasn't flipped
            if(flip1 == 0){
                file1 << -1 << "   " << i * (2 * M_PI / dati) - M_PI
                      << "   " << j * (2 * M_PI / dati) - M_PI << endl;
            }
            if(flip2 == 0){
                file2 << -1 << "   " << i * (2 * M_PI / dati) - M_PI
                      << "   " << j * (2 * M_PI / dati) - M_PI << endl;
            }
            //Going back to initial conditions
            t = 0.;
            flip1 = 0;
            flip2 = 0;
            }
    }
    //Close files
    file1.close();
    file2.close();
#endif
}

//========================RK4=========================

void RK4(double t, double *Y, double dt, int neq,
         void (*RHS_Func)(double, double*, double*)){
    // Runge-Kutta numerical method for integrating differential equations

    // Intermediate variables for RK4
    double Y1[neq], Y2[neq], Y3[neq];
    double k1[neq], k2[neq], k3[neq], k4[neq];

    // Evaluate the function at the initial point
    dYdt(t, Y, k1);

    // Calculate intermediate values for the Runge-Kutta steps
    for(int i = 0; i < neq; i++){
        Y1[i] = Y[i] + 0.5 * dt * k1[i]; // First intermediate step
    }
```

```
344     dYdt(t + 0.5 * dt, Y1, k2); // Evaluate at the midpoint
345 
346     for(int i = 0; i < neq ; i++){
347         Y2[i] = Y[i] + 0.5 * dt * k2[i]; // Second intermediate step
348     }
349     dYdt(t + 0.5 * dt, Y2, k3); // Evaluate at the midpoint again
350 
351     for(int i = 0; i < neq ; i++){
352         Y3[i] = Y[i] + dt * k3[i]; // Third intermediate step
353     }
354     dYdt(t + dt, Y3, k4); // Evaluate at the end of the interval
355 
356     // Update the solution
357     for(int i = 0; i < neq ; i++){
358         Y[i] +=  dt / 6. * (k1[i] + 2 * k2[i] + 2 *k3[i] + k4[i]);
359     }
360 }
361 
362 //=======================dYdt================================
363 
364 void dYdt(double t, double *Y, double *R){
365     double theta1 = Y[0], theta2 = Y[1]; //Angels
366     double v1 = Y[2], v2 = Y[3], den; //Angular velocities
367     //Denominator
368     den = g_l * (3. * g_m - g_m * cos((2. * theta1) - (2. * theta2)));
369     //Set of differential equations
370     R[0] = v1;
371     R[1] = v2;
372     R[2] = (- 3. * g_g * g_m * sin(theta1) - g_m * g_g * sin(theta1 - 2. * theta2)
373             - 2. * g_m * sin(theta1 - theta2) * (v2 * v2 * g_l +
374             v1 * v1 * g_l * cos(theta1 - theta2))) / den;
375     R[3] = ((2. * sin(theta1 - theta2)) * (2. * g_m * g_l * v1 * v1 + 2.* g_g * g_m
376              * cos(theta1) + v2 * v2 * g_l * g_m * cos(theta1 - theta2))) / den;
377 
378 }
```