Project 2: Moto del proiettile realistico

Umberto C. Chiapasco

April 2024

Abstract

Si presentano le caratteristiche di un metodo numerico utilizzato per calcolare la traiettoria seguita da un proiettile realistico (soggetto all'attrito dell'aria oltre che all'attrazione di gravità) per raggiungere un dato target a partire da un punto stabilito. Il metodo permette di studiare alcune caratteristiche di questo sistema, quali l'energia del proiettile e la traiettoria seguita, in diverse configurazioni.

1 Introduzione

Il moto di un proiettile soggetto all'attrazione gravitazionale ed all'attrito dell'aria è un problema ben noto in cinematica classica. Il sistema consiste di un oggetto, in prima approssimazione considerato puntiforme, che inizialmente si trova in un punto, ad esempio l'origine degli assi, con una certa velocità iniziale. L'oggetto subisce gli effetti dell'attrazione di gravità, che causa un'accelerazione costante e verticale, e dell'attrito dell'aria, la cui accelerazione è sempre orientata anti-parallelamente alla velocità istantanea. L'oggetto seguirà dunque una traiettoria non esattamente parabolica, e raggiungerà il suolo in un punto P. Ci si chiede, a partire dalla conoscenza delle coordinate di P e del modulo della velocità iniziale, quale debba essere, se esiste, l'angolo di cui deve essere inclinato rispetto al suolo il vettore velocità iniziale affinché il proiettile raggiunga il punto P.
In assenza dell'attrito (viscoso) dell'aria, il problema presenta una simmetria rispetto all'angolo $\frac{\pi}{4}$ radianti. Questa simmetria implica che, data una soluzione del problema diversa da $\frac{\pi}{4}$ radianti, ne deve esistere un'altra, simmetrica a questa rispetto a tale angolo. Ad esempio, se una soluzione fosse $\frac{\pi}{3}$, si potrebbe assumere che una seconda soluzione sia $\frac{\pi}{6}$. In presenza di attrito questa simmetria si perde, come anche la forma esattamente parabolica della traiettoria. I risultati vanno quindi ricercati singolarmente. Inoltre, il sistema di equazioni differenziali che descrive il sistema è:

$$\begin{cases} \ddot{x} = -Bvv_x \\ \ddot{y} = -g - Bvv_y \end{cases} \tag{1}$$

in cui v è il modulo della velocità istantanea, con componenti v_x e v_y, B il coefficiente di attrito dell'aria e g l'accelerazione di gravità. Una soluzione analitica di questo sistema è particolarmente difficile da ottenere. Per questo motivo risulta particolarmente interessante l'utilizzo di un metodo numerico per ottenere risultati con un buon grado di approssimazione. Si osserva inoltre che, introducendo una forza d'attrito (non conservativa) l'energia del sistema non è più conservata.
Al fine di determinare la traiettoria seguita da un proiettile in moto, con velocità iniziale $\vec{v}_0$, viene utilizzato un metodo di Runge-Kutta (e quindi *single-step*) del quarto ordine. Questo permette di stimare la soluzione di un'equazione differenziale in un certo istante, a partire dallo stato del sistema in un momento precedente e dal valore delle derivate delle componenti dello stato in prossimità di quell'istante.
La stima dei valori di theta viene ottenuta con un metodo di *single-shooting* per il *boundary value problem* definito dalle condizioni iniziali e finali sulla posizione del proiettile. Questo metodo consiste nel costruire una funzione `Residual(theta)` che costruisca la soluzione per un dato theta e determini la differenza tra il punto di arrivo di questa soluzione e il target richiesto. A questo punto il problema consiste nell'ottenere uno zero della funzione `Residual(theta)`, che corrisponderà al valore di theta per cui il bersaglio viene colpito.

2 Struttura del codice

Il codice contiene cinque funzioni principali, oltre alla funzione `main()`. Queste sono:

- `RK4Step`
- `dYdt`
- `bisection_rf`
- `bracketing`
- `Residual`

La funzione `RK4Step` è una funzione di tipo `void` che prende come argomento tre `double` (il tempo, la lunghezza temporale di uno step e un array contenente gli elementi dello stato), un `int` (il numero di equazioni) ed una funzione di tipo `void` con tre argomenti a sua volta, in questo codice questa funzione sarà sempre `dYdt`. La funzione `RK4Step` implementa il metodo di Runge-Kutta del quarto ordine che permette di stabilire il valore di una soluzione (ovvero calcolare gli elementi dello stato lungo la curva che è soluzione dell'equazione differenziale scelta) ad un istante di tempo t.
La funzione `dYdt` è una funzione di tipo `void` che prende come variabili un

`double` (il tempo) e due vettori `double` (il vettore stato Y ed un secondo vettore R) ed associa a ogni elemento del vettore R la derivata dell'elemento corrispondente per il vettore Y. Questa funzione è necessaria per il funzionamento del metodo di Runge-Kutta implementato e contiene la struttura dell'equazione differenziale fisica che descrive il sistema. Si osserva che per questo motivo l'equazione differenziale (1) viene riscritta come sistema di quattro equazioni differenziali del primo ordine:

$$\begin{aligned} \dot{x} &= v_x, \\ \dot{y} &= v_y, \\ \dot{v_x} &= -Bvv_x, \\ \dot{v_y} &= -Bvv_y - g. \end{aligned} \tag{2}$$

La funzione `bisection_rf`, in cui `rf` è acronimo di *root finder*, implementa per l'appunto un metodo di ricerca degli zeri di una funzione. Data una funzione continua con valori di segno opposto agli estremi di un intervallo (chiuso e limitato), il teorema di esistenza degli zeri afferma infatti che dovrà esistere almeno una radice della funzione tra i valori interni all'intervallo.
La funzione `bracketing` permette di suddividere il dominio considerato di una funzione in molteplici intervalli, in modo da poter applicare il metodo di bisezione su ognuno di essi, aumentando la capacità di questo metodo di trovare il reale numero di radici presenti nell'intervallo.
`Residual`, infine, come già accennato, costruisce a partire dall'angolo *theta* la traiettoria passo per passo, utilizzando la funzione `RK4Step` per determinare l'evoluzione temporale del sistema, e restituisce la distanza (orizzontale) tra il punto raggiunto ed il bersaglio. Al fine di evitare di porre condizioni di arrivo che il sistema potrebbe mal interpretare, la funzione separa la valutazione dell'evoluzione temporale in due "periodi". Il primo si conclude quando la velocità verticale si annulla (entro la tolleranza `ytol` richiesta). Una volta superata questa condizione, il programma riduce il passo dr dividendolo per 5. Questa scelta è motivata dall'osservazione del fatto che la valutazione della prima parte della traiettoria ha poca influenza sulla precisione del risultato finale[1]. Si sceglie quindi di aumentare la precisione in prossimità del target al fine di aumentare la sensibilità senza aumentare eccessivamente il numero di operazioni richieste. Il ciclo ha come condizione di uscita l'oltrepassamento del valore della coordinata y del target. Il valore della coordinata y non fa quindi parte del residuo stimato, ma serve solo per stabilire quando la massa raggiunge il target. Questo sistema funziona meglio nell'approssimazione in cui B, parametro dell'attrito, è piccolo (come nel caso che si vuole considerare), ma causa dei problemi per valori di questo parametro elevati. In questo caso risulta adeguato modificare i valori di tolleranza degli errori ed il passo dr per ottenere risultati più precisi. Il risultato finale viene ottenuto da un'interpolazione lineare tra gli ultimi due punti valutati.
Il codice, nel `main()`, quindi, suddivide il dominio della funzione `Residual` in

[1] vedere osservazioni sulla convergenza nel seguito

intervalli. Tra questi intervalli vengono considerati solo quelli in cui è possibile applicare `bisection_rf` e utilizzando quest'ultima si ottengono gli zeri della funzione `Residual`. A questo punto si produce un file contenente i dati della traiettoria seguita dal proiettile per i valori di theta che sono radici della funzione. Uno pseudocodice riassuntivo del funzionamento del `main()` è riportato alla fine del paragrafo.
I parametri liberi, facilmente modificabili nel codice, sono il **modulo della velocità iniziale**, la **posizione del target**, le costanti **g** (accelerazione di gravità) e **B** (coefficiente d'attrito viscoso). È quindi possibile produrre dei risultati variando non solo il modulo della velocità iniziale e la posizione del target, ma anche i parametri fisici dovuti all'ambiente esterno al sistema.

```
// definizione delle funzioni:
void RK4Step(double, double *, void(*)(double, double *, double *),
  double, int);
void dYdt(double, double *, double *);
int bracketing(double(*)(double),double, double, int, int&,
  double*, double*);
double Residual(double);
int bisection_rf(double(*)(double), double, double, double,
  int&, double, double&);

main(){

  bracketing(Residual(),N); // divide il dominio di Residual in N parti
  for(){ // in ognuna delle parti in cui e' possibile, viene utilizzato
      bisection_rf
    bisection_rf(Residual(),root[i]); // inserisce nell'array root[i] le
        radici di residual
  }
  ofstream fdata;
  fdata.open("chiapasco_project2.1_data.dat"); // i dati vengono
      inseriti in un file .dat
  for(roots){ // per ogni radice di Residual() ottenuta si studia la
      traiettoria
    while(Y[1]>y_target){
      E = ... // l'energia del sistema viene valutata all'istante r
      fdata << r << E << Y << endl;
      // Y e' l'array contenente le informazioni cinematiche del sistema
      RK4Step(Y, r);
    }

  }
  fdata.close();
  return 0;
}
```

---

3 Analisi dei risultati

Simmetria a $\pi/4$

Come osservato nei paragrafi precedenti, l'introduzione del termine di attrito porta diverse conseguenze fisicamente osservabili. Tra queste, le più evidenti sono la rottura della simmetria a $\frac{\pi}{4}$ del punto in cui il proiettile raggiunge il terreno in funzione dell'angolo di partenza e la non conservazione dell'energia. Queste conseguenze sono osservabili sperimentalmente nei sistemi reali e sono ben riprodotte dai dati generati in questa simulazione.
A titolo di esempio si riportano in Tabella 1 una serie di coppie di angoli, ottenuti per diversi valori dei parametri liberi, ed il loro valore medio. Si osserva che la differenza dal valore $\frac{\pi}{4} \approx 0.78539816$ è maggiore nel caso in cui viene scelto un coefficiente d'attrito maggiore.

parametri liberi	θ_1	θ_2	θ_{media}
B = 4.e-5, v = 10.0, x = 5.0	0.256090	1.31439	0.78524
B = 4.e-5, v = 15.0, x = 10.0	0.225692	1.34506	0.78537
B = 4.e-5, v = 8.0, x = 4.0	0.329929	1.24097	0.78544
B = 4.e-2 v = 15.0, x = 5.0	0.126685	1.38521	0.75594

Tabella 1: Valori di theta ottenuti per diversi B, v, x. L'altezza del target e l'accelerazione di gravità sono assunte costanti e pari, rispettivamente, a $0m$ e $9.81\frac{m}{s^2}$.

Energia

Una seconda importante conseguenza della presenza dell'attrito dell'aria è la non conservazione dell'energia del proiettile, parte della quale viene trasmessa all'ambiente circostante. L'andamento dell'energia simulata in funzione della distanza percorsa (in orizzontale) è riportato nel grafico in Figura 1.

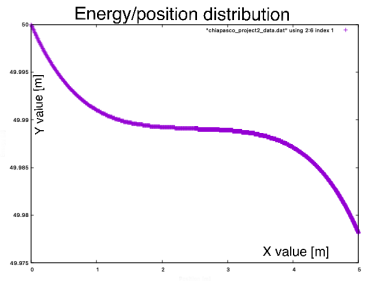
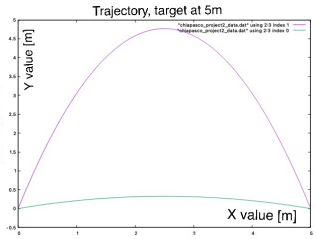
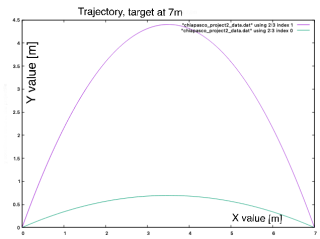


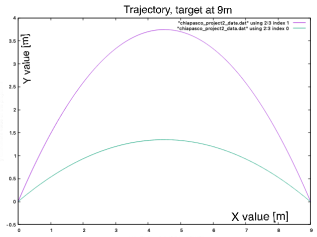
Figura 1: Grafico dell'energia in funzione della coordinata spaziale orizzontale. I parametri liberi sono scelti pari a: $v = 10m/s$, $B = 4.0 \cdot 10^{-5}\frac{1}{m}$, posizione del target: $x = 5m, y = 0m$. L'energia non resta costante durante il tragitto



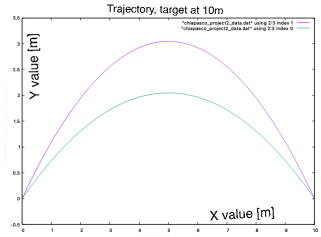
(a) Target: $5m$



(b) Target: $7m$



(c) Target: $9m$



(d) Target: $10m$

Figura 2: Traiettorie simulate per il lancio del proiettile in presenza di attrito per diverse scelte della posizione del target. La velocità iniziale del proiettile è fissata a $v = 10.0m/s$, il coefficiente d'attrito e l'accelerazione di gravità sono nelle condizioni standard ($B = 4.0 \cdot 10^{-5} \frac{1}{m}$, $g = 9.81 \frac{m}{s^2}$. Per ogni grafico sono riportate entrambe le traiettorie ottenute. All'aumentare della distanza del target, e quindi in prossimità alla gittata massima del proiettile, le due parabole tendono ad avvicinarsi

Si può osservare che l'energia tende a diminuire, e che essa subisce variazioni maggiori nelle zone del percorso in cui il proiettile si muove più rapidamente, ovvero nei punti più lontani dal punto di massimo. Questa osservazione qualitativa è in accordo con ciò che ci si aspetta di ottenere dalla teoria.

Traiettoria

La scelta di ridurre il passo nella seconda parte della traiettoria permette di migliorare la sensibilità dei risultati, senza aumentare eccessivamente il numero di operazioni richieste. La condizione di uscita dal loop per il calcolo della traiettoria è il superamento della coordinata verticale del proiettile con quella del target. In prossimità di questo valore, si richiede di aumentare la precisione al fine di evitare una valutazione eccessivamente approssimativa del punto di arrivo.
I grafici della traiettoria calcolata per diversi valori dei parametri liberi sono riportati in Figura 2.

La forma dei grafici ottenuti per diversi valori della distanza del target è quasi parabolica. Questo è dovuto al valore ridotto del coefficiente di attrito viscoso B. Considerando quest'ultimo come parametro libero è possibile studiare qualitativamente la variazione della forma della traiettoria all'aumentare del coefficiente di attrito. I risultati sono riportati in Figura 3

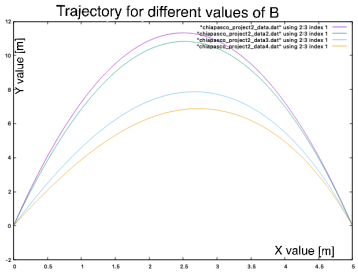


Figura 3: Traiettoria del proiettile per diversi valori del coefficiente di attrito B, all'aumentare di quest'ultimo la curva seguita perde gradualmente le proprietà di simmetria assiale della parabola. I valori di B sono (dall'alto verso il basso): $4.0 \cdot 10^{-5} \frac{1}{m}$, $4.0 \cdot 10^{-3} \frac{1}{m}$, $4.0 \cdot 10^{-2} \frac{1}{m}$ e $6.0 \cdot 10^{-2} \frac{1}{m}$. La velocità iniziale e la posizione del target sono fissati e valgono, rispettivamente, $15m/s$ e $5m$.

Si osserva che, all'aumentare del coefficiente di attrito, le traiettorie perdono la loro simmetria rispetto all'asse verticale passante per il punto di massimo. Inoltre, l'angolo ottenuto tende, in questo contesto, ad avvicinarsi all'angolo di gittata massima al fine di raggiungere il target. Scegliendo la classe di curve con angolo iniziale maggiore di quello della traiettoria di gittata massima, le curve tendono quindi ad "abbassarsi", oltre che a perdere la simmetria.
Infine, il codice è stato costruito per poter simulare anche il moto di un proiettile diretto verso un target sopraelevato. In questo sistema la traiettoria viene comunque calcolata nel modo visto in precedenza, determinando il punto di massimo in una prima parte e poi (dopo aver ridotto il passo temporale) valutando il raggiungimento della coordinata y richiesta. Si riporta, in Figura 4, una traiettoria calcolata per un bersaglio sopraelevato di un metro rispetto al suolo.

Convergenza

Si osserva che la differenza tra il valore di x ottenuto ed il target si riduce quando il passo viene diminuito. Riducendo valore di dr si suppone che questa differenza tenda a ridursi gradualmente e che la convergenza venga raggiunta.
Per studiare questo fatto si sceglie un set di condizioni iniziali e si studia la differenza, ad angolo fissato, tra il valore dell'ascissa ottenuto e quello del target. Si scelgono quindi le condizioni standard riportate nei paragrafi precedenti per il valore di g e di B, la posizione del target ed il modulo della velocità iniziale. Mantenendo costante il valore ottenuto dei due angoli θ, si studia quindi

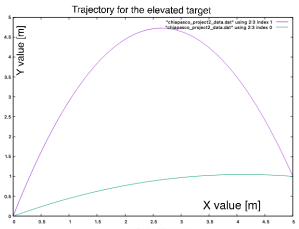


Figura 4: Traiettorie del proiettile ottenute per un bersaglio elevato di $1m$ rispetto al suolo, la velocità iniziale ha modulo $10m/s$ e l'ascissa del target è $x = 5m$, g e B hanno i valori standard indicati in precedenza

la deviazione dalla posizione del target. I risultati di alcune simulazioni sono riportati in Tabella 2.

dr $[s]$	Δ_1 $[m]$	Δ_2 $[m]$
0.005	0.009149	0.0022202
0.01	0.009178	0.0047342
0.03	0.009149	0.0098255
0.05	0.067205	0.0047342
0.1	0.163917	0.0300853
0.15	0.067205	0.0554365
0.2	0.357342	0.0300853
0.5	0.163917	0.1821940
0.01	0.067205	0.004734
0.02	0.163917	0.030085
0.1	0.163917	0.182194
0.2	1.131060	0.435715

Tabella 2: Deviazioni dal valore di $x = 5m$ del target per diversi valori del passo temporale dr. Le prime 8 righe sono ottenute con la scelta di ridurre il passo temporale nella seconda parte della traiettoria, le successive 4 sono invece riferite alla scelta di non ridurre il passo temporale.

Osservando i dati riportati, si può osservare che la deviazione tende a stabilizzarsi quando i valori di dr si avvicinano a $\simeq 0.01s$. Simulazioni con valori di dr inferiori riportano risultati simili. Come atteso, al diminuire del valore di dr diminuisce anche la distanza tra il valore dell'ascissa ottenuto con il *single-shooting method* e l'ascissa del target.
Dalla tabella si evince inoltre il motivo per il quale si sceglie di ridurre il passo temporale durante la valutazione della seconda parte della traiettoria. Si osser-

va infatti che il valore ottenuto di Δ per un dato dr che utilizzi il metodo visto fino ad ora è estremamente simile nel caso in cui si scelga un dr che sia 5 volte inferiore senza la riduzione del passo temporale.
In questi due casi, il passo temporale in prossimità del raggiungimento del target è lo stesso, ma nel primo il numero di operazioni richieste è inferiore, in quanto utilizza un dr maggiore per la prima parte della traiettoria. Si può quindi affermare che, fatta eccezione per alcuni casi patologici, la precisione nella prima parte della valutazione della traiettoria è meno rilevante che nella seconda parte. Per questo motivo si è deciso di utilizzare il metodo proposto.

Conclusioni

Si è mostrato come il codice in esame possa simulare il moto di un oggetto sottoposto ad un campo gravitazionale e all'attrito di un mezzo viscoso. L'algoritmo viene utilizzato per studiare l'evoluzione temporale e la traiettoria di un proiettile realistico in aria, questo problema è infatti difficilmente risolvibile con metodi analitici. La traiettoria seguita differisce da quella classicamente ottenuta per sistemi conservativi. In questo caso l'energia non è conservata e la simmetria della traiettoria rispetto al suo asse principale viene rotta. Inoltre, la simmetria a $\frac{\pi}{4}$ dell'angolo a cui il proiettile deve essere lanciato per raggiungere un target fisso con una certa velocità iniziale viene a sua volta rotta, come conseguenza della presenza di attrito. Il codice è stato utilizzato per studiare le caratteristiche di questo tipo di sistema al variare dei parametri liberi di: velocità iniziale (in modulo), posizione del target (con la possibilità di scegliere un target sopraelevato), accelerazione di gravità e coefficiente di attrito. Si mostra che l'andamento dei risultati rispecchia, qualitativamente, quanto atteso dalla conoscenza teorica del sistema fisico reale.

Codice

```
//
//  chiapasco_project2.cpp
//  Project2
//
//  Created by Umberto Carlo Chiapasco
//

#include <iostream>
#include <cmath>
#include <fstream>
#include <iomanip>

using namespace std;

#define DEBUG 0

const double g_vel_i = 10.0; // the total speed of the projectile is
// initially assumed equal (in module) to 10 m/s
const double g_xf = 5.0; // the target is the point (5.0m,1.0m)
const double g_yf = 0.0;
const double g_grav = 9.81;
// note: the bullet mass is assumed equal to 1 kg


void RK4Step(double, double *, void(*)(double, double *, double *),
  double, int);
void dYdt(double, double *, double *);
int bracketing(double(*)(double),double, double, int, int&,
  double*, double*);
double Residual(double);
int bisection_rf(double(*)(double), double, double, double,
  int&, double, double&);

int main(){
  double a = 0., b = M_PI*0.5, tol = 1e-7, xtol = 0.01;
  // note that:
  // tol is used in bisection as the width of the interval around the
  // root
  // xtol is the tolerance for the distance between the x-result at
  // the chosen theta and the actual target
  int N = 10, nroots, nbisec, op;
  double sr[64], sl[64], root[64];
  // part 1: identify the roots of Residual()
  op = bracketing(Residual, a, b, N, nroots, sl, sr);
  if(op != 0) { cout << "Error occurred on bracketing()\n"; }
  #if DEBUG
  cout << "Bracketing executed. Number of roots: " << nroots << endl;
  #endif
  for(int i=0; i < nroots; i++){
    op = bisection_rf(Residual,sl[i],sr[i],tol,nbisec,xtol,root[i]);
    if(op != 0) { cout << "Error occurred on bisection_rf()\n"; }
    #if DEBUG
    cout << "Bisection: root n. " << i+1 << ": " << root[i] << endl;
    #endif
  }

  // part 2: check solutions and save trajectory data
  double ytol = 0.001, y0, y1, x0;
  // ytol is the tolerance on the y-axis, useful to determine
  // the proximity to the ground in the 2nd cycle
  int nstepsmax = 10000, neq = 4, n;
  double rb = 0.0, r = rb, dr = 0.01, E; // dr = 0.01*g_xf/g_vel_i;
  double v_iy;
  double Y[4];
  bool halfreached = 0;
  ofstream fdata;
```

```
  fdata.open("chiapasco_project2.1_data.dat");
  // loop over the roots of Residual() found in part 1:
  for(int i=0; i < nroots; i++){
    n = 0;
    r = rb;
    v_iy = g_vel_i*sin(root[i]);
    Y[0] = 0.0; Y[1] = 0.0; // initial point
    Y[2] = g_vel_i*cos(root[i]);   // initial x speed
    Y[3] = v_iy;                   // initial y speed
    ////////////////////////////////////////////////////
    // Compute the result obtained with RK4 shooting:
    // The loop has been split in order to avoid requiring the
    // condition on Y in the first part of the trajectory.
    // 1st half of the trajectory:
    while(n < nstepsmax && Y[1] >= g_yf) {
      // save trajectory data on "chiapasco_project2.1_data.dat"
      E = 0.5*Y[2]*Y[2] + 0.5*Y[3]*Y[3] + g_grav*Y[1];
      fdata << r << "  " << Y[0] << "  " << Y[1] << "  " << Y[2];
      fdata << "  " << Y[3] << "  " << E << endl;
      x0 = Y[0];
      RK4Step(r,Y,dYdt,dr,neq);
      r+=dr;
      if(Y[3] < ytol && !halfreached){ dr/=5.0; halfreached = 1; }
      #if DEBUG
      cout << "*";
      #endif
      n++;
    }
    E = 0.5*Y[2]*Y[2] + 0.5*Y[3]*Y[3] + g_grav*Y[1];
    fdata << r << "  " << Y[0] << "  " << Y[1] << "  ";
    fdata << Y[2] << "  " << Y[3] << "  " << E << endl;
    // we now build the value of x obtained to compare it with
    // the x of the target. To do this, we use linear interpolation
    // with the last two values of x (the one just above and that just
    // below the target y):
    //// y0: height of point 0 above the target
    //// y1: height of point 1 below the target
    //// x0: ascissae of point 0
    y0 = y0 - g_yf;
    y1 = g_yf - Y[1];
    // the value of x that has been reached is obtained by linear
    // interpolation between the last two points:
    Y[0] = (Y[0]*y0 + x0*y1)/(y1 + y0);
    // Note: division by zero is excluded since the step (y1-y0) != 0
    // and y1, y0 are >= 0. Therefore at least one of them must be > 0
    ////////////////////////////////////////////////////
    #if DEBUG
    cout << endl;
    if(n >= nstepsmax){ cout << "nstepsmax reached.\n"; }
    else if(Y[1] - g_yf < ytol){cout << "mass reached the ground.\n";}
    else { cout << "an error occurred.\n"; }
    #endif
    fdata << endl << endl;
    cout << "Theta value " << i + 1 << ": " << root[i];
    cout << ".\n  x result: " << Y[0] << ", target: " << g_xf;
    cout << ".\n  Difference: " << fabs(Y[0] - g_xf);
    if(fabs(Y[0] - g_xf) < xtol){
      cout << " has acceptable deviation\n";
    }
    else{
      cout << " is beyond the required tolerance on x: ";
      cout << xtol << "\n";
    }
  }
  fdata.close();
  return 0;
}

void RK4Step(double t, double *Y, void (*RHSFunc)(double, double *,
  double *), double h, int Neq){
  double Y1[256], k1[256], k2[256], k3[256], k4[256];
  int i;
```

```
  RHSFunc(t,Y,k1);
  for (i=0;i<Neq;i++) Y1[i] = Y[i] + 0.5*h*k1[i];
  RHSFunc(t+0.5*h,Y1,k2);
  for (i=0;i<Neq;i++) Y1[i] = Y[i] + 0.5*h*k2[i];
  RHSFunc(t+0.5*h,Y1,k3);
  for (i=0;i<Neq;i++) Y1[i] = Y[i] + h*k3[i];
  RHSFunc(t+h,Y1,k4);
  for (i=0;i<Neq;i++) Y[i] += h/6.0*(k1[i]+2.0*k2[i]+2.0*k3[i]+k4[i]);
}

int bisection_rf(double (*F_fr)(double), double a, double b,
  double tol, int &n, double ytol, double &root){
  // on return, bisection_rf returns:
  // 0: no errors ocurred
  // 1: initial values problems
  // 2: result problems
  if(a>b){
    cout << "! bisection_rf: Error: b must be greater than a\n";
    return 1;
  }
  if(tol<0){
    cout << "! bisection_rf: Error: tol must be greater than 0\n";
    return 1;
  }
  double f_l, f_r;
  f_l = F_fr(a);
  f_r = F_fr(b);

  if(f_l*f_r >= 0){
    cout << "! bisection_rf: Error: interval must have positive";
    cout << " & negative extremes in order to use this function;\n";
    return 1;
  }
  double x_mid,f_mid;
  int i;
  n = log2(1./tol);
  #if DEBUG
  cout << "f_l: " << f_l << " || f_r: " << f_r << endl;
  #endif
  for(i=0; i<n; i++){
    x_mid = (a+b)/2.0;
    f_mid = F_fr(x_mid);
    #if DEBUG
    cout << "x_mid: " << x_mid << " || f_mid: " << f_mid << endl;
    #endif
    if(f_l*f_mid>=0){a = x_mid;}
    else{b = x_mid;}
  }
  root = a;
  if(F_fr(root) >= ytol){
    cout << "! bisection_rf: An error occurred, the function did ";
    cout << "not converge to zero. Try again with lower tolerance.\n";
    return 2;
  }
  return 0;
}

int bracketing(double (*F_fr)(double), double a, double b,
  int N_intervals, int &n_roots, double *xL_arr,double *xR_arr){
  // on return, bracketing returns:
  // 0: no errors ocurred
  // 1: initial values problems
  // 2: result problems

  if(a>b){
    cout << "! bracketing: Error: b must be greater than a\n";
    return 1;
  }
  int i;
  double len = (b - a)/N_intervals, f_l,f_r;
  n_roots = 0;
  f_r = F_fr(a);
```

```
  for(i=1;i<=N_intervals;i++){
    f_l = f_r;
    f_r = F_fr(a + i*len);
    if(f_l*f_r<0){ // => the function changes sign in this interval
      *xL_arr = a + (i-1.0)*len;
      *xR_arr = a + i*len;
      xL_arr++;
      xR_arr++;
      n_roots++;
    }
  }
  return 0;
}

void dYdt(double t, double *Y, double *R){
  double B = 4.e-5;
  R[0] = Y[2];
  R[1] = Y[3];
  R[2] = -B*Y[2]*sqrt(Y[2]*Y[2] + Y[3]*Y[3]);
  R[3] = -g_grav - B*Y[3]*sqrt(Y[2]*Y[2] + Y[3]*Y[3]);
}

double Residual(double theta){
  // Residual function returns the error committed on the
  // shooting at angle theta
  // Note that the only residual (and therefore tolerance)
  // we consider is that on the x result. The height is only
  // useful in order to calculate the resulting x range.
  // note that a lot of lines in this function are commented.
  // This is mainly useful if one wants to debug this function
  // separately from the rest of the code
  double ytol = 0.001, x0, y0, xf, y1;
  // ytol has the same meaning as in main()
  int nstepsmax = 10000, neq = 4, n = 0, E;
  double rb = 0.0, r = rb, dr = 0.01; // dr = 0.01*g_xf/g_vel_i;
  double v_iy = g_vel_i*sin(theta);
  double Y[4];
  bool halfreached = 0;
  Y[0] = 0.0; Y[1] = 0.0; // initial point
  Y[2] = g_vel_i*cos(theta);
  Y[3] = v_iy;
  // halfway can be chosen in different ways (see main()). We
  // choose it as approximed top point of the trajectory;
  // halfway = Y[2]*Y[3]/g_grav;
  // #if DEBUG
  // cout << "I cicle\n";
  // #endif
  // ofstream fdata;
  // fdata.open("chiapasco_project2_data.dat");
  ////////////////////////////////////////////////////
  // 1st half of the trajectory:
  while(n < nstepsmax && Y[1] >= g_yf) {
    // save data on file to show the trajectory:
    // E = 0.5*Y[2]*Y[2] + 0.5*Y[3]*Y[3] + g_grav*Y[1];
    // fdata << r << "  " << Y[0] << "  " << Y[1] << "  " << Y[2];
    // fdata << "  " << Y[3] << "  " << E << endl;
    x0 = Y[0];
    y0 = Y[1];

    RK4Step(r,Y,dYdt,dr,neq);
    r+=dr;
    if(Y[3] < ytol && !halfreached){ dr/=5.0; halfreached = 1; }

    // #if DEBUG
    // cout << "*";
    // #endif
    n++;
  }
  // y0: height of point 0 above the target
  // y1: height of point 1 below the target
  // x0: ascissae of point 0; xf: esteemed value of x
  y0 = y0 - g_yf;
```

```
  y1 = g_yf - Y[1];
  // the value of x that has been reached is obtained by linear
  // interpolation between the last two points
  xf = (Y[0]*y0 + x0*y1)/(y1+y0);
  // #if DEBUG
  // cout << endl;
  // if(n >= nstepsmax){ cout << "nstepsmax reached.\n"; }
  // else if(Y[1]-g_yf < ytol){cout << "mass reached the ground.\n";}
  // else { cout << "an error occurred.\n"; }
  // #endif
  // fdata.close();
  return xf - g_xf;
}
```