Laboratorio di Software per il Calcolo Scientifico 2

INTERFACCIA GRAFICA PER IL METODO DI SCHMIDT

Calcolo e visualizzazione della traiettoria di un proiettile con l'interfaccia grafica Matlab

Sebastiano Ferraris

Anno Accademico 2007-2008

La natura e le leggi della natura giacevano nascoste nella notte; Dio disse: "Che Newton sia!", e luce fu.
- Alexander Pope

Indice

Prefazione

Questa tesi di LAB CS2 intende essere una evoluzione della tesi di presentata l'anno scorso durante l'esame di LAB CS1. I primi tre capitoli sono dedicati alle argomentazioni della tesi precedentemente sviluppata con le opportune modifiche, per poter trattare, nell'ultimo capitolo, l'argomento centrale di LAB CS2: l'interfaccia grafica GUI di MATLAB.

Capitolo 1

Problema

Nella prima parte di questa ricerca si vuole trovare un modo per calcolare e visualizzare la traiettoria di un proiettile, con Matlab.
Prima di affrontare la parte relativa all'algoritmo e al calcolo numerico, è opportuno richiamare alcuni concetti di balistica, e di fisica, sui quali sarà basata la soluzione del problema [http://www.earmi.it/balistica/fisica.htm].

1.1 Nomenclature

La **balistica** è quel ramo della fisica meccanica che studia il moto dei proiettili.
La **balistica interna** studia il moto all'interno della canna dell'arma, la **balistica esterna** studia il moto fra la canna dell'arma e il bersaglio, e la **balistica terminale** studia il moto nel bersaglio colptio.
La **traiettoria** è la line curva percorsa dal centro di gravità della pallottola durante il suo movimento.
La **linea di proiezione** è il prolungamento dell'asse della canna, nel momento in cui il proettile viene sparato; ovvero la tangente alla traiettoria nell'origine.
L'**angolo di proiezione** è l'angolo compreso fra la linea di proiezione e l'orizzonte.
La **gittata** è la distanza fra l'origine e il punto di caduta.

T	Linea di proiezione
α	Angolo di proiezione
OG	Gittata
V	Vertice della traiettoria

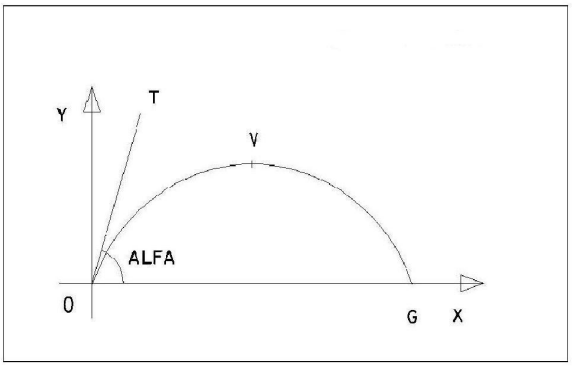

Figura 1.1: Figura delle nomenclature disegnata con il sofrware AutoCAD

1.2 Fisica: moto di un proietto nel vuoto e nell'aria

Nel vuoto, un proiettile durante il suo tragitto, è sottoposto alla somma di due forze: il vettore velocità iniziale, che imprime al proiettile un moto rettilineo uniforme lungo l'asse orizzontale, e la forza di gravità, rivolta verso il basso, che tende a far cadere il proiettile con un moto accelerato uniforme. Dato il vettore velocità iniziale $\overline{v}$ che avrà direzione parallela alla linea di proiezione, e sarà esprimibile in componenti come somma di due vettori paralleli agli assi $\overline{v}_x$ e $\overline{v}_y$, posso calcolare il vettore spostamento $\overline{s}$, che unisce due punti successivi della traiettoria, come la somma di due vettori, paralleli agli assi coordinati: $\overline{s} = \overline{s}_x + \overline{s}_y$. Dove $\overline{s}_x = \overline{v}_x \cdot t$ ed $\overline{s}_y = \overline{v}_y \cdot t - 1/2 \cdot \overline{g} \cdot t^2$. La traiettoria descritta dai successivi vettori spostamento, è una parabola simmetrica, rivolta verso il basso ad asse verticale [Paul Tipler "Fisica I "Zanichelli].

Nell'aria (in assenza di vento), oltre al vettore velocità iniziale, e alla forza di gravità, il proiettile è sottoposto anche alla resistenza dell'aria, opposta alla direzione del moto del proiettile. La traietoria finale, in due dimensioni non sarà più una parabola simmetrica, ma sarà una parabola asimmetrica, con il ramo ascendente più lungo di quello discendente. Per disegnare una approssimazione della traiettoria nell'aria, si può usare il metodo di Schmidt.

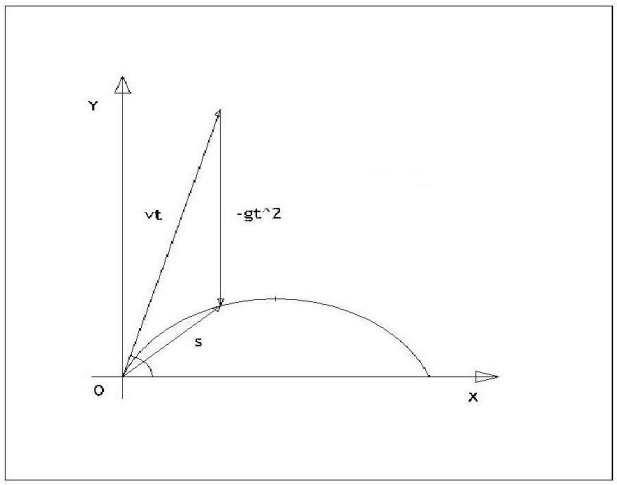

Figura 1.2: Traiettoria

Capitolo 2

Soluzione

Per il calcolo di una traiettoria parabolica si può utilizzare il metodo di Schmidt.

2.1 Metodo di Schmidt

Il metodo di Schmidt, parte dall'idea di approssimare la traiettoria di un proiettile nell'aria, con la traiettoria di un proiettile nel vuoto, il quale sarà però anche sottoposto ad una forza di verso opposto alla direzione del suo moto, che simula la forza resistente impressa dall'aria.
Il proiettile risulterà quindi sottoposto a tre vettori diversi: la gravità, la velocità istantanea e la forza resistente. Il punto di applicazione di questi vettori è il baricentro della pallottola.

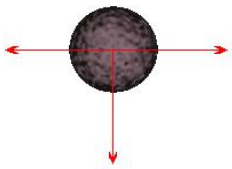

Figura 2.1: Proiettile sottoposto al vettore gravità, velocità e resistenza disegnata con il sofrware POV-RAY

Per applicare il metodo di Schmidt, che si basa sulla seguente proprietà 1, servono l'angolo di proiezione, la gittata ed l'ordinata del vertice.

Proprietà 1. *Date due tangenti ad una parabola, la retta che congiunge la metà dei due tratti, misurati dal punto di contatto al punto di incontro, è a sua volta tangente alla parabola.*

Per disegnare la traiettoria parabolica si parte dalla linea di proiezione, misurata fra l'origine la sua intersezione B con la retta perpendicolare ad OG passante per il vertice. Una volta definito il segmento OB, lo si raddoppi, sulla linea di proiezione, per trovare il punto C. La seconda tangente alla parabola, nel punto di caduta, sarà dato dalla retta passante per CZ, e il punto D sarà la sua intersezione con la retta perpendicolare ad OG passante per il vertice.
A questo punto si prendano in considerazione i quattro segmenti trovati, tangenti alla parabola: OB, BV, VD, DG, e i trovino i punti medi di ciascun segmento, rispettivamente A_1, A_2, A_3, A_4. Per la proprietà 1, so che la parabola è tangente ai quattro punti trovati. Trovando i punti medi dei segmenti formati da OA_1, A_1B, BA_2, A_2V, VA_3, A_3D, DA_4, A_4G si ottengono sei ulteriori punti di tangenza alla parabola B_1, B_2, B_3, B_4, B_5, B_6. Reiterando il ragionamento, si possono trovare ad ogni $n-$esimo passaggio $(n+1)\cdot 2$ nuovi punti di tangenza.
Quando si è raggiunto un numero di punto soddisfacente, si potrà costruire la parabola passante per i punti trovati, che corrisponde ad una approssimazione della parabola reale, con il metodo di interpolazione.

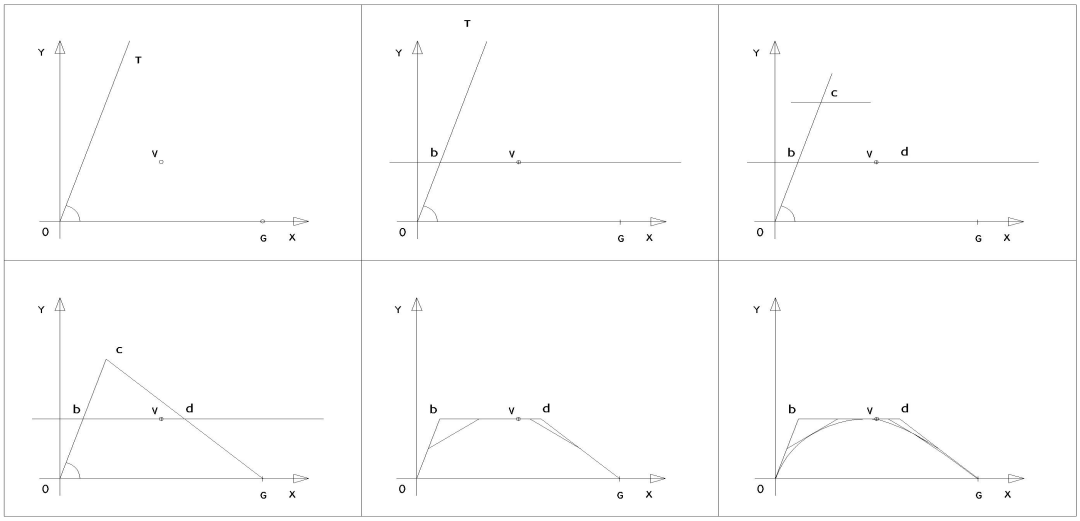

Figura 2.2: Figura dell'algoritmo di Schmidt

Capitolo 3

Algoritmi

3.1 Algoritmo di Schmidt in C

Per trovare i punti di approssimazione di una traiettoria con l'algoritmo di Schmidt usando Matlab, può essere utile scrivere prima il programma in linguaggio C. Esso si articola in due fasi diverse: la prima consiste nel trovare i 5 punti di partenza O, b, v, d, G della figura 2.2, che formano la "struttura iniziale".
La seconda fase consiste in un ciclo nel quale si trovano i punti medi dei punti medi dei punti della "struttura iniziale ", in modo da avere così i punti di tangenza cercati. Aumentando il numero di iterazioni del ciclo, si aumenterà il livello di precisione nell'approssimazione della traiettoria.

```
//programma per il calcolo del metodo di Schmidt

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define EPSILON 1

struct Punto
{
  double x;
  double y;
};

//Funzione della distanza fra due punti
```

```
double Dist(Punto uno, Punto due)
{
  return(sqrt((uno.x - due.x)* //vado a capo
  *(uno.x - due.x)+(uno.y - due.y)*(uno.y - due.y)));
}

// Funzione che calcola il punto medio tra sin e des

Punto PuntoMedio(Punto sin, Punto des)
{
  Punto centro;

  centro.x = (sin.x + des.x) / 2.0;
  centro.y = (sin.y + des.y) / 2.0;

  return(centro);
}

// funzione che calcola i punti medi e il punto medio dei medi

void CalcSeg(Punto tang1, Punto centro, Punto tang2)
{
  Punto CentroSinistra, CentroDestra;
  Punto CentroCentro;

  if(Dist(tang1, tang2) < EPSILON)
  //dove EPSILON e' l'approssimazione cercata
    return;

  CentroSinistra = PuntoMedio(tang1, centro);
  CentroDestra = PuntoMedio(tang2, centro);
  CentroCentro = PuntoMedio(CentroSinistra, CentroDestra);

  CalcSeg(tang1, CentroSinistra, CentroCentro);
  CalcSeg(CentroCentro, CentroDestra, tang2);

  printf("(%10.5f, %10.5f)\n", CentroCentro.x, CentroCentro.y);
}

main()
```

```
  {
  double gittata, ordvert;
  Punto origine, destinazione, vertice;

  printf("Programma per il calcolo della traiettoria di un proiettile \n");
  printf("con il metodo di Schmidt. \n");
  printf("Saranno forniti in ordine, tutti i punti consecutivi \n");
  printf("per i quali passa la traiettoria \n");
  printf("con distanza reciprica minore di EPSILON \n");
  printf("dati iniziali:  \n");

  printf("introdurre la gittata: \n");
  scanf("%lf",&gittata);

  printf("introdurre l'ordinata del vertice:\n");
  scanf("%lf",&ordvert);

  // Calcolo del trapezio iniziale con il metododi Schmidt
  // Per ora li mettiamo a mano
  origine.x = 0.0;
  origine.y = 0.0;

  destinazione.x = gittata;
  destinazione.y = 0.0;

  vertice.x = gittata * 0.555; //l'ordinata del vertice e' approssimata.
  vertice.y = ordvert;

  printf("Punti della traiettoria:\a\n\n");
  CalcSeg(origine, vertice, destinazione);

  system("pause");
return 0;
}
```

3.2 Algoritmo di Schmidt in Matlab

Ora ci spostiamo in ambiente Matlab. Per calcolare una traiettoria con l'algoritmo di Schmidt, si può usare un procedimento ricorsivo, che si basa su sette function. Le prime tre servono da supporto per trasformare i dati iniziali nei primi tre pun-

ti ai quali applicare il procedimento ricorsivo per una serie di volte. La quarta e la quinta function supportano il procedimento ricorsivo della sesta, e l'ultima si occupa di acquisire i dati in entrata, dirigere le function di supporto e fornire sia un risultato sotto forma di matrice direttamente nel workspace, sia di fornire un grafico del plottaggio dei punti trovati.

1. Function per trovare coefficente angolare e termine noto di una retta parallela ad una retta data per un punto dato.
   Modello: [m,n] = RettaP (M1, Xp, Yp).
2. Function per trovare coefficente angolare e termine noto di una retta passente per due punti.
   Modello: [m1,n1] = RettaPer2 (X1, Y1, X2, Y2).
3. Function per trovare l'intersezione fra due rette.
   Modello [Xa,Ya] = InterRetta(ma , na, mb, nb).
4. Function per trovare la distanza fra due punti che serve ad uscire dal ciclo del metodo ricorsivo.
   Modello [d] = Distanza(Ax, Ay, Bx, By)
5. Function per trovare il punto medio, dalle coordinate di due punti.
   Modello [Mx,My] = PuntoMedio(Psx, Psy, Pdx, Pdy).
6. Function ricorsiva, per applicare l'algoritmo di Schmidt, anche se localmente sarebbe più corretto chiamarlo algoritmo di de Casteljau.
   Modello [Mfx,Mfy] = MedioDiMedioRicorsivo(Ax, Ay, Bx, By, Cx, Cy).
7. Function finale Schmidt, che coordina le funzioni precedenti e stampa il risultato sotto forma di matrice e di grafico della matrice.

3.2.1 Retta parallela ad una retta data, per un punto dato

```
%Retta parallela ad una retta data, per un punto dato
%nel piano ortonormale.
%
%Inputs:
%   M1 = coeff. angolare della retta nota
%   x = ascissa per la quale passa la retta cercata
%   y = ordinata per la quale passa la retta cercata
%
%Outputs:
%   [m,n] dove m Ë il coeff. angolare della retta
%         nota, e n Ë la quota sull'asse y.
%
```

```
%Usage:
%   [m,n] = RettaP (M1, Xp, Yp)

function [m,n] = RettaP (M1, Xp, Yp)
  m = M1;
  n = Yp - m*Xp;
```

Esempio 1. *Usare la funcion RettaP per trovare la retta R* $[m, n]$ *parallela alla retta* $y = 5x + 2$ *passante per l'origine.*

```
>> [m,n] = RettaP(5,0,0)

m =

     5


n =

     0
```

La retta cercata è $y = 5x$.

3.2.2 Retta per due punti dati

```
%Retta per due punti dati nel piano ortonormale
%
%Inputs:
%   (X1, Y1) = primo punto di passaggio
%   (X2, Y2) = secondo punto di passaggio
%
%Outputs:
%   m1 = coeff. angolare della retta cercata
%   n1 = termine noto della retta cercata
%
%Usage:
%   [m1,n1] = RettaPer2 (X1, Y1, X2, Y2)

function [m1,n1] = RettaPer2 (X1, Y1, X2, Y2)
  m1 = (Y2 - Y1)/(X2 - X1);
  n1 = Y1 -X1*m1;
```

Esempio 2. *Usare la funcion RettaPer2 per trovare la retta passante per* $(1;1)$ $(-2;-3)$.

```
>> [m,n] = rettaper2(1,1,-2,-3)

m =

    1.3333


n =

   -0.3333
```

3.2.3 Intersezione fra due rette

```
%Intersezione fra due rette nel piano ortonormale
%
%Inputs:
%   ma = coeff. angolare della prima retta
%   na = termine noto della prima retta
%   mb = oeff. angolare della seconda retta
%   nb = termine noto della seconda retta
%
%Outputs:
%   [Xa,Ya] = punti di intersezione fra le rette date
%
%Usage:
%   [Xa,Ya] = InterRetta(ma , na, mb, nb)

function [X,Y] = InterRetta(ma , na, mb, nb)
  if  ma == mb
      'ATTENZIONE, le due rette sono parallele'

  else
      X = (nb - na)/(ma - mb);
      Y = ma*X + na;
  end
```

Esempio 3. *Usare la funcion InterRetta per trovare il punto di intersezione fra la retta* $y = 2x + 3$ *ed* $y = -x + 6$ *.*

```
>> [x,y] = InterRetta(2,3,-1,6)

x =

     1


y =

     5
```

Il punto di intersezione cercato è $(1; 5)$*.*

3.2.4 Distanza fra due punti

```
%Funcion per trovare la distanza fra due punti
%Users
%   [d] = Distanza(Ax, Ay, Bx, By)

function [d] = Distanza(Ax, Ay, Bx, By)
d = sqrt((Ax-Bx)^2+(Ay-By)^2);
```

Esempio 4. *Usare la funcion Distanza per trovare la distanza fra i punti (0;0) e (2;2) .*

```
>> [d] = Distanza(0, 0, 2, 2)

d =

    2.8284
```

3.2.5 Punto medio

```
%Funzione che da due punti in coordinate bidimensionali
%fornisce il punto medio.
%output : (Mx;My) sono le coordinate del punto in uscita.
```

```
%input : (Psx;Psy), (Pdx;Pdy) sono le coordinate dei due punti.

function [Mx,My] = PuntoMedio(Psx, Psy, Pdx, Pdy)
  Mx = (Psx + Pdx) / 2.0;
  My = (Psy + Pdy) / 2.0;
```

Esempio 5. *Usare la funcion PuntoMedio per trovare il punto medio dei punti* $(1, 2)$, $(-1; -2)$.

```
>> [x,y] = PuntoMedio(1,2,-1,-2)

x =

     0


y =

     0
```

Il risultato è l'origine.

3.2.6 Funzione ricorsiva che trova il punto medio dei punti medi di due segmenti contigui

```
% Calcolo ricorsivo dei punti medi di due segmenti AC e CB,
% ed il punto medio dei medi.
%
%Inputs:
% A (Ax;Ay) = punto iniziale
% B (Bx;By) = punto finale
% C (Cx;Cy) = punto centrale
%
%Implicit outputs:
% mAC (mACx; mACy) = punto medio sul segmento AC
% mBC (mBCx; mBCy) = punto medio sul segmento CB
% che servir‡ a costruire M.
%
%Outputs:
```

```
% il punto medio M (Mfx;Mfy) sul segmento fra mAC ed mBC.
%
%Usage:
% [Mfx,Mfy] = MedioDiMedio(Ax, Ay, Bx, By, Cx, Cy)

function MedioDiMedioRicorsivo(Ax, Ay, Bx, By, Cx, Cy)

global Risultato;
global Indice;

if Distanza(Ax, Ay, Bx, By) < 0.1
    return
end
[mACx, mACy] = PuntoMedio(Ax, Ay, Cx, Cy);
[mBCx, mBCy] = PuntoMedio(Bx, By, Cx, Cy);
%punto medio finale:
[Mfx, Mfy] = PuntoMedio(mACx, mACy, mBCx, mBCy);

MedioDiMedioRicorsivo(Ax, Ay, Mfx, Mfy, mACx, mACy)
Risultato(Indice,:) = [Mfx Mfy];
Indice = Indice + 1;
MedioDiMedioRicorsivo(Mfx, Mfy, Bx, By, mBCx, mBCy)
```

3.2.7 Function finale

```
%Schmidt fornisce un grafico della traiettoria
%
%Inputs
% Gx = gittata (solo ascissa, l'ordinata Ë 0 di default)
% Vy = ordinata del vertice
% Tan = angolo di proiezione
%
%Outputs
%  Matrice Risultato su due colonne con le ascisse sulla prima colonna
%  e le ordinate sulla seconda
%
%Altre functions usate:
%  [m,n] = RettaP (M1, Xp, Yp)
%  [m1,n1] = RettaPer2 (X1, Y1, X2, Y2)
%  [Xa,Ya] = InterRetta(ma , na, mb, nb)
%  [Mfx,Mfy] = MedioDiMedioRicorsivo(Ax, Ay, Bx, By, Cx, Cy)
```

```
function Schmidt
'Funzione per il calcolo della traiettoria di un proiettile'
'Inserire i dati:'
Gx =input('Ascissa del punto finale -> ');
Vy =input('Ordinata del vertice -> ');
Tan =input('Tangente alla retta di proiezione  -> ');
%Linea di proiezione (retta 1)
m1 = Tan;
n1 = 0;
%retta per OG [mog, nog]
[mog, nog] = RettaPer2 (0, 0, Gx, 0);
%parallela alla retta OG passante per Vy (retta 2)
[m2, n2] = RettaP (mog, 0, Vy);
%punto P1 (intersezione retta 1 e retta 2)
[P1x,P1y] = InterRetta (m1 , n1, m2, n2);
%punto massimo C
Cx = 2*P1x;
Cy = 2*P1y;

% Per valutare se i dati sono realistici (condizione necessaria
% ma non sufficiente), l'ascissa del punto C deve essere maggiore del
% della meta' della traietoria. Una traiettoria che non soddisfa almeno
% questa condizione e' inverosimile.

if Cx < Gx/2
    'I dati iniziali non sono corretti'
else

global Risultato;
global Indice;

Risultato(:,:) = [];
Indice = 1;
%Uso la funzione ricorsiva MedioDiMedioRicorsivo che trova n
%volte il punto medio del medio. Stampo prima e dopo il punto iniziale
%e finale
Risultato(Indice,:) = [0 0];
Indice = Indice + 1;
MedioDiMedioRicorsivo(0, 0, Gx, 0, Cx, Cy)
Risultato(Indice,:) = [Gx 0];
```

```
Indice = Indice + 1;
%Comando per stampare il risultato sotto forma di matrice nel workspace:
assignin('base','Risultato',Risultato)
%Comando per plottare il risultato trovato: [Plot(x,y)]
plot(Risultato(:,1), Risultato(:,2))
end
```

Esempio 6. *Usare la funcion Schmidt per disegnare la traiettoria di un proiettile con gittata* 100 *m, ordinata massima* 40*, e tangente alla linea di tiro* 1*.*

```
ans =

Inserire i dati:

Ascissa del punto finale -> 100
Ordinata del vertice -> 40
Tangente alla retta di proiezione  -> 1
```

Compare il grafico 3.1, e la matrice Risultato di dimensioni 1994 *per* 2 *nel command window di Matlab.*

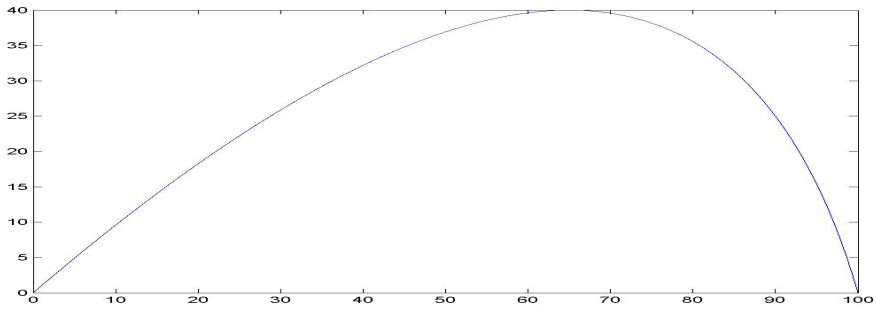

Figura 3.1: Traiettoria di un proiettile

3.2.8 Filmato della traiettoria

La funcion finale può essere anche eseguita con un filmato che disegni la traiettoria del proiettile, in questo caso salvando un frame ogni 100 punti della traiettoria.

```
%SchmidtFILM fornisce un grafico della traiettoria
```

```
%e inoltre fornisce in tempo reale il filmato della traiettoria
% Rifacendosi alla funcion Schmidt


function SchmidtFILM
'Funzione per il calcolo della traiettoria di un proiettile'
'Insertire i dati:'
Gx =input('Ascissa del punto finale -> ');
Vy =input('Ordinata del vertice -> ');
Tan =input('Tangente alla retta di proiezione  -> ');
m1 = Tan;
n1 = 0;
[mog, nog] = RettaPer2 (0, 0, Gx, 0);
[m2, n2] = RettaP (mog, 0, Vy);
[P1x,P1y] = InterRetta (m1 , n1, m2, n2);
Cx = 2*P1x;
Cy = 2*P1y;

if Cx < Gx/2
    'I dati iniziali non sono corretti'
else

global Risultato;
global Indice;

Risultato(:,:) = [];
Indice = 1;

Risultato(Indice,:) = [0 0];
Indice = Indice + 1;
MedioDiMedioRicorsivo(0, 0, Gx, 0, Cx, Cy)
Risultato(Indice,:) = [Gx 0];
Indice = Indice + 1;

assignin('base','Risultato',Risultato)

M = moviein(100);
L = length(Risultato);

%primo plottaggio
subplot(2,1,1)
```

```
plot(Risultato(:,1), Risultato(:,2))
axis equal

%secondo plottaggio
for j=1:100
  subplot(2,1,2)
  plot(Risultato([1:j*L/100],1), Risultato([1:j*L/100],2));
  axis equal
  M(j)= getframe;
end
movie(M)
assignin('base','M',M)
end
```

In questa funcion mi sono servito di due comandi: il primo

```
subplot(a,b,c)
```

mi ha permesso di plottare nello stesso screen due figure differenti, la prima quella già plottata con la function precedente, che fornisce la traiettoria fissa, e la seconda è il filmato del volo del proiettile che esegue la sua traiettoria. a corrisponde al numero di righe nelle quali viene suddiviso lo screen, b corrisponde al numero di colonne, e c corrisponde alla posizione della figura in questione nello screen
Il secondo comnado che ho usato è

```
moviein(n)
```

il quale inizializza il filmato ed associa ad M ogni fotogramma acquisito dal ciclo for. i fotogrammi vengono poi ristampati in sequenza con il comando

```
movie(M)
```

.

Capitolo 4

Interfaccia grafica per il metodo di Schmidt

Si può anche compilare l'algoritmo di Schmidt per mezzo delle interfacce grafiche di MATLAB. Questo ha lo scopo di "nascondere"il linguaggio del MATLAB, per rendere l'accesso al programma più comodo e pratico all'utente.

4.1 Introduzione alla GUI

L'interfaccia grafica (Graphical User Interface) è una finestra grafica che contiene dispositivi o componenti, utilizzabili in modo interattivo. Per crearne una si può agire in due modi: partire dalla scrittura di un m-file, che andrà successivamente compilato (Compile → Run), oppure con l'uso dell'editor GUIDE, attraverso il quale si può facilmente visualizzare e costruire l'interfaccia. Per accedere all'editor, premere il bottone in basso a sinistra della finestra principale, selezionare il menù a tendina "MATLAB"e cliccare su "GUIDE".

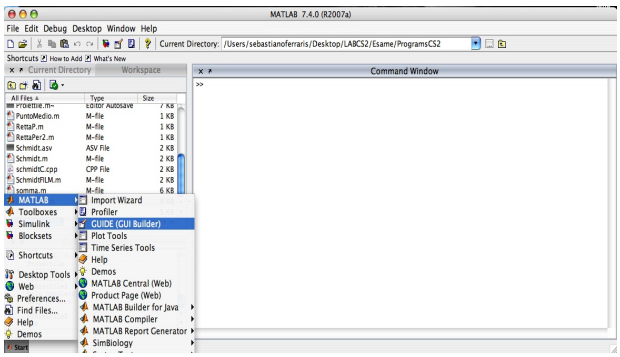

Figura 4.1: Come aprire GUIDE

Dalla finestra che si apre, si possono selezionare i vari comandi e i vari bottoni da inserire nella GUI finale.

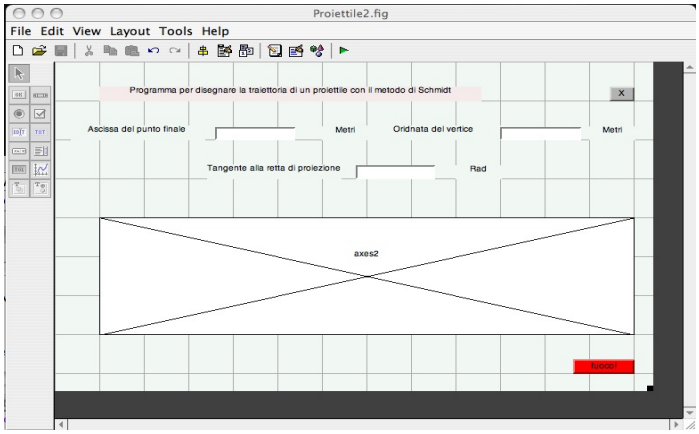

Figura 4.2: Editor

Una volta composta la GUI nell'editor, si può visualizzare il linguaggio MATLAB associato in un m-file dalla barra degli strumenti. Quindi sul monitor si avrà la finestra .fig, compilata dall'editor, la finestra dell'm-file e l'editor stesso.

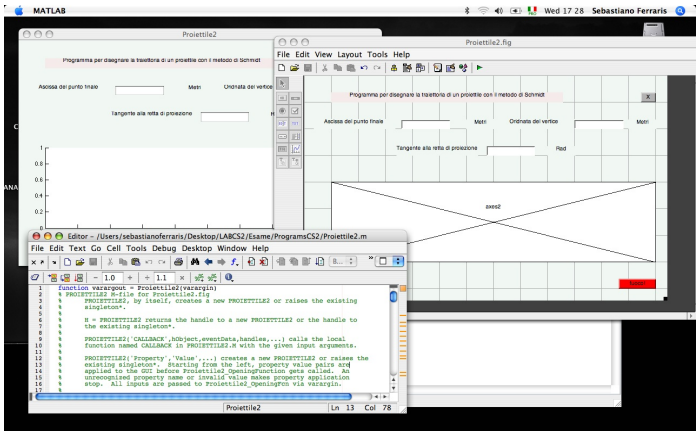

Figura 4.3: Tutti gli strumenti necessari

Ora quello che si deve fare è collegare i bottoni e le finestre grafiche dell'm-file al programma vero e proprio: nella prossima sezione viene presentato la function già completa, e nella sezione successiva vengono presentate le modifiche che sono state effettuate per far funzionare il file .fig.

4.2 Applicazione

4.2.1 M-File per l'interfaccia grafica

Dall'editor GUIDE di MATLAB, viene compilato l'm-file senza nessun collegamento con il programma che si vuole implementare. Di seguito si presenta il programma relativo all'algoritmo di Schmidt completo dei collegamenti al programma Schmidt.m

```
function varargout = Proiettile2(varargin)
% PROIETTILE2 M-file for Proiettile2.fig
%      PROIETTILE2, by itself, creates a new PROIETTILE2 or raises the existing
%      singleton*.
%
%      H = PROIETTILE2 returns the handle to a new PROIETTILE2 or the handle to
%      the existing singleton*.
%
%      PROIETTILE2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PROIETTILE2.M with the given input arguments.
%
%      PROIETTILE2('Property','Value',...) creates a new PROIETTILE2 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Proiettile2_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Proiettile2_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Proiettile2

% Last Modified by GUIDE v2.5 31-Mar-2008 19:52:54

% Begin initialization code - DO NOT EDIT
if nargin == 0  % LAUNCH GUI

hObject = openfig(mfilename,'reuse');

% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(hObject);
```

```
    guidata(hObject, handles)

if nargout > 0
varargout{1} = hObject;
end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

try
if (nargout)
[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
else
feval(varargin{:}); % FEVAL switchyard
end
catch
disp(lasterr);
end

end
% End initialization code - DO NOT EDIT


% --- Executes just before Proiettile2 is made visible.
function Proiettile2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Proiettile2 (see VARARGIN)

% Choose default command line output for Proiettile2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Proiettile2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
```

```
function varargout = Proiettile2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

Gx = str2double(get(handles.edit1,'String'));
handles.Gx=Gx;
guidata(hObject,handles);


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
... get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double

Vy = str2double(get(handles.edit2,'String'));
handles.Vy=Vy;
guidata(hObject,handles);


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
 ... get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


Tan = str2double(get(handles.edit3,'String'));
handles.Tan=Tan;
guidata(hObject,handles);


% --- Executes during object creation, after setting all properties.
```

```
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
... get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

m1 = handles.Tan;
n1 = 0;
[mog, nog] = RettaPer2 (0, 0, handles.Gx, 0);
[m2, n2] = RettaP (mog, 0, handles.Vy);
[P1x,P1y] = InterRetta (m1 , n1, m2, n2);
Cx = 2*P1x;
Cy = 2*P1y;

% if Cx < Gx/2
%     'I dati iniziali non sono corretti'
% else

global Risultato;
global Indice;

Risultato(:,:) = [];
Indice = 1;

Risultato(Indice,:) = [0 0];
Indice = Indice + 1;
MedioDiMedioRicorsivo(0, 0, handles.Gx, 0, Cx, Cy)
Risultato(Indice,:) = [handles.Gx 0];
```

```
Indice = Indice + 1;

assignin('base','Risultato',Risultato)

M = moviein(100);
L = length(Risultato);

% %primo plottaggio
% subplot(2,1,1)
%
% plot(Risultato(:,1), Risultato(:,2))
% axis equal

%secondo plottaggio
axes(handles.axes2)
cla
hold on
for j=1:100
   plot(Risultato([1:j*L/100],1), Risultato([1:j*L/100],2));
   axis equal
   M(j)= getframe;
end
movie(M)
assignin('base','M',M)


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


delete(handles.figure1);
display Goodbye
close(figure1);
```

4.2.2 Commenti all'm-file

In questa sezione saranno commentati i comandi dell' m-File "Proiettile"passo passo. Per prima cosa, nell' m-File si deve modificare la prima parte della **function vargout** : può dipendere dalle versioni di MATLAB, ma ci si deve assicurare che al posto di

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @somma_OpeningFcn, ...
                   'gui_OutputFcn',  @somma_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

ci sia scritto:

```
% Begin initialization code - DO NOT EDIT
if nargin == 0  % LAUNCH GUI

hObject = openfig(mfilename,'reuse');

% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(hObject);
    guidata(hObject, handles)

if nargout > 0
varargout{1} = hObject;
end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK
```

```
try
if (nargout)
[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
else
feval(varargin{:}); % FEVAL switchyard
end
catch
disp(lasterr);
end

end
% End initialization code - DO NOT EDIT
```

Se non ci fosse scritta già la seconda parte, la si deve sostituire manualmente. Sotto alla function edit1 Callback si deve aggiungere la variabile che viene inserita nella prima casella dell'interfaccia grafica, ovvero

```
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

Gx = str2double(get(handles.edit1,'String'));
handles.Gx=Gx;
guidata(hObject,handles);
```

Dove a Gx viene associata la stringa da inserire manualmente nella prima casella. Per gli altri due dati inseriti nell'interfaccia grafica si fa un ragionamento analogo:

```
Vy = str2double(get(handles.edit2,'String'));
handles.Vy=Vy;
guidata(hObject,handles);


Tan = str2double(get(handles.edit3,'String'));
handles.Tan=Tan;
guidata(hObject,handles);
```

Sotto a pushbutton1 Callback si scrive la function di Schmidt opportunamente modificata, tendendo conto che i dati iniziali sono già inseriti:

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

m1 = handles.Tan;
n1 = 0;
[mog, nog] = RettaPer2 (0, 0, handles.Gx, 0);
[m2, n2] = RettaP (mog, 0, handles.Vy);
[P1x,P1y] = InterRetta (m1 , n1, m2, n2);
Cx = 2*P1x;
Cy = 2*P1y;

% if Cx < Gx/2
%     'I dati iniziali non sono corretti'
% else

global Risultato;
global Indice;

Risultato(:,:) = [];
Indice = 1;

Risultato(Indice,:) = [0 0];
Indice = Indice + 1;
MedioDiMedioRicorsivo(0, 0, handles.Gx, 0, Cx, Cy)
Risultato(Indice,:) = [handles.Gx 0];
Indice = Indice + 1;

assignin('base','Risultato',Risultato)

M = moviein(100);
L = length(Risultato);

% %primo plottaggio
% subplot(2,1,1)
%
% plot(Risultato(:,1), Risultato(:,2))
% axis equal

%secondo plottaggio
```

```
axes(handles.axes2)
cla
hold on
for j=1:100
   plot(Risultato([1:j*L/100],1), Risultato([1:j*L/100],2));
   axis equal
   M(j)= getframe;
end
movie(M)
assignin('base','M',M)
```

Per stampare la traiettoria del proiettile nella finestra grafica, si è aggiunto:

```
axes(handles.axes2)
cla
hold on
```

E per ultimo, per fare in modo che il tasto di uscita in alto a destra dell'interfaccia grafica funzioni, si scrive

```
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


delete(handles.figure1);
display Goodbye
close(figure1);
```

Per ulteriori informazioni sui comandi utilizzati consultare l'help di MATLAB.

4.2.3 Risultato finale

Ora che l'm-file è stato costruito e completato, si può richiamare l'interfaccia grafica da MATLAB dal command window, usando il nome della funzione.

Inserendo i dati iniziali della traiettoria e premendo il tasto "fuoco", il programma esegue la traiettoria:

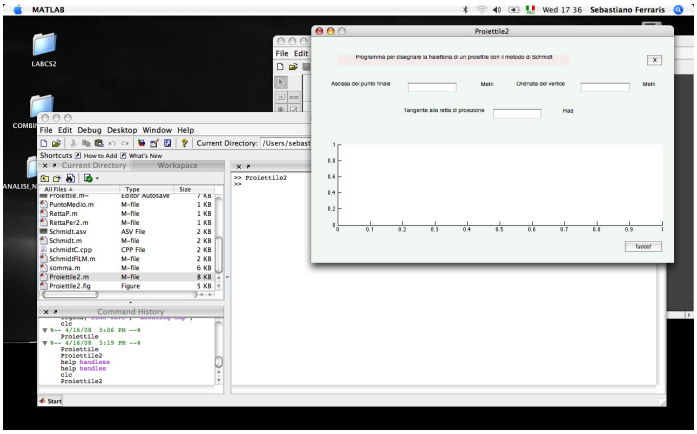

Figura 4.4: Accedere all'interfaccia grafica

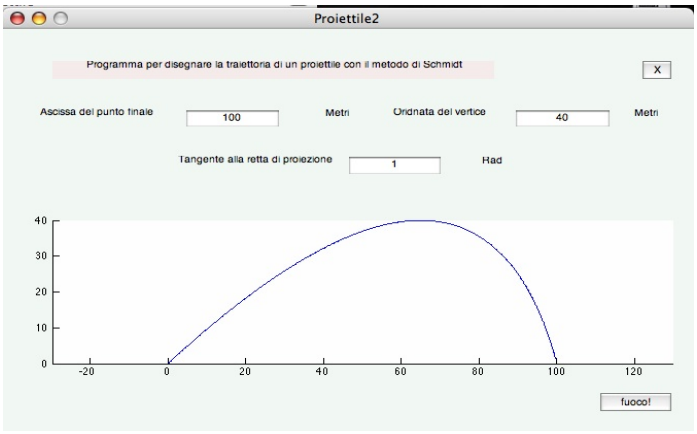

Figura 4.5: Interfaccia grafica finale

Bibliografia

[1] Manuale Matlab:

http://www.mathworks.com/academia/student_center/tutorials/launchpad.html

[2] Sito sulle GUI

http://xoomer.alice.it/gciabu/matlab/index.html

[3] Balistica

http://www.earmi.it/

In particolare:

http://www.earmi.it/balistica/balest.htm