PROGRAMMA IN C++ SULL'ATTRATTORE DI LORENZ

Marco Uguccioni

Abstract

Questo programma, scritto in C++, simula graficamente l'attrattore di Lorenz nel sistema di riferimento cartesiano tridimensionale (x,y,z), date le coordinate iniziali di due punti (rosso e blu), dopo una determinata variazione della variabile indipendente (t). Questa particolare soluzione dell'equazione di Lorenz, capace di generare un comportamento caotico, è stata ottenuta numericamente attraverso l'algoritmo di Runge-Kutta. La parte grafica del programma è stata realizzata attraverso le librerie OpenGL e GLUT (GL Utility Toolkit).

Introduzione

L'attrattore di Lorenz è una soluzione particolare di un sistema di 3 equazioni differenziali alle derivate parziali non lineari, formulate dal meteorologo Edward Norton Lorenz nel 1963. Nella forma più generale, il sistema è il seguente:

$$\begin{aligned}\dot{x} &= \sigma(y - x)\\ \dot{y} &= \rho x - xz - y\\ \dot{z} &= xy - \beta z\end{aligned}$$

Equazione 1- Equazione di Lorenz: i parametri σ e β sono costanti di valore rispettivamente 10 e 8/3, mentre ρ viene lasciato libero ed è quindi l'unico parametro da cui il sistema effettivamente dipende.

Tale sistema viene utilizzato per spiegare il movimento termico di convezione in un fluido, infatti, nel presente caso fisico, le variabili x,y,z non sono spaziali, ma indicano le modalità di movimento del fluido (rispettivamente: velocità, variazione di temperatura e distorsione lineare del gradiente termico). Per poterne individuare un contenuto fisico plausibile, il parametro ρ deve assumere valori vicinissimi, se non uguali ad 1. Portando invece il sistema fuori dall'appropriato regime fisico ed assegnando a ρ il valore 28, si individua un primo esempio storico di sistema a 3 gradi di libertà a presentare un comportamento caotico. L'attrattore di Lorenz, infatti, individua una forte sensibilità dalle condizioni iniziali: una minima variazione delle ultime comporta variazioni significative del comportamento futuro; ciò porta, ad esempio, all'impossibilità di fare predizioni corrette a lungo termine in campo meteorologico, essendo i dati iniziali sempre approssimati, a causa della complessità del sistema considerato. Dalla particolare forma dell'attrattore e dal suo comportamento caotico deriva la celebre locuzione "effetto farfalla" e fu inoltre lo stesso Lorenz, nel 1972, ad intitolare una sua conferenza "Può, il batter d'ali di una farfalla in Brasile, provocare un tornado in Texas?".
Un'altra peculiarità di questo sistema di equazioni è quello di presentare, come soluzione, un attrattore, cioè un insieme verso il quale un sistema evolve dopo un determinato periodo di tempo: infatti le curve individuate in R^3 da un sistema di punti nello spazio, ciascuno dei quali con condizioni iniziali anche molto diverse, convergeranno sulla tipica forma a farfalla.
L'attrattore di Lorenz quindi, per il suo comportamento caotico, viene definito "attrattore strano".
La soluzione dell'equazione non è ottenibile analiticamente, ma solo attraverso integrazione numerica: proprio per questo motivo la soluzione è stata ottenuta attraverso l'algoritmo di Runge-Kutta, utilizzato per risolvere sistemi di equazioni differenziali del tipo $\acute{X} = f(X, t)$, commettendo un errore proporzionale ad h^4, dove h è un incremento finito della variabile indipendente t. Lo schema dell'algoritmo applicato al sistema di Lorenz è mostrato in Equazione 2:

$$X(t_0 + h) = X(t_0) + \frac{h}{6}\left[f\left(X(t_0)\right) + 2f(Y) + 2f(Z) + f(W)\right]$$
$$Y = X(t_0) + \frac{h}{2}f\left(X(t_0)\right)$$
$$Z = X(t_0) + \frac{h}{2}f(Y)$$
$$W = X(t_0) + hf(Z)$$

*Equazione 2- Algoritmo di Runge-Kutta applicato al sistema di Lorenz. Noto il vettore tridimensionale **X**(x,y,z) nella condizione iniziale, è possibile conoscere il vettore dopo un incremento h, piccolo e finito. E' necessario introdurre 3 campi ausiliari **Y**(x,y,z), **Z**(x,y,z) e **W**(x,y,z), tutti valutabili a partire dai dati iniziali.*

Per la trattazione dell'attrattore da un punto di vista puramente speculativo e matematico, il programma mostra l'evoluzione di due punti nello spazio euclideo: pertanto le coordinate x,y,z, e la variabile indipendente t, presentate durante l'esecuzione del codice, non sono stati del sistema fisico, ma coordinate cartesiane a valori reali.

Elaborazione Codice

Il programma elaborato è di tipo procedurale, costituito dunque da diverse funzioni, gestite a loro volta in main tramite appropriate funzioni di openGL/GLUT o tramite invocazioni dirette.

Una delle più importanti è `void display()`, per il controllo della finestra grafica: gestita a sua volta da `glutDisplayFunc(display)` in main , oltre alla gestione dei buffers (banchi di memoria utilizzati per contenere le informazioni sui disegni), permette di modificare la visuale della finestra grafica, attraverso una moltiplicazione di matrici, con:

```
glRotatef((GLfloat)angolo1, 1.0, 0.0, 0.0) //rotazione x
glRotatef((GLfloat)angolo2, 0.0, 1.0, 0.0) //rotazione y
glTranslatef((GLfloat) distanza,0.0,0.0) //traslazione x
```

Esse individuano, rispettivamente, la rotazione intorno l'asse x, y e la traslazione lungo x di una quantità di valore pari ad `angolo1`, `angolo2` e `distanza`, variabili globali intere qualificate static ed inizializzate a zero, opportunamente modificate tramite la pressione di appositi tasti. Inoltre, la funzione di display, una volta impostati i due colori (blu e rosso), realizza le primitive grafiche come sequenza di punti collegati da una retta, oltre a realizzare lo spazio euclideo tridimensionale (in bianco), su cui vengono stampate le lettere 'X','Y','Z'. Si riporta qui il segmento di codice che permette il disegno dell'attrattore per un punto nello spazio:

```
glColor3f(1,0,0); //rosso
glBegin(GL_LINE_STRIP); //punti collegati da linea
for(int i=0 ; i<conteggio && i<N ; ++i) //particella 1
glVertex3dv(punti[i]) //disegno del punto i-esimo
glEnd();
```

dove `N` è una costante definita al preprocessore tramite `#define N 100000`; `conteggio` è una variabile globale intera static, inizializzata a zero (analogamente alla variabile `incremento`); infine `punti[N][3]` è un array di double bidimensionale, dichiarato anch'esso come static nell'ambito globale (analogamente a `punti2[N][3]` per il secondo punto), che permette di memorizzare in esso i vari vettori tridimensionali, generati dall'algoritmo di Runge-Kutta, e quindi

di stamparli come punti colorati nella finestra grafica.
Per quanto riguarda la realizzazione grafica dello spazio euclideo, si riporta il seguente segmento:

```
glColor3f(1,1,1); // bianco
glBegin(GL_LINES); //linea
glVertex3d(0,0,0); glVertex3d(50,0,0);//asse x
glVertex3d(0,0,0); glVertex3d(0,50,0);//asse y
glVertex3d(0,0,0); glVertex3d(0,0,50); //asse z
glEnd();
glRasterPos3d(50,0,0); //posizione dell'immagine digitale
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'X'); //asse x
glRasterPos3d(0,50,0);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'Y'); //asse y
glRasterPos3d(0,0,50);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'Z'); //asse z
```

Si noti la seconda espressione all'interno del precedente ciclo for: grazie a quest'ultima, insieme all'istruzione condizionata seguente,

```
if(conteggio < N) {
if(conteggio + incremento > N) conteggio=N;
else conteggio += incremento; }
```

è possibile realizzare un disegno in modo graduale, senza stampare tutti i punti insieme, in modo da generare una vera e propria animazione della curva generata con l'algoritmo.

L'animazione è resa possibile anche dalla funzione `void animazione`() , gestita in main da `glutIdleFunc(animazione)`: nel suo ambito viene eseguito l'importante `glutPostRedisplay()`, che permette continuamente il ridisegno della finestra ed il richiamo della funzione di display.

Il corretto posizionamento della visuale è garantito dalla funzione `void rimodella_finestra(int w, int h)`, sempre gestita in main, da `glutReshapeFunc(rimodella_finestra)`. Questa riceve come argomenti 2 interi che corrispondono, rispettivamente, alla larghezza ed all'altezza della finestra grafica: lavorando con questi due parametri, matrici di proiezione (proiezione geometria sul piano) e di model-view (per portare oggetti nel riferimento della camera), è stato possibile, dopo numerosi tentativi, trovare una posizione adeguata della visuale nel sistema di riferimento considerato. La stessa funzione inoltre, garantisce un corretta gestione dell'immagine, qualora la finestra venga ridimensionata.

Le ultime due funzioni riguardanti la grafica, sono quelle utilizzate per l'input da tastiera ed il controllo interattivo dell'animazione: `void tastiera(unsigned char tasto,int x,int y)` e `void tastiera2(int tasto,int x,int y)`, invocate rispettivamente da `glutKeyboardFunc(tastiera)` e `glutSpecialFunc(tastiera2)`. Attraverso la pressione di determinati tasti, è possibile effettuare numerose azioni, riportate qui sotto:

```
if(tasto=='f') conteggio=N; //finisci animazione
if(tasto=='r') conteggio=0; //ricomincia animazione
if(tasto=='a') distanza+=5; //trasla
if(tasto=='s') distanza-=5;
if(tasto=='v') incremento+=5; //velocizza
if(tasto=='c') if( incremento-5 >0) incremento-=5; //rallenta
if(tasto=='i') {angolo1=0,angolo2=0,distanza=0;} //riposiziona camera
if(tasto==27) exit(0); //esci/termina programma
if(tasto==13) incremento=10; //invio: comincia animazione

if(tasto==GLUT_KEY_LEFT) angolo2+=5; //frecce: ruota
if(tasto==GLUT_KEY_RIGHT) angolo2-=5;
if(tasto==GLUT_KEY_UP) angolo1-=5;
if(tasto==GLUT_KEY_DOWN) angolo1+=5;
```

Aldilà delle funzioni OpenGL/GLUT, la più importante e consistente funzione del programma è senz'altro

```
void LORENZ(double *x, double *x0, double t, int &conta);
```

in quanto è proprio quest'ultima a risolvere l'equazione di Lorenz, e dunque ad effettuare l'algoritmo di Runge-Kutta nella sua completezza. Questa riceve da main due array di 3 double, `x0[3]` e `x[3]` , corrispondenti rispettivamente al vettore iniziale $\mathbf{X}(t_0)$ e quello dopo un incremento h, $\mathbf{X}(t_0+h)$. Oltre a queste, riceve anche una variabile double `t`, che rappresenta la variazione della variabile indipendente dell'equazione; inoltre, per riferimento sinistro, viene trasmessa la variabile intera `conta`, di valore iniziale 0, utilizzata per effettuare due diversi disegni dell'attrattore, tramite un'unica funzione (cfr. appresso).
All'interno della funzione LORENZ sono effettuate numerose dichiarazioni, utili per l'algoritmo:

```
const double a=10.0,b=8.0/3.0,c=28.0; //sigma,beta,rho
double f0[3]; //funzione calcolata in X(t0)
double Y[3];  //campo ausiliario Y
double f1[3]; //funzione calcolata in Y
double Z[3];  //campo ausiliario Z
double f2[3]; //funzione calcolata in Z
double W[3];  //campo ausiliario W
double f3[3]; //funzione calcolata in W
double h;     //incremento piccolo e finito della variabile t
h=t/(double)N;
```

I vari significati degli identificatori dichiarati sono opportunamente commentati nel precedente segmento: l'incremento h si ottiene dalla divisione tra la variabile t, ricevuta da main, e la costante definita N, di valore molto elevato, così da ottenere un incremento sufficientemente piccolo da non commettere errori significativi.
Successivamente alle dichiarazioni, si effettua un ciclo for, ripetuto N (100000) volte, che effettua i calcoli dell'algoritmo, passo dopo passo, nel modo qui sotto mostrato:

```
//equazione di Lorenz calcolata in x0
f0[0] = a*(x0[1]-x0[0]),
f0[1] = c*x0[0]-x0[0]*x0[2]-x0[1],
f0[2] = x0[0]*x0[1]-b*x0[2];

//campo ausiliario Y
for(int i=0 ; i<3 ; ++i) Y[i]=x0[i]+(h/2)*f0[i];

//equazione di Lorenz calcolata in Y
f1[0] = a*(Y[1]-Y[0]),
f1[1] = c*Y[0]-Y[0]*Y[2]-Y[1],
f1[2] = Y[0]*Y[1]-b*Y[2];

//campo ausiliario Z
for(int i=0 ; i<3 ; ++i) Z[i]=x0[i]+(h/2)*f1[i];

//equazione di Lorenz calcolate in Z
f2[0] = a*(Z[1]-Z[0]),
f2[1] = c*Z[0]-Z[0]*Z[2]-Z[1],
f2[2] = Z[0]*Z[1]-b*Z[2];

//campo ausiliario W
 for(int i=0 ; i<3 ; ++i) W[i]=x0[i]+h*f2[i];

//equazione di Lorenz calcolate in W
f3[0] = a*(W[1]-W[0]),
f3[1] = c*W[0]-W[0]*W[2]-W[1],
f3[2] = W[0]*W[1]-b*W[2];

//ALGORITMO DI RUNGE KUTTA
for(int i=0 ; i<3 ; ++i)
x[i]=x0[i]+(h/6)*(f0[i]+2*f1[i]+2*f2[i]+f3[i]);
```

In questo modo si assegnano agli elementi dell'array `x[3]`, ad ogni iterazione del ciclo, i risultati dei calcoli dell'algoritmo. Tuttavia, per aggiornare i valori dell'array e trattarli come "nuovi" valori iniziali, prima di chiudere l'ambito del ciclo principale, viene inserita la seguente istruzione condizionata:

```
if(conta) {for(int t=0; t<3; ++t) {punti2[i][t]=x0[t];x0[t]=x[t];} }
else { for(int t=0; t<3; ++t) {punti[i][t]=x0[t];x0[t]=x[t];} }
}//fine ciclo runge-kutta
++conta;
```

Così, oltre ad aggiornare i valori in modo da ottenere, punto dopo punto, tutta la funzione, i vettori iniziali ed i consecutivi 99999 dopo un incremento h, vengono memorizzati nell'array globale.
Si noti il ruolo della variabile conta: poiché in main la funzione LORENZ viene invocata due volte per disegnare due funzioni, è la variabile conta a decidere, grazie al costrutto if, in quale dei due array memorizzare tali valori. Ciò permette, come discusso precedentemente, di effettuare due

diverse rappresentazioni semplicemente invocando la stessa funzione due volte e trasmettendo diverse condizioni iniziali.

I compiti svolti della funzione `int main()` dunque, oltre a dichiarare le variabili trasmesse a `void LORENZ()` durante le invocazioni, sono sostanzialmente due.
Il primo di questi è l'assegnamento per lettura delle condizioni inziali, tramite un ciclo range for, e della variabile t; tutto ciò avviene tramite l'oggetto `cin` e l'estrattore >>:

```
while(true)
  {
   cin>>t;
   if(!cin)
    {
     cin.clear( );
     cout<<"parametro non valido,riprova:\n";
     do; while(c=cin.get() != '\n');
    }
   else break;
  }
if(t==0) {cout<<"incremento nullo, nessuna evoluzione del
sistema.\n";return 0;}
if(t<0) {cout<<"incremento non valido, fornito un valore negativo.\n";
return 1;}
if(t<1) {cout<<"incremento troppo piccolo per individuare un'evoluzione
significativa.\n"; return 2;}
if(t>200) {cout<< "valore troppo elevato\n"; return 3;}
cout<<"valore di t inserito: "<<setprecision(10)<<t<<"\n"<<endl;


while(true)
 {
   for(double&i : x0)
   {
    cin>>i; //condizioni iniziali particella rossa
    if(i>100 || i<-100) {cout<< "valore fuori range\n"; return 4;}
   }
   if(!cin)
    {
     cin.clear( );
     cout<<"terna non valida,riprovare:\n";
     do; while(c=cin.get() != '\n');
    }
    else break;
  }
if(x0[0]==0. && x0[1]==0. ) cout<<"nessuna evoluzione significativa per
il punto rosso."<<endl;
cout<<"coordinate punto rosso inserite:("<<setprecision(10)<<x0[0]<<" ,
"<<x0[1]<<" , "<<x0[2]<<")\n"<<endl;
```

Il programma aderisce alla “scuola permissiva” per quanto riguarda la digitazione di dati che causano la restituzione falsificata dell’oggetto cin; tuttavia, qualora l’esecutore del codice, fornisca dati fuori dal range, ben specificato durante la fase di lettura, farà fede la conversione alla “scuola draconiana”, conducendo direttamente alla terminazione del programma.
Le istruzioni all’interno dei vari if sono dovute a numerosi tentativi di esecuzione: inserendo ad esempio le prime due coordinate iniziali nulle (0,0,z), non si nota alcuna evoluzione significativa del sistema, individuando solo una retta coincidente con l’asse Z stesso. Inserendo invece un incremento t minore di 1, il grafico non permette di individuare nulla di particolare: è per valori maggiori di t=50 che è possibile apprezzare l’ “effetto farfalla” per cui l’attrattore è noto.
Si verifica infine che, digitando valori molto alti, si rischia il problema dell’overflow; per cui è stato stabilito un range di valori reali in cui si potesse essere al sicuro da questa problematica, ma, allo stesso tempo, individuare il miglior intervallo per la realizzazione del grafico.
Si noti infine l’uso del manipolatore di output setprecision(10), per mostrare i dati decimali con una precisione fino a 10 cifre dopo la virgola.

Il secondo e ultimo compito svolto da main, prima di concludere il programma, è l’inizializzazione di GLUT e la gestione delle varie funzioni di cui si è parlato precedentemente:

```
glutInit(&arg1,arg2); //inizializzazione
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
glutInitWindowPosition(500,500); //creazione finestra
glutInitWindowSize(700, 700);
glutCreateWindow("Attrattore di Lorenz");


/*varie funzioni glut per la gestione di funzioni esterne…*/

glutMainLoop();  //entra nel loop di GLUT
return 0;
```

Conclusioni

Il programma realizzato mette in evidenza due aspetti molto interessanti di questa particolare soluzione, fornita dall’equazione di Lorenz:

- Se le condizioni iniziali sono identiche a meno di una piccola variazione, ad esempio (1,1,1) e (1.001,1,1), in un determinato momento, si avrà una divergenza esponenziale tra i punti, inizialmente quasi coincidenti, in correlazione con l’ipotesi sul comportamento caotico.
- Se le condizioni iniziali sono molto diverse tra loro, ad esempio (100,1,-50) e (-30 40 10), le due funzioni convergeranno sulla stessa forma a farfalla, in correlazione con la definizione di attrattore.

Questi aspetti, nel loro insieme, mostrano in modo semplice ma efficace il motivo per cui questa affascinante curva prende il nome di “attrattore strano”.