Generatori di numeri pseudo-casuali e tests statistici

Pellegrin Fabio

22 giugno 2005

Indice

Capitolo 1

Numeri pseudo-casuali

1.1 Perchè "pseudo".

Un generatore di numeri casuali è uno strumento capace di fornire una sequenza di numeri appunto casuali, ovvero non deterministici. Questi numeri sono idealmente infiniti e non sono influenzabili da alcun fattore esterno. Il computer o qualsiasi macchina non ha la possibilità di generare questo tipo di sequenza. L'unico modo è utilizzare opportuni algoritmi che generenano numeri apparentemente casuali. Vengono quindi chiamati numeri pseudo-casuali poichè venendo a conoscenza dell'algoritmo e il seme (primo elemento) utilizzati è possibile determinare la sequenza che verrà generata. Inoltre la sequenza non è infinita, o meglio, la sequenza di numeri generati si ripete ciclicamente con un'intervallo fisso (periodo del generatore). L'unico elemento puramente aleatorio è il seme, poichè scelto dai dati casuali presenti nella macchina (p.e. numero di file presenti, numero di battute della tastiera, l'ora etc..).

1.2 Bontà di un generatore.

Per valutare la bontà vengono considerati la "casualità" della sequenza ed il periodo del generatore. Per il periodo è semplice: più lungo è il periodo, migliore è il generatore; mentre per la "casualità", com'è possibile attribuire l'aggettivo casuale ad una sequenza di numeri? Prendiamo come esempio due serie di 1 e 0.

$$0, 1, 0, 1, 0, 1, 0, 1, 0, 1...$$

$$1, 1, 0, 1, 0, 1, 1, 0, 0, 1, ...$$

Apperentemente si è portati e definire la prima sequenza deterministica, poichè si riconosce una certa periodicità o un algoritmo capace di generarla; la seconda sembra essere "più casuale" e non si trova alcuna regola capace di formarla. In realtà entrambe potrebbero essere generate dal lancio di una moneta come

da un algoritmo. Il "buon senso" quindi non basta per dare una valutazione; esistono altri criteri più obiettivi fra i quali:

- criterio di Turing(un'alternativa pratica è offerta da un criterio ricavato dal test di Turing):

  se una sequenza numerica generata da estrazioni meccaniche è apparentemente indistinguibile da quella generata da un algoritmo, allora anche quest'ultima puo' considerarsi casuale.

- criterio di von Neumann:

  esistono dei procedimenti matematici per determinare la "bontà" di un generatore di numeri casuali. Questi metodi si fondano sull'assunto che ogni numero casuale sia equiprobabile e dunque l'estrazione di N numeri diversi dia luogo ad una distribuzione uniforme (p. es.χ^2).

Il primo resta comunque legato al "buon senso", mentre il secondo è piu obiettivo ed introduce all'addattamento.

1.3 Utilizzo dei numeri casuali.

L'utilizzo maggiore dei numeri casuali si trova nella crittografia e nella simulazione. Nella crittografia c'è la necessita di generare numeri che non possano esser determinati da terzi; nella simulaziane è necessario creare campioni "virtuali" capaci di rappresentare una reale popolazione o un suo aspetto.

Capitolo 2

Generatori di numeri pseudo-casuali.

2.1 Metodo delle congruenze lineari

Tale metodo permette, dato un valore iniziale X_0, di ottenere una sequenza di numeri pseudo-casuali mediante l'applicazione ripetuta della seguente formula:

$$X_{i+1} = (a * X_i + c)\ mod\ M$$

dove *a,c* ed M sono coefficienti interi non negativi chiamati rispettivamente moltiplicatore, incremento e modulo. L'operazione *mod* è il resto della divisione per M, quindi i numeri generati sono compresi fra 0 ed *m-1*. Il metodo prende il nome dalla seguente definizione:

> due numeri x e y si dicono congrui modulo m, e scriveremo $x = y(mod\ m)$, se essi differiscono per un multiplo intero di m, ossia se $x(mod\ m) = y(mod\ m)$. Nel nostro caso X_{i+1} sarà congruo modulo m a $(a * X_i + c)$.

Se $c = 0$, il metodo viene detto moltiplicativo; Se $a = 1$ il metodo viene detto additivo; Se $a \neq 1$ e $c \neq 0$ il metodo è misto. Prove effettuate variando i parametri *a*, c e M hanno portato le seguenti osservazioni:

- La lunghezza massima raggiungibile dalla sequenza generata, senza ripetizione, vale M;
- Particolari scelte di a e c possono ridurre notevolmente la lunghezza utile della sequenza;
- Il valore di X_0 può essere determinante nella lunghezza della sequenza.

Alcuni studiosi analizzando le sequenze generate hanno individuato alcuni valori di *a,* c e M per i quali il metodo "funziona particolarmente bene":

- KNUT $M = 2^{31};\ a = int(*10^8);\ c = 453806245$
- GOODMAN e MILLER $M = 2^{31} - 1;\ a = 7^5;\ c = 0$
- GORDON $M = 2^{31};\ a = 5^{13};\ c = 0$
- LEORMONT e LEWIS $M = 2^{31};\ a = 2^{16} + 3;\ c = 0$

2.2 LFG Lagged Fibonacci Generator

Le argomentazioni che seguono sono tratte da:

- D. Knuth *The art of Computer Programming* vol. 2 *Seminumerical Algoritms.*
- D.V. Pryor,S.A. Cuccaro, M. Mascagni, M.L. Robinson *Implementation of a Portable and Reproducible Parallel Pseudorandom Number Generator*

2.2.1 Lagged Fibonacci

L'algoritmo lagged-Fibonacci per generare numeri pseudo-casuali nasce dal tentativo di generalizzare il metodo delle congruenze lineari che, come noto, è dato dalla ricorrenza

$$X_{n+1} = (aX_n + c) \mod m$$

Uno dei motivi che spingono alla ricerca di generatori nuovi è la necessità - utile per molte applicazioni, specie nel calcolo parallelo - di allungare il periodo del generatore. Il periodo di un generatore lineare quando m è approssimativamente pari alla grandezza della parola del computer, è all'incirca dell'ordine di 10^9, suffciente per molte applicazioni ma non tutte.

Una delle tecniche indagate è quella di far dipendere X_{n+1} da entrambi X_n e X_{n-1} invece che solo da X_n; allora il periodo può arrivare vicino al valore m^2 perchè la sequenza non si ripeterà finchè non si avrà

$$(X_{n+\lambda}, X_{n+\lambda+1}) = (X_n, X_{n+1})$$

Il più semplice generatore di questo tipo è la successione di Fibonacci

$$X_{n+1} = (X_n + X_{n-1}) \mod m$$

questo generatore è stato analizzato negli anni '50 e fornisce un periodo ¿ m, ma la sequenza non supera i più semplici test statistici.

Anche i generatori del tipo

$$X_{n+1} = (X_n + X_{n-k}) \mod m$$

pur migliori della successione di Fibonacci, non danno risultati soddisfacenti. Bisogna attendere il 1958 quando Mitchell e Moore propongono la sequenza

$$X_n = (X_{n-24} + X_{n-55}) \mod m, n \geq 55$$

dove m è pari e $X_0, \ldots, X_{54}$ sono interi arbitrari non tutti pari. Le costanti 24 e 55 non sono scelte a caso, sono numeri che definiscono una sequenza i cui bit meno significativi ($X_n \mod 2$) hanno un periodo di lunghezza $2^{55}-1$; perciò la successione (X_n) deve avere un periodo di lunghezza almeno $2^{55}-1$. In realtà la successione ha periodo $2^{M-1}(2^{55}-1)$ dove $m = 2^M$.

I numeri 24 e 55 sono comunemente chiamati *lags* e la (X_n) si dice essere una successione lagged-Fibonacci (LFG).

La successione LFG può essere generalizzata in

$$X_n = (X_{n-l} + X_{n-k}) \mod 2^M$$
$$l > k > 0$$

ma solo alcune coppie (k, l) danno periodi sufficientemente lunghi; in questi casi si dimostra[1] che il periodo è $2^{M-1}(2^l - 1)$. Studi approfonditi sulle proprietà di questi generatori sono dovuti a Marsaglia [1984,1985]. Le coppia (k, l) devono essere scelte in modo opportuno. L'unica condizione sui primi l valori è che almeno uno di essi deve essere dispari (altrimenti tutta la successione sarà composta di numeri pari). Osserviamo che l'implementazione di un generatore LFG è piuttosto semplice: una addizione intera, un AND logico (per la divisione modulo 2^M) e il decremento di due puntatori sono le uniche operazioni richieste dal programma (vedi codice allegato).

2.2.2 Generatori LFG paralleli

Indichiamo con LGF(l,k,M) un generatore lagged-Fibonacci con lags $l > k > 0$ del tipo

$$X_n = (X_{n-l} + X_{n-k}) \mod 2^M$$

Supponiamo di disporre verticalmente gli l numeri iniziali in modo da formare una matrice Mxl di bit. Il generatore farà uscire dalla matrice la prima colonna a sinistra e ne farà entrare una nuova a destra. Il numero di possibili matrici distinte di Mxl bit è 2^{Ml} mentre, come già detto, il periodo massimo di un LFG(k,l,M) è $(2^l - 1)2^{M-1}$. Le condizioni per cui un generatore ha periodo massimo sono che almeno uno degli l residui modulo 2^M sai dispari, in altre parole se tutti i bit meno significativi degli l numeri iniziali sono 0 allora il generatore non avrà periodo massimo. Quante sono le possibili configurazioni della matrice Mxl che soddisfano questa condizione? se mettiamo tutti zeri nell'ultima riga avremo $2^{(M-1)l}$ configurazioni possibili (non massime) quindi ne restano

$$2^{Ml} - 2^{(M-1)l} = 2^{l(M-1)}(2^l - 1)$$

[1] Abbiamo trovato una interessante e semplice dimostrazione di questa formula ma sarebbe troppo lungo riportarla nel testo.

di valide. Essendoci $(2^l - 1)2^{M-1}$ elementi in una sequenza massima, allora ci saranno

$$\frac{2^{l(M-1)}(2^l - 1)}{(2^l - 1)2^{M-1}} = 2^{(M-1)(l-1)}$$

possibili sequenze distinte di periodo massimo.

Questo è un fatto notevole per quanto riguarda la necessità di generare sequenze pseudo-casuali in un calcolatore parallelo: in tale contesto ogni singolo processore dovrebbe generare una sequenza riproducibile e non correlata con quelle contemporaneamente generate negli latri processori. Come osservato, un LFG ha queste possibilità ma è necessario che l'inizializzazione del genratore segua delle regole ben precise che corrispondono ad una analisi teorica solida e ben fondata. Descriviamo quì le regole pratiche di implementazione che si devono seguire e che abbiamo seguito nella stesura del nostro generatore rimandando le considerazioni teoriche all'articolo citato all'inizio.

2.2.3 Inizializzazione

In riferimento alla sopracitata matrice Mxl composta da l numeri di M bit disposti in colonna, le regole di inizializzazione che garantiscono il periodo massimo sono le seguenti:

- La colonna di sinistra è composta da tutti 0
- L'ultima riga in basso è composta da tutti 0 salvo uno o più 1 nelle posizioni prescritte dalla tabella che segue, dipendenti dallo specifico LFG.
- I rimanenti bit della regione di ampiezza (M-1)x(l-1) possono essere scelti arbitrariamente; ogni scelta darà origine ad una sequenza distinta di periodo massimo.

l	k	valore con l.s.b. =1
3	2	0
5	3	1, 2
10	7	7
17	5	10
35	2	0
55	24	11
71	65	1
93	91	1, 2
127	97	21
185	128	63

Notiamo che questo schema pone i seguenti problemi:

- Se lo schema di inizializzazione dell'area di ampiezza (M-1)(l-1) è troppo semplice, la differenza nei primi numeri di sequenze diverse non è precisamente molto casuale. Il problema si risolve non utilizzando i primi 100 numeri di ogni sequenza.

- Si noterà anche che, per un particolare LFG, tutti i bit meno significativi sono gli stessi per qualsiasi sequenza e questo è un prezzo che si deve pagare per avere l'unicità delle sequenze. Il problema si può risolvere in due modi: non utilizzando il bit meno significativo di ogni numero oppure sostituendolo con un bit generato casualmente con un apposito generatore.

Le scelte per inizializzare la regione di ampiezza (M-1)(l-1) in un calcolatore parallelo sono molteplici. Quella che abbiamo scelto noi prevede l'uso di un generatore a congruenze lineari (quello delle librerie standard del compilatore C va benissimo) per generare i primi l numeri che riempiono la regione Mxl per poi procedere agli aggiustamenti necessari nella prima colonna e ultima riga. Provvedendo un seed diverso per ogni processore (usando il tempo o il n. di processo a seconda delle esigenze di riproducibilità) abbiamo una buona probabilità che l'inizializzazione sia corretta per ogni processore e quindi la quasi certezza che le sequenze siano indipendenti.

Capitolo 3

Test χ^2 di Pearson

3.1 Introduzione

Il test χ^2 di Pearson (o della bontà dell'adattamento) è un test non parametrico applicato a grandi campioni quando si vuole verificare se il campione è stato estratto da una popolazione con una predeterminata distribuzione o che due o più campioni derivino dalla stessa popolazione. Fa parte di un'ampia classe di test detti test χ^2 in quanto hanno in comune le formule e la variabile casuale χ^2. Nello studio di un generatore di numeri pseudo-casuale viene utilizzato per valutare l'adattamento dei numeri generati alla distribuzione uniforme.

3.2 La distribuzione χ^2

La distribuzione χ^2 viene definita come somma di g variabili normali con media nulla e varianza unitaria

$$\chi^2_g = N_1(0,1)^2 + N_2(0,1)^2 + ... + N_g(0,1)^2$$

dove g rappresenta il numero di gradi di libertà. La funzione di densità f(x) per χ^2_g(chi-quadro con g gradi di libertà) è:

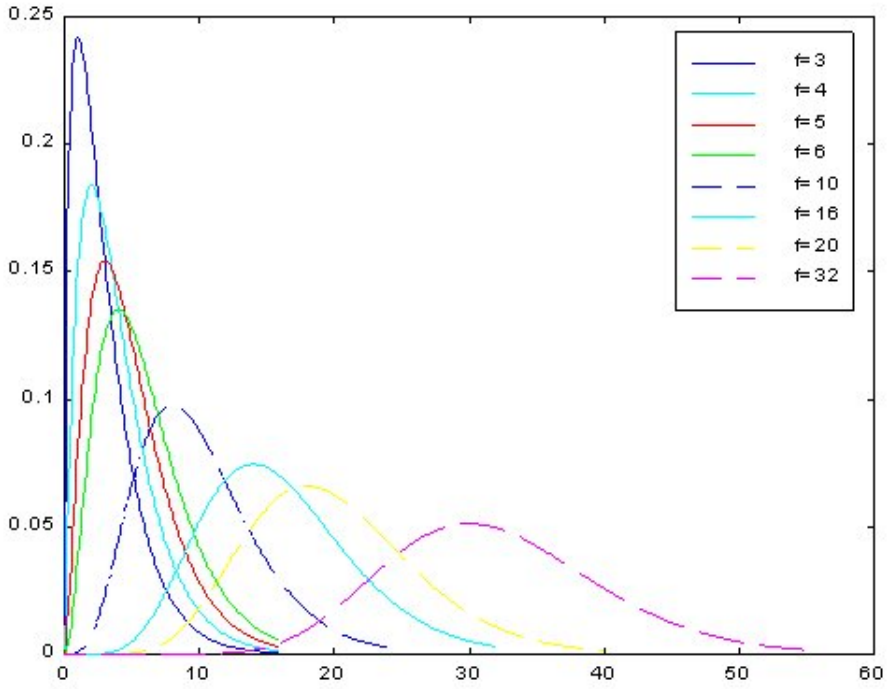
$$f(x) = \frac{e^{-\frac{x}{2}}\ x^{\frac{g}{2}-1}}{2^{\frac{g}{2}}\ \Gamma(\frac{g}{2})}\ \ per\ x > 0$$

dove $\Gamma(x)$è la funzione gamma. Questa particolare funzione, nel nostro caso $\Gamma(\frac{g}{2})$ restituisce $(\frac{g}{2}-1)!$ se g è pari, e $(\frac{g}{2}-1)! * \sqrt{\Pi}$ se g è dipari.

3.3 Calcolo χ^2 di Pearson

Per poter eseguire questo test è necessario effettuare il campionamento e suddividere l'intervallo in classi, registrando per ognuna di esse il numero di casi

Figura 3.1: Densità distribuzione χ^2 al variare dei gradi di libertà (f)



osservati che ricadono in essa. Viene poi effettuato un confronto fra il numero di casi attesi (in base alla distribuzione decisa) e il numero di casi osservati. Il coefficiente χ^2di Pearson si occupa di effettuare questo confronto tramite la seguente formula:

$$X^2_{Pear.} = \sum_{i=1}^{k} \frac{(f_i - Np_i)^2}{Np_i}$$

Dove:

- f_i è il numero di casi osservati che fanno parte dell' i-esima classe
- Np_iè il numero di casi attesi
- k è il numero di classi - 1

Si può notare che viene utilizzato l'elevamento al quadrato anzichè l'operatore modulo (valore assoluto) per rendere più "sensibile" alle differenze il coefficiente (aumenta le grandi differenze e riduce le piccole).

3.4 χ^2 critico e confronto

Si può dimostrare che il χ^2 di Pearson, con N tendente all'infinito, tende ad una distribuzione X^2 con gradi di libertà $g = k - 1 - m$ dove m è il numero di parametri da stimare (0 per la distribuzione uniforme). Una volta deciso $1 - \alpha$, ovvero il livello di significatività del test,se

$$X^2_{Pears.} < X^2_{1-\alpha}$$

il campione segue la ditribuzione scelta, se invece

$$X^2_{Pears.} \geq X^2_{1-\alpha}$$

il campione non segue la distribuzione scelta.

3.5 Implementazione

Poichè non esiste nelle librerie C una funzione capace di restituire il valore critico fissati k e $1 - \alpha$, si è dovuto utilizzare il metodo numerico dei rettangoli per l'approssimazione delle aree. Questo metodo consiste nel suddividere l'intervallo di integrazione $[a,\ b]$ in n parti uguali, e sommare le aree dei rettangoli aventi per base $\frac{b-a}{n}$ e per altezza il valore della funzione nell'estremo sinistro di ciascuna parte.

Tale metodo è stato però applicato in modo inverso: l'area viene calcolta incrementando l'intervallo di integrazione di un passo alla volta; quando l'area calcolata supera $1-\alpha$viene ristituita la media degli ultimi due valori dell'ascissa.

Figura 3.2: Integrazione con metodo dei rettangoli

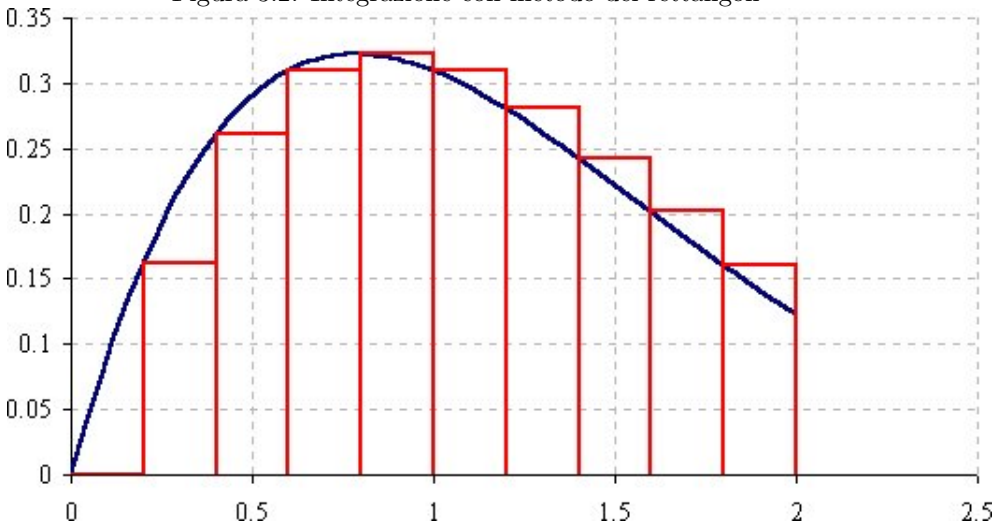
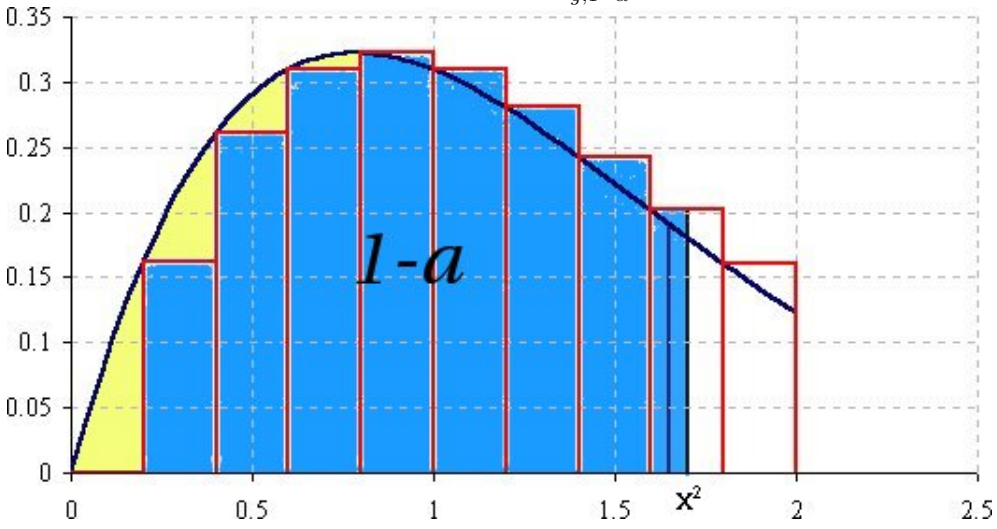


Figura 3.3: Valore di $\chi^2_{g,1-\alpha}$



Appendice A

Codice sorgente & output

A.1 Libreria Lagged Fibonacci

Il codice è una modifica della libreria c scritta da Sartorello Enrico e Guido Aguliari.

```
#include <math.h>
#include <time.h>
#include <stdlib.h>

static unsigned long  lf_vet[55];
static int  lf_head=0;
static int lf_count=55;

void lf_init ();
unsigned long lf_rand();

void lf_init () {
int i;
srand(time(NULL)); // utilizza per rand il seme del tempo
lf_vet[0]=0;
for (i=1;i<55;i++)
lf_vet[i]=rand() & 0xFFFFFFFE; // tutti uni e 1110
lf_vet[10]=lf_vet[10] | 0x00000001; // tutti zeri e 0001
for (i=0; i<200; i++) lf_rand(); // scarta i primi 200 valori

}

unsigned long lf_rand () {
lf_vet[lf_head] = (lf_vet[(lf_count-55)%55]+lf_vet[(lf_count++-24)%55]);
if (++lf_head >= 55) {
lf_head = 0;
```

```
return lf_vet[54] >> 1; }
return lf_vet[lf_head-1] >> 1;
}
```

A.2 Libreria ChiQuadro

```
#include<math.h>

const
double INC = 0.001;
double gammad (double t) {
if ((t > 0.4999999999999999) && (t < 0.5000000000000001))
return sqrt(M_PI);
return (t-1) * gammad(t-1);
}

double gammap (double t) {
if (t == 1)
return 1;
return ((t-1) * gammap(t-1));
}

double gammax (int t) {
if (t%2 == 0)
return gammap (t/2);
else
return gammad ((double)t/2);
}

double chi(double x, int g) {
return pow(x,(double)g/2-1)*pow(M_E,-x/2)/
(pow(2,(double)g/2)*gammax(g));
}

double inv_chi (double a, int g) {
double area = 0;
double i = 0;
while (area < a) {
i+=INC;
area += INC * chi(i,g);
}
return i-INC/2;
}
```

A.3 Codice ChiQuadro.c

```
#include<stdio.h>
#include "lf_lib_single.h"
#include "mielib.h"

int n_dati = 0;
int n_classi = 0;
long *classi;
int i;
double comodo;
long tot;
double chi_pearson;
double chi_teo;
double liv_fiducia;
const long MAX = 2147483647; //2^31 - 1


int main (int argc, char *argv[]) {
if (argc != 4) {
printf("Sintassi: %s <n_dati> <n_classi> <liv_fiducia>\n",argv[0]);
return 1;
}

n_dati = atoi(argv[1]);
n_classi = atoi(argv[2]);
liv_fiducia = atof(argv[3]);

if ((liv_fiducia>=1) || (liv_fiducia<=0)) {
printf("Livello di fiducia deve essere in ]0,1[\n");
return 2;
}

classi = (long *) malloc(n_classi-1);
for (i=0;i<n_classi;i++) classi[i]=0;
lf_init();
for (i=0;i<n_dati;i++) {
comodo = lf_rand()/(double)MAX; //fra 0 e 1
classi[(int)(comodo*n_classi)] ++;
//Assegnazione classe di appartenenza
}

tot =0;
```

```
for (i=0;i<n_classi;i++) {
printf("Classe %d\t%d\n",i,classi[i]);
tot += classi[i];
chi_pearson += ((double)(n_dati/n_classi)-classi[i])*
((double)(n_dati/n_classi)-classi[i]);
//chi-quadro = (f_teorica - f_sperimentale)^2 / f_teorica
}

chi_pearson = chi_pearson/(double)(n_dati/n_classi);
printf("--------------------\n");
printf("CHI_pearson\t\t=\t%3.3f\n",chi_pearson);
chi_teo = inv_chi(liv_fiducia,n_classi-1);
printf("CHI_teorico(%3.3f,%d)\t=\t%3.3f\n",liv_fiducia,n_classi-1,chi_teo);
if (chi_teo < chi_pearson) printf("Non distribuito secondo uniforme\n");
else printf("Distribuito secondo uniforme\n");

}
```

L'output del programma risulata il seguente

```
ospel@iris c $ ./chiQuadro
Sintassi: ./chiQuadro <n_dati> <n_classi> <liv_fiducia>
ospel@iris c $
ospel@iris c $ ./chiQuadro 1000 10 0.95
Classe 0        99
Classe 1        98
Classe 2        109
Classe 3        114
Classe 4        107
Classe 5        88
Classe 6        97
Classe 7        96
Classe 8        97
Classe 9        95
--------------------
CHI_pearson             =       5.340
CHI_teorico(0.950,9)    =       16.918
Distribuito secondo uniforme

ospel@iris c $ ./chiQuadro 10000 10 0.95
Classe 0        982
Classe 1        950
Classe 2        991
Classe 3        1027
```

```
Classe 4        1036
Classe 5        987
Classe 6        1083
Classe 7        992
Classe 8        971
Classe 9        981
--------------------
CHI_pearson             =       13.254
CHI_teorico(0.950,9)    =       16.918
Distribuito secondo uniforme
```