Simulazione

4 ottobre 2007

Indice

1. Perchè simulare? 5
- 1.1. L'esempio della farmacia . . . 5
- 1.2. L'esempio del Risiko . . . 9
- 1.3. L'esempio del dilemma di Monty Hall . . . 9
- 1.4. L'esempio della distribuzione t di student . . . 10

2. Generazione di numeri pseudocasuali 13
- 2.1. Algoritmo middle-square . . . 13
- 2.2. Generatori congruenziali lineari . . . 15
- 2.3. Generatori basati su shift register . . . 20
- 2.4. Esempi di applicazioni . . . 22
  - 2.4.1. Approssimazione di integrali . . . 22
  - 2.4.2. Approssimazione di π . . . 23
  - 2.4.3. Approssimazione di e . . . 24

3. Generazione di variabili casuali discrete 26
- 3.1. Metodo della trasformata inversa . . . 26
  - 3.1.1. La variabile casuale uniforme . . . 27
  - 3.1.2. La variabile casuale geometrica . . . 28
  - 3.1.3. La variabile casuale bernoulliana . . . 29
  - 3.1.4. La variabile casuale poissoniana . . . 29
  - 3.1.5. La variabile casuale binomiale . . . 30

4. Generazione di variabili casuali continue 31
- 4.1. Metodo della trasformata inversa . . . 31
  - 4.1.1. La variabile casuale esponenziale . . . 31
- 4.2. Metodi per la distribuzione normale . . . 33
  - 4.2.1. Metodo di Box-Muller . . . 33
  - 4.2.2. Usando il teorema centrale . . . 36

5. Altri metodi per variabili casuali discrete e continue 39
- 5.1. Metodi plug-in . . . 39
  - 5.1.1. Metodo plug-in per la bernoulliana . . . 39
  - 5.1.2. Metodo plug-in per la binomiale . . . 39
  - 5.1.3. Metodo plug-in per la poissoniana . . . 39
  - 5.1.4. Metodo plug-in per la chi-quadro . . . 40
  - 5.1.5. Metodo composizionale . . . 42

5.2. Metodo di acceptance-rejection 44
5.2.1. La variabile casuale logaritmica 46
5.2.2. La variabile casuale Beta 47

6. Simulazione di sistemi complessi **53**
6.1. Esempio: una coda in farmacia 53
6.2. Esempio: un'assicurazione contro i rischi 56

7. Analisi statistiche di dati provenienti da sistemi simulati **61**
7.1. Stima puntuale 61
7.2. Stime per intervalli di confidenza 64
7.3. Tecniche bootstrap 66
7.3.1. Esempio: tempo trascorso dal cliente in farmacia 67
7.3.2. Esempio: stima bootstrap della varianza 68

8. Tecniche di riduzione della varianza **71**
8.1. Uso di variabili antitetiche 71
8.1.1. Simulazione della funzione affidabilità 71
8.1.2. Approssimazione di un integrale secondo il metodo Monte Carlo 73
8.1.3. Approssimazione di e 77
8.2. Uso di variate di controllo 83
8.2.1. Stimatore per la funzione affidabilità a varianza ridotta 85
8.2.2. Approssimazione di un integrale 85
8.3. Riduzione tramite condizionamento 85
8.3.1. Stima di π 86
8.3.2. La variabile casuale esponenziale 88
8.3.3. Simulazione di variabili compound 91
8.3.4. Simulazione di una terapia anti tumorale 94
8.4. Campionamento stratificato 98
8.4.1. Applicazioni del campionamento stratificato 99

9. Validazione statistica di ipotesi **104**
9.1. Test Chi-quadro 104
9.2. Test di Kolmogorov-Smirnov 108

A. Risorse utili **113**
A.1. Simulazione 113
A.2. Scilab 113

B. Riconoscimenti **114**

C. Licenza **115**

Percorso d'apprendimento

1. perchè simulare?
2. come il computer genera numeri casuali uniformemente distribuiti tra 0 e 1?
3. come generare i valori di una variabile casuale avente distribuzione arbitraria?
4. eventi discreti: il comportamento di un modello stocastico nel tempo
5. ottenere stimatori delle quantità d'interesse desiderate generando ripetutamente il comportamento del sistema
6. quando fermare la simulazione e quale confidenza dare agli stimatori ottenuti
7. tecniche per ottenere stimatori migliori
8. usare la simulazione per determinare se il modello stocastico ipotizzato è consistente con i dati reali

1. Perchè simulare?

Nello studio di fenomeni reali, il primo passo da compiere è certamente la formulazione di un modello. Solitamente si ricorre ad un compromesso tra un modello realistico che descrive fedelmente il fenomeno ed uno meno preciso ma più facilmente trattabile dal punto di vista matematico. Con l'avvento dei calcolatori, e la crescita continua della loro potenza computazionale, è possibile invece modellare un fenomeno in maniera quanto più fedele possibile e ricorrere alla simulazione per analizzarlo.

1.1. L'esempio della farmacia

Si consideri ad esempio una farmacia. L'arrivo dei nuovi clienti segue una certa distribuzione, infatti i tempi che intercorrono tra l'arrivo di un cliente e del successivo sono esponenziali. Le richieste di ciascun cliente sono soddisfatte dal farmacista in tempi aventi distribuzione normale.
Naturalmente, in alcune situazioni, può crearsi una coda. Per quanto il farmacista si impegni, può capitare che talvolta debba fare straordinario per soddisfare le richieste dei clienti che si erano messi in coda prima del termine dell'orario di lavoro.
Proviamo a simulare una giornata tipica di lavoro del farmacista, al fine di stimare i minuti di straordinario che dovrà fare. Sappiamo che

- l'intervallo di tempo tra l'arrivo di un cliente e di quello successivo (`deltat`) è una variabile casuale di distribuzione esponenziale di parametro n;
- il tempo richiesto per soddisfare la richiesta di un cliente (`filltime`) è una variabile casuale di distribuzione normale con media m e deviazione standard s.

Usiamo Scilab per eseguire la simulazione. Innanzitutto definiamo una funzione che riceve in ingresso il parametro n (assumiamo che la giornata di lavoro sia di 8 ore, e quindi di 480 minuti, in tal caso $480/n$ è il tempo medio di interarrivo dei clienti) e restituisce una singola (`1,1`) specificazione di una variabile casuale con distribuzione esponenziale (`'exp'`) di parametro n e quindi media $480/n$:

```
function [x] = interTempo(n)
  x = grand(1,1,'exp',480/n)
endfunction
```

Definiamo quindi una seconda funzione che riceve in ingresso i due parametri m e s e restituisce una singola (`1,1`) specificazione di una variabile casuale con distribuzione normale (`'nor'`) di media m e deviazione standard s (dal momento che si tratta di

un ritardo, non accettiamo specificazioni di valore negativo... in tal caso tronchiamo il risultato a 0):

```
function [x] = fillTime(m,s)
  x = max(0,grand(1,1,'nor',m,s))
endfunction
```

Definiamo ancora una funzione che riceve in ingresso i parametro n, m, s, e restituisce una tabella costituita dai tempi di interarrivo dei clienti (prima colonna) e dai tempi necessari alla soddisfazione delle richieste dei clienti (seconda colonna). Vengono aggiunte righe (`$+1`) finchè la somma dei tempi di interarrivo non supera 480 (perchè al termine dell'orario di lavoro nessun cliente può più mettersi in coda).

```
function [simtable] = simulatePharma(n,m,s)
  t = 0;
  while (t <= 480) do
    iT = interTempo(n);
    t = t + iT;
    simtable($+1,[1:2]) = [iT,fillTime(m,s)];
  end
endfunction
```

Definiamo una funzione accessoria che riceve in input una tabella e un numero di colonna e restituisce un vettore in cui l'i-esimo elemento è somma di tutti gli elementi precedenti della sua stessa colonna:

```
function [cumtable] = cumulate(table,col)
  i = 2;
  rows_number = size(table,'r');
  cumtable(1) = simtable(1,col);
  while (i <= rows_number) do
    cumtable(i) = cumtable(i-1) + simtable(i-1,col);
    i = i + 1;
  end
endfunction
```

Definiamo una funzione che riceve in input una tabella e restituisce un vettore il cui i-esimo elemento rappresenta il tempo in cui la richiesta dell'i-esimo cliente sarà stata soddisfatta.
`max(0,old-table(i,1))` rappresenta il tempo che l'i-esimo cliente passa in coda.
`table(i,2)` rappresenta il tempo necessario al farmacista per soddisfare la richiesta dell'i-esimo cliente.

```
function [workt] = worktime(table)
  workt(1) = table(1,2);
  old = table(1,2);
  i = 2;
```

```
  while (i <= size(table,'r')) do
    new = max(0,old-table(i,1)) + table(i,2);
    workt(i) = new;
    old = new;
    i = i + 1;
  end
endfunction
```

Definiamo la funzione che riceve in input n, m, s, avvia la simulazione completa (richiamando tutte le funzioni precedenti) e restituisce il numero dei minuti lavorati dal farmacista, cioè

- un valore pari a 480 se non dovrà fare straordinario;
- altrimenti un valore pari alla somma del minuto in cui è arrivato in farmacia l'ultimo cliente (`times`($,$)) e i minuti di lavoro arretrato che si sono accumulati nel corso della giornata (`workt`($,$)).

```
function [min_work] = goSimulate(n,m,s)
  simtable = simulatePharma(n,m,s);
  times = cumulate(simtable,2);
  workt = worktime(simtable);
  min_work = max(480,times($,$) + workt($,$))
endfunction
```

Ecco infine un esempio di simulazione numerica, la ripetiamo 365 volte:

```
n = 32;
m = 10;
s = 4;
k = 1;
while (k <= 365) do
  workdays(k) = goSimulate(n,m,s);
  k = k + 1;
end
```

Al termine del calcolo, `workdays` è un vettore di 365 righe contenente il numero di minuti lavorati in ogni singolo giorno dell'anno, quindi 365 valori maggiori o uguali a 480.
Per riassumere i risultati, possono essere raggruppati in modo da evidenziarne la frequenza:

```
--> tabul(workdays)

  ans=

        583.4238     1.
```

```
    550.72706    1.
    550.66163    1.
    513.2591     1.
    499.7507     1.
    497.06635    1.
    496.98472    1.
    490.45547    1.
    490.1833     1.
    480.         356.
```

ed è infine possibile disegnarne un grafico normalizzato (1.1):

```
histplot([min(workdays):ceil(max(workdays))],workdays);
```

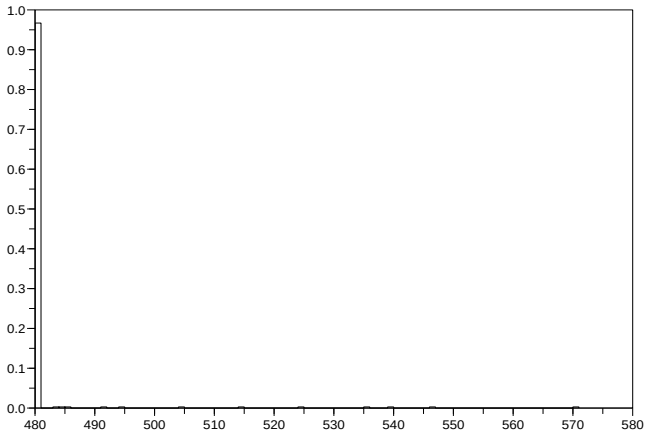

Figura 1.1.: Minuti di lavoro giornaliero effettivo del farmacista

Come si vede dai risultati, nella maggior parte dei casi (356/365 giorni di lavoro), il farmacista terminerà il suo orario di lavoro regolarmente, ma in alcuni giorni sarà costretto a fare straordinario.

Calcoliamo infine quanti minuti il farmacista lavorerà in media ogni giorno:

```
--> mean(workdays)

    ans  =

       480.96579
```

1.2. L'esempio del Risiko

Come secondo esempio di simulazione, consideriamo il gioco del Risiko. L'attaccante ha a disposizione 3 dadi, il difensore 2. L'esito dell'attacco è determinato confrontando i due dadi di valore maggiore di ciascun giocatore. A parità di valore, il difensore ha la meglio.
Siamo interessati a determinare, tramite simulazione, la probabilità che il difensore esca indenne da un attacco. In Scilab, possiamo scrivere questa semplice funzione:

```
function [esito] = resiste()
  attacco = gsort(grand(1,3,'uin',1,6));
  difesa = gsort(grand(1,2,'uin',1,6));
  if (attacco(1) <= difesa(1) & attacco(2) <= difesa(2)) then
    esito = 1 // il difensore esce indenne dalla battaglia
  else
    esito = 0 // il difensore perde almeno un'armata
  end
endfunction
```

Ripetendo l'esperimento più volte, stimiamo la probabilità dell'evento d'interesse con la media campionaria dei risultati.

```
--> mean(feval([1:100000],resiste))
 ans  =

    0.29093
```

1.3. L'esempio del dilemma di Monty Hall

Dimostriamo con una simulazione la soluzione del dilemma di Monty Hall. In questo problema, il conduttore di un concorso a premi televisivo ci chiede di indovinare quale porta, delle tre disponibili, cela il premio. Dopo la nostra scelta, apre una porta (dietro la quale non c'è il premio) e ci dà la possibilità di cambiare la nostra scelta iniziale. Conviene cambiare o no?
Si può dimostrare che conviene effettivamente cambiare, perchè in questo modo la nostra possibilità di vincita aumenta da 1/3 a 2/3.
Vediamo come una simulazione ci permette di confermare la soluzione del dilemma data.
Simuliamo per 10000 volte l'evento:

```
clear;

N=10000;
correctWhenStick = 0;
correctWhenChange = 0;
```

```
for i=[1:N] do
  rightAnswer = grand(1,1,'uin',1,3);
  initialGuess = grand(1,1,'uin',1,3);
  if (initialGuess<>rightAnswer) then
    correctWhenChange = correctWhenChange+1;
  else
    if (initialGuess==rightAnswer) then
      correctWhenStick = correctWhenStick+1;
    end;
  end;
end
```

Vediamo ora qual è la probabilità di vincita se adotto la politica di non cambiare mai la porta scelta:

```
--> correctWhenStick/N
 ans  =

    0.3392
```

cioè circa 1/3.
Se invece adotto la politica di cambiare la mia scelta, ho probabilità di vincita pari a

```
--> correctWhenChange/N
 ans  =

    0.6608
```

cioè circa 2/3.

1.4. L'esempio della distribuzione t di student

Agli inizi del '900 William Gosset lavorava alla fabbrica di birra Guinness con il compito di applicare metodi statistici per verificare la qualità della produzione. Durante il suo lavoro arrivò a sviluppare una nuova distribuzione, ora nota come distribuzione t di Student (lo pseudonimo che Gosset dovette adottare per pubblicare nel 1908 questo risultato sulla rivista Biometrika, in quanto la Guinness considerava l'utilizzo di metodi statistici per assistere la produzione come un segreto industriale).
Questa distribuzione doveva, tra le altre cose, descrivere il rapporto tra una variabile normale standard e una variabile chi-quadro divisa per il numero dei suoi gradi di libertà. Per verificare empiricamente che la distribuzione teorica fosse in accordo con questo fatto, Gosset utilizzò un campione di 3000 misurazioni fatte su persone come campione (approssimato) di una distribuzione normale, che dopo un procedimento di standardizzazione divise in 750 sotto-campioni di quattro elementi, e per ognuno di questi sotto-campioni calcolò:

- la media campionaria, di cui standardizzò il valore (moltiplicandola per la radice quadrata della taglia del sotto-campione, quindi 2), ottenendo in questo modo una specificazione di una variabile normale standard;

- la somma degli scarti quadratici dalla media, che sapeva seguire una distribuzione chi-quadro i cui gradi di libertà erano pari al numero di elementi nel sotto-campione meno uno (quindi 3);

Possiamo ripetere questo esperimento partendo da una tabella di 3000 dati simulati aventi una distribuzione normale standard.

La simulazione si basa sul fatto che se Z è una variabile casuale normale standard e U è una variabile casuale chi-quadro con k gradi di libertà, allora $\frac{Z}{\sqrt{U/k}}$ segue una distribuzione t di Student con k gradi di libertà. Si parte quindi da un campione di *numData* elementi suddivisi in sotto-campioni di dimensione n (nel nostro caso, $n = 4$). Di ciascuno di questi da una parte si calcola la media campionaria e la si moltiplica per $\sqrt{n}$ (per normalizzare il risultato, ottenendo una specificazione di Z), dall'altra si calcola la somma degli scarti quadratici dalla media (ottenendo una specificazione di U che in questo caso ha $n - 1$ gradi di libertà). Alla fine si divide il primo valore (Z) per la radice del secondo (U) diviso $n - 1$:

```
numData = 3000;
data = grand(1,numData,'nor',0,1);
t = [];
i = 1;
while (i <= numData - 3) do
  xm = mean(data(1,i:i+3));
  chi = 0;
  for j = [0:3],
    chi = chi + (data(1,i+j) - xm)^2;
  end;
  t(1,$+1) = (xm*2) / sqrt(chi/3);
  i = i + 4;
end
```

Confrontiamo ora graficamente (1.2) la distribuzione così ottenuta e quella teorica t di student (in rosso):

```
// densità empirica
histplot(75,t)

// densità t di student
function [f] = fT(t,nu)
  divisore = sqrt(%pi * nu) .* gamma(nu/2);
  f = gamma((nu+1)./2) .* (1+t.^2./nu).^(-(nu+1)/2) / divisore;
endfunction
tt = [min(t):0.1:max(t)];
```

```
ff = fT(tt,3);
plot2d(tt,ff,style=5);
p = get("hdl");
p.children.thickness = 2;
```

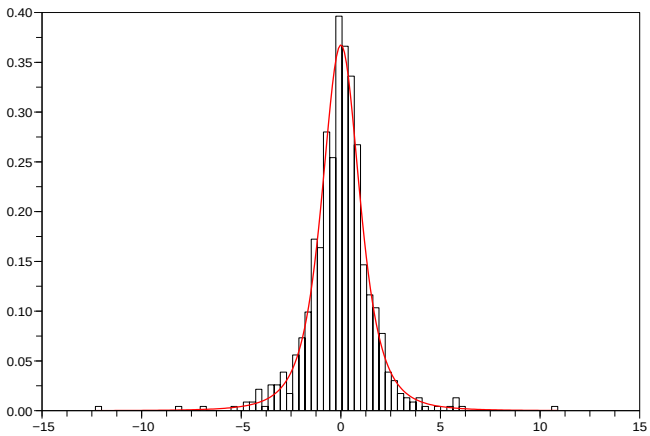

Figura 1.2.: Confronto tra densità di probabilità empirica generata e t di student

2. Generazione di numeri pseudocasuali

Passo iniziale di ogni simulazione è la generazione di un numero casuale, specificazione di una variabile casuale uniformemente distribuita nell'intervallo $(0, 1)$.
Ecco, velocemente, alcune tecniche. Senza approfondire, in seguito si supporrà che sia dato un metodo che genera variabili casuali uniformi in $(0, 1)$.

2.1. Algoritmo middle-square

Semplice algoritmo di Von Neumann per generare numeri pseudocasuali.

Algorithm 1 Algoritmo middle-square

Require: un seme s
Ensure: una sequenza di numeri casuali r
  loop
    $s \Leftarrow s^2$
    $r \Leftarrow n$ cifre centrali di s
    restituisci r
    $s \Leftarrow r$
  end loop

Non è un buon algoritmo: il periodo non può essere più lungo di 10^n e quindi dopo qualche computazione l'algoritmo restituisce gli stessi valori.

In Scilab può essere realizzato così:

```
clear;

function [n] = vonNeumannNextRand(n)
  l = ceil(log10(n));
  sn = "";
  for i = floor(l/2)+1:floor(3*l/2),
    j = part(string(n^2),i);
    sn = sn + j;
  end;
  n = evstr(sn);
endfunction
```

```
function [result] = MS_gen(N,seed)
  result = [];
  n = seed;
  for i=[1:N],
    n = vonNeumannNextRand(n);
    result($+1,1) = n/9999;
  end
endfunction
```

Segue un esempio di generazione di 50 numeri casuali con seme 5132. Si notino le ripetizioni:

```
 -->MS_gen(50,5132)
 ans  =

    0.3374337
    0.3838384
    0.7302730
    0.3192319
    0.1888189
    0.6454645
    0.6541654
    0.7846785
    0.5597560
    0.3264326
    0.6536654
    0.7192719
    0.7248725
    0.5335534
    0.4622462
    0.3628363
    0.1623162
    0.3412341
    0.6417642
    0.1778178
    0.6128613
    0.5523552
    0.5035504
    0.3512351
    0.3341334
    0.1622162
    0.3088309
    0.3574357
    0.7734773
    0.8147815
```

```
0.3736374
0.9576958
0.6997700
0.9580958
0.7764776
0.2796280
0.1761176
0.0112011
0.0254025
0.0451045
0.0034003
0.0015002
0.0025003
0.0025003
0.0025003
0.0025003
0.0025003
0.0025003
0.0025003
0.0025003
```

2.2. Generatori congruenziali lineari

Algorithm 2 Generatore congruenziale lineare

Require: il seme x_0 e i parametri a, m e c
Ensure: una sequenza di numeri casuali x_1
  loop
    $x_1 \Leftarrow (ax_0 + c) \bmod m$ $\{a \in N, m \in N\}$
    restituisci x_1
    $x_0 \Leftarrow x_1$
  end loop

Il numero di valori restituiti dal generatore (**modulo**) è m se $c \neq 0$, altrimenti $m-1$ perchè 0 non può essere un numero casuale valido: infatti, nel caso $c = 0$ e $x = 0$ il generatore continuerebbe a restituire lo stesso valore.
a, m e c vanno scelti accuratamente, in modo che l'algoritmo soddisfi tre criteri:

- per qualsiasi x_0, la sequenza generata deve sembrare una sequenza di variabili casuali con distribuzione uniforme in $(0, 1)$;
- per qualsiasi x_0, il numero di variabili generate prima che si incorra in una ripetizione deve essere grande;
- i valori devono poter essere calcolati efficientemente da un calcolatore.

Nel caso del generatore congruenziale moltiplicativo ($c = 0$), il valore di m è solitamente il più grande numero primo rappresentabile in una parola di memoria del calcolatore: se questa è di 32 bit, allora è comune la scelta $m = 2^{31} - 1$. In questo caso, si usa porre $a = 7^5$ al fine di ottenere un periodo completo.
Riguardo la scelta dei parametri, esiste una trattazione matematica piuttosto complessa, vedere come riferimento le seguenti risorse:

- `http://cnx.org/content/m13103/latest/`
- `http://en.wikipedia.org/wiki/Linear_congruential_generator`

In Scilab, un generatore congruenziale lineare misto può essere realizzato così:

```
global storedRandom
global ret
global aCongr
global cCongr
global mCongr

function setCongruentialGenerator(a,c,m)
  global aCongr
  global cCongr
  global mCongr
  aCongr = a;
  cCongr = c;
  mCongr = m;
endfunction

function setCongruentialSeed(s)
  global storedRandom
  storedRandom = s;
endfunction

function [ret] = nextRandom()
  global storedRandom
  global ret
  ret = storedRandom / mCongr;
  storedRandom = modulo(aCongr * storedRandom + cCongr,mCongr);
endfunction

function init(a,c,m,s)
  setCongruentialGenerator(a,c,m);
  setCongruentialSeed(s);
endfunction

function [vect] = genera(dim,a,c,m,s)
```

```
  init(a,c,m,s);
  i = 1;
  vect = [];
  for i=1:dim,
    vect($+1,1) = nextRandom();
  end
endfunction
```

Possiamo prendere ad esempio questi parametri

- $a = 7^5$
- $c = 1$
- $m = 2^{31} - 1$
- $s = 0.3$

e generare 50 numeri casuali:

```
-->y = genera(50,7^5,1,2^31-1,.3)
 ans  =

    1.397D-10
    0.0000023
    0.0394692
    0.3582272
    0.7247460
    0.8056184
    0.0282977
    0.6002536
    0.4619313
    0.6799885
    0.5661014
    0.4662917
    0.9646308
    0.5495215
    0.8078074
    0.8195042
    0.4072422
    0.5197050
    0.6815840
    0.3831011
    0.7807103
    0.3980075
    0.3115891
    0.8778285
```

```
    0.6643356
    0.4881090
    0.6474658
    0.9573364
    0.9524013
    0.0079492
    0.6022155
    0.4355034
    0.5048765
    0.4592549
    0.6967237
    0.8349342
    0.7387421
    0.0390351
    0.0627497
    0.6343398
    0.3484593
    0.5546944
    0.7494604
    0.1815758
    0.7442983
    0.4210426
    0.4626043
    0.9912541
    0.0072823
    0.3928338
```

Possiamo disegnare il grafico della distribuzione dei numeri casuali (2.1)

```
plot2d([1:50],y)
c = get("hdl");
p = c.children;
p.mark_mode = "on";
p.line_mode = "off";
p.mark_style = 0;
p.mark_size = 1;
```

anche come istogramma delle frequenze (2.2)

```
set("current_figure",1)
histplot([0:.02:1],y)
```

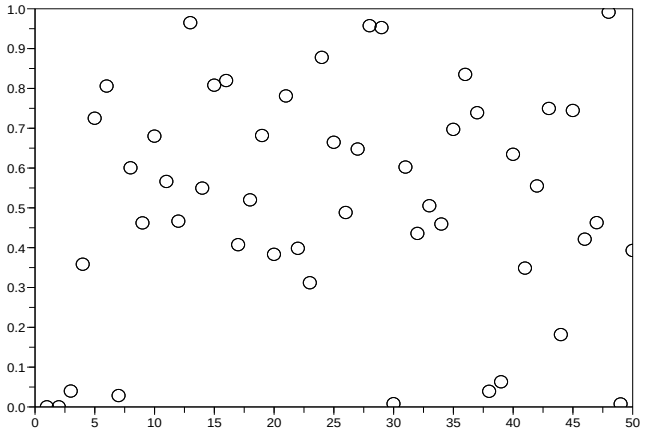

Figura 2.1.: Esempio di distribuzione di numeri casuali

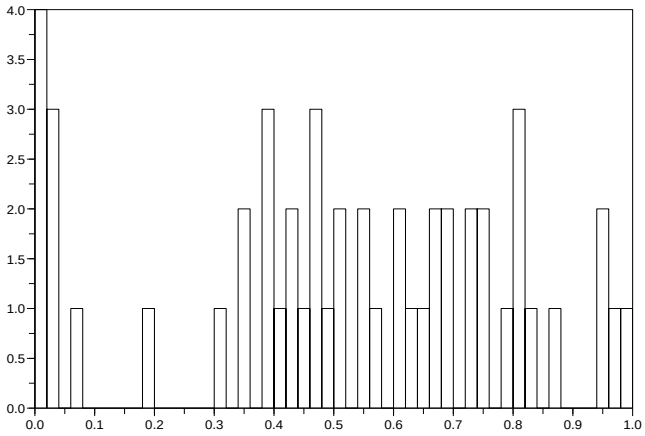

Figura 2.2.: Istogramma della distribuzione di numeri casuali

2.3. Generatori basati su shift register

Algorithm 3 Generatore basato su shift register a 3 bit

Require: una stringa binaria $b_0 b_1 b_2$ in un registro di dimensione 3
Ensure: una sequenza di stringhe binarie casuali di dimensione 3
  loop
    $b_1 \Leftarrow b_0$
    $b_2 \Leftarrow b_1$
    $b_0 \Leftarrow b_1$ XOR b_2
  end loop

Un generatore basato su shift register è il generatore Tausworthe: il numero casuale u_i in output, codificato da L bits, viene generato a partire da un vettore binario b il cui i-esimo elemento viene calcolato applicando la funzione XOR ai due elementi binari di indici $i - p$ e $i - q$.

$$b_i = b_{i-p} \text{ XOR } b_{i-q}$$

$$u_i = \sum_{S=1}^{L} 2^{-S} b_{it+S}$$

L, p e q sono parametri del generatore.
Ecco come possiamo realizzarlo in Scilab:

```
p=10;
q=7;

// inizializzo il vettore binario
b=round(grand(1,max(p,q),'def'));

// numero di bits che codificano il valore casuale
L=32;
t=L;

// numero di valori casuali che voglio generare
m=200;

function [bx]=XOR(b1,b2)
  if (b1==b2) then
    bx=0
  else
    bx=1;
  end
endfunction
```

```
// estendo il vettore binario iniziale calcolando
// b(i) con la funzione XOR sui bits b(i-p) e b(i-q)
for i=[max(p,q)+1:m*t+L+1];
  b(i)=XOR(b(i-p),b(i-q));
end;

// preparo il vettore dei valori casuali finali
sb=zeros(1:m);

// genero sb(i) usando L bits del vettore binario b
for i=[1:m];
  for s=[1:L];
    sb(i)=sb(i)+2^(-s)*b(i*t+s);
  end;
end

// restituisco i primi venti valori casuali ottenuti
sb(1:20)'
```

Ecco un esempio dei primi 20 numeri casuali generati:

```
-->sb(1:20)'

 ans  =

    0.1576331
    0.1465209
    0.7149014
    0.1310341
    0.4046445
    0.2970441
    0.8866771
    0.7625523
    0.9127383
    0.4510829
    0.0383582
    0.2581280
    0.9806034
    0.7635223
    0.6430727
    0.6470058
    0.5684728
    0.7925944
    0.6708615
    0.5330044
```

2.4. Esempi di applicazioni

2.4.1. Approssimazione di integrali

Integrali in $(0, 1)$

Sia $g(x)$ una funzione. Vogliamo calcolare l'integrale

$$\theta = \int_0^1 g(x)\,\mathrm{d}x$$

Se U è una variabile casuale distribuita uniformemente in $(0, 1)$, possiamo scrivere

$$\theta = \mathrm{E}\left[g(U)\right]$$

Se le variabili casuali $U_1, \ldots, U_k$ sono indipendenti e identicamente distribuite (iid) come U, allora anche $g(U_1), \ldots, g(U_k)$ sono iid con media θ, e per la legge dei grandi numeri vale con probabilità 1 per $k \to \infty$

$$\sum_{i=1}^{k} \frac{g(U_i)}{k} \to \mathrm{E}\left[g(U)\right] = \theta$$

Possiamo quindi stimare θ generando un elevato numero di U_i e approssimare l'integrale con la media dei valori di $g(U_i)$ (**metodo Monte Carlo**).
Questa tecnica è particolarmente utile nel caso di integrali multipli.

In Scilab, se vogliamo approssimare l'integrale di una funzione g usando m numeri casuali, possiamo procedere in questo modo:

```
clear;
function [y] = integrale(m,g)
  x = grand(m,1,'def');
  y = mean(feval(x,g))
endfunction
```

Approssimiamo ad esempio l'integrale di $\dfrac{x^4+1}{x^3+1}$ usando 100 numeri casuali:

```
 deff('[y] = g(x)','y = (x^4 + 1) / (x^3 + 1)');
 m = 100;
-->integrale(m,g)
 ans  =

    0.9606645
```

Il valore reale dell'integrale è $\frac{4\log 2+3}{6}$, cioè circa 0.9620981.

Integrali in (a, b)

Se vogliamo calcolare

$$\theta = \int_a^b g(x)\,\mathrm{d}x$$

possiamo fare la sostituzione

$$y = \frac{x-a}{b-a}$$

Dobbiamo quindi ricalcolare x, $\mathrm{d}x$ e gli estremi di integrazione in funzione di y. Dopo il cambio di variabile, gli estremi di integrazione diventano 0 e 1, e si torna quindi al caso dell'integrale precedente.

Integrali in $(0, +\infty)$

Se vogliamo calcolare

$$\theta = \int_0^{+\infty} g(x)\,\mathrm{d}x$$

operiamo la sostituzione

$$y = \frac{1}{x+1}$$

e, procedendo come sopra, torniamo al caso dell'integrale in $(0, 1)$.

2.4.2. Approssimazione di π

Supponiamo di avere un quadrato di lato 2 e un cerchio inscritto (che avrà quindi raggio 1). Il rapporto tra il numero di punti del piano racchiusi nel cerchio e quelli nel quadrato è pari al rapporto tra le loro aree: $\pi/4$. Se quindi generiamo un elevato numero di punti, il rapporto tra quelli che cadono nel quadrato e quelli che cadono nel cerchio stimerà $\pi/4$, ed in tal modo è possibile ottenere una stima di π.
Dal punto di vista pratico, iniziamo generando specificazioni di $X = 2U_1 - 1$ e $Y = 2U_2 - 1$, variabili casuali compresi tra $(-1, 1)$. Ciascuna coppia rappresenta le coordinate di un punto, che è interno al cerchio se $X^2 + Y^2 \leq 1$. Non resta che contare i punti nel cerchio con la variabile casuale I e prenderne il valore atteso moltiplicato per 4 per ottenere la stima di π.
Quanti punti devo estrarre perchè l'errore dell'approssimazione sia sotto un certo valore?
Poniamo di voler approssimare π con un errore relativo inferiore a ϵ. Posso sfruttare la disuguaglianza di Markov

$$\mathrm{P}\left(g\left(X\right) \geq k\right) \leq \frac{\mathrm{E}\left[g\left(X\right)\right]}{k} \tag{2.1}$$

dove

$$g(X) = \frac{|\,\overline{I} - \pi/4\,|}{\pi/4}$$

e $k = \epsilon$. Per risolvere, conviene elevare $g(x) \geq k$ al quadrato (per eliminare il modulo) ed invertire la disuguaglianza di Markov ($\mathrm{P}(A) = 1 - \mathrm{P}(\overline{A})$).

In Scilab, se vogliamo approssimare π con n simulazioni di m punti ciascuna, possiamo procedere in questo modo:

```
clear;

function [pi] = approx_pi(m,n)
  c = zeros(m,n);
  cn = 1;
  while (cn <= n),
    x = grand(m,1,'unf',-1,1);
    y = grand(m,1,'unf',-1,1);
    for i=[1:m]
      if (x(i)^2 + y(i)^2 <= 1),
        c(i,cn) = c(i,cn) + 1;
      end;
    end;
    cn = cn + 1;
  end;
  pi = 4 * mean(c)
endfunction
```

Approssimiamo π eseguendo 100 simulazioni con 1000 punti:

```
-->approx_pi(1000,100)
 ans  =

    3.1338
```

Il valore reale di π è circa 3.1415927.

2.4.3. Approssimazione di e

Ricordiamo innanzitutto una delle definizioni del numero di Nepero:

$$e = \sum_{k=0}^{+\infty} \frac{1}{k!}$$

Data una sequenza di numeri $U_1, U_2, \ldots, U_n, \ldots$, consideriamo ora la variabile casuale N definita come la prima posizione in cui vale $U_n > U_{n-1}$. Dimostriamo che $\mathrm{E}[N] = e$. La probabilità che una sequenza di n numeri abbia un ordine strettamente decrescente ($N > n$) è

$$\mathrm{P}(N > n) = \frac{1}{n!}$$

poichè esiste una sola permutazione di questo tipo.
Determiniamo la funzione di massa di probabilità di N:

$$\mathrm{P}(N = n) = \mathrm{P}(N > n - 1) - \mathrm{P}(N > n) = \frac{1}{(n-1)!} - \frac{1}{n!} = \frac{n-1}{n!}$$

Concludiamo la dimostrazione calcolando quindi il valore atteso di N

$$\mathrm{E}[N] = \sum_{n=2}^{+\infty} n\mathrm{P}(N = n) = \sum_{n=2}^{+\infty} n\frac{n-1}{n!} = \sum_{n=2}^{+\infty} \frac{1}{n-2!} = \sum_{k=0}^{+\infty} \frac{1}{k!} = e$$

Per simulare e, possiamo quindi generare tante sequenze $U_1, U_2, \ldots, U_n, \ldots$ e fare la media delle posizioni in cui per ciascuna sequenza è valsa la condizione $U_n > U_{n-1}$.

In Scilab, se vogliamo approssimare e con n simulazioni, possiamo procedere in questo modo:

```
clear;

function [e] = approx_e(n)
  i = ones(1,n) * 2;
  cn = 1;
  while (cn <= n),
    u1 = grand(1,1,'def');
    u2 = grand(1,1,'def');
    while (u2 < u1),
      u1 = u2;
      u2 = grand(1,1,'def');
      i(cn) = i(cn) + 1;
    end;
    cn = cn + 1;
  end;
  e = mean(i)
endfunction
```

Approssimiamo e eseguendo 100 simulazioni:

```
-->approx_e(1000)
ans  =

   2.738
```

Il valore reale di e è circa 2.7182818.

3. Generazione di variabili casuali discrete

3.1. Metodo della trasformata inversa

Supponiamo di voler generare una specificazione di una variabile casuale discreta X con funzione di massa di probabilità

$$\mathrm{P}(X = x_j) = p_j \qquad \text{dove } j = 0, 1, \ldots, \sum_j p_j = 1$$

Possiamo utilizzare il metodo della trasformata inversa, implementata dall'algoritmo 4.

Algorithm 4 Metodo della trasformata inversa per v.c. discrete - algoritmo estensivo

Require: i valori p_j della funzione di massa di probabilità desiderata
Ensure: una specificazione di una variabile casuale X con probabilità desiderata

```
genera un numero casuale U uniforme in (0, 1)
if U < p_0 then
    X = x_0
else if U < p_0 + p_1 then
    X = x_1
else if U < p_0 + p_1 + p_2 then
    X = x_2
⋮

end if
```

Per spiegare l'origine del nome del metodo, notiamo che è basato sull'inversione della funzione di ripartizione. Infatti, essendo noto che vale

$$\mathrm{F}(x_k) = \sum_{i=0}^{k} p_i$$

possiamo scrivere

$$X = x_j \quad \text{se } \mathrm{F}(x_{j-1}) \leq U < \mathrm{F}(x_j)$$

Il procedimento è quindi il seguente:

1. si genera U, uniforme in $(0, 1)$
2. si partiziona l'intervallo $(0, 1)$ con i valori della funzione di ripartizione di X
3. si determina in quale sottointervallo di $(0, 1)$ si trova U
4. si assegna il valore a X di conseguenza

Se gli x_j non sono in numero finito, allora è necessario un algoritmo intensivo (5).

Algorithm 5 Metodo della trasformata inversa per v.c. discrete - algoritmo intensivo

Require: i valori $p_0, p_1, \ldots$ della funzione di massa di probabilità desiderata
Ensure: un valore di una variabile casuale X con probabilità desiderata

```
genera un numero casuale U uniforme in (0, 1)
i = 1
sp = p_1 {sp è la somma dei primi p_i}
fine=false
while fine=true do
  if U < sp then
    x = x_i
    fine=true
  else
    i = i + 1
    sp = sp + p_i
  end if
end while
```

Una nota: l'efficienza di questi algoritmi è migliore se le probabilità vengono ordinate in maniera decrescente, in modo da rendere minimo il numero di test e di cicli necessari.

3.1.1. La variabile casuale uniforme

In alcuni casi non è necessario cercare l'intervallo in cui cade il numero casuale.
Una variabile casuale X con distribuzione uniforme nell'intervallo $(1, n)$ può essere infatti generata facilmente a partire da una variabile casuale U con distribuzione uniforme nell'intervallo $(0, 1)$, semplicemente ponendo:

$$X = \lceil nU \rceil$$

Esempio: permutazione casuale

Se ci interessa generare una permutazione casuale di N elementi, possiamo procedere come descritto nell'algoritmo numero 6.

Inoltre, si può osservare che si ottiene una sottosequenza casuale di dimensione n interrompendo l'algoritmo dopo $n < N$ passi.

Algorithm 6 Permutazione casuale di N elementi

Require: un vettore di N elementi
Ensure: una permutazione casuale del vettore in ingresso
  $i = 1$
  for $i \leq N$ **do** {ordinamento del vettore in ingresso}
    $x[i] = x_i$
    $i = i + 1$
  end for
  for $i \leq N$ **do** {permutazione}
    $j = \lceil N * random() \rceil$
    $t = x[i],\ x[i] = x[j],\ x[j] = t$ {scambio x_i con x_j}
    $i = i + 1$
  end for

Un caso particolare di generazione di una sottosequenza casuale Data una sequenza di N elementi, desideriamo creare una sottosequenza casuale di dimensione n. La scelta degli elementi da inserire nella sottosequenza avviene con probabilità uniforme (tutti gli elementi hanno pari probabilità di essere selezionati). L'inserimento effettivo nella sottosequenza è però regolato dalla probabilità

$$\mathrm{P}\left(\frac{n-r}{N-K}\right)$$

dove r (funzione del numero di iterazione-scelta) è il numero di elementi già inseriti nella sottosequenza, e K (funzione del numero di iterazione-scelta) è il numero di elementi della sequenza già selezionati ma non inseriti nella sottosequenza.
Dopo aver scelto un elemento nella sequenza con probabilità uniforme, K viene incrementato di una unità e viene generato un altro valore di U. Se vale

$$U \leq \mathrm{P}\left(\frac{n-r}{N-K}\right)$$

l'elemento della sequenza viene inserito nella sottosequenza e r viene incrementato di una unità.

3.1.2. La variabile casuale geometrica

Anche in questo caso non è necessario cercare l'intervallo in cui cade il numero casuale. Ricordiamo che una variabile casuale geometrica rappresenta il tempo del primo successo in ripetuti esperimenti indipendenti, e calcoliamo la probabilità che il successo arrivi all'i-esimo tentativo, con $i = 1, \ldots, j-1$:

$$\mathrm{F}(j-1) = \mathrm{P}(X \leq j-1) = \sum_{i=1}^{j-1} \mathrm{P}(X = i) = 1 - \mathrm{P}(X > j-1) = 1 - q^{j-1}$$

Usando il metodo tradizionale, potremmo quindi ottenere una variabile geometrica X generando un numero casuale U e impostando $X = j$ in modo tale che

$$1 - q^{j-1} \leq U < 1 - q^j$$

X è quindi il valore minore di j per cui vale $q^j < 1 - U$.
Se invece non vogliamo invertire la funzione di ripartizione e cercare l'intervallo in cui U si trova, possiamo partire dalla considerazione che è possibile applicare il logaritmo (funzione monotona) alla condizione precedente

$$j \log q < \log(1 - U)$$

e dividere per $\log q$ invertendo il segno della disequazione (perchè $\log q$ è negativo per $0 < q < 1$)

$$j > \frac{\log(1 - U)}{\log q}$$

Possiamo quindi esprimere X come

$$X = \left\lceil \frac{\log(1 - U)}{\log q} \right\rceil$$

oppure, in maniera equivalente dal momento che U e $1 - U$ sono identicamente distribuite,

$$X = \left\lceil \frac{\log U}{\log q} \right\rceil$$

3.1.3. La variabile casuale bernoulliana

Anche in questo caso non è necessario cercare l'intervallo in cui cade il numero casuale. Supponiamo infatti di voler generare una variabile casuale bernoulliana, che assume valore 1 con probabilità p. Posso quindi generare una variabile casuale uniforme U e restituire 1 se ha valore $u < p$, 0 altrimenti.
Se invece voglio generare una sequenza $b_1, b_2, \ldots, b_n$ di variabili casuali bernoulliane, posso farlo più efficientemente generando la specificazione j di una variabile casuale geometrica, ed impostando $b_j = 1$, $b_i = 0$ per $i < j$.

3.1.4. La variabile casuale poissoniana

A differenza dei casi precedenti, in questo è necessario usare il metodo della trasformata inversa.
La generazione di una variabile casuale poissoniana è resa più efficiente dal calcolo ricorsivo della probabilità:

$$p_{i+1} = \frac{\lambda}{i + 1} p_i$$

L'algoritmo 7 esegue $1 + \lambda$ ricerche, potrebbe essere ottimizzato se il controllo non partisse da 0 ma intorno al valor medio della poissoniana (λ, appunto). Si può determinare con l'approssimazione normale che il numero medio di ricerche con questa ottimizzazione sarebbe pari a $1 + \sqrt{\frac{2\lambda}{\pi}}$.

Algorithm 7 Metodo della trasformata inversa per la v.c. poissoniana

Require: il parametro λ della poissoniana
Ensure: il valore di una variabile casuale poissoniana
  genera un numero casuale U
  $i = 0, p = \mathrm{e}^{-\lambda}, F = p$ {F è il valore della funzione di ripartizione}
  loop
    if $U < F$ **then**
      $X = i$, stop
    else
      $p = \frac{\lambda}{i+1} p$
      $F = F + p$
      $i = i + 1$
    end if
  end loop

3.1.5. La variabile casuale binomiale

Anche in questo caso è necessario usare il metodo della trasformata inversa ed è possibile rendere più efficiente la generazione della variabile casuale calcolandone ricorsivamente la probabilità:

$$p_{i+1} = \frac{n-i}{i+1} \frac{p}{1-p} p_i$$

Algorithm 8 Metodo della trasformata inversa per la v.c. binomiale

Require: i parametri n, p della binomiale
Ensure: il valore di una variabile casuale binomiale
  genera un numero casuale U
  $c = \frac{p}{1-p}, i = 0, p = (1-p)^n, F = p$ {F è il valore della funzione di ripartizione}
  loop
    if $U < F$ **then**
      $X = i$, stop
    else
      $p = \left(c \frac{n-i}{i+1}\right) p$
      $F = F + p$
      $i = i + 1$
    end if
  end loop

L'algoritmo 8 esegue in media $1 + np$ ricerche. Come nel caso della poissoniana, è possibile migliorare l'algoritmo eseguendo la ricerca di u intorno al valore atteso della distribuzione. Se $p > 1/2$, è conveniente generare i fallimenti (cioè una binomiale con parametri $n, 1-p$).
Infine, per generare una specificazione di cariabile casuale binomiale posso anche utilizzare n prove di Bernoulli e contare il numero di successi.

4. Generazione di variabili casuali continue

4.1. Metodo della trasformata inversa

Supponiamo di voler generare una specificazione di una variabile casuale continua avente funzione di ripartizione F qualsiasi. Se U è una variabile casuale uniforme in $(0, 1)$, la variabile casuale X definita da

$$X = F^{-1}(U)$$

ha funzione di ripartizione F.
Si noti che, essendo la funzione di ripartizione continua monotona per definizione, esiste sempre la sua funzione inversa.

4.1.1. La variabile casuale esponenziale

Una variabile casuale esponenziale ha funzione di ripartizione

$$F_X(x; \lambda) = 1 - \mathrm{e}^{-\lambda x}$$

quindi

$$u = 1 - \mathrm{e}^{-\lambda x}$$

La funzione inversa è

$$x = -\frac{1}{\lambda} \ln(1 - u)$$

oppure

$$x = -\frac{1}{\lambda} \ln(u)$$

essendo u e $1 - u$ specificazioni delle variabili casuali U e $1 - U$ identicamente distribuite.
In Scilab, per generare n specificazioni di una variabile casuale con distribuzione esponenziale di parametro $\lambda = 0.5$ possiamo procedere come segue:

```
function [y] = esponenziale(l,n)
  y = -1/l .* log(grand(n,1,'def'))
endfunction

n = 1000;
dati = esponenziale(.5,n);
```

Verifichiamo la bontà dei valori simulati (in nero) rispetto a quelli reali (in rosso): disegniamo la funzione di distribuzione empirica (4.1)

```
// distribuzione empirica
dati = gsort(dati,'g','i');
plot2d(dati,[1:n]/n)

// distribuzione reale
deff('[EE] = EEX(x,l)','EE = 1 - exp(-l .* x)')
tt = [min(dati):.1:max(dati)];
ff = EEX(tt,.5);
plot2d(tt,ff,style=5);
```

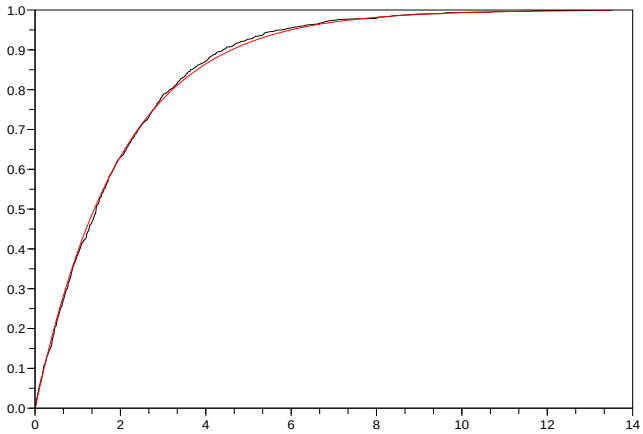

Figura 4.1.: Funzione di distribuzione empirica e teorica (in rosso)

e la funzione di densità di probabilità empirica 4.2

```
// densità empirica
set("current_figure",1)
histplot([0:max(dati)],dati)

// densità reale
deff('[ee] = eex(x,l)','ee = l .* exp(-l .* x)')
tt=[min(dati):.1:max(dati)];
ff = eex(tt,.5);
plot2d(tt,ff,style=5);
```

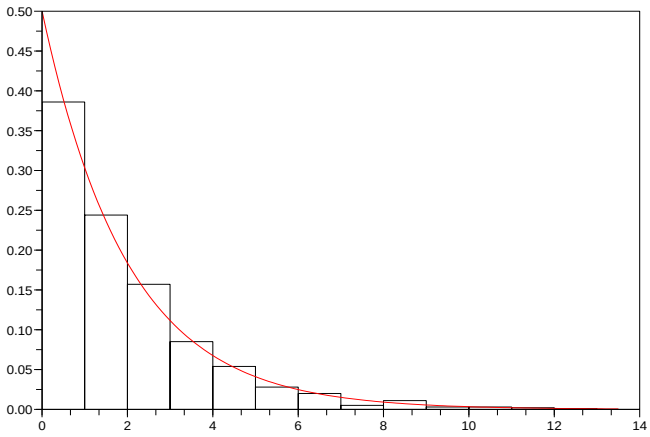

Figura 4.2.: Funzione di densità di probabilità empirica e teorica (in rosso)

4.2. Metodi per la distribuzione normale

Mostriamo ora come generare variabili casuali normali standard, quindi con media 0 e varianza 1. A partire da una variabile normale standard Z, è possibile ottenere una variabile casuale normale X generica, con media μ e varianza σ^2, usando la seguente formula:

$$X = \mu + \sigma Z$$

4.2.1. Metodo di Box-Muller

Siano X e Y normali standard indipendenti, allora la loro densità di probabilità congiunta vale

$$\mathrm{f}_{X,Y}(x,y) = \mathrm{f}_X(x)\,\mathrm{f}_Y(y) = \frac{1}{\sqrt{2\pi}}\mathrm{e}^{-x^2/2}\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-y^2/2} = \frac{1}{2\pi}\mathrm{e}^{-\frac{x^2+y^2}{2}}$$

Passiamo ora alle coordinate polari:

$$R^2 = X^2 + Y^2$$

$$\tan\theta = \frac{Y}{X}$$

Per ottenere la densità di probabilità congiunta di (R^2, Θ) (chiamiamola $\mathrm{f}(d,\theta)$), devo fare il cambio di variabili

$$d = x^2 + y^2$$

$$\theta = \arctan\left(\frac{y}{x}\right)$$

Dopo aver determinato che il determinante Jacobiano (il determinante delle derivate parziali di d e θ rispetto a x e a y) vale 2, posso scrivere la densità di probabilità congiunta di (R^2, Θ)

$$\mathrm{f}(d, \theta) = \frac{1}{2}\frac{1}{2\pi}\mathrm{e}^{-d/2}$$

Si noti che è prodotto di una densità uniforme su $(0, 2\pi)$ (rappresentato nella formula da $\frac{1}{2\pi}$) e di una densità esponenziale di media 2 (rappresentato nella formula da $\frac{1}{2}\mathrm{e}^{-d/2}$): ne segue che R^2 e Θ sono indipendenti, con R^2 esponenziale di media 2 e Θ uniforme in $(0, 2\pi)$.
Possiamo quindi generare una coppia di variabili casuali normali standard generando prima le loro coordinate polari e quindi trasformandole in coordinate rettangolari.

Algorithm 9 Algoritmo di Box-Muller

Ensure: una coppia di variabili casuali normali standard
genera U_1 e U_2
$R^2 = -2\log U_1$
$\Theta = 2\pi U_2$
$X = R\cos\Theta = \sqrt{-2\log U_1}\cos(2\pi U_2)$
$Y = R\sin\Theta = \sqrt{-2\log U_1}\sin(2\pi U_2)$

Si osservi che l'algoritmo 9 computazionalmente non è molto efficiente, dal momento che richiede il calcolo delle funzioni seno e coseno.

In Scilab, possiamo generare n coppie di variabili casuali normali standard con queste istruzioni:

```
function [y1,y2]=normale()
  u1 = grand(1,1,'def');
  u2 = grand(1,1,'def');
  d = -2 * log(u1);
  theta = 2 * %pi * u2;
  y1 = sqrt(d) * cos(theta);
  y2 = sqrt(d) * sin(theta);
endfunction

function [y] = repeat(n,g)
  for i=[1:n]
    [y1,y2] = g();
    y($+1) = y1;
    y($+1) = y2;
  end
endfunction
```

```
dati = repeat(1000, normale);
```

Disegnamo la funzione di distribuzione teorica (in rosso) ed empirica (4.3)

```
// funzione di distribuzione empirica
dati = gsort(dati,'g','i');
m = size(dati,'r');
plot2d(dati,[1:m]/m)

// funzione di distribuzione teorica
x = [-4:.1:4];
size_x = size(x,'c');
plot2d(x,cdfnor("PQ",x,zeros(1,size_x),ones(1,size_x)),style=5)
```

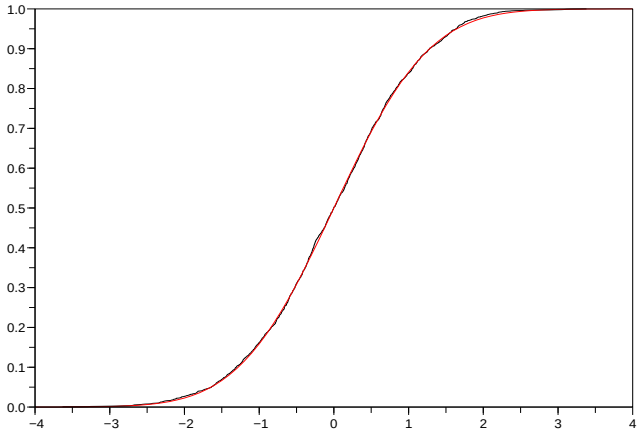

Figura 4.3.: Funzione di distribuzione empirica e teorica (in rosso)

e la funzione di densità di probabilità teorica (in rosso) ed empirica (4.4)

```
// funzione di densità empirica
set("current_figure",1)
Max = max(dati);
Min = min(dati);
histplot([Min:(Max-Min)/100:Max],dati)

// funzione di densità teorica
function [f] = fN(x,mu,sigma)
  f = exp(-(x-mu)^2/(2*sigma^2)) / (sigma*sqrt(2*%pi));
endfunction
```

```
tt = [Min:0.1:Max];
ff = fN(tt,0,1);
plot2d(tt,ff,style=5);
p = get("hdl");
p.children.thickness = 2;
```

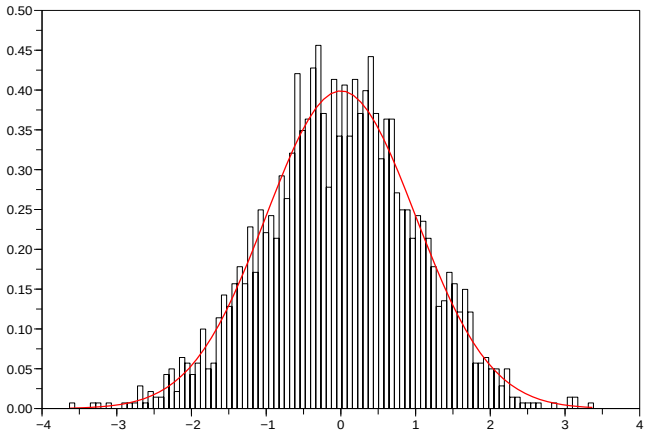

Figura 4.4.: Funzione di densità di probabilità empirica e teorica (in rosso)

4.2.2. Usando il teorema centrale

Ricordiamo il teorema del limite centrale: la somma di n variabili casuali identicamente distribuite con media μ e varianza σ^2, standardizzata sottraendo $n\mu$ e dividendo per $\sigma\sqrt{n}$, ha funzione di distribuzione normale standard per $n \to \infty$.
Possiamo quindi approssimare una variabile casuale normale standard a partire da m variabili casuali iid con distribuzione qualsiasi.

In Scilab, proviamo a simulare una variabile casuale normale standard a partire da n bernulliane. Ripetiamo l'esperimento m volte:

```
function [y] = normale_approx(n,m)
  for i=[1:m],
    u = grand(n,1,'def');
    y($+1) = (sum(u) - n/2) * sqrt(12/n);
  end
endfunction
```

```
dati = normale_approx(1000,1000);
```

Disegnamo la funzione di distribuzione teorica (in rosso) ed empirica (4.5)

```
// funzione di distribuzione empirica
dati = gsort(dati,'g','i');
m = size(dati,'r');
plot2d(dati,[1:m]/m)

// funzione di distribuzione teorica
x = [-4:.1:4];
size_x = size(x,'c');
plot2d(x,cdfnor("PQ",x,zeros(1,size_x),ones(1,size_x)),style=5)
```

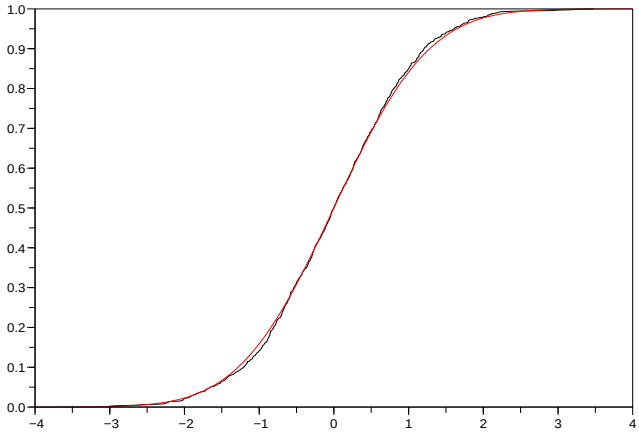

Figura 4.5.: Funzione di distribuzione empirica e teorica (in rosso)

e la funzione di densità di probabilità teorica (in rosso) ed empirica (4.6)

```
// funzione di dentità empirica
set("current_figure",1)
Max = max(dati);
Min = min(dati);
histplot([Min:(Max-Min)/100:Max],dati)

// funzione di densità teorica
function [f] = fN(x,mu,sigma)
  f = exp(-(x-mu)^2/(2*sigma^2)) / (sigma*sqrt(2*%pi))
endfunction
```

```
tt = [Min:0.1:Max];
ff = fN(tt,0,1);
plot2d(tt,ff,style=5);
p = get("hdl");
p.children.thickness = 2;
```

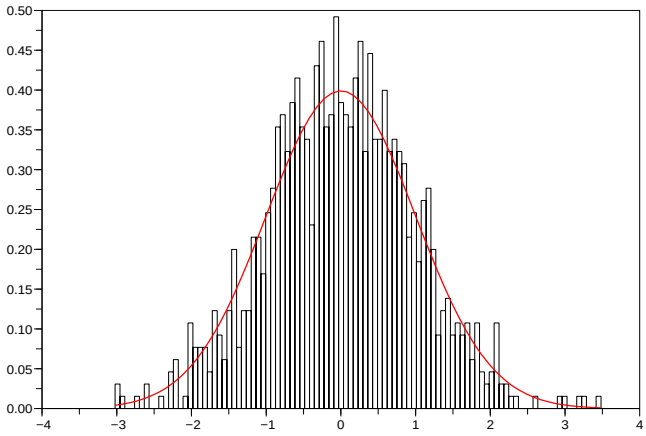

Figura 4.6.: Funzione di densità di probabilità teorica (in rosso) ed empirica

5. Altri metodi per variabili casuali discrete e continue

5.1. Metodi plug-in

5.1.1. Metodo plug-in per la bernoulliana

Come visto in precedenza, è possibile ottenere una sequenza di variabili casuali bernoulliane generando una variabile casuale avente distribuzione geometrica.
Sia ad esempio Y una variabile casuale geometrica, e valga $Y = i$ (successo all'i-esimo tentativo). La sequenza di variabili casuali bernoulliane è composta da $i - 1$ valori 0 (e quindi $i - 1$ fallimenti), e da un valore 1 (il successo all'i-esima prova).

5.1.2. Metodo plug-in per la binomiale

Come visto in precedenza, è possibile simulare una binomiale a partire da una bernoulliana, poichè date le variabili casuali $X_1, \ldots, X_n \overset{\text{iid}}{\sim} \mathrm{B}(p)$, allora

$$\sum_{i=1}^{n} X_i \sim \mathrm{Bin}(n, p)$$

5.1.3. Metodo plug-in per la poissoniana

Ricordiamo che un processo di Poisson con tasso λ genera eventi con tempi di interarrivo esponenziali con tasso λ.
$N(1)$, il numero di eventi al tempo 1, ha distribuzione poissoniana con media λ.
Se chiamiamo $X_1, X_2, \ldots$ i tempi di interarrivo, allora l'n-esimo evento avverrà al tempo $\sum_{i=1}^{n} X_i$, e quindi il numero di eventi al tempo 1 può essere espresso come

$$N(1) = Max\left\{n\colon \sum_{i=1}^{n} X_i \leq 1\right\}$$

Possiamo riscrivere $N(1)$ esplicitando che i tempi di interarrivo seguono una distribuzione esponenziale e possono essere generati a partire da numeri casuali uniformi in (0,1):

$$N(1) = Max\left\{n\colon \sum_{i=1}^{n} -\frac{1}{\lambda}\ln U_i \leq 1\right\} = Max\left\{n\colon \prod_{i=1}^{n} U_i \geq \mathrm{e}^{-\lambda}\right\}$$

A questo punto è chiaro che possiamo simulare una variabile casuale di Poisson (in questo caso $N(1)$) di media λ generando continuamente numeri casuali finchè il loro prodotto supera il valore $\mathrm{e}^{-\lambda}$, e quindi restituire il numero di numeri casuali generati meno uno:

$$N = Min\left\{n\colon \prod_{i=1}^{n} X_i < \mathrm{e}^{-\lambda}\right\} - 1$$

5.1.4. Metodo plug-in per la chi-quadro

Possiamo simulare una variabile casuale Y con distribuzione $\chi^2(k)$ (chi-quadro con k gradi di libertà) usando la normale standard, poichè vale

$$Y = \sum_{i=1}^{k} X_i^2$$

dove $X_i \sim \mathrm{N}(0,\ 1)$.
Proviamo ad utilizzare Scilab per generare una variabile casuale con distribuzione $\chi^2(4)$. Procediamo in questo modo: generiamo $m = 4$ variabili casuali normali standard usando il risultato del teorema centrale (a questo scopo, approssimiamo ciascuna normale con $n = 1000$ bernulliane). Successivamente, usiamo la formula data sopra per generare $M = 1000$ variabili casuali chi-quadro.

```
function [y] = normale_approx(n,m)
  for i=[1:m],
    u = grand(n,1,'def');
    y($+1) = (sum(u) - n/2) * sqrt(12/n);
  end
endfunction

function [y] = chi2(k,M)
  for i=[1:M]
    nor = normale_approx(1000,k);
    y(i) = sum(nor.^2)
  end
endfunction

dati = chi2(4,1000);
```

Disegnamo la funzione di distribuzione teorica (in rosso) ed empirica (5.1)

```
// funzione di distribuzione empirica
dati = gsort(dati,'g','i');
m = size(dati,'r');
plot2d(dati,[1:m]/m)
```

```
// funzione di distribuzione teorica
x = [0:.1:25];
size_x = size(x,'c');
plot2d(x,cdfchi("PQ",x,ones(1,size_x)*4),style=5)
```

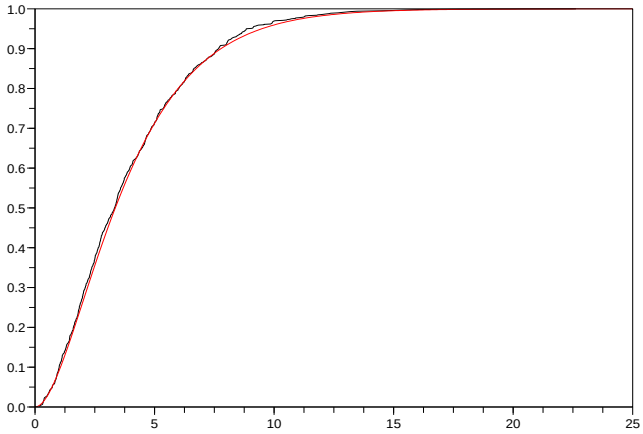

Figura 5.1.: Funzione di distribuzione teorica (in rosso) ed empirica

e la funzione di massa di probabilità teorica (in rosso) ed empirica (5.2)

```
// funzione di massa di probabilità empirica
set("current_figure",1)
Max = max(dati);
Min = min(dati);
histplot([Min:(Max-Min)/30:Max],dati)

// funzione di massa di probabilità teorica
function [f] = fC(x,nu)
  f = x.^(nu/2-1).*exp(-x/2) ./ (2.^(nu/2).*gamma(nu./2))
endfunction

tt = [Min:0.1:Max];
ff = fC(tt,4);
plot2d(tt,ff,style=5)
p = get("hdl");
p.children.thickness = 2;
```

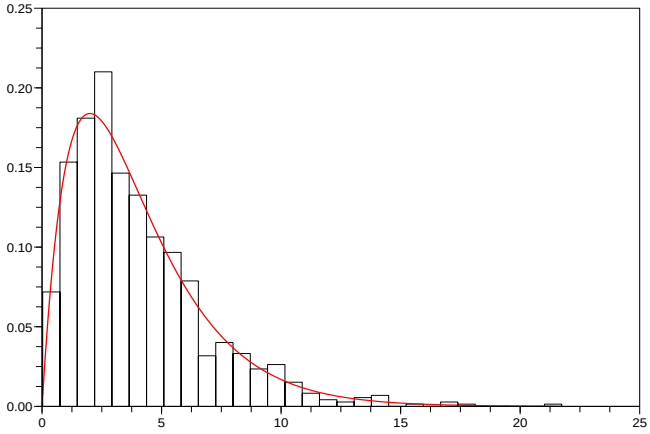

Figura 5.2.: Funzione di massa di probabilità teorica (in rosso) ed empirica

5.1.5. Metodo composizionale

Supponiamo di saper simulare due funzioni di massa di probabilità $\left\{p_j^{(1)}, j \geq 0\right\}$ e $\left\{p_j^{(2)}, j \geq 0\right\}$, e che vogliamo simulare il valore di una variabile casuale con funzione di massa di probabilità

$$\mathrm{P}\left(X = j\right) = \alpha p_j^{(1)} + (1-\alpha) p_j^{(2)}, \quad j \geq 0$$

dove $0 < \alpha < 1$.
X può essere pensata come composizione di X_1 e X_2, di massa di probabilità (rispettivamente) $p_j^{(1)}$ e $p_j^{(2)}$, e può quindi essere ottenuta generando un numero casuale U e poi ponendo $X = X_1$ se $U \geq \alpha$ e $X = X_2$ se $U < \alpha$.

In Scilab, proviamo a generare la variabile casuale con funzione di massa di probabilità composta

$$\mathrm{P}\left(X = j\right) = 0.3 p_j^{(1)} + 0.6 p_j^{(2)} + 0.1 p_j^{(3)}$$

dove $p_j^{(i)}$, con $i = 1, 2, 3$, sono funzioni di densità di probabilità normali di media $1, 2, 3$ e deviazione standard $.1, .2, .1$:

```
n = 3;
medie = [1,2,3];
devstd = [.1,.2,.1];
pesi = [.3,.6,.1];
```

```
// approssima la normale con n bernoulliane
function [y] = normale_approx(n)
  u = grand(n,1,'def');
  y = (sum(u) - n/2) * sqrt(12/n);
endfunction

// simula una variabile casuale normale standard
function [y] = simZ()
  y = normale_approx(20);
endfunction

// simula una variabile casuale normale
function [y] = simNormal(mu,sigma)
  y = mu + sigma * simZ();
endfunction

// simula la varibabile casuale mediante composizione di normali
function [y] = simMix()
  u = grand(1,1,'def');
  cm = pesi(1);
  i = 1;
  while (cm < u),
    i=i+1;
    cm=cm+pesi(i);
  end
  y = simNormal(medie(i),devstd(i));
endfunction

// ripete n volte la funzione g
function [y] = repeat(n,g)
  for i=[1:n]
    y($+1) = g();
  end
endfunction

m = 1000;
dati = repeat(m,simMix);
```

Disegnamo la funzione di ripartizione empirica (5.3)

```
dati = gsort(dati,'g','i');
m = size(dati,'r');
plot2d(dati,[1:m]/m)
```

e la funzione di densità di probabilità empirica (5.4)

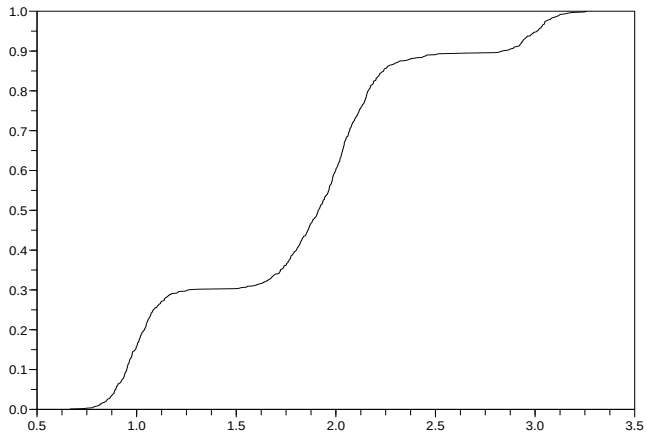

Figura 5.3.: Funzione di ripartizione empirica

```
Max = max(dati);
Min = min(dati);
set("current_figure",1)
histplot([Min:(Max-Min)*10/m:Max],dati)
```

5.2. Metodo di acceptance-rejection

Il metodo di acceptance-rejection consente di generare variabili casuali discrete o continue. Descriviamo il metodo considerando il caso discreto, in seguito vedremo esempi con variabili casuali sia discrete che continue.
Supponiamo di avere un metodo efficiente per simulare una variabile casuale Y con funzione di massa di probabilità q_j, con $j \geq 0$. Possiamo simulare una variabile casuale discreta X con funzione di massa di probabilità p_j, con $j \geq 0$, simulando Y ed accettando il valore simulato con una probabilità proporzionale a $\frac{p_Y}{q_Y}$.

Dimostriamo ora che l'algoritmo 10 genera effettivamente una variabile casuale $\mathrm{P}(X = j) = p_j$, con $j \geq 0$, e che il numero di iterazioni necessarie a generare X è una variabile casuale geometrica di media c.
Determiniamo innanzitutto la probabilità che una singola iterazione restituisca il valore j:

$$\mathrm{P}(Y = j, \text{accettato}) = \mathrm{P}(Y = j)\,\mathrm{P}(\text{accettato}|Y = j) = q_j \frac{p_j}{cq_j} = \frac{p_j}{c}$$

Sommando su j si ottiene la probabilità che la variabile casuale generata venga accet-

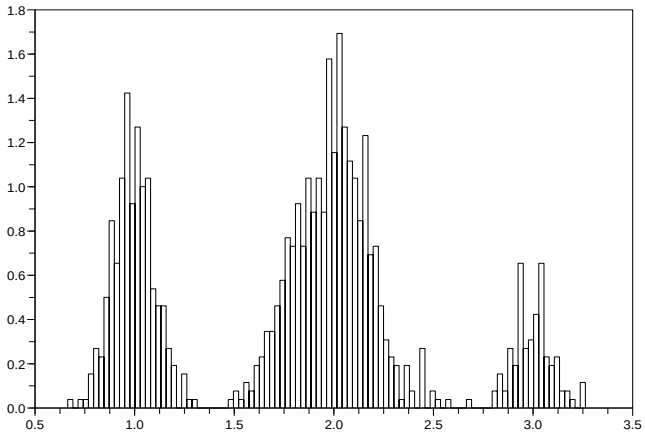

Figura 5.4.: Funzione di densità di probabilità

Algorithm 10 Metodo di acceptance-rejection

Require: p_j e q_j, con $j \geq 0$
Ensure: una variabile casuale con massa di probabilità p_j
  simula Y, avente massa di probabilità q_j
  loop
    genera un numero casuale U
    if $U < \dfrac{p_Y}{cq_Y}$ **then**
      $X = Y$, stop
    end if
  end loop

tata:

$$\mathrm{P}\left(\text{accettato}\right) = \sum_j \frac{p_j}{c} = \frac{1}{c}$$

Quindi è evidente che sono necessari c cicli perchè la variabile casuale generata venga accettata. Inoltre

$$\mathrm{P}\left(X = j\right) = \sum_n \mathrm{P}\left(j \text{ accettato all'}n\text{-esima iterazione}\right) = \sum_n \left(1 - \frac{1}{c}\right)^{n-1} \frac{p_j}{c} = p_j$$

5.2.1. La variabile casuale logaritmica

Vediamo ad esempio come simulare con questo metodo una variabile casuale con distribuzione logaritmica

$$\mathrm{f}_X\left(x\right) = \frac{(1-p)^x}{-\log(p)x} I_{N\setminus\{0\}}(x)$$

a partire da una distribuzione geometrica:

$$\mathrm{f}_X\left(x\right) = (1-p)^{(x-1)} p I_{N\setminus\{0\}}(x)$$

Riscriviamo la densità di probabilità logaritmica

$$p_X = \frac{(1-p)^x}{-\log(p)x} = \frac{1}{-\log(p)}(1-p)^{x-1}\frac{1-p}{x}$$

Se $p \geq 1/2$, allora quanto sopra è minore o uguale a

$$\frac{1}{-\log(p)}(1-p)^{x-1}p$$

che può quindi essere considerato cq_X. La condizione di accept sarà quindi

$$\frac{p_X}{cq_X} = \frac{1-p}{xp}$$

Vediamo come applicare in Scilab il risultato appena ottenuto per generare una variabile casuale logaritmica:

```
clear;

function [y] = geometrica(p)
  y = ceil(log(grand(1,1,'def')) / log(1-p))
endfunction

function [y] = logaritmica(p)
  exit = %F;
  while (~ exit),
```

```
    u = grand(1,1,'def');
    y = geometrica(p);
    exit = (u <= (1 - p)/(p * y));
  end
endfunction

function [y] = repeat(n,g,p1)
  for i=[1:n]
    y($+1) = g(p1);
  end
endfunction

m = 1000;
p = .6;
t = repeat(m,logaritmica,p);
```

Disegnamo la funzione di densità di probabilità teorica (i cui punti sono contrassegnati con delle X) e la funzione di densità di probabilità empirica:

```
// funzione di densità empirica
histplot([min(t):max(t)]-1,t)

// funzione di densità teorica
deff('[y] = Logaritmica(x)','y = (1-p)^x / (-log(p)*x)')
fplot2d([min(t):max(t)],Logaritmica,style=-2)
```

5.2.2. La variabile casuale Beta

Vediamo ora come simulare una variabile casuale con distribuzione Beta

$$g(x) = \frac{1}{B(a,b)} x^{a-i}(1-x)^{b-1} I_{(0,1)}(x)$$

in due modi possibili, cioè partendo da due differenti distribuzioni.
Iniziamo con una distribuzione uniforme, la cui densità in $(0,1)$ (dominio della Beta) è $f(x) = 1$.
Per determinare la costante c, calcoliamo il massimo valore del rapporto $\frac{f(x)}{g(x)}$:

1. calcolo la derivata prima

$$\frac{1}{B(a,b)} \left[(a-1)x^{a-2}(1-x)^{b-1} - x^{a-1}(b-1)(1-x)^{b-2}\right]$$

2. trovo il punto in cui si annulla

$$x_0 = \frac{a-1}{a+b-2}$$

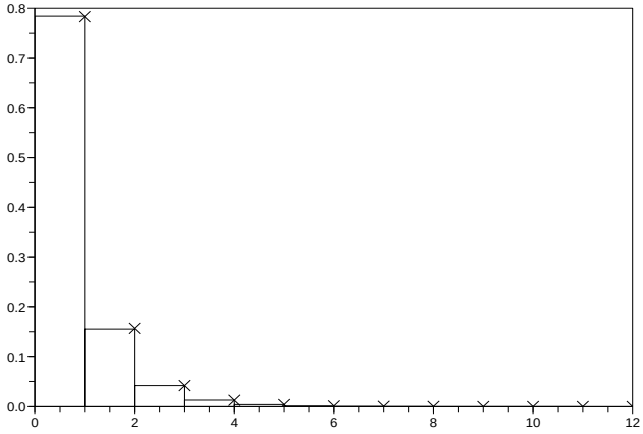

Figura 5.5.: Funzione di densità di probabilità teorica (le croci) ed empirica

3. calcolo il valore del rapporto $\frac{f(x)}{g(x)}$ nel punto x_0:

$$\frac{1}{B(a,b)}\left[\left(\frac{a-1}{a+b-2}\right)^{a-1}\left(\frac{b-1}{a+b-2}\right)^{b-1}\right]$$

Poniamo c uguale al valore ottenuto.
Posso ora calcolare il valore $\frac{f(x)}{cg(x)}$ da inserire nella condizione di accept:

$$\frac{f(x)}{cg(x)} = \left[\left(y\frac{a+b-2}{a-1}\right)^{a-1}\left((1-y)\frac{a+b-2}{b-1}\right)^{b-1}\right]$$

Applichiamo in Scilab il risultato appena ottenuto:

```
clear;

function [y] = Beta()
  cont = %T;
  while cont,
    u1 = grand(1,1,'def');
    u2 = grand(1,1,'def');
    fattore_1 = (u1 * (a+b- 2) / (a-1))^(a-1);
    fattore_2 = ((1-u1) * (a+b-2) / (b-1))^(b-1);
    cont= ~(u2 <= (fattore_1 * fattore_2));
```

```
  end;
  y = u1;
endfunction

function [y] = repeat(n,g)
  for i=[1:n]
    y($+1) = g();
  end
endfunction

m = 10000;
a = 2;
b = 5;

t = repeat(m,Beta);
```

Possiamo quindi disegnare la funzione di densità di probabilità teorica (in rosso) ed empirica (5.6)

```
// funzione di densità empirica
x = [floor(min(t)):.01:ceil(max(t))];
histplot(x,t)

// funzione di densità teorica
function [bb] = bet(x,a,b)
  bb = (gamma(a+b)/(gamma(a)*gamma(b)))*x.^(a-1).*(1-x).^(b-1)
endfunction

plot2d(x,bet(x,2,5),style=5);
p = get("hdl");
p.children.thickness = 2;
```

Possiamo migliorare l'algoritmo scegliendo $f(x)$ in modo che c e la condizione di accept siano più semplici. Consideriamo ad esempio

$$f(x) = ax^{a-1} I_{(0,1)}(x)$$

Notiamo innanzitutto che possiamo facilmente simulare la variabile casuale avente la funzione di densità $f(x)$ usando il metodo di inversione:

$$\mathrm{F}(x) = \int_0^x az^{a-1}\,\mathrm{d}z = x^a \implies x = u^{\frac{1}{a}}$$

Calcoliamo ora c come il massimo del rapporto $\dfrac{f(x)}{g(x)}$:

$$max\left(\frac{f(x)}{g(x)}\right) = \frac{1}{bB(a,b)}$$

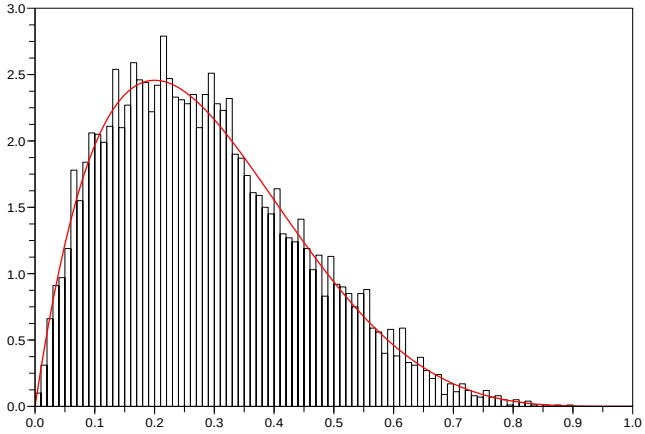

Figura 5.6.: Funzione di densità di probabilità teorica (in rosso) ed empirica

La condizione di accept risultante,

$$\frac{f(x)}{cg(x)} = (1-x)^{b-1}$$

è evidentemente differente dalla precedente e probabilmente più semplice dal punto di vista computazionale (a dimostrazione di questa affermazione, si possono confrontare i valori di c delle due condizioni di accept).

Generiamo in Scilab una variabile casuale logaritmica usando questo secondo risultato:

```
clear;

function [out] = Beta2()
  cont = %T;
  while cont,
    u1 = grand(1,1,'def');
    y = u1^(1/a);
    u2 = grand(1,1,'def');
    cont= ~(u2 <= (1-y)^(b-1));
  end;
  out = y;
endfunction

function [y] = repeat(n,g)
```

```
  for i=[1:n]
    y($+1) = g();
  end
endfunction

m = 10000;
a = 4.9;
b = 1.5;

t2 = repeat(m,Beta2);
```

Anche in questo caso possiamo disegnare la funzione di densità di probabilità teorica (in rosso) ed empirica (5.7)

```
// funzione di densità empirica
x = [floor(min(t2)):.01:ceil(max(t2))];
histplot(x,t2)

// funzione di densità teorica
function [bb] = bet(x,a,b)
  bb = (gamma(a+b)/(gamma(a)*gamma(b)))*x.^(a-1).*(1-x).^(b-1)
endfunction

plot2d(x,bet(x,4.9,1.5),style=5);
p = get("hdl");
p.children.thickness = 2;
```

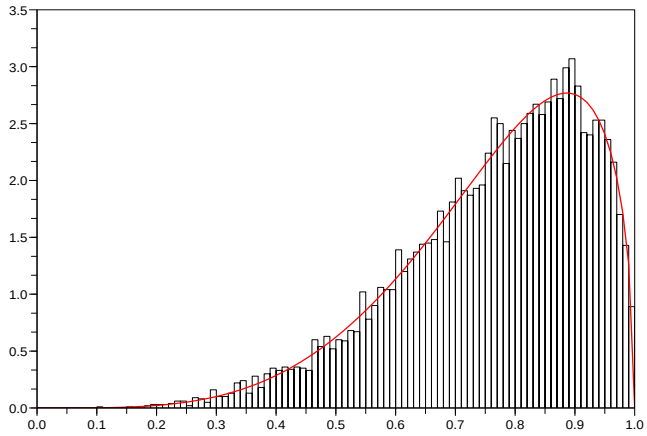

Figura 5.7.: Funzione di densità di probabilità teorica (in rosso) ed empirica

6. Simulazione di sistemi complessi

Nei precedenti capitoli abbiamo presentato come è possibile generare variabili casuali di distribuzione arbitraria. Possiamo ora usare queste tecniche per simulare sistemi complessi, generando il meccanismo stocastico che li regola e tenendo traccia dei valori delle quantità a cui siamo interessati.
Uno dei paradigmi di simulazione più comuni è la **simulazione a eventi discreti**, i cui elementi chiave sono gli eventi e le variabili.
Le variabili, ad esempio il tempo e i contatori delle quantità d'interesse, vengono aggiornate ogni volta che accade uno degli eventi definito nel modello, ed in questo modo seguono lo sviluppo del sistema nel tempo.

6.1. Esempio: una coda in farmacia

Come esempio di simulazione di un sistema con un solo servitore, simuliamo una coda in farmacia (algoritmo 11), dove l'arrivo dei clienti segue una legge di distribuzione esponenziale e il tempo che il farmacista dedica a soddisfare la richiesta di un cliente segue una legge di distribuzione normale.

- t = tempo
- $T(i)$ = tempo di arrivo del cliente i-esimo
- T_{max} = tempo di chiusura della farmacia
- N_A = numero di arrivi nel sistema al tempo corrente
- N_D = numero di partenze dal sistema al tempo corrente
- t_A = tempo di arrivo del prossimo cliente
- t_D = tempo di partenza del cliente in coda
- n numero di clienti nel sistema (clienti in coda + cliente in servizio) al tempo corrente

In Scilab può essere realizzato così:

```
function [y] = generaTempo()
  lambda = .2;
  y= -1/lambda * log(grand(1,1,'def'));
endfunction
```

Algorithm 11 Simulazione di una coda in farmacia

{inizializzazione}
$t = N_A = N_D = 0$
$T(0) =$ generaTempo()
$t_D = \infty$ {non ci sono clienti in coda}
if $t_A \leq t_D$ e $t_A \leq T_{max}$ **then**
  $t = t_A$
  N_A++
  $n++$
  $T(N_A) =$ generaTempo()
  $t_A = T(N_A)$
  if n==1 **then**
    $y =$ generaServizio()
    $t_D = t + y$
  end if
end if
if $t_D > t_A$ e $t_D \leq T_{max}$ **then**
  $t = t_D$
  $n--$
  N_D++
  if $n == 0$ **then**
    $t_D = \infty$
  else
    $y =$generaServizio()
    $t_D = t + y$
  end if
  if $min(t_A, t_D) > T_{max}$ e $n > 0$ **then**
    $t = t_D$
    $n--$
    N_D++
    if $n == 0$ **then**
      $t_D = \infty$
    else
      $y =$generaServizio()
      $t_D = t + y$
    end if
  end if
  if $min(t_A, t_D) > T_{max}$ e $n == 0$ **then**
    $T_p = max(t - T_{max}, 0)$
    exit()
  end if
end if

```
function [y] = generaServizio()
  mu = .5;
  sigma = .1;
  y = mu + sigma * (sum(grand(12,1,'def')) - 6);
endfunction

Tmax = 10;

function [A,D,Tp] = simulaCoda()
  tc = 0;                // tempo corrente
  NA = 0;                // arrivi nel tempo corrente
  ND = 0;                // partenze nel tempo corrente
  n = 0;                 // clienti nel sistema al tempo corrente
  T = [];                // tempo di arrivo del cliente i-esimo
  T(1) = generaTempo();
  tA = T(1);             // tempo di arrivo del prossimo cliente
  tD = %inf;             // tempo di partenza del prossimo cliente
  exit = %F;
  while (~ exit),
    // un cliente si mette in coda
    if (tA <= tD & tA < Tmax),
      tc = tA;
      NA = NA + 1;
      n = n + 1;
      T(NA) = tc + generaTempo();
      tA = T(NA);
      if (n == 1),
        Y = generaServizio();
        tD = tc + Y;
      end;
      A(NA) = tc;
    end;
    // un cliente lascia la farmacia
    if (tD < tA & tD <= Tmax)
      tc = tD;
      n = n - 1;
      ND = ND + 1;
      D(ND) = tc;
      if (n == 0)
        tD = %inf;
      else
        Y = generaServizio();
        tD = tc + Y;
      end;
```

```
    end;
    // gestione dei clienti ancora in farmacia
    // al termine dell'orario di lavoro regolare
    if (min(tA,tD) > Tmax & n>0)
      tc = tD;
      n = n - 1;
      ND = ND + 1;
      D(ND) = tc;
      if (n > 0)
        Y = generaServizio();
        tD = tc + Y;
      end;
    end;
    // svuotata la coda, calcolo dei minuti di straordinario
    if (min(tA,tD) > Tmax & n == 0)
      Tp = max(tc-Tmax,0);
      exit = %T;
    end;
  end;
endfunction;

[A,D,Tp] = simulaCoda();
```

In questo caso, in output otteniamo il vettore degli orari di arrivo `A` e di uscita `B` dei clienti, e i minuti di lavoro straordinario `Tp` eventualmente necessari a soddisfare le richieste dei clienti ancora in farmacia al termine dell'orario di lavoro regolare.
Il modello può essere ulteriormente arricchito e complicato considerando più servitori, più servizi e quindi più code.

6.2. Esempio: un'assicurazione contro i rischi

Simuliamo ora un'assicurazione contro i rischi. Scopo della simulazione è sapere se dopo un tempo t l'assicurazione è fallita oppure no.
Definiamo alcune variabili:

- n = numero clienti corrente;
- n_0 = numero clienti iniziale;
- a = capitale corrente;
- a_0 = capitale iniziale;
- $\mathrm{P}(\nu)$ = nuovi clienti (variabila casuale con distribuzione poissoniana);
- $\mathrm{P}(\mu)$ = clienti in uscita (variabila casuale con distribuzione poissoniana);

- $\mathrm{P}(\lambda)$ = risarcimenti (variabila casuale con distribuzione poissoniana);
- F = entità del risarcimento (variabile casuale nota con distribuzione normale);
- c = euro pagati da ogni cliente, per unità di tempo.

Definiamo ora gli eventi:

- viene guadagnato un nuovo cliente;
- un cliente viene perso;
- viene effettuato un risarcimento.

Si assume inoltre che

- tutte le richieste di risarcimento vengono soddisfatte;
- gli eventi coinvolti sono indipendenti.

L'intertempo tra ciascun tipo di evento segue una legge di probabilità esponenziale. L'evento successivo viene definito come il valore minimo tra gli intertempi, e segue anch'esso una legge esponenziale: infatti, dati $X_1, X_2, \ldots, X_n$ esponenziali di parametro λ_i e $M = min_{i=1,2,\ldots,n}\{X_i\}$, allora M ha la seguente funzione di ripartizione esponenziale:

$$\mathrm{F}_M(x) = 1 - \mathrm{e}^{-\sum_{i=1}^{n} \lambda_i x}$$

Definiamo ora la probabilità che si verifichi uno dei tre tipi di eventi:

- nuovo cliente:
$$\frac{\nu}{\nu + n\mu + n\lambda}$$
- cliente in uscita:
$$\frac{n\mu}{\nu + n\mu + n\lambda}$$
- un risarcimento:
$$\frac{n\lambda}{\nu + n\mu + n\lambda}$$

Si può ora simulare l'andamento del sistema con l'algoritmo 12:

In Scilab, la simulazione avviene in questo modo:

```
function [y] = generaTempo(nu,mu,lambda,n)
  param = nu + n*mu + n*lambda;
  y = -1/param * log(grand(1,1,'def'));
endfunction

function [y] = generaEvento(nu,mu,lambda,n)
  u = grand(1,1,'def');
  if (u < nu/(nu + n*mu + n*lambda)),
```

Algorithm 12 Simulazione di un'assicurazione contro i rischi

```
loop
  {inizializzazione}
  t = 0, a = 0, n = n_0
  t_E = simulaEvento(X); {X è esponenziale con parametro ν + nμ + nλ}
  if t_E > T_max then
    out = 1, exit()
  else
    a = a + nc(t_E − t)
    t = t_E
    i =simula(I) {I = {1, 2, 3} con probabilità di ciascun evento}
    if i == 1 then
      n + +
    else
      if i == 2 then
        n − −;
      else
        if i == 3 then
          y = simula(Y) {Y è l'entità del risarcimento, distribuita come F}
          if a > y then
            a = a − y
          else
            out = 0, exit()
          end if
        end if
      end if
    end if
    t_E = simula(X);
  end if
end loop
```

```
    y = 1;
  elseif (u < (nu + n*mu)/(nu + n*mu + n*lambda)),
    y = 2;
  else
    y = 3;
  end;
endfunction

function [Y] = generaRisarcimento()
  Y = grand(1,1,'nor',5,3);
endfunction

Tmax = 100;

tc = 0;          // tempo corrente
a = 10;          // capitale iniziale
n = 10;          // numero clienti iniziale
c = .01;         // tasso di pagamento del premio
nu = 1;          // tasso di arrivo dei nuovi clienti
mu = 1;          // tasso di abbandono dei clienti
lambda = .3;     // tasso di richieste di risarcimento
I = 0;           // 1 se non c'è stato fallimento entro Tmax
// generiamo il tempo del primo evento
tE = generaTempo(nu,mu,lambda,n);
exit = %F;
while (~ exit),
  // il prossimo evento è fuori tempo massimo
  if (tE > Tmax)
    I = 1;
    exit = %T;
  // il prossimo evento si verifica durante il periodo di simulazione
  else
    a = a + n*c*(tE - tc);
    tc = tE;
    j = generaEvento(nu,mu,lambda,n);
    // evento generato: nuovo cliente
    if (j == 1)
      n = n + 1;
    // evento generato: cliente abbandona
    elseif (j == 2)
      n = n - 1;
    // evento generato: risarcimento
    else
      Y = generaRisarcimento();
      // entità risarcimento eccede capitale?
```

```
      if (Y > a)
        // sì: l'assicurazione fallisce
        I = 0;
        exit = %T;
      else
        // no: il capitale viene decrementato dell'entità del risarcimento
        a = a - I;
      end;
    end;
  end;
  // generiamo il tempo del prossimo evento
  tE = tc + generaTempo(nu,mu,lambda,n);
end;
I // controlliamo infine se l'assicurazione non è fallita (I=1)
```

7. Analisi statistiche di dati provenienti da sistemi simulati

7.1. Stima puntuale

Date n variabili casuali $X_1, \ldots, X_n \overset{\text{iid}}{\sim} X$, vogliamo stimare una funzione

$$\theta = g(x_1, \ldots, x_n)$$

di X. Poniamo ad esempio di essere interessati alla media di X

$$\theta = \mathrm{E}[X]$$

Come **stimatore** possiamo utilizzare la media campionaria

$$\overline{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

Poichè vale $\mathrm{E}\left[\overline{X}\right] = \theta$, lo stimatore si dice **non distorto**.
Per verificare la bontà dello stimatore, possiamo calcolarne l'errore quadratico medio. In generale, se uno stimatore è non distorto il suo errore quadratico medio è uguale alla sua varianza. A verifica che la media campionaria è uno stimatore non distorto di $\mathrm{E}\left[\overline{X}\right]$, calcoliamone l'errore quadratico medio:

$$MSE = \mathrm{E}\left[(\overline{X} - \theta)^2\right] = \frac{\sigma^2}{n} = \mathrm{var}\left[\overline{X}\right]$$

Supponiamo ora di volere una stima affetta da un errore quadratico medio non superiore ad una quantità d: se è nota la varianza σ^2 posso scegliere n in modo opportuno, tale che

$$n = \frac{\sigma^2}{d}$$

Posso inoltre sfruttare la disuguaglianza di Chebyshev

$$\mathrm{P}\left(\mid \overline{X} - \theta \mid \geq k \frac{\sigma}{\sqrt{n}} \right) \leq \frac{1}{k^2}$$

per sapere con che probabilità massima la stima $\overline{X}$ si discosta da θ di una quantità maggiore di $k\frac{\sigma}{\sqrt{n}}$.

Possiamo altrimenti ottenere un risultato asintotico usando il teorema del limite centrale, che ricordiamo velocemente:

$$Z = \frac{\overline{X}_n - \mu}{\sigma/\sqrt{n}}$$

dove $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$, tende ad assumere una distribuzione di probabilità normale standard per $n \to \infty$.
In particolare, dalla disuguaglianza di Chebyshev si ricava che

$$\mathrm{P}\left(\mid \overline{X} - \theta \mid \geq k\frac{\sigma}{\sqrt{n}}\right) = \mathrm{P}\left(\left|\frac{\overline{X} - \theta}{\frac{\sigma}{\sqrt{n}}}\right| \geq k\right) = \mathrm{P}\left(|Z| \geq k\right) = 2(1 - \Theta(k))$$

In un problema di simulazione, la varianza σ^2 non è generalmente nota e va quindi anch'essa stimata. Come stimatore potremmo usare

$$\frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2$$

ma è distorto, quindi è preferibile usare lo stimatore non distorto S^2 (**varianza campionaria**)

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$$

Nelle formule precedenti è quindi possibile sostituire la deviazione standard σ con la sua stima non distorta S.

Il procedimento completo di simulazione è il seguente:

1. scegliamo k in modo che $2(1 - \Theta(k))$ sia sufficientemente piccolo
2. scegliamo l'errore di approssimazione accettabile d
3. simuliamo il sistema n volte ottenendo $x_1, \ldots, x_n$ (di solito $n = 100$)
4. continuiamo a simulare il sistema ottenendo x_i
5. calcoliamo $\overline{X} = \frac{1}{i}\sum_{j=1}^{i} X_j$ e $S^2 = \frac{1}{i-1}\sum_{j=1}^{i}(X_j - \overline{X})^2$
6. se $\frac{kS}{\sqrt{i}} \geq d$, allora incrementiamo i e ripartiamo dal punto 4
7. il valore della stima ottenuta è $\overline{x}$, con probabilità minore di $2(1 - \Theta(k))$ di aver commesso un errore maggiore di d.

Durante la simulazione, viene utile poter calcolare media e varianza campionaria in maniera ricorsiva con le seguenti formule:

$$\overline{X}_{n+1} = \frac{n\overline{X}_n + X_{n+1}}{n+1}$$

$$S^2_{n+1} = \left(1 - \frac{1}{n}\right) S^2_n + \frac{1}{n+1}(X_{n+1} - \overline{X}_n)^2$$

con valori iniziali $\overline{X}_0 = 0$ e $S^2_0 = 0$.

In Scilab, possiamo mostrare con un esempio la validità del risultato asintotico derivante dal teorema del limite centrale:

1. generiamo 1000 variabili casuali esponenziali;
2. ne standardizziamo la somma, ottenendo una specificazione di variabile casuale normale standard;
3. ripetiamo il procedimento 1000 volte;
4. disegniamo la funzione di ripartizione e di densità, confrontando i grafici empirici dei valori ottenuti con quelli teorici di una normale standard.

```
function [y] = repeat(n,g)
  for i=[1:n]
    y($+1) = g();
  end
endfunction

function [y] = genera()
  y = (sum(grand(n,1,'exp',1/lambda))/n -exm) / stdevxm;
endfunction

n = 1000;
m = 1000;
lambda = 3;
exm = 1/lambda;
stdevxm = 1/(lambda * sqrt(n));
dati = repeat(m,genera);
datis = gsort(dati,'g','i');
```

Disegnamo ora la funzione di ripartizione teorica (in rosso) ed empirica (7.1)

```
// funzione di ripartizione empirica
plot2d(datis,[1:m]/m)

// funzione di ripartizione teorica
```

```
x = [-4:.1:4];
size_x = size(x,'c');
plot2d(x,cdfnor("PQ",x,zeros(1,size_x),ones(1,size_x)),style=5)
```

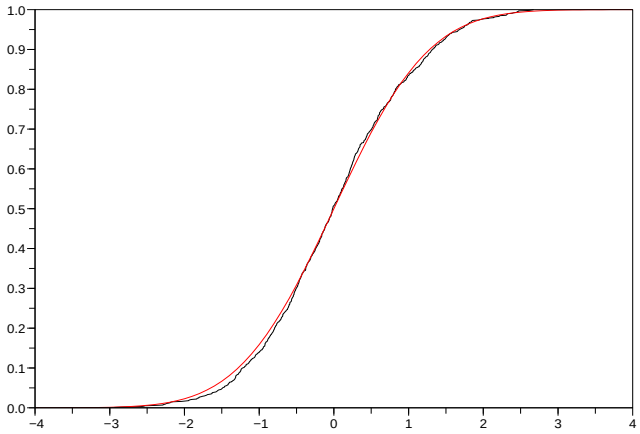

Figura 7.1.: Funzione di ripartizione teorica (in rosso) ed empirica

e la funzione di densità di probabilità teorica (in rosso) ed empirica (7.2)

```
// funzione di densità empirica
histplot([min(datis):100/m:max(datis)],datis)

// funzione di densità teorica
function [f] = fN(x,mu,sigma)
  f = exp(-(x-mu)^2/(2*sigma^2)) / (sigma*sqrt(2*%pi))
endfunction

ff = fN(x,0,1);
plot2d(x,ff,style=5);
p = get("hdl");
p.children.thickness = 2;
```

7.2. Stime per intervalli di confidenza

Piuttosto che restituire un valore di stima approssimato, in alcuni casi può essere più utile specificare un intervallo a cui la funzione da stimare θ appartiene con un certo

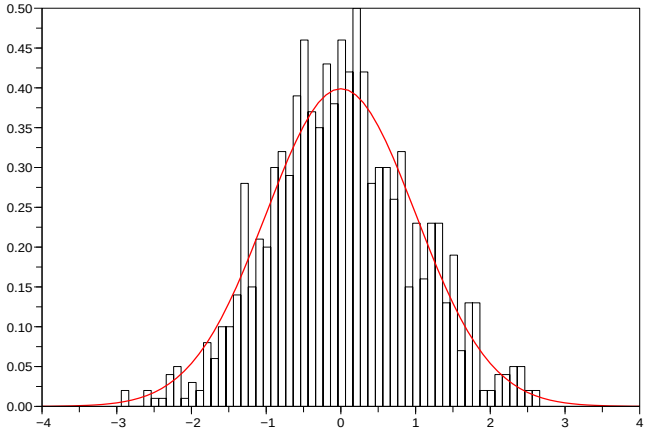

Figura 7.2.: Funzione di densità di probabilità teorica (in rosso) ed empirica

grado di confidenza.
Innanzitutto è possibile dimostrare che nel teorema del limite centrale

$$\frac{\overline{X} - \theta}{\sigma/\sqrt{n}} \sim \mathrm{N}\,(0,\ 1) \quad \text{se } n \to \infty$$

possiamo sostituire la varianza campionaria S alla varianza σ senza invalidare il risultato.
Ora, $\forall \alpha \in (0,1)$, definiamo z_α in modo che valga

$$\mathrm{P}\,(Z > z_\alpha) = \alpha$$

Per simmetria della normale standard, vale $z_{1-\alpha} = -z_\alpha$, quindi

$$\mathrm{P}\left(-z_{\alpha/2} < Z < z_{\alpha/2}\right) = 1 - \alpha$$

Sostituendo $\dfrac{\overline{X} - \theta}{\sigma/\sqrt{n}}$ a Z e moltiplicando per -1, otteniamo

$$\mathrm{P}\left(\overline{X} - z_{\alpha/2}\frac{S}{\sqrt{n}} < \theta < \overline{X} + z_{\alpha/2}\frac{S}{\sqrt{n}}\right) \approx 1 - \alpha$$

Quindi possiamo affermare che il valore atteso da stimare θ si troverà nella regione

$$\overline{X} \pm z_{\alpha/2}\frac{S}{\sqrt{n}}$$

con probabilità $1 - \alpha$.

Il procedimento completo di stima per intervalli di confidenza è il seguente:

1. scelgo il valore di $\alpha \in (0,1)$
2. scelgo l'ampiezza l dell'intervallo di confidenza
3. simulo il sistema 100 volte ottenendo $x_1, \ldots, x_{100}$
4. $k = 1$
5. simulo il sistema ottenendo x_k
6. calcolo $\overline{X}$ e s
7. se $2z_{\alpha/2}\frac{s}{\sqrt{n}} > l$, allora incremento k e riprendo dal punto 5
8. la stima di θ è $\overline{x} \pm z_{\alpha/2}\frac{s}{\sqrt{n}}$

Il risultato non è esatto, ma asintoticamente corretto.
Per questioni di efficienza, il calcolo di $\overline{x}$ e s può essere effettuato ricorsivamente.

7.3. Tecniche bootstrap

Supponiamo di essere interessati a studiare una funzione di ripartizione F, e a stimare un suo parametro $\theta(F)$ per il quale è stato proposto uno stimatore $g(X_1, \ldots, X_m)$. Vogliamo verificarne la bontà stimando l'errore quadratico medio

$$MSE(F) = \mathrm{E}_F\left[(g(X_1, \ldots, X_m) - \theta(F))^2\right]$$

In alcuni casi il suo calcolo può essere semplice: se ad esempio la quantità da stimare è il valore atteso e lo stimatore non distorto è la media campionaria, come noto l'errore quadratico medio sarà $\frac{S^2}{n}$. In altri casi conviene invece usare una **tecnica bootstrap**:

1. estraggo uniformemente da un campione di X;
2. costruisco progressivamente la funzione di ripartizione empirica $\hat{F}(x)$, stima di $\mathrm{F}(X)$ a cui converge asintoticamente.

Poichè vale $MSE(F) \simeq MSE(\hat{F})$, posso calcolare l'errore quadratico medio usando la funzione di ripartizione empirica:

$$MSE(\hat{F}) = \mathrm{E}_{\hat{F}}\left[\left(g(X_1, \ldots, X_n) - \theta(\hat{F})\right)^2\right]$$

7.3.1. Esempio: tempo trascorso dal cliente in farmacia

In questo esempio vogliamo stimare nel lungo termine il tempo ω che un cliente passa mediamente in farmacia.
Essendo ω_i il tempo totale trascorso dall'i-esimo cliente nel sistema, siamo interessati a

$$\theta = \lim_{n \to +\infty} \frac{\omega_1 + \ldots + \omega_n}{n}$$

Poichè $\omega_1, \ldots, \omega_n$ non sono indipendenti né identicamente distribuite, dimostriamo innanzitutto che il limite esiste davvero.
Sia N_i il numero dei clienti nel giorno i, e $D_i = \omega_{N_{i-1}+1} + \ldots + \omega_{N_i}$ la somma dei tempi passati nel sistema dei clienti arrivati nel giorno i.
Possiamo esprimere la quantità di interesse θ, cioè il tempo passato dal cliente nel sistema, come

$$\theta = \lim_{m \to +\infty} \frac{D_1 + \ldots + D_m}{N_1 + \ldots + N_m} = \lim_{m \to +\infty} \frac{(D_1 + \ldots + D_m)/m}{(N_1 + \ldots + N_m)/m}$$

Essendo $D_1, \ldots, D_m$ indipendenti e identicamente distribuite, grazie alla legge dei grandi numeri sappiamo che la loro media convergerà con probabilità 1 al loro valore atteso comune (lo stesso discorso vale anche per N), e quindi possiamo scrivere:

$$\theta = \frac{\mathrm{E}[D]}{\mathrm{E}[N]}$$

Per ottenere la stima di θ possiamo simulare il sistema per k giorni, raccogliere i dati $(d_1, n_1), \ldots, (d_k, n_k)$, calcolarne la media campionaria $\mathrm{E}_{\hat{F}}[D] = \overline{d}$ e $\mathrm{E}_{\hat{F}}[N] = \overline{n}$ ed infine concludere che vale

$$\theta = \frac{\overline{d}}{\overline{n}}$$

Va osservato che la stima del tempo medio trascorso nel sistema dai clienti è calcolata soltanto con i dati dei primi k giorni.
Se vogliamo invece calcolare l'errore quadratico medio

$$MSE = \mathrm{E}\left[\left(\frac{\sum_{i=1}^{k} D_i}{\sum_{i=1}^{k} N_i} - \theta\right)^2\right]$$

conviene usare la tecnica bootstrap.
Consideriamo la funzione di massa di probabilità empirica uniforme sulle coppie (D, N):

$$\mathrm{P}(D = d_i, N = n_i) = \frac{1}{k}$$

In accordo a questa funzione di probabilità, possiamo estrarre delle coppie di valori e calcolarne così i due valori attesi

$$\mathrm{E}_{F_e}[D] = \overline{d}$$

e

$$\mathrm{E}_{F_e}[N] = \overline{n}$$

ed infine stimare il valore atteso a partire da questi dati:

$$\theta(F_e) = \frac{\overline{d}}{\overline{n}}$$

Calcoliamo ora l'errore quadratico medio

$$MSE(F_e) = \mathrm{E}_{F_e}\left[\left(\frac{\sum_{i=1}^{k} D_i}{\sum_{i=1}^{k} N_i} - \frac{\overline{d}}{\overline{n}}\right)^2\right]$$

Un calcolo esatto richiederebbe k^k somme. Possiamo allora approssimare il risultato estraendo in accordo alla funzione di distribuzione empirica coppie di vettori casuali D_i^1, N_i^1 e calcolare

$$Y_1 = \left(\frac{\sum_{i=1}^{k} D_i}{\sum_{i=1}^{k} N_i} - \frac{\overline{d}}{\overline{n}}\right)^2$$

Allo stesso modo possiamo calcolare $Y_2, \ldots, Y_r$ ($r = 100$ dovrebbe essere sufficiente), e con questi valori stimare quindi MSE_{F_e}, che a sua volta stima MSE, l'errore quadratico medio della quantità di tempo che un cliente passa mediamente nel sistema.

7.3.2. Esempio: stima bootstrap della varianza

Siano date $X_1, \ldots, X_n$ di varianza σ^2 non nota, stimata con la varianza campionaria

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2$$

Poniamo per semplicità che valga $n = 2$, $x_1 = 1$, $x_2 = 3$.
Ci interessa calcolare l'errore quadratico medio

$$MSE(F) \simeq MSE(\hat{F}) = \mathrm{E}_{\hat{F}}\left[\left(\frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2 - \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2\right)^2\right]$$

Cominciamo con l'osservare che vale

$$\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2 = \frac{1}{2}\left((1-2)^2 + (3-2)^2\right) = 1$$

mentre

$$\frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2$$

viene calcolato estraendo con probabilità uniforme $\frac{1}{4}$ una coppia di valori (X_1, X_2) da $\{x_1, x_2\}$, calcolandone la media $\overline{X}$ e quindi il valore atteso rispetto alla funzione di ripartizione empirica.
In questo modo stimo l'errore quadratico medio di uno stimatore usando un'altra stima (metodo bootstrap).

Possiamo utilizzare Scilab per fare un ulteriore esempio di calcolo della stima della varianza con il metodo bootstrap. Innanzitutto definiamo i dati su cui lavoreremo:

```
dati = [5,4,9,6,21,17,11,20,7,10,21,15,13,16,8];
n = size(dati,'c');
```

Definiamo poi una funzione che genera un nuovo vettore di dati, di pari dimensione, campionando con reimmissione

```
function [repl] = replica()
  repl = sample(n,dati)
endfunction
```

e una funzione che calcola la varianza

```
function [y] = Var(d)
  xm = sum(dati) / n;
  y= 1/n * sum((d-xm)^2);
endfunction
```

ed infine una funzione che calcola la varianza campionaria

```
function [y] = Var_camp(d)
  xm = sum(d) / n;
  y= 1/(n-1) * sum((d-xm)^2);
endfunction
```

Calcoliamo ora la varianza dei dati definiti in precedenza, questo sarà il valore che intendiamo approssimare:

```
-->var = Var(dati)
 var  =

    32.026667
```

In teoria potremmo estrarre $15^{15} = 437.893.890.380.859.375$ campioni con reimmissione dal vettore `dati`, ma il calcolo risultante sarebbe troppo laborioso. Consideriamo quindi solo 10000 campioni, calcoliamone la varianza campionaria ed infine applichiamo l'operatore di media:

```
function [y] = repeat(n,g,h)
  for i=[1:n]
    y($+1) = g(h());
```

```
  end
endfunction

r = 10000;
Y = mean(repeat(r,Var_camp,replica));
```

In questo modo, abbiamo ottenuto la nostra approssimazione della varianza mediante la tecnica bootstrap:

```
-->Y
 Y  =

    32.02689
```

8. Tecniche di riduzione della varianza

Finora abbiamo stimato $\mathrm{E}[X]$ con la media campionaria $\overline{X}$, stimatore non deviato di varianza $\frac{\sigma^2}{n}$. Studiamo ora alcune tecniche per ottenere stimatori con varianza ridotta.

8.1. Uso di variabili antitetiche

Date X_1 e X_2 identicamente distribuite con media μ, anche $\frac{X_1+X_2}{2}$ ha media μ e, in generale, varianza

$$\mathrm{var}\left[\frac{X_1+X_2}{2}\right] = \frac{1}{4}\left(\mathrm{var}[X_1] + \mathrm{var}[X_2] + 2\mathrm{cov}[X_1X_2]\right)$$

Se X_1 e X_2 sono correlate in maniera opportuna (antitetica), la covarianza assume valori negativi e quindi la varianza totale diminuisce.
Come esempio di variabili antitetiche, possiamo considerare

$$X_1 = h(u_1, \ldots, u_n) \quad \text{e} \quad X_2 = h(1-u_1, \ldots, 1-u_n)$$

Infatti, se h è monotona, X_2 cambia in maniera antitetica al variare di X_1 dalla quale è dipendente. Poichè U e $1-U$ hanno lo stesso valore atteso la media dello stimatore $\frac{X_1+X_2}{2}$ resta uguale, ma la sua varianza diminuisce.

Un altro esempio: poniamo di avere una variabile casuale normale Y di media μ e varianza σ^2. Possiamo usare il metodo delle variabili antitetiche considerando la variabile casuale

$$X = 2\mu - Y$$

che è identicamente distribuita a Y, in quanto la covarianza di X e Y sarà negativa come desiderato.

8.1.1. Simulazione della funzione affidabilità

Consideriamo un sistema di n componenti, ciascuna delle quali può funzionare o meno. Sia $s_i = 1$ se la componente i-esima funziona, 0 altrimenti, e chiamiamo $\boldsymbol{s} = (s_1, \ldots, s_n)$ il vettore di stato del sistema.
Supponiamo inoltre che esista una funzione non decrescente $\theta(s_1, \ldots, s_n)$ (funzione di struttura) con valore 1 se il sistema funziona nello stato $s_1, \ldots, s_n$, 0 altrimenti. Alcune comuni funzioni di struttura sono la struttura in serie (funziona se tutte le componenti funzionano)

$$\theta(s_1, \ldots, s_n) = Min_i(s_i)$$

la struttura in parallelo (funziona se almeno un componente funziona)

$$\theta(s_1, \ldots, s_n) = Max_i(s_i)$$

il sistema k-di-n (funziona se k componenti su un totale di n funzionano):

$$\theta(s_1, \ldots, s_n) = 1 \quad \text{se} \quad \sum_{i=1}^{n} s_i \geq k$$

e la struttura a ponte, ad esempio a cinque componenti

$$\theta(s_1, \ldots, s_5) = Max(s_1 s_3 s_5, s_2 s_3 s_4, s_1 s_4, s_2 s_5)$$

Supponiamo ora che gli stati S_i dei componenti siano variabili casuali indipendenti, e definiamo la funzione di affidabilità

$$r(p_1, \ldots, p_n) = \mathrm{P}\left(\theta(S_1, \ldots, S_n) = 1\right) = \mathrm{E}\left[\theta(S_1, \ldots, S_n)\right]$$

Per i sistemi in serie vale

$$r(p_1, \ldots, p_n) = \prod_{i=1}^{n} p_i$$

mentre per i sistemi in parallelo vale

$$r(p_1, \ldots, p_n) = 1 - \prod_{i=1}^{n} (1 - p_i)$$

In generale è però difficile calcolare la funzione di affidabilità. A questo scopo, possiamo quindi usare la simulazione e scrivere $S_i = 1$ se $U_i < p_i$. Vale allora

$$\theta(S_1, \ldots, S_n) = h(U_1, \ldots, U_n)$$

dove h è una funzione non decrescente, e di conseguenza

$$\mathrm{cov}\left[h(\boldsymbol{U}), h(1 - \boldsymbol{U})\right] \leq 0$$

Allora è possibile usare l'approccio delle variabili antitetiche per generare set di variabili casuali con varianza ridotta da utilizzare nella stima.
Poniamo ad esempio che l'output rilevante di una simulazione sia funzione di variabili casuali di input: $X = h(Y_1, \ldots, Y_m)$. Se Y_i ha distribuzione F_i ed è stata generata con il metodo della trasformata inversa, allora è possibile scrivere

$$X = h\left(F_1^{-1}(U_1), \ldots, F_m^{-1}(U_m)\right)$$

Essendo F non decrescente, lo è anche F^{-1}. Se inoltre h è monotona, lo è anche $h\left(F_1^{-1}(U_1), \ldots, F_m^{-1}(U_m)\right)$. Allora il metodo delle variabili antitetiche, che genera X_1 a partire da $U_1, \ldots, U_m$ e X_2 a partire da $1-U_1, \ldots, 1-U_m$, porta ad uno stimatore con una varianza minore (perchè $\mathrm{cov}\left[X_1 X_2\right] < 0$) di quanto avremmo ottenuto generando X_2 con un differente insieme di numeri casuali.

8.1.2. Approssimazione di un integrale secondo il metodo Monte Carlo

Supponiamo di voler stimare

$$\theta = \mathrm{E}\left[\mathrm{e}^{U}\right] = \int_0^1 \mathrm{e}^{u}\,\mathrm{d}u$$

che sappiamo valere $e - 1$.
La funzione e^{u} è monotona, quindi possiamo usare il metodo delle variabili antitetiche per ottenere uno stimatore con varianza ridotta. Si noti che

$$\mathrm{cov}\left[\mathrm{e}^{U}, \mathrm{e}^{1-U}\right] = \mathrm{E}\left[\mathrm{e}^{U}\mathrm{e}^{1-U}\right] - \mathrm{E}\left[\mathrm{e}^{U}\right]\mathrm{E}\left[\mathrm{e}^{1-U}\right] = \mathrm{e} - (\mathrm{e}-1)^2 = -0.2342$$

Poichè

$$\mathrm{var}\left[\mathrm{e}^{U}\right] = \mathrm{E}\left[\mathrm{e}^{2U}\right] - \left(\mathrm{E}\left[\mathrm{e}^{U}\right]\right)^2 = \int_0^1 \mathrm{e}^{2x}\,\mathrm{d}x - (\mathrm{e}-1)^2 = \frac{e^2-1}{2} - (e-1)^2 = 0.2420$$

l'uso di due variabili casuali indipendenti porta ad una varianza di

$$\mathrm{var}\left[\frac{\mathrm{e}^{U_1} + \mathrm{e}^{U_2}}{2}\right] = \frac{\mathrm{var}\left[\mathrm{e}^{U}\right]}{2} = 0.1210$$

mentre l'uso di due variabili antitetiche porta ad ottenere una varianza di

$$\mathrm{var}\left[\frac{\mathrm{e}^{U} + \mathrm{e}^{1-U}}{2}\right] = \frac{\mathrm{var}\left[\mathrm{e}^{U}\right]}{2} + \frac{\mathrm{cov}\left[\mathrm{e}^{U}, \mathrm{e}^{1-U}\right]}{2} = 0.0039$$

In Scilab, possiamo procedere come segue per verificare l'efficacia della tecnica delle variabili antitetiche:

```
function [y] = stima()
  u = grand(1,1,'def');
  y = exp(u);
endfunction

function [y] = stimaVA()
  u = grand(1,1,'def');
  y = (exp(u) + exp(1-u)) / 2;
endfunction

function [y] = repeat(n,g)
  for i = [1:n]
    y($+1) = g();
  end
```

```
endfunction

m = 100;
dati = repeat(2*m,stima);
datiVA = repeat(2*m,stimaVA);

mean(dati)
mean(datiVA)
variance(dati)
variance(datiVA)
```

Otteniamo i seguenti risultati:

```
-->mean(dati)
 ans  =

    1.7247371

-->mean(datiVA)
 ans  =

    1.7116268

-->variance(dati)
 ans  =

    0.2437242

-->variance(datiVA)
 ans  =

    0.0035324
```

Possiamo osservare che la media è rimasta praticamente inalterata, mentre la varianza si è ridotta in maniera consistente.
Proviamo ora a ripetere la simulazione più volte:

```
function [y] = euEstimate()
  m = 100;
  y = mean(repeat(m,stima));
endfunction

function [y] = euEstimateVA()
  m = 100;
  y = mean(repeat(m,stimaVA));
endfunction
```

Disegnamo un istogramma che mette in evidenza la frequenza con cui la stima dell'integrale, ottenuta con lo stimatore classico, si distribuisce sull'intervallo $[1.55, 1.85]$ suddiviso in classi di dimensione 0.01 (8.1):

```
n = 1000;
histplot([1.55:0.01:1.85],repeat(n,euEstimate));
```

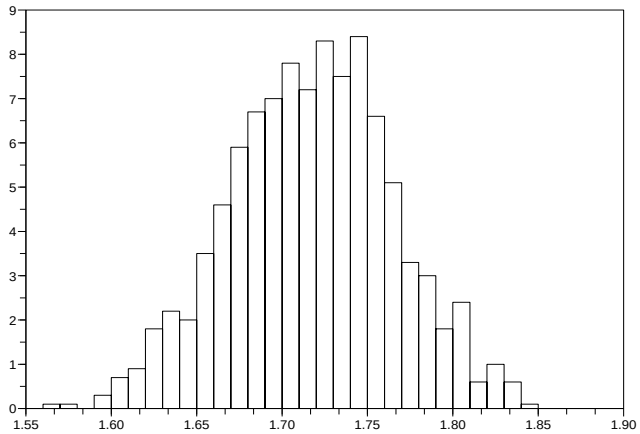

Figura 8.1.: Istogramma dei valori della stima dell'integrale usando lo stimatore classico

Facciamo la stessa cosa per la stima dell'integrale ottenuta con lo stimatore a varianza ridotta (8.2):

```
set("current_figure",1)
histplot([1.55:0.01:1.85],repeat(n,euEstimateVA));
```

Come ci attendavamo, la varianza della stima dell'integrale si è ridotta notevolmente usando la tecnica delle variabili antitetiche.
Per rendere più evidente questa osservazione, possiamo eseguire nuovamente la simulazione e confrontare i grafici (8.3):

```
set("current_figure",3)
histplot([1.55:0.01:1.85],repeat(n,euEstimate));
histplot([1.55:0.01:1.85],repeat(n,euEstimateVA),style=5);
```

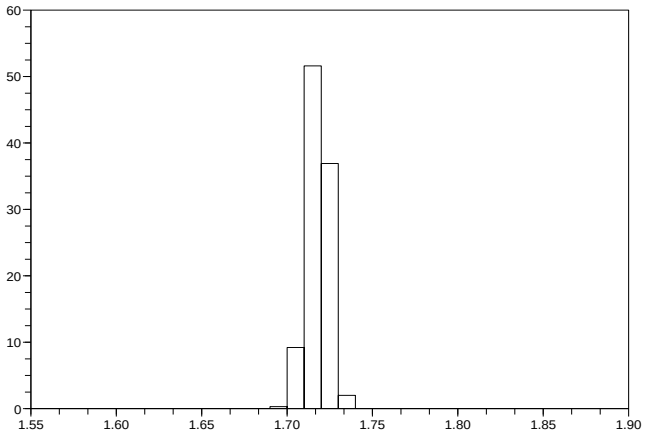

Figura 8.2.: Istogramma dei valori della stima dell'integrale usando lo stimatore a varianza ridotta

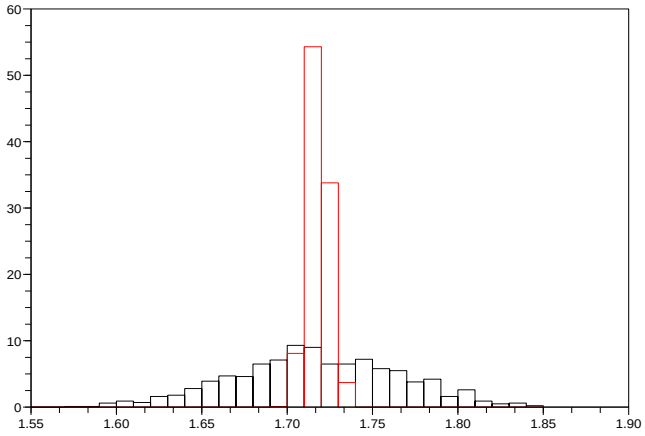

Figura 8.3.: Istogrammi dei valori della stima dell'integrale (in rosso la stima con le variabili antitetiche)

8.1.3. Approssimazione di e

Si consideri una sequenza di numeri casuali e sia N la posizione del primo numero maggiore del precedente:

$$N = min\{n\colon n \geq 2, U_n > U_{n-1}\}$$

Vale

$$\mathrm{P}\left(N > n\right) = \frac{1}{n!}$$

e quindi

$$\mathrm{P}\left(N = n\right) = \mathrm{P}\left(N > n-1\right) - \mathrm{P}\left(N > n\right) = \frac{n-1}{n!}$$

Il valore atteso di N è

$$\mathrm{E}\left[N\right] = \sum_{n=2}^{\infty} n\frac{n-1}{n!} = \frac{1}{(n-2)!} = \mathrm{e}$$

e vale anche

$$\mathrm{E}\left[N^2\right] = \sum_{n=2}^{\infty} \frac{n}{(n-2)!} = \sum_{n=2}^{\infty} \frac{2}{(n-2)!} + \sum_{n=2}^{\infty} \frac{n-2}{(n-2)!} = 2\mathrm{e} + \sum_{n=3}^{+\infty} \frac{1}{(n-3)!} = 3\mathrm{e}$$

La varianza di N è quindi uguale a $3\mathrm{e} - \mathrm{e}^2 \approx 0.7658$
Nel corso della simulazione, invece di continuare a generare N finchè la condizione precedente non è soddisfatta, posso generare anche

$$M = min\{n \geq 2, 1 - U_n > 1 - U_{n-1}\} = min\{n \geq 2, U_{n-1} < U_n\}$$

cioè l'opposto di N. Si noti innanzitutto che una delle due variabili casuali N e M varrà sicuramente 2. Inoltre, lo stimatore $\frac{N+M}{2}$ avrà varianza inferiore rispetto a $\frac{N_1+N_2}{2}$ perchè N e M sono antitetiche.
Per dimostrare questa affermazione, consideriamo innanzitutto la variabile N_a, che rappresenta il numero di numeri osservati prima che uno sia maggiore del precedente, e ridefiniamo N in modo che assuma valori $2, 2 + N_a$ con probabilità $\frac{1}{2}$.
Calcoliamo il valore atteso di N

$$\mathrm{E}\left[N\right] = 2 + \frac{1}{2}\mathrm{E}\left[N_a\right]$$

e di N^2

$$\mathrm{E}\left[N^2\right] = \frac{1}{2}4 + \frac{1}{2}\mathrm{E}\left[(2+N_a)^2\right] = 4 + 2\mathrm{E}\left[N_a\right] + \frac{1}{2}\mathrm{E}\left[N_a^2\right]$$

Da questi risultati ricaviamo quindi i seguenti valori attesi:

$$\mathrm{E}\left[N_a\right] = 2e - 4$$

$$\mathrm{E}\left[N_a^2\right] = 8 - 2e$$

quindi

$$\text{var}[N_a] = 14e - 4e^2 - 8 \approx 0.4997$$

$$\text{var}[N + M] = \text{var}[4 + N_a] = \text{var}[N_a]$$

$$\frac{\text{var}[N_1 + N_2]}{\text{var}[N + M]} \approx \frac{1.5316}{0.4997} \approx 3.065$$

Possiamo concludere che il metodo delle variabili antitetiche riduce la varianza dello stimatore di $\frac{1}{3}$.
Possiamo verificare in Scilab questo risultato. Calcoliamo la stima di e con lo stimatore classico:

```
function [y] = repeat(n,g)
  for i=[1:n]
    y($+1) = g();
  end
endfunction

function [i] = minEstimate()
  i = 2;
  u1 = grand(1,1,'def')
  u2 = grand(1,1,'def')
  while (u2 < u1),
    u1 = u2;
    u2 = grand(1,1,'def');
    i = i + 1;
  end;
endfunction

m = 1000;
datiMin = repeat(m,minEstimate);
```

Dopo aver ripetuto 1000 simulazioni di N, visualizziamo media e varianza:

```
-->mean(datiMin)
 ans  =

    2.737

-->variance(datiMin)
 ans  =

    0.7786096
```

Ripetiamo l'esperimento simulando 1000 volte M

```
function [i] = maxEstimate()
  i = 2;
  u1 = grand(1,1,'def')
  u2 = grand(1,1,'def')
  while (u2 > u1),
    u1 = u2;
    u2 = grand(1,1,'def');
    i = i + 1;
  end;
endfunction

datiMax = repeat(m,maxEstimate);
```

e visualizziamo nuovamente media e varianza

```
-->mean(datiMax)
 ans  =

    2.717

-->variance(datiMax)
 ans  =

    0.727638
```

Possiamo ora calcolare la stima di e usando N e M insieme

```
datiMinMax = (datiMin + datiMax) / 2;

mean(datiMinMax)
variance(datiMinMax)
```

ottenendo

```
-->mean(datiMinMax)
 ans  =

    2.727

-->variance(datiMinMax)
 ans  =

    0.3728438
```

Si può osservare che la varianza si è dimezzata.
Anche l'errore quadratico medio si è ridotto:

```
-->mean((datiMin - %e)^2)
 ans  =

    0.7781814

-->mean((datiMax - %e)^2)
 ans  =

    0.7269126

-->mean((datiMinMax - %e)^2)
 ans  =

    0.3725470
```

e la stima di e è abbastanza vicina al suo valore reale

```
-->%e
 %e  =

    2.7182818
```

Possiamo ora scrivere alcune funzioni che ripetano m volte la simulazione

```
function [y] = eEstimateMin()
  y = mean(repeat(m,minEstimate));
endfunction

function [y] = eEstimateMax()
  y = mean(repeat(m,maxEstimate));
endfunction

function [y] = eEstimateMinMax()
  minimi = repeat(m,minEstimate);
  massimi = repeat(m,maxEstimate);
  y = mean((minimi + massimi) / 2);
endfunction
```

e disegnare gli istogrammi dei valori della stima di e ottenuti con N (8.4) e M (8.5)

```
histplot([2.5:0.01:3],repeat(m,eEstimateMin));


set("current_figure",1)
histplot([2.5:0.01:3],repeat(m,eEstimateMax));
```

ed infine disegnare l'istogramma dei valori della stima di e ottenuti con lo stimatore migliorato con la tecnica delle variabili antitetiche (8.6)

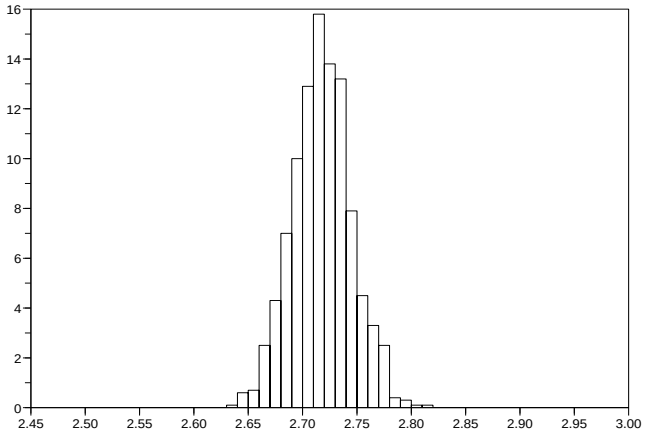

Figura 8.4.: Istogrammi dei valori della stima di e ottenuti con N

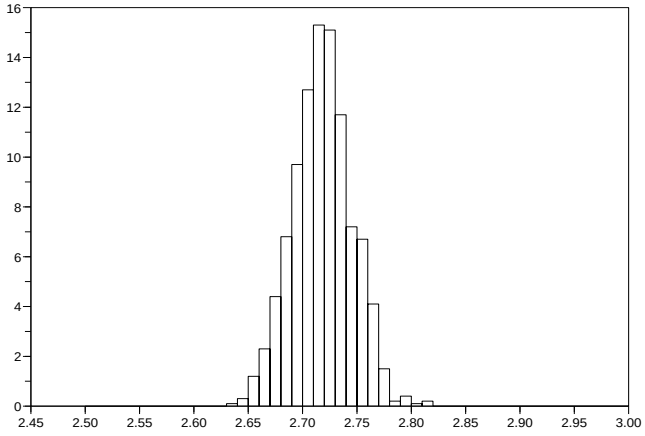

Figura 8.5.: Istogrammi dei valori della stima di e ottenuti con M

```
set("current_figure",2)
histplot([2.5:0.01:3],repeat(m,eEstimateMinMax));
```

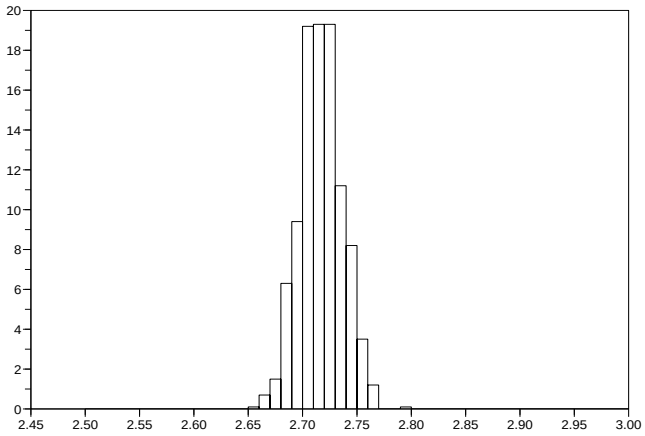

Figura 8.6.: Istogrammi dei valori della stima di e ottenuti con $\frac{N+M}{2}$

Ripetiamo nuovamente la simulazione e confrontiamo graficamente la stima con uno stimatore classico e la stima con uno stimatore a varianza ridotta (8.7)

```
histplot([2.5:0.01:3],repeat(m,eEstimateMin));
histplot([2.5:0.01:3],repeat(m,eEstimateMinMax),style=5);
```

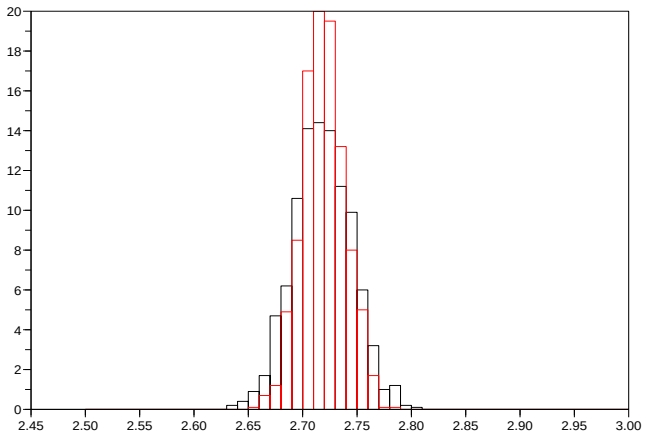

Figura 8.7.: Istogrammi dei valori della stima di e (in rosso la stima con le variabili antitetiche)

8.2. Uso di variate di controllo

Supponiamo di voler stimare $\theta = \mathrm{E}[X]$, dove X è l'output di una simulazione, e di conoscere un'altra variabile di output Y il cui valore atteso μ_y è noto. Allora, per ogni costante c, anche

$$X + c(Y - \mu_y)$$

è uno stimatore non distorto di θ, di varianza

$$\mathrm{var}[X + c(Y - \mu_y)] = \mathrm{var}[X + cY] = \mathrm{var}[X] + c^2\mathrm{var}[Y] + 2c\,\mathrm{cov}[X, Y]$$

Per scegliere c in modo da minimizzare la varianza, ne calcolo la derivata e considero il valore di c in cui la derivata della varianza si annulla:

$$c^* = -\frac{\mathrm{cov}[X, Y]}{\mathrm{var}[Y]}$$

Con questo valore di c la varianza vale

$$\mathrm{var}[X + c^*(Y - \mu_y)] = \mathrm{var}[X] - \frac{[\mathrm{cov}[X, Y]]^2}{\mathrm{var}[Y]}$$

Come desiderato, la varianza di $X + c^*(Y - \mu_y)$ è minore della varianza di X perchè il termine $\frac{[\mathrm{cov}[X,Y]]^2}{\mathrm{var}[Y]}$ è sempre positivo.
La quantità Y, correlata con X, è chiamata **variata di controllo**.

Si supponga che la correlazione sia positiva: in questo caso si può approssimativamente dire che X è grande quando lo è anche Y. Osserviamo inoltre che c^* è negativo (positivo) quando la correlazione è positiva (negativa). Di conseguenza, quando Y si discosta dal suo valor medio μ_y (noto) di una certa quantità, anche X si discosta dal suo valor medio θ (per correlazione) ed allora c^* (negativo) corregge lo stimatore in direzione opposta.
Si osservi inoltre che, dividendo la varianza dello stimatore per $\text{var}[X]$, si ottiene

$$\frac{\text{var}[X + c^*(Y - \mu_y)]}{\text{var}[X]} = 1 - (\text{corr}[X, Y])^2$$

Di norma le quantità $\text{cov}[X, Y]$ e $\text{var}[Y]$ non sono note in anticipo e devono essere stimate dai dati simulati:

$$\widehat{\text{cov}[X, Y]} = \sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})/(n-1)$$

e

$$\widehat{\text{var}[Y]} = \sum_{i=1}^{n} (Y_i - \overline{Y})^2/(n-1)$$

Possiamo quindi approssimare di conseguenza c^* con $\hat{c}^*$, ed infine approssimare la varianza dello stimatore controllato.

Si noti la similitudine con la regressione lineare

$$X = a + bY + e$$

dove

- e è una variabile casuale con media 0 e varianza σ^2
- $\hat{a}$, lo stimatore ai minimi quadrati di a sui dati X_i, Y_i è

$$\hat{a} = \overline{X} - \hat{b}\overline{Y}$$

- $\hat{b}$, lo stimatore ai minimi quadrati di b sui dati X_i, Y_i è

$$\hat{b} = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}$$

quindi $\hat{b} = -\hat{c}^*$.
Vale infine

$$\overline{X} + \hat{c}^* = \overline{X} - \hat{b}(\overline{Y} - \mu_y) = \hat{a} + \hat{b}\mu_y$$

8.2.1. Stimatore per la funzione affidabilità a varianza ridotta

Supponiamo di voler stimare tramite simulazione la funzione affidabilità

$$r(p_1, \ldots, p_n) = \mathrm{E}\left[\theta(S_1, \ldots, S_n)\right]$$

dove $S_i = 1$ se $U_i < p_i$, 0 altrimenti.
Da $\mathrm{E}\left[S_i\right] = p_i$ segue che

$$\mathrm{E}\left[\sum_{i=1}^{n} S_i\right] = \sum_{i=1}^{n} p_i$$

Si può quindi usare il numero delle componenti funzionanti, $Y \equiv \sum S_i$, come una variata di controllo dello stimatore $X \equiv \theta(S_1, \ldots, S_n)$. Dal momento che queste due funzioni sono positivamente correlate, c^* ha segno negativo.

8.2.2. Approssimazione di un integrale

Supponiamo di voler approssimare, come in precedenza, l'integrale

$$\int_0^1 \mathrm{e}^u \, \mathrm{d}u$$

equivalente al valore atteso $\mathrm{E}\left[\mathrm{e}^U\right]$. Usiamo U come variata di controllo e calcoliamo ora la varianza del nuovo stimatore

$$\mathrm{var}\left[X\right] - \frac{\left[\mathrm{cov}\left[X, Y\right]\right]^2}{\mathrm{var}\left[Y\right]} \text{ con } X = \mathrm{e}^U \text{ e } Y = U$$

La covarianza vale

$$\mathrm{cov}\left[\mathrm{e}^U, U\right] = \mathrm{E}\left[U\mathrm{e}^U\right] - \mathrm{E}\left[U\right]\mathrm{E}\left[\mathrm{e}^U\right] = \int_0^1 x\mathrm{e}^x \, \mathrm{d}x - \frac{e-1}{2} = 1 - \frac{e-1}{2} = 0.14086$$

Poichè $\mathrm{var}\left[U\right] = \frac{1}{12}$, ne segue che

$$\mathrm{var}\left[\mathrm{e}^U + c^*\left(U - \frac{1}{2}\right)\right] = \mathrm{var}\left[\mathrm{e}^U\right] - 12(0.14086)^2 = 0.2420 - 0.2380 = 0.0039$$

8.3. Riduzione tramite condizionamento

Supponiamo di voler effettuare una simulazione per stimare il valore di $\theta = \mathrm{E}\left[X\right]$, e di avere a disposizione un'altra variabile casuale Y tale che $\mathrm{E}\left[X|Y\right]$ è noto e assume un valore che può essere determinato dalla simulazione stessa. Poichè

$$\mathrm{E}\left[\mathrm{E}\left[X|Y\right]\right] = \mathrm{E}\left[X\right] = \theta$$

segue che anche $\mathrm{E}\left[\mathrm{E}\left[X|Y\right]\right]$ è uno stimatore non distorto di θ.
Possiamo dimostrare inoltre che la sua varianza è inferiore a quella dello stimatore $\mathrm{E}\left[X\right]$ di X. Ricordiamo infatti la formula della varianza condizionata

$$\mathrm{var}\left[X\right] = \mathrm{E}\left[\mathrm{var}\left[X|Y\right]\right] + \mathrm{var}\left[\mathrm{E}\left[X|Y\right]\right]$$

Poichè tutti i termini sono non negativi, vale

$$\mathrm{var}\left[X\right] \geq \mathrm{var}\left[\mathrm{E}\left[X|Y\right]\right]$$

8.3.1. Stima di π

Ricordiamo come avevamo stimato π in precedenza: avevamo calcolato il rapporto tra i punti che cadono in un cerchio di raggio 1 e quelli che cadono in un quadrato ad esso circoscritto. In particolare, avevamo definito $V_i = 2U_i - 1$ e $I = 1$ se $V_1^2 + V_2^2 \leq 1$, 0 altrimenti, e calcolato $\mathrm{E}\left[I\right] = \frac{\pi}{4}$.
Possiamo migliorare lo stimatore usando $I|V_1$ invece di I:

$$\begin{aligned}\mathrm{E}\left[I|V_1 = v\right] &= \mathrm{P}\left(V_1^2 + V_2^2 \leq 1|V_1 = v\right) = \mathrm{P}\left(V_2^2 \leq 1 - v^2\right) \\ &= \int_{(1-v^2)^{1/2}}^{(1+v^2)^{1/2}} \left(\frac{1}{2}\right) \mathrm{d}x = (1-v^2)^{1/2}\end{aligned}$$

quindi lo stimatore $\mathrm{E}\left[I|V_1 = v\right] = (1 - V^2)^{1/2}$ ha sempre media $\pi/4$ ma varianza ridotta rispetto a I.
Poichè $\mathrm{E}\left[(1-V^2)^{1/2}\right] = \mathrm{E}\left[(1-U^2)^{1/2}\right]$, possiamo semplificare lo stimatore usando $(1-U^2)^{1/2}$.
Il miglioramento nella varianza è facilmente determinato:

$$\mathrm{var}\left[(1-U^2)^{1/2}\right] = \mathrm{E}\left[1-U^2\right] - \left(\frac{\pi}{4}\right)^2 = \frac{2}{3} - \left(\frac{\pi}{4}\right)^2 \approx 0.0498$$

a differenza di

$$\mathrm{var}\left[I\right] = \left(\frac{\pi}{4}\right)\left(1 - \frac{\pi}{4}\right) \approx 0.1686$$

Lo stimatore potrebbe essere ulteriormente migliorato con la tecnica delle variabili antitetiche

$$\frac{1}{2}\left[(1-U^2)^{1/2} + (1-(1-U)^2)^{1/2}\right]$$

o delle variate di controllo:

$$\left(1-U^2\right)^{1/2} + c\left(U^2 - 1/3\right)$$

Realizziamo ora l'approssimazione di π in Scilab. Usiamo inizialmente lo stimatore classico:

```
function [y] = stimaPi()
  v1 = grand(1,1,'unf',-1,1);
  v2 = grand(1,1,'unf',-1,1);
  if (v1^2 + v2^2 <= 1),
    y = 1;
  else
    y = 0;
  end;
endfunction

function [y] = repeat(n,g)
  for i=[1:n]
    y($+1) = g();
  end
endfunction

m = 1000;
dati = repeat(m,stimaPi);
```

Dopo aver eseguito m simulazioni, abbiamo ottenuto un vettore di valori di cui ora calcoliamo media e varianza:

```
-->mean(dati)*4
 ans  =

    3.14

-->variance(dati)
 ans  =

    0.1689439
```

Proviamo ora ad approssimare π usando lo stimatore condizionato:

```
function [y] = stimaPiCond()
  y = sqrt(1 - grand(1,1,'def')^2)
endfunction

dati = repeat(m,stimaPiCond);
```

Dopo m simulazioni, ecco media e varianza dei valori ottenuti:

```
-->mean(dati)*4
 ans  =

    3.1466257
```

```
-->variance(dati)
 ans  =

    0.0497869
```

La media è rimasta approssimativamente costante, mentre la varianza si è ridotta sensibilmente.
Possiamo migliorare ulteriormente lo stimatore facendo uso della tecnica delle variabili antitetiche

```
function [y] = stimaPiCondVA()
  u = grand(1,1,'def');
  y = (sqrt(1-u^2) + sqrt(1-(1-u)^2)) / 2;
endfunction

dati = repeat(m,stimaPiCondVA);
```

con i seguenti risultati

```
-->mean(dati)*4
 ans  =

    3.1380716

-->variance(dati)
 ans  =

    0.007150
```

8.3.2. La variabile casuale esponenziale

Supponiamo che Y sia una variabile casuale esponenziale di media 1 e che la variabile casuale X sia normale con media y e varianza 4 se condizionata con $Y = y$. Vediamo come si può usare la simulazione per stimare efficientemente $\theta = \mathrm{P}(X > 1)$.
L'approccio più semplice è il seguente:

1. $Y = -\log U$
2. se $Y = y$, allora genera X normale con media y e varianza 4
3. se $X > 1$, allora $I = 1$, 0 altrimenti

La media di I su varie simulazioni è lo stimatore della quantità desiderata.
Lo stimatore può essere migliorato notando che, se $Y = y$, allora

$$Z = \frac{X - Y}{2}$$

è una variabile casuale normale standard. Di conseguenza

$$\mathrm{E}\left[I|Y=y\right]=\mathrm{P}\left(X>1|Y=y\right)=\mathrm{P}\left(Z>\frac{1-y}{2}\right)=\overline{\Theta}\left(\frac{1-y}{2}\right)$$

dove $\overline{\Theta}(x)=1-\Theta(x)$ è la probabilità che una variabile casuale normale standard ecceda x. Lo stimatore ottenuto è migliore rispetto al precedente.
Essendo lo stimatore $\overline{\Theta}\left(\frac{1-y}{2}\right)$ monotono in Y, può essere ulteriormente migliorato usando la tecnica delle variabili antitetiche:

$$\mathrm{E}\left[I|Y=y\right]=\frac{\overline{\Theta}\left(\frac{1+\log(U)}{2}\right)+\overline{\Theta}\left(\frac{1+\log(1-U)}{2}\right)}{2}$$

Utilizziamo ora Scilab per verificare questi risultati. Innanzitutto generiamo 1000 variabili casuali I condizionate da $Y=y$:

```
function [y] = ySim()
  y = -log(grand(1,1,'def'));
endfunction

function [i] = iSim()
  nor = grand(1,1,'nor',ySim(),2);
  if (nor <= 1),
    i = 0;
  else
    i = 1;
  end;
endfunction

function [y] = repeat(n,g)
  for i=[1:n]
    y($+1) = g();
  end
endfunction

m = 1000;
dati = repeat(m,iSim);
```

Calcoliamo media e varianza dei dati ottenuti:

```
-->mean(dati)
 ans  =

    0.491

-->variance(dati)
```

```
 ans  =

    0.2501692
```

Generiamo altri 1000 valori sfruttando ora la distribuzione normale:

```
function [y] = iSimCond()
  y = 1-cdfnor("PQ",(1-ySim())/2,0,1)
endfunction

dati = repeat(m,iSimCond);
```

La media è rimasta approssimativamente uguale mentre la varianza è diminuita in maniera evidente:

```
-->mean(dati)
 ans  =

    0.4890479

-->variance(dati)
 ans  =

    0.0248531
```

Infine, miglioriamo ulteriormente l'approssimazione usando la tecnica delle variabili antitetiche:

```
function [y] = iSimCondAV()
  u = grand(1,1,'def');
  nor1 = cdfnor("PQ",(1+log(u))/2,0,1);
  nor2 = cdfnor("PQ",(1+log(1-u))/2,0,1);
  y = (1 - nor1 + 1 - nor2) / 2;
endfunction

dati = repeat(m,iSimCondAV);
```

Ancora una volta, la media resta sostanzialmente invariata mentre la varianza si riduce di un'ordine di grandezza.

```
-->mean(dati)
 ans  =

    0.4906723

-->variance(dati)
 ans  =

    0.0032037
```

8.3.3. Simulazione di variabili compound

Supponiamo che $X_1, X_2, \ldots$ sia una sequenza di variabili casuali iid, e indipendenti dalla variabile casuale non negativa a valori interi N. La variabile casuale

$$S = \sum_{i=1}^{N} X_i$$

viene chiamata **variabile casuale compound**.
Ad esempio, nel caso di un'assicurazione, X_i potrebbe essere l'ammontare di un risarcimento, e N potrebbe essere il numero di risarcimenti nel tempo t specificato. In situazioni simili, N è solitamente una variabile casuale di Poisson (nel qual caso S è chiamata variabile casuale compound di Poisson) o una variabile casuale di Poisson mista, se esiste un'altra variabile Λ tale che la distribuzione condizionata di N, dato $\Lambda = \lambda$, è poissoniana con media λ. Se Λ ha funzione di densità di probabilità $g(\lambda)$, allora la funzione di massa di probabilità di N mista è

$$\mathrm{P}(N = n) = \int_0^{+\infty} \frac{\mathrm{e}^{-\lambda}\lambda^n}{n!} g(\lambda)\,\mathrm{d}\lambda$$

Una variabile casuale di Poisson mista si presenta quando una situazione ambientale casuale determina la media del numero di eventi nel periodo di tempo di interesse. La funzione di distribuzione di Λ è chiamata *mixing distribution*.
Supponiamo ora di voler stimare la quantità

$$p = \mathrm{P}\left(\sum_{i=1}^{N} X_i > c\right)$$

per qualche c costante positiva prefissata. Nel caso di un'assicurazione, significa determinare la probabilità che la somma di N rimborsi, l'importo di ciascuno dei quali è stocastico, ecceda il capitale disponibile.
Seguendo l'approccio standard di simulazione, potremmo generare il valore di N (poniamo n), poi i valori di $X_1, \ldots, X_n$ ed infine usarli per determinare il valore dello stimatore I che vale 1 se $\sum_{i=1}^{N} X_i > c$, 0 altrimenti. Il valore medio di I dopo varie simulazioni sarebbe lo stimatore di p.
Possiamo migliorare la simulazione usando il metodo di valore atteso condizionato, quindi generando i valori degli X_i in sequenza finché la somma dei valori generati eccede c. Sia quindi

$$M = min\left\{n\colon \sum_{i=1}^{n} X_i > c\right\}$$

Se il valore generato di M è m, allora usiamo $\mathrm{P}(N \geq m)$ come stimatore di p. Lo stimatore così ottenuto ha varianza minore perchè, dal momento che le variabili casuali X_i sono positive, vale

$$I = 1 \Longleftrightarrow N \geq M$$

e quindi

$$\mathrm{E}[I|M] = \mathrm{P}(N \geq M|M)$$

Ora

$$\mathrm{P}(N \geq M|M = m) = \mathrm{P}(N \geq m|M = m) = \mathrm{P}(N \geq m)$$

Di conseguenza, se il valore di M ottenuto dalla simulazione è $M = m$, allora il valore $\mathrm{E}[I|M]$ ottenuto è $\mathrm{P}(N \geq m)$.
Il precedente stimatore può essere migliorato ulteriormente usando una variata di controllo. Sia ad esempio $\mu = \mathrm{E}[X_i]$, e definiamo

$$Y = \sum_{i=1}^{M} (X_i - \mu)$$

Può essere dimostrato che $\mathrm{E}[Y] = 0$.

Utilizziamo ancora una volta Scilab per verificare questi risultati. Innanzitutto generiamo il valore n di una variabile casuale N poissoniana di parametro 5, poi generiamo n esponenziali $X_1, \ldots, X_n$ di parametro 1:

```
function [y] = xi(n)
  y = grand(n,1,'exp',1);
endfunction

function [y] = nSim()
  y = grand(1,1,'poi',5);
endfunction

function [y] = directSim(c)
  n = nSim();
  if (sum(xi(n)) > c),
    y = 1;
  else
    y = 0;
  end;
endfunction

c = 5;
```

Simuliamo ora il sistema, cioè determiniamo se i rimborsi hanno ecceduto il capitale dell'assicurazione:

```
-->directSim(c)
 ans  =

    1
```

Ripetiamo la simulazione 1000 volte

```
function [y] = repeat(n,g,c)
  for i=[1:n]
    y($+1) = g(c);
  end
endfunction

m = 1000;
dati = repeat(m,directSim,c);
```

e otteniamo media e varianza dello stimatore

```
-->mean(dati)
 ans  =

    0.434

-->variance(dati)
 ans  =

    0.2458899
```

Possiamo migliorare lo stimatore usando il metodo di valore atteso condizionato:

```
function [y] = vrSim(c)
  m = 0;
  somma = 0;
  while (somma <= c),
    m = m + 1;
    somma = somma + xi(1);
  end;
    y = 1 - cdfpoi("PQ",m-1,5);
endfunction

dati = repeat(m,vrSim,c);
```

Infine calcoliamo media e varianza del nuovo stimatore.

```
-->mean(dati)
 ans  =

    0.4499032

-->variance(dati)
 ans  =

    0.0797658
```

8.3.4. Simulazione di una terapia anti tumorale

Supponiamo di avere $n + m$ cellule, delle quali n sono malate di cancro e m sono normali. A ciascuna cellula assegnamo un peso ω_i.
Supponiamo di uccidere una cellula per unità di tempo, e chiamiamo S l'insieme di cellule vive in ogni istante di tempo.
La prossima cellula $i \in S$ verrà uccisa con probabilità $\frac{\omega_i}{\sum_{j \in S} \omega_j}$.
La cellula i viene uccisa per prima con probabilità $\frac{\omega_i}{\sum_{j=1}^{n+m} \omega_j}$.
Posto che la cellula i è stata uccisa, la cellula seguente k viene uccisa con probabilità $\frac{\omega_k}{\sum_{j \neq i} \omega_j}$.
Questo processo continua finchè le prime n cellule malate di cancro vengono uccise. Definiamo in questo istante la variabile casuale N rappresentante il numero di cellule sane ancora vive. Vogliamo determinare $\mathrm{P}(N \geq k)$.
Prima di sviluppare una procedura efficiente di simulazione, consideriamo un modello analogo in cui la cellula i viene uccisa al tempo casuale T_i, con $T_1, \ldots, T_{N+M}$ variabili casuali esponenziali indipendenti di tasso $\omega_1, \ldots, \omega_{n+m}$. Poichè le variabili casuali esponenziali sono senza memoria, allora la probabilità che la cellula i-esima venga uccisa è proprio $\frac{\omega_i}{\sum_{j \in S} \omega_j}$, come definito in precedenza.
Ora, se $T^{(k)}$ è il k-esimo valore più grande di $T_{n+1}, \ldots, T_{n+m}$, allora $T^{(k)}$ è il primo istante in cui ci sono meno di k cellule normali vive. Perchè N sia almeno k, tutte le cellule cancerose devono essere state uccise al tempo $T^{(k)}$, cioè

$$\mathrm{P}(N \geq k) = \mathrm{P}\left(Max_{i \leq n}\{T_i\} < T^{(k)}\right)$$

quindi

$$\mathrm{P}\left(N \geq k | T^{(k)}\right) = \mathrm{P}\left(Max_{i \leq n}\{T_i\} < T^{(k)} | T^{(k)}\right) = \prod_{i=1}^{n}\left(1 - \mathrm{e}^{-\omega_i T^{(k)}}\right)$$

In conclusione, generando m variabili casuali esponenziali $T_{m+1}, \ldots, T_{m+n}$ possiamo ottenere lo stimatore non distorto di $\mathrm{P}(N \geq k)$

$$\prod_{i=1}^{n}\left(1 - \mathrm{e}^{-\omega_i T^{(k)}}\right)$$

dove $T^{(k)}$ è il k-esimo valore più grande.
Inoltre, essendo lo stimatore funzione crescente di $T_{n+1}, \ldots, T_{n+m}$, è possibile applicare altri metodi di riduzione della varianza, ad esempio la tecnica delle variabili antitetiche.
Il procedimento di simulazione completo potrebbe essere il seguente:

1. generiamo i numeri casuali $U_1, \ldots, U_m$

2. sia $T^{(k)}$ il k-esimo più grande valore degli m valori

$$-\frac{1}{\omega_{n+i}} \log U_i$$

3. sia $S^{(k)}$ il k-esimo più grande valore degli m valori

$$-\frac{1}{\omega_{n+i}} \log(1 - U_i)$$

4. lo stimatore risultante è

$$\frac{1}{2}\left[\prod_{i=1}^{n}\left(1 - \mathrm{e}^{-\omega_i T^{(k)}}\right) - \prod_{i=1}^{n}\left(1 - \mathrm{e}^{-\omega_i S^{(k)}}\right)\right]$$

Proviamo a realizzare questa simulazione in Scilab.
La funzione seguente sceglie la cellula che verrà uccisa, tenendo in considerazione il peso ad essa associato.

```
function [y] = genDisc(pf)
  pfNorm = pf/(sum(pf));
  u = grand(1,1,'def');
  s = 0;
  i = 0;
  while (u > s),
    i = i + 1;
    s = s + pfNorm(i);
  end;
  y = ws(2,i)
endfunction
```

La funzione seguente elimina la cellula uccisa da `ws`, il vettore che associa ad ogni cellula il suo peso.

```
function [y] = Drop(ws,killed)
  killid = (find(ws(2,:)==killed));
  if (killid==1),
    y = ws(:,[2:$]);
  else
    for i=[1:killid-1],
      y(:,i) = ws(:,i);
    end;
    for i=[killid+1:size(ws,'c')],
      y(:,$+1) = ws(:,i)
    end;
  end;
endfunction
```

La funzione seguente simula il risultato della cura, dove `n` è il numero di cellulare sane, `m` il numero di cellule malate e `k` il numero di cellule sane che intendiamo preservare a cura terminata.

```
function [y] = simDirect(n,m,k,ws)
  status = zeros(1,n+m);
  while (sum(status(1:n)) < n)
    killed = genDisc(ws(1,:));
    status(killed) = 1;
    ws = Drop(ws,killed)
  end;
  if (sum(1-status(n+1:n+m)) >= k),
    y = 1;
  else
    y = 0;
  end;
endfunction
```

La simulazione che vogliamo realizzare prevede inizialmente 5 cellule sane e 15 malate. Vogliamo calcolare la probabilità di preservare tutte le 5 cellule sane dopo aver ucciso tutte quelle malate. Per ottenere la stima, ripetiamo la simulazione 1000 volte.

```
function [y] = simula()
  n = 5;
  m = 15;
  k = 5;
  ws = [grand(n+m,1,'def')';[1:n+m]];
  y = simDirect(n,m,k,ws);
endfunction

function [y] = repeat(n,g)
  for i=[1:n]
    y($+1) = g();
  end
endfunction

mm = 1000;
```

Otteniamo il seguente risultato:

```
-->mean(y)
 ans  =

    0.195

-->variance(y)
 ans  =
```

```
    0.1569907
```

Ripetiamo l'esperimento ottimizzato, facendo uso delle variabili casuali esponenziali:

```
function [y] = simVR(n,m,k,ws)
  n = 5;
  m = 15;
  k = 5;
  ws = [grand(n+m,1,'def')';[1:n+m]];
  for i=[n+1:n+m]
    tkk(i-n) = grand(1,1,'exp',1/ws(1,i));
  end;
  tkk_sort = gsort(tkk,'g');
  tk = tkk_sort(k);
  y = 1;
  for i=[1:n]
    y = y * (1-exp(-ws(1,i)*tk));
  end;
endfunction

mm = 1000;
dati = repeat(mm,simVR);
```

Otteniamo i seguenti risultati:

```
-->mean(dati)
 ans  =

    0.1810820

-->variance(dati)
 ans  =

    0.0464059
```

Ottimizziamo ulteriormente il risultato precedente usando il metodo delle variabili antitetiche:

```
function [y] = simVRVA(n,m,k,ws)
  n = 5;
  m = 15;
  k = 5;
  ws = [grand(n+m,1,'def')';[1:n+m]];
  us = grand(m,1,'def');
  for i=[n+1:n+m]
    tkk(i-n) = -1/ws(1,i) * log(us(i-n));
```

```
  end;
  for i=[n+1:n+m]
    skk(i-n) = -1/ws(1,i) * log(1-us(i-n));
  end;
  tkk_sort = gsort(tkk,'g');
  tk = tkk_sort(k);
  skk_sort = gsort(skk,'g');
  sk = skk_sort(k);
  y1 = 1;
  y2 = 1;
  for i=[1:n]
    y1 = y1 * (1-exp(-ws(1,i)*tk));
  end;
  for i=[1:n]
    y2 = y2 * (1-exp(-ws(1,i)*sk));
  end;
  y = (y1 + y2) / 2;
endfunction

mm = 1000;
dati = repeat(mm,simVRVA);
```

Otteniamo una varianza ulteriormente ridotta:

```
-->mean(dati)
 ans  =

    0.1861267

-->variance(dati)
 ans  =

    0.0359829
```

8.4. Campionamento stratificato

Supponiamo di voler stimare $\theta = \mathrm{E}[X]$ e di disporre di una variabile discreta Y, con valori $y_1, \ldots, y_k$, tali che

- le probabilità $p_i = \mathrm{P}(Y = i)$ sono note
- per ogni i, possiamo simulare il valore di X condizionato da $Y = y_i$

Se vogliamo stimare $\mathrm{E}[X]$, solitamente si procede generando n indipendenti repliche di X e quindi calcolandone la media campionaria, stimatore non distorto del valore

atteso avente varianza $\frac{\sigma^2}{n}$.
Scrivendo invece

$$\mathrm{E}[X] = \sum_{i=1}^{k} \mathrm{E}[X|Y = y_i]\, p_i$$

si nota che è possibile stimare $\mathrm{E}[X]$ a partire dalle quantità $\mathrm{E}[X|Y = y_i]$. Invece di generare n volte X, eseguiamo np_i volte le simulazioni condizionate all'evento $Y = y_i$, per ogni i. Se definiamo $\overline{X}_i$ come la media degli np_i valori condizionati di X osservati, allora abbiamo lo stimatore non distorto

$$\varepsilon = \sum_{i=1}^{k} \overline{X}_i p_i$$

Lo stimatore ε è chiamato stimatore a campionamento stratificato di $\mathrm{E}[X]$. Calcoliamone la varianza cominciando a determinare $\mathrm{var}\left[\overline{X}_i\right]$:

$$\mathrm{var}\left[\overline{X}_i\right] = \frac{\mathrm{var}[X|Y = y_i]}{np_i}$$

quindi

$$\mathrm{var}[\varepsilon] = \sum_{i=1}^{k} p_i^2 \mathrm{var}\left[\overline{X}_i\right] = \frac{1}{n}\sum_{i=1}^{k} p_i \mathrm{var}[X|Y = y_i] = \frac{1}{n}\mathrm{E}[\mathrm{var}[X|Y]]$$

Poichè $\mathrm{var}\left[\overline{X}\right] = \frac{1}{n}\mathrm{var}[X]$, mentre $\mathrm{var}[\varepsilon] = \frac{1}{n}\mathrm{E}[\mathrm{var}[X|Y]]$, dalla formula della varianza condizionata

$$\mathrm{var}[X] = \mathrm{E}[\mathrm{var}[X|Y]] + \mathrm{var}[\mathrm{E}[X|Y]]$$

si comprende che il nuovo stimatore ε ha varianza minore rispetto alla media campionaria $\overline{X}$.

8.4.1. Applicazioni del campionamento stratificato

Approssimazione di π

In precedenza abbiamo mostrato che

$$\frac{\pi}{4} = \mathrm{E}\left[\sqrt{1 - U^2}\right]$$

Possiamo quindi stimare π generando $U_1, \ldots, U_n$ e usando lo stimatore

$$est = \frac{4}{n}\sum_{j=1}^{n} \sqrt{1 - [(U_j + j - 1)/n]^2}$$

Possiamo ulteriormente migliorare lo stimatore utilizzando variabili antitetiche

$$\hat{\pi} = \frac{2}{n}\sum_{j=1}^{n} \left(\sqrt{1 - [(U_j + j - 1)/n]^2} + \sqrt{1 - [(j - U_j)/n]^2}\right)$$

Dimostriamo questo risultato con Scilab. Innanzitutto stimiamo π usando la tecnica del campionamento stratificato:

```
function [y] = stima(n)
  y1 = 0;
  for j = [1:n]
    y1 = y1 + sqrt(1-((grand(1,1,'def')+j-1)/n)^2);
  end;
  y= 4/n * y1;
endfunction

function [y] = repeat(n,g,p1)
  for i=[1:n]
    y($+1) = g(p1);
  end
endfunction

n = 1000;
dati = repeat(n,stima,20);
```

Otteniamo i seguenti risultati:

```
-->mean(dati)
 ans  =

    3.1413658

-->variance(dati)
 ans  =

    0.000392
```

Ora miglioriamo la stima precedente utilizzando le variabili antitetiche:

```
function [y] = stimaVA(n)
  y1 = 0;
  for j=[1:n]
    add1 = sqrt(1-((grand(1,1,'def')+j-1)/n)^2);
    add2 = sqrt(1-((j-grand(1,1,'def'))/n)^2);
    y1=y1 + add1 + add2;
  end;
  y = 2/n * y1;
endfunction

n = 1000;
dati = repeat(n,stimaVA,20);
```

Otteniamo i seguenti risultati:

```
-->mean(dati)
 ans  =

    3.1414594

-->variance(dati)
 ans  =

    0.0001763
```

Approssimazione di integrali multidimensionali di funzioni monotone

Supponiamo di voler usare la simulazione per calcolare l'integrale n-dimensionale

$$\theta = \int_0^1 \cdots \int g(x_1, \ldots, x_n)\, \mathrm{d}x_1 \ldots \mathrm{d}x_n$$

Se consideriamo $U_1, \ldots, U_n$ variabili casuali indipendenti uniformi in (0,1), il precedente integrale può essere scritto come $\theta = \mathrm{E}\left[g(U_1, \ldots, U_n)\right]$.
Supponiamo che g sia una funzione non decrescente di $U_1, \ldots, U_n$.
Definiamo $Y = \prod_{i=1}^n U_i$. Poichè sia $g(U_1, \ldots, U_n)$ che Y sono funzioni crescenti di $U_1, \ldots, U_n$, sembrerebbe che $\mathrm{E}\left[\mathrm{var}\left[g(U_1, \ldots, U_n)|Y\right]\right]$ sia relativamente piccolo. Proviamo allora a stimare θ stratificando su $\prod_{i=1}^n U_i$. Dobbiamo mostrare come

1. generare $U_1, \ldots, U_n$ condizionato da $\prod_{i=1}^n U_i$ avente un valore specificato;
2. generare il valore di $\prod_{i=1}^n U_i$ in modo stratificato.

Per fare entrambe le cose, colleghiamo U_i ad un processo di Poisson, ricordando che $-\log U$ è esponenziale con tasso 1 ed interpretandolo come il tempo tra l'$(i-1)$-esimo evento e l'i-esimo evento di un processo di Poisson con tasso 1. Con questa interpretazione, il j-esimo evento di un processo di Poisson avverrà al tempo T_j, dove

$$T_j = \sum_{i=1}^{j} -\log(U_i) = -\log(U_i \cdots U_j)$$

Poichè la somma di n indipendenti variabili casuali esponenziali con tasso 1 è una variabile casuale gamma $(n, 1)$, possiamo generare (in modo stratificato, che dobbiamo ancora vedere) il valore di $T_n = -\log(U_i \cdots U_j)$ con una variabile casuale gamma $(n, 1)$. Questo porta ad un valore generato per $\prod_{i=1}^n U_i$, esattamente

$$\prod_{i=1}^{n} U_i = \mathrm{e}^{T_n}$$

Per generare le variabili casuali individuali $U_1, \ldots, U_n$ condizionate dal loro prodotto, usiamo il seguente risultato dei processi di Poisson: la sequenza dei primi $n-1$ eventi, condizionata all'n-esimo evento di un processo di Poisson al tempo t, sono distribuiti nel tempo come una sequenza ordinata di $n-1$ variabili casuali indipendenti uniformi in $(0,t)$. Di conseguenza, una volta generato il valore di T_n, la singola variabile casuale U_i può essere ottenuta prima generando $n-1$ numeri casuali $V_1, \ldots, V_{n-1}$, e quindi ordinandoli in maniera crescente $V_{(1)} < V_{(2)} < \ldots < V_{(n-1)}$.
Rappresentando il tempo dell'evento j con $T_n V_{(j)}$, vale

$$T_n V_{(j)} = -\log(U_1 \cdots U_j) = -\log(U_1 \cdots U_{j-1}) - \log(U_j) = T_n V_{(j-1)} - \log(U_j)$$

Quindi, con $V_{(0)} = 0$, $V_{(n)} = 1$, vale

$$U_j = \mathrm{e}^{-T_n\left(V_{(j)} - V_{(j-1)}\right)}$$

Di conseguenza vediamo come generare $U_1, \ldots, U_n$ condizionato da $\prod_{i=1}^{n} U_i$. Per fare la stratificazione, sfruttiamo il fatto che $T_n = -\log\left(\prod_{i=1}^{n} U_i\right)$ è una variabile casuale gamma $(n,\ 1)$. Se abbiamo pianificato di fare m simulazioni, allora durante la k-esima un numero casuale U deve essere generato e T_n deve essere posto uguale a $G_n^{-1}\left(\frac{U+k-1}{m}\right)$. Per questo valore di T_n, usiamo allora il precedente risultato per simulare i valori di $U_1, \ldots, U_n$ e calcolare $g(U_1, \ldots, U_n)$ (generiamo cioè $n-1$ numeri casuali, li ordiniamo in ordine crescente e definiamo U_j come fatto sopra).
La media dei valori di g ottenuti nelle m esecuzioni della simulazione è lo stimatore a campionamento stratificato di $\mathrm{E}\left[g(U_1, \ldots, U_n)\right]$.

Vediamo un esempio in Scilab: approssimiamo l'integrale

$$\theta = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{x_1 + x_2}{1 + x_3 - x_4}\, \mathrm{d}x_1\, \mathrm{d}x_2\, \mathrm{d}x_3\, \mathrm{d}x_4$$

```
n = 4;
function [y] = g(u1,u2,u3,u4)
  y = (u1 + u2) / (1 + u3 - u4)
endfunction

function [y] = tn()
  U = grand(4,1,'def');
  y = -log(prod(U));
endfunction

function [y] = simG()
  vs = gsort(grand(n-1,1,'def'),'g','i');
  vs($+1) = 1;
  t = tn();
  us(1) = exp(-t*vs(1));
```

```
  for j=[2:n],
    us(j) = exp(-t*(vs(j)-vs(j-1)));
  end;
  y = g(us(1),us(2),us(3),us(4))
endfunction

function [y] = repeat(n,G)
  for i=[1:n]
    y($+1) = G();
  end
endfunction

m = 1000;
dati = repeat(m,simG);
```

Ecco media e varianza dei risultati delle 1000 simulazioni eseguite:

```
-->mean(dati)
 ans  =

    1.9919374

-->variance(dati)
 ans  =

    0.000214
```

Infine, notiamo che esiste un'altra alternativa per simulare una variabile casuale con distribuzione gamma $(n,1)$, poichè questa ha la stessa distribuzione di $\frac{1}{2}\chi^2_2$, dove χ^2_{2n} è una variabile casuale di distribuzione chi-quadro con $2n$ gradi di libertà. Vale quindi

$$G_n^{-1} = \frac{1}{2} F^{-1}_{\chi^2_2} x$$

dove l'inversa della funzione di chi-quadro può essere ottenuta in vari modi.

9. Validazione statistica di ipotesi

Generalmente, un'analisi statistica ha inizio con l'assegnazione di una particolare distribuzione di probabilità agli eventi in studio. Ipotesi di questo tipo possono essere statisticamente testate osservando i dati reali e poi verificando se la particolare distribuzione di probabilità ipotizzata è consistente con essi. Vediamo due test di questo genere.

9.1. Test Chi-quadro

Il test chi-quadro viene utilizzato nel caso di distribuzioni discrete.
Nel corso di una simulazione, supponiamo di osservare n variabili casuali indipendenti $Y_1, \ldots, Y_n$, ciascuna con valore $1, \ldots, k$, e di aver ipotizzato che queste variabili hanno massa di probabilità $\{p_i, i = 1, \ldots, k\}$. Se Y rappresenta ciascuna delle Y_j e H_0 è l'ipotesi da testare (ipotesi nulla), possiamo scrivere:

$$H_0 \colon \mathrm{P}(Y = i) = p_i$$

Per testare l'ipotesi, sia N_i il numero di Y_j uguale a i. Poichè ogni Y_j vale i con probabilità $\mathrm{P}(Y = i)$, ne segue che N_i è binomiale con parametri (n, p_i) sotto l'ipotesi H_0. Vale dunque

$$\mathrm{E}[N_i] = np_i$$

e quindi $(N_i - np_i)^2$ è un indice di quanto verosimilmente p_i vale la probabilità $Y = i$: se è grande (relativamente a np_i), allora l'ipotesi H_0 non è corretta. Consideriamo quindi la quantità

$$T = \sum_{i=1}^{k} \frac{(N_i - np_i)^2}{np_i}$$

e respingiamo l'ipotesi H_0 quando T è grande.
Supponiamo ora che i dati attuali portino T ad assumere il valore t. Definiamo allora il p-value

$$p - \text{value} = \mathrm{P}_{H_0}(T \geq t)$$

dove la probabilità è calcolata nell'assunzione che H_0 sia corretta.
Il p-value quindi è la probabilità che un valore di T sia più grande di t osservato se l'ipotesi H_0 fosse vera. Solitamente si rifiuta l'ipotesi H_0 se il valore del p-value risulta minore di 0.05 o 0.01.
Per ottenere il p-value, possiamo approssimarlo sapendo che, per grandi valori di n,

T ha approssimativamente una distribuzione chi-quadro con $k-1$ gradi di libertà. Possiamo quindi scrivere

$$p - \text{value} \approx \mathrm{P}\left(\chi^2_{k-1} \geq t\right)$$

dove χ^2_{k-1} è una variabile casuale chi-quadro con $k-1$ gradi di libertà.
Quando l'approssimazione chi-quadro restituisce un p-value vicino ad un valore critico (ad esempio non maggiore di 0.15), possiamo approssimare in maniera migliore generando r insiemi di variabili $Y_1, \ldots, Y_n$, quindi ottenendo $T_1, \ldots, T_r$, e concludendo che vale:

$$\frac{\text{numero di } l\text{: } T^{(l)} \geq t}{r} \approx \mathrm{P}\left(\chi^2_{k-1} \geq t\right)$$

Facciamo qualche esempio usando Scilab. Innanzitutto scriviamo una funzione che calcoli il valore di t

```
function [t] = chiValue(dati,ps)
  mm = size(dati,'r');
  [a,ns] = dsearch(dati,[1:k],"d");
  t = sum((ns-(mm*ps))^2 ./(mm*ps));
endfunction
```

e una funzione che calcoli il $p-$value usando la funzione di ripartizione della distribuzione chi-quadro:

```
function [y] = chiTest()
  t = chiValue(dati,ps);
  y = 1 - cdfchi("PQ",t,k-1);
endfunction
```

Generiamo 1000 valori interi di una variabile casuale uniforme nell'intervallo $[1, 10]$, e calcoliamo il $p-$value nell'ipotesi (corretta) che la distribuzione sia uniforme.

```
k = 10;
m = 1000;
ps = ones(1,k) * 1/k;
dati = grand(m,1,'uin',1,k);

-->chiTest()
 ans  =

    0.0882255
```

Siamo interessati a ripetere il test 100 volte, cambiando continuamente i dati di cui disponiamo:

```
function [y] = repeat(n,g)
  for i=[1:n]
    y($+1) = g();
  end
```

```
endfunction

function [y] = ct()
  dati = grand(m,1,'uin',1,k);
  y = chiTest();
endfunction

n = 100;
p_values = repeat(n,chiTest);
```

Calcoliamo la media dei 1000 $p-$ value:

```
-->mean(p_values)
 ans  =

    0.5516629
```

Ripetiamo l'esperimento generando dati con distribuzione binomiale $(9, 1/2)$:

```
function [y] = ct_bin()
  dati = grand(m,1,'bin',k-1,1/2);
  y = chiTest()
endfunction

p_values_bin = repeat(n,ct_bin);
```

Otteniamo una media di $p-$ value pari a:

```
-->mean(p_values_bin)
 ans  =

    0.
```

Consideriamo ora un altro esempio. Definiamo questi dati

```
dati = [1, 10, 2, 5, 7, 6, 1, 4, 7, 5, ...
        3, 5, 8, 1, 6, 6, 3, 10, 7, 2, ...
        10, 10, 2, 10, 5, 10, 5, 9, 2, 3, ...
        5, 3, 9, 1, 6, 6, 5, 10, 2, 2, ...
        1, 6, 8, 9, 8, 10, 10, 6, 10, 10, ...
        7, 2, 7, 9, 10, 10, 2, 1, 9, 10, ...
        2, 8, 10, 7, 5, 10, 4, 10, 6, 1, ...
        5, 3, 7, 7, 1, 10, 3, 2, 8, 2, ...
        10, 8, 5, 10, 9, 3, 8, 10, 3, 1, ...
        5, 3, 10, 7, 7, 7, 6, 9, 4, 1];
dati = dati';
histplot([0:k],dati)
```

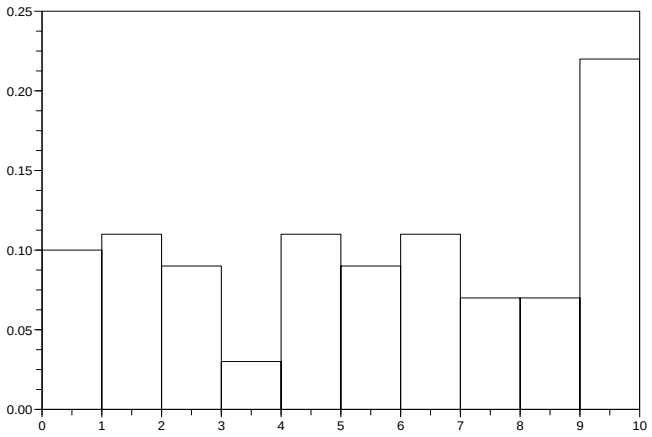

Figura 9.1.: Distribuzione di massa di probabilità dei dati considerati

aventi la distribuzione raffigurata nel grafico 9.1. Ipotizziamo che provengano da una distribuzione uniforme. La distribuzione dei dati è in effetti abbastanza simile ad una distribuzione uniforme, come dimostra il valore del $p-$value molto prossimo alla soglia di accettabilità dell'ipotesi:

```
-->chiTest(dati,ps)
 ans  =

    0.0102369
```

Come ultimo esperimento, generiamo casualmente dati estratti da una distribuzione uniforme e dimostriamo che il test chi-quadro effettivamente conferma la distribuzione di provenienza. Diversamente da quanto fatto finora, non useremo però la distribuzione chi-quadro per calcolare il $p-$value finale, ma ripeteremo l'esperimento più volte e determineremo in quante occasioni il valore di T eccederà t calcolato in precedenza:

```
-->t=chiValue(dati,ps)
 t  =

    21.6
```

Ecco il codice per la simulazione

```
k = 10;
m = size(dati,'r');
```

```
function [y] = simDist()
  y = grand(m,1,'uin',1,k)
endfunction

function [y] = tSim()
  [a,NS] = dsearch(simDist(),[1:k],"d");
  y = sum((NS-(m*ps))^2 ./ (m*ps));
endfunction

function [y] = simula(mm)
  r = repeat(mm,tSim);
  c = 0;
  for i=[1:mm]
    if (r(i) >= t)
      c = c + 1;
    end;
  end;
  y= c / mm;
endfunction
```

e i risultati, nel caso di 1000 e 10000 simulazioni:

```
 -->simula(1000)
 ans  =

    0.012

-->simula(10000)
 ans  =

    0.0102
```

9.2. Test di Kolmogorov-Smirnov

Supponiamo di avere dei dati $Y_1, \ldots, Y_n$ che ipotizziamo provengano da una particolare distribuzione continua.
Un possibile approccio per testare l'ipotesi H_0 potrebbe essere partizionare i valori di Y_j in k intervalli distinti, e quindi considerare la variabile casuale discretizzata $Y_j^d = i$ se Y_j è nell'intervallo (y_{i-1}, y_i). L'ipotesi nulla implica che $\mathrm{P}\left(Y_j^d = i\right) = \mathrm{F}\left(y_i\right) - \mathrm{F}\left(y_{i-1}\right)$ e potrebbe essere verificata con il test chi-quadro.
Un altro possibile test, che non richiede di discretizzare i dati, è il seguente. Dopo aver osservato $Y_1, \ldots, Y_n$, sia F_e la funzione di distribuzione empirica definita da

$$\mathrm{F}_e\left(x\right) = \frac{\#i\colon Y_i \leq x}{n}$$

$\mathrm{F}_e(x)$ è uno stimatore naturale di $\mathrm{F}(x)$, quindi è ad esso vicino per ogni x. Una misura semplice per testare l'ipotesi H_0 è quindi la distanza massima tra la funzione di ripartizione empirica dei dati e la funzione di ripartizione teorica, cioè

$$D \equiv max_x \{|\mathrm{F}_e(x) - \mathrm{F}(x)|\}$$

chiamata **statistica del test di Kolmogorov-Smirnov**.
Vediamo ora come calcolare concretamente D per un insieme di $Y_j = y_j$. Siano $y_{(1)}, \ldots, y_{(n)}$ i valori di Y_j in ordine crescente.
$\mathrm{F}_e(x)$ è costante negli intervalli $(y_{(j-1)}, y_{(j)})$, e salta di $1/n$ nei punti $y_{(1)}, \ldots, y_{(n)}$.
Poichè $\mathrm{F}(x)$ è una funzione crescente e sempre minore di 1, il valore massimo di $\mathrm{F}_e(x) - \mathrm{F}(x)$ è non negativo e vale

$$max_x\{\mathrm{F}_e(x) - \mathrm{F}(x)\} = max_{j=1,\ldots,n}\left\{\frac{j}{n} - \mathrm{F}\left(y_{(j)}\right)\right\}$$

In maniera simile, il valore massimo di $\mathrm{F}(x) - \mathrm{F}_e(x)$ è anch'esso non negativo e si trova immediatamente prima di uno dei punti di salto $y_{(j)}$:

$$max_x\{\mathrm{F}(x) - \mathrm{F}_e(x)\} = max_{j=1,\ldots,n}\left\{\mathrm{F}\left(y_{(j)}\right) - \frac{j-1}{n}\right\}$$

Unendo le due ultime equazioni, possiamo finalmente determinare D:

$$D = max_{j=1,\ldots,n}\left\{\frac{j}{n} - \mathrm{F}\left(y_{(j)}\right), \mathrm{F}\left(y_{(j)}\right) - \frac{j-1}{n}\right\}$$

Supponiamo ora di aver calcolato $D = d$. Poichè un grande valore di D apparirebbe inconsistente con l'ipotesi che F è la distribuzione dei dati reali, ne segue che il p-value per questo insieme di dati vale

$$p\text{-value} = \mathrm{P}_F(D \geq d)$$

dove la probabilità viene calcolata assumendo che H_0 sia corretta.
Si può dimostrare che $\mathrm{P}_F(D \geq d)$ non dipende in realtà dalla distribuzione F dei dati reali, quindi possiamo calcolare il p-value usando nel corso della simulazione una qualsiasi distribuzione continua F (anche la distribuzione uniforme in (0,1), che ovviamente preferiremo alle altre per motivi di efficienza).
Possiamo quindi generare un insieme di n numeri casuali $U_1, \ldots, U_n$ e controllare se è valida la disuguaglianza

$$max_{0\leq y\leq 1}\left\{\left|\frac{\#i\colon U_i \leq y}{n} - y\right|\right\} \geq d$$

Questo test viene ripetuto più volte e la proporzione delle volte che è soddisfatto diventa la stima del p-value dell'insieme di dati.

Facciamo un esempio in Scilab. Generiamo 10 valori di una variabile casuale esponenziale di media 1/5 e calcoliamone la funzione di ripartizione empirica:

```
m = 10;
dati = grand(m,1,'exp',1/5);

function [y] = fEmp(dati)
  m = size(dati,'r');
  datis = gsort(dati,'g','i');
  for i=[1:m]
    y(i,1) = i / m;
    y(i,2) = datis(i);
  end;
  y = y';
endfunction

y = fEmp(dati);
```

Calcoliamo i valori della funzione di ripartizione esponenziale nei punti in cui è definita la funzione di ripartizione empirica:

```
deff('[y] = cdfexp(lambda,x)','y = 1 - exp(-lambda * x)')

fs = cdfexp(5,y(2,:));
```

Calcoliamo infine d:

```
m1 = [1:m]/m - fs;
m2 = fs - [0:m-1]/m;

-->d = max([m1,m2])
 d  =

    0.3463117
```

Simuliamo ora D 10000 volte e calcoliamo quindi il $p-$value

```
function [d] = simPValue()
  us = gsort(grand(m,1,'def'),'g','i');
  m1 = [1:m]/m - us'
  m2 = us' - [0:m-1]/m
  d = max([m1,m2])
endfunction

function [y] = meanP(mm)
  c = 0;
  for i=[1:mm],
    if (simPValue() >= d),
      c = c + 1;
    end;
```

```
  end;
  y= c / mm;
endfunction

-->meanP(10000)
 ans  =

    0.1491
```

Il valore che otteniamo è maggiore di 0.01 e 0.05, quindi possiamo confermare l'ipotesi. Proviamo ora a ripetere il procedimento usando dei dati estratti da una distribuzione chi-quadro con 5 gradi di libertà:

```
m = 10;
dati = grand(m,1,'chi',5);

function [y] = fEmp(dati)
  m = size(dati,'r');
  datis = gsort(dati,'g','i');
  for i=[1:m]
    y(i,1) = i / m;
    y(i,2) = datis(i);
  end;
  y = y';
endfunction

y = fEmp(dati);
```

Calcoliamo i valori della funzione di ripartizione esponenziale nei punti in cui è definita la funzione di ripartizione empirica precedente:

```
deff('[y] = cdfexp(lambda,x)','y = 1 - exp(-lambda * x)')

fs = cdfexp(5,y(2,:));
```

Calcoliamo infine d:

```
m1 = [1:m]/m - fs;
m2 = fs - [0:m-1]/m;

-->d = max([m1,m2])
 d  =

    0.9988764
```

Simuliamo ora D 10000 volte e calcoliamo quindi il $p-$value

```
function [d] = simPValue()
  us = gsort(grand(m,1,'def'),'g','i');
  m1 = [1:m]/m - us'
  m2 = us' - [0:m-1]/m
  d = max([m1,m2])
endfunction

function [y] = meanP(mm)
  c = 0;
  for i=[1:mm],
    if (simPValue() >= d),
      c = c + 1;
    end;
  end;
  y = c / mm;
endfunction

-->meanP(10000)
 ans  =

    0.
```

Come atteso, il test non conferma l'ipotesi che i dati, da noi estratti da una distribuzione chi-quadro, abbiano una distribuzione esponenziale.

A. Risorse utili

A.1. Simulazione

- Sheldon M. Ross, Simulation, fourth edition, Elsevier/Academic Press, Amsterdam, 2006

A.2. Scilab

- `http://www.scilab.org`, sito ufficiale
- `http://h0.web.u-psud.fr/orscilab/index.html`, tutorial passo-passo
- `http://www.engineering.usu.edu/cee/faculty/gurro/Scilab.html`, informazioni generali, scripts e funzioni
- `http://wiki.axiom-developer.org/RosettaStone`, sinonimi per alcune comune operazioni dei più diffusi CAS, tra cui Scilab
- `http://www.mi.imati.cnr.it/~marco/springer/index.html`, materiale aggiuntivo in Scilab relativo al libro "Probabilità, Statistica e Simulazione"
- `http://kiwi.emse.fr/SCILAB/sci-bot/book1.htm`, *Scilab bag of tricks*
- `http://ljk.imag.fr/membres/Bernard.Ycart/polys/demarre_scilab/`, tutorial

B. Riconoscimenti

La prima edizione di questo documento, realizzata con LaTeX, è stata curata da Walter Mottinelli (wmotti(at)gmail.com) e completata nel Febbraio 2007.

L'ultima modifica del documento è avvenuta il giorno 4 ottobre 2007.

C. Licenza