Uso del generatore di numeri casuali nei programmi di simulazione

Giorgio Meini

Una simulazione al calcolatore è spesso l'unico modo per prevedere l'evoluzione dei sistemi complessi ed in particolare dei sistemi non deterministici: le tecniche descritte in questo articolo sono applicabili a una vasta gamma di situazioni

Introduzione

In via di principio è sempre possibile descrivere i fenomeni della realtà, naturale o artificiale, in termini di sistemi e di processi. Tradizionalmente i modelli di rappresentazione dei fenomeni reali vengono classificati, in funzione di alcune loro caratteristiche notevoli, come statici o dinamici, continui o discreti, deterministici o stocastici. Quasi sempre un modello rappresenta l'evoluzione nel tempo di un sistema reale ed è quindi "dinamico" e non statico; inoltre i modelli continui si distinguono dai modelli discreti per il carattere delle variabili in gioco.

Una differenza concettualmente importante è quella che intercorre tra processi deterministici - nei quali, indipendentemente dalla complessità, ogni evento è determinato da una causa o da un insieme di cause che si verificano contemporaneamente o in sequenza temporale (principio di causa-effetto o di causalità) - e processi stocastici, nei quali un evento è determinato da una causa, o da un insieme di cause, solo come effetto casuale che si realizza con una certa probabilità e in alternativa ad effetti diversi aventi proprie probabilità di realizzazione. Spesso si ricorre ad un modello stocastico per descrivere un sistema complesso che è in realtà deterministico al fine di evitare una rappresentazione dettagliata e proibitiva di molti oggetti, o eventi, e delle loro innumerevoli interazioni. La prima applicazione di questa metodologia - che permette in alcune situazioni di evitare la complessità ricorrendo alla casualità - è stata la meccanica statistica che fin dal secolo scorso ha per scopo la determinazione delle proprietà degli oggetti macroscopici come conseguenza delle proprietà dei loro componenti microscopici.

In un articolo del 1985 divenuto poi famoso [1] Stephen Wolfram sostiene che la simulazione al calcolatore è spesso l'unico modo per prevedere l'evoluzione dei sistemi complessi ed in particolare dei sistemi non deterministici (o stocastici). Nell'articolo Wolfram confronta un modello teorico per il fenomeno del moto browniano (una equazione differenziale risolvibile esattamente o per mezzo di una approssimazione numerica fornita da un algoritmo iterativo) con un modello computazionale consistente in un programma di simulazione.

Struttura di un semplice programma di simulazione

Per raggiungere lo stesso scopo che Wolfram si proponeva nell'articolo citato - prendere cioè in esame la struttura di un semplice programma di simulazione - analizziamo un programma codificato in C (per compilatore C/C++ Microsoft v8.0, ambiente target MS-DOS,) realizzato alcuni anni fa dai miei studenti dell'ITIS "Galilei" di Livorno come modello computazionale del fenomeno della disintegrazione atomica di un elemento radioattivo (**listato 1**, *radio.c*). Il programma è composto da tre cicli annidati denominati "ciclo degli esperimenti", "ciclo del tempo" e "ciclo dello spazio". Il "ciclo del tempo" permette di simulare il flusso continuo del tempo *t* per mezzo di singoli avanzamenti discreti di ampiezza *dt*: per ogni iterazione del ciclo sono simulati tutti gli eventi (deterministici o stocastici) che si verificano nell'intervallo di tempo (*t0*,*t1*) dove $t_0=t$ e $t_1=t_0+dt$. Oggetto della simulazione sono in questo caso i singoli atomi soggetti al fenomeno non deterministico della disintegrazione radioattiva: nel programma sono rappresentati mediante il vettore *Atomi[]* di valori pseudo-booleani (il valore *TRUE* contraddistingue un atomo ancora esistente, mentre il valore *FALSE* indica un atomo già disintegrato) ed il "ciclo dello spazio" consente di microsimularne l'eventuale disintegrazione prendendo in esame ogni singolo atomo. Il programma si comporta come un contatore Geiger virtuale visualizzando il numero di disintegrazioni atomiche registrate per ogni intervallo di tempo di un intero esperimento di durata definita. Trattandosi di un fenomeno aleatorio (vedremo tra poco come viene microsimulata ogni singola disintegrazione a partire dalla probabilità che essa si verifichi in un particolare intervallo di tempo) non è possibile trarre conclusioni valide dall'analisi dei risultati prodotti nel corso di un solo esperimen-

to virtuale: è quindi necessario ripetere più volte il singolo esperimento - per mezzo del "ciclo degli esperimenti" - e mantenere al tempo stesso una traccia dei risultati ottenuti in precedenza (a questo scopo il programma registra nei singoli elementi del vettore *Freq[]* il numero di volte che, in un singolo intervallo di tempo, è stato registrato un numero di disintegrazioni pari all'indice dell'elemento stesso). La funzione *Prob()* calcola e restituisce la probabilità - espressa come un valore compreso tra 0 e 1 - che un atomo si disintegri nell'intervallo (t_0,t_1) se al tempo *t0* il numero degli atomi non ancora disintegrati è *N* (N_0 rappresenta il numero degli atomi all'inizio dell'esperimento, mentre *k* è una costante che caratterizza ogni particolare elemento radioattivo). Per lo scopo che si propone questo articolo l'espressione calcolata da *Prob()* ha un'importanza del tutto secondaria (per i dettagli relativi si veda comunque il riquadro fuori testo).

Microsimulazione di un evento stocastico

Ma come è possibile simulare l'eventuale disintegrazione di un atomo, evento che nella realtà si verifica solo casualmente con probabilità p? Si deve realizzare il programma di simulazione in modo da avere in fase di esecuzione un evento che, per analogia, si verifichi solo con probabilità p: è a questo scopo che molti linguaggi di programmazione e molte librerie di uso generale o per applicazioni numeriche o di simulazione implementano un generatore di numeri casuali (per una sommaria descrizione del generatore di numeri casuali della ANSI *standard library* del C si veda il relativo riquadro).
Ogni esecuzione dell'istruzione C:

```
nc=((double)rand())/RAND_MAX;
```

assegna alla variabile *nc* di tipo *double* un valore casuale - di volta in volta diverso - uniformemente distribuito nell'intervallo che si estende da 0 a 1 (per il significato esatto di "distribuzione uniforme" si veda il riquadro di testo dedicato alle variabili casuali). L'eventuale condizione per cui il valore di *nc* risulta minore della probabilità p associata all'evento stocastico reale da simulare costituirà di conseguenza un evento simulato che si verifica solo con probabilità p:

```
if (nc<p)
  {
   /*
   simulazione dell'evento stocastico
   avente probabilità p di verificarsi
   */
  }
```

Volendo per esempio simulare il classico lancio di una moneta e registrarne l'esito come valore 0 in caso di testa e come valore 1 in caso di croce si avrà:

```
nc=((double)rand())/RAND_MAX;
if (nc<.5) esito=0; // testa
else esito=1; // croce
```

Questa tecnica è nota come "metodo di Monte Carlo" ed è stata utilizzata, fin dagli anni '40, per risolvere numericamente complessi problemi di fisica teorica da parte di scienziati come Fermi, Ullman e Von Neuman. La caratteristica denominazione è da attribuirsi al fatto che per la generazione dei numeri casuali, in mancanza dei calcolatori elettronici, si ricorreva alle registrazioni delle "uscite" alla *roulette* del casinò di Monte Carlo. Un semplice e classico esempio didattico di applicazione della tecnica al campo della computazione numerica è dato dal calcolo del valore approssimato di π.

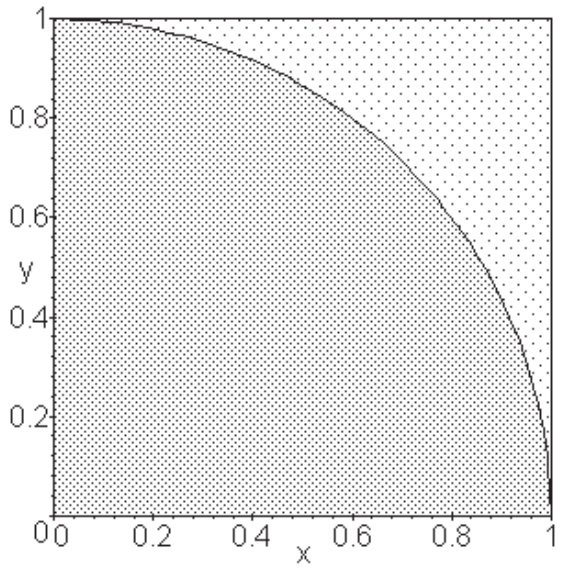

Figura 1 - Applicazione del metodo di "Monte Carlo" alla stima statistica del valore approssimativo del rapporto tra l'area del quadrato e l'area del quadrante circolare.

Si consideri (**figura 1**) il quadrato di vertici (0,0), (0,1), (1,1) e (1,0) e il settore circolare con centro (0,0) e raggio 1 compreso all'interno del quadrato stesso: l'area del quadrato è ovviamente 1, mentre quella del settore circolare è la quarta parte dell'area del cerchio di raggio 1 e cioè π/4. Ora il rapporto tra l'area del quadrato e quella del settore circolare è 1:π/4 e, di conseguenza, la probabilità che un punto del quadrato scelto a caso cada all'interno del settore circolare è esattamente π/4, cioè circa 0.7854: ripetendo per "molte" volte la scelta di un punto interno al quadrato si avrà che il 78-79% delle volte esso sarà anche interno al settore circolare. Il rapporto tra il numero di punti che cadono all'interno del settore circolare e il numero di punti scelti a caso all'interno del quadrato fornisce in definitiva una stima statistica del valore numerico approssimato di π/4 e, moltiplicando per 4 il risultato, anche di π. Dato che scegliere a caso un punto interno al quadrato significa scegliere un punto avente entrambe le coordinate comprese tra 0 e 1, il seguente frammento di programma C permette

di stimare il valore di π con approssimazione tanto maggiore quanto è maggiore *N*:

```
cont=0;
for (i=0; i<N; i++)
   {
    x=((double)rand())/RAND_MAX;
    y=((double)rand())/RAND_MAX;
    if (sqrt(pow(x,2)+pow(y,2))<1)
         /* distanza del punto
(x,y) da (0,0) minore di 1 */
       cont++;
   }
pg=(cont/N)*4;
```

Generazione di variabili casuali discrete

La tecnica di microsimulazione stocastica esposta nel paragrafo precedente permette di simulare un singolo evento aleatorio avente probabilità p di verificarsi ricorrendo al metodo di Monte Carlo, basandosi cioè sul fatto che un valore compreso tra 0 e 1 fornito dal generatore di numeri casuali ha esattamente probabilità p di cadere nell'intervallo che si estende da 0 a p (generando un numero N molto grande di valori casuali compresi tra 0 e 1 statisticamente si ha che circa N X p di essi sono minori di p ed i restanti N X (1-p) maggiori).

Ma non sempre un evento aleatorio ha solo due possibili esiti mutuamente esclusivi di cui uno avente probabilità p e l'altro, di conseguenza, probabilità q=1-p: per convincersene è sufficiente pensare al lancio di un dado che presenta sei possibili esiti distinti. Nel caso del lancio di un dado i diversi esiti possibili hanno tutti la stessa probabilità di verificarsi (sono, come si dice, "equiprobabili"), ma anche questa condizione non sempre è verificata ed un evento stocastico che può presentare esiti diversi deve in definitiva essere descritto elencando tutti gli esiti possibili ed associando a ciascuno di essi la relativa probabilità di verificarsi (naturalmente la somma di tutte le probabilità indicate deve essere pari a 1: è questa di fatto la definizione di variabile aleatoria discreta finita, si veda in proposito il relativo riquadro proposto in fondo all'articolo). Il concetto di variabile aleatoria - oltre a consentire l'elaborazione di una tecnica standard per l'appropriata generazione dei valori di una variabile casuale - permette in alcuni casi di evitare la microsimulazione dei singoli eventi aleatori del fenomeno da simulare fornendo un modello stocastico del fenomeno complessivo. Per esempio i risultati forniti dal programma di simulazione *radio.c* sono immediatamente ricavabili dalla seguente formula che permette di calcolare la probabilità di ogni singolo esito dell'esperimento complessivo, in altre parole la probabilità che si verifichino esattamente *n* disintegrazioni atomiche nell'unità di tempo (si tratta della distribuzione di probabilità di Poisson per gli "eventi rari": si veda in proposito il riquadro dedicato alle variabili aleatorie ed alle distribuzioni teoriche di probabi-

$$P(n) = \frac{k^n}{n!} e^{-k}$$

dove il fattoriale n! è dato da 1 X 2 X 3 X ... X (n-1) X n.

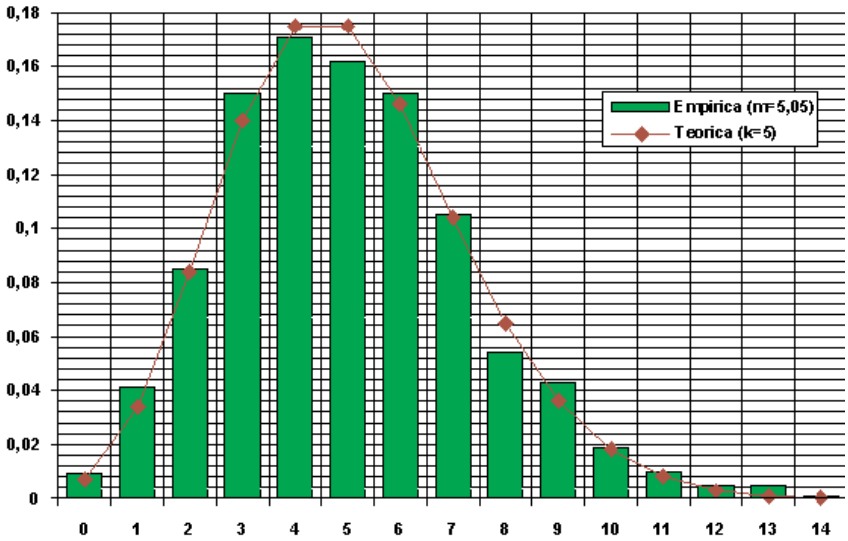

Figura 2 - Confronto tra i valori empirici delle frequenze relative prodotti dal programma di simulazione radio c (vedi testo) ed i valori teorici di probabilità calcolati con la formula di Poisson.

Nel grafico della **figura 2** sono riportati i risultati forniti dal programma *radio.c* (si tratta delle frequenze relative i cui valori compesi tra 0 e 1 sono direttamente confrontabili con le probabilità) e i valori di probabilità calcolati con la formula precedente.

Abbiamo accennato in precedenza all'esistenza di una tecnica standard per la generazione dei valori di una variabile casuale discreta e abbiamo già avuto modo di osservare che, dal punto di vista computazionale, questa non è altro che un insieme di valori per ciascuno dei quali è specificata una probabilità di presentazione in modo che la somma di tutte le probabilità sia esattamente 1 (in alcuni casi la probabilità può essere associata al valore per mezzo di una formula, ma non è necessariamente così: è infatti sempre possibile definire una variabile aleatoria discreta come un elenco di coppie valore-probabilità). Come può essere generato di volta in volta il valore casuale da attribuire alla variabile rispettando la distribuzione dei valori di probabilità che la caratterizza?

Prendiamo in esame il classico esempio del lancio di un dado. La variabile casuale che descrive il fenomeno è rappresentata dalla seguente tabella che elenca tutti i possibili eventi con le relative probabilità:

Evento	Probabilita'
"1"	$\frac{1}{6}$
"2"	$\frac{1}{6}$
"3"	$\frac{1}{6}$
"4"	$\frac{1}{6}$
"5"	$\frac{1}{6}$
"6"	$\frac{1}{6}$

Volendo applicare il metodo di Monte Carlo occorre suddividere l'intervallo [0,1) in cui è necessariamente compreso il numero casuale fornito dal generatore in segmenti di lunghezza proporzionale ai valori di probabilità di ogni singolo evento possibile. Nell'esempio del dado si tratterebbe dei segmenti [0,1/6), [1/6,1/3), [1/3,1/2), [1/2,2/3), [2/3,5/6) e [5/6,1). Gli estremi superiori di questi intervalli rappresentano le cosiddette probabilità cumulate di ciascuno evento, definite come la somma delle probabilità associate a tutti gli eventi di valore inferiore più la probabilità relativa all'evento considerato:

Esito	Probabilita' cumulata
"1"	$\frac{1}{6}$
"2"	$\frac{2}{6}=\frac{1}{3}$
"3"	$\frac{3}{6}=\frac{1}{2}$
"4"	$\frac{4}{6}=\frac{2}{3}$
"5"	$\frac{5}{6}$
"6"	$\frac{6}{6}=1$

Il valore da assegnare in fase di simulazione alla variabile aleatoria *dado* è quello associato all'intervallo nel quale è compreso il numero casuale *nc* fornito di volta in volta dal generatore:

```
double
ProbCumul[ 7] ={ 0.,.167,.333,.500,.667,.833,1.};
nc=((double)rand())/RAND_MAX;
dado=1;
while (nc>ProbCumul[ dado] )
      dado++;
```

L'implementazione di questa tecnica non dipende dal numero dei valori che la variabile casuale può assumere (se i valori dei possibili esiti non sono consecutivi è sufficiente assegnare probabilità 0 ai valori che non sono mai assunti dalla variabile in modo che due o più valori successivi di probabilità cumulata siano uguali) e non è legata alla proprietà di "equiprobabilità" dei singoli eventi possibili (le probabilità cumulate possono essere infatti facilmente calcolate anche per valori diversi delle singole probabilità associate agli eventi). Questo è quindi un metodo del tutto generale per la generazione dei valori assunti da una variabile aleatoria discreta. Per esempio, la variabile casuale che descrive il lancio di due dadi è specificata nella seguente tabella:

Evento	Probabilita'	Probabilita' cumulata
"2"	$\frac{1}{36}$	$\frac{1}{36}$
"3"	$\frac{2}{36}=\frac{1}{18}$	$\frac{3}{36}=\frac{1}{12}$
"4"	$\frac{3}{36}=\frac{1}{12}$	$\frac{6}{36}=\frac{1}{6}$
"5"	$\frac{4}{36}=\frac{1}{9}$	$\frac{10}{36}=\frac{5}{18}$
"6"	$\frac{5}{36}$	$\frac{15}{36}=\frac{5}{12}$
"7"	$\frac{6}{36}=\frac{1}{6}$	$\frac{21}{36}=\frac{7}{12}$
"8"	$\frac{5}{36}$	$\frac{26}{36}=\frac{13}{18}$
"9"	$\frac{4}{36}=\frac{1}{9}$	$\frac{30}{36}=\frac{5}{6}$
"10"	$\frac{3}{36}=\frac{1}{12}$	$\frac{33}{36}=\frac{11}{12}$
"11"	$\frac{2}{36}=\frac{1}{18}$	$\frac{35}{36}$
"12"	$\frac{1}{36}$	$\frac{36}{36}=1$

e può essere generata con il seguente codice C:

```
double
ProbCum[ 13] ={ 0.,0.,.0278,.0833,.1667,\

.2778,.4167,.5833,.7222,\
                  .8333,.9167,.9722,1.};
nc=((double)rand())/RAND_MAX;
dadi=2;
while (nc>ProbCum[ dadi] )
      dadi++;
```

o, in modo computazionalmente equivalente, con il seguente frammento di programma:

```
double
ProbCumul[ 7] ={ 0.,.167,.333,.500,.667,.833,1.};
nc=((double)rand())/RAND_MAX;
dado1=1;
```

```
while (nc>ProbCumul[ dado1] )
      dado1++;
nc=((double)rand())/RAND_MAX;
dado2=1;
while (nc>ProbCumul[ dado2] )
      dado2++;
dadi=dado1+dado2;
```

Quest'ultima versione della simulazione di un lancio di due dadi può essere facilmente estesa al lancio di N dadi:

```
double
ProbCumul[ 7] ={ 0.,.167,.333,.500,.667,.833,1.} ;
dadi=0;
for (cont=0; cont<N; cont++)
   {
    nc=((double)rand())/RAND_MAX;
    dado=1;
    while (nc>ProbCumul[ dado] )
          dado++;
    dadi=dadi+dado;
   }
```

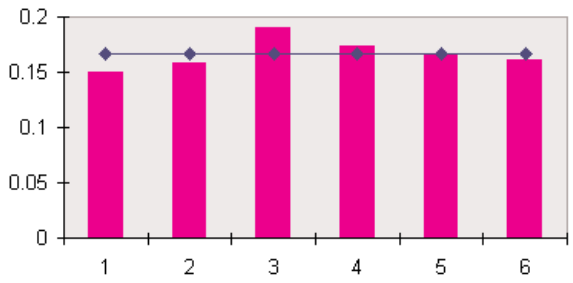

Figura 3 - Confronto tra i valori empirici delle frequenze relative prodotti da un programma di simulazione del lancio di un dado ed i valori teorici di probabilità.

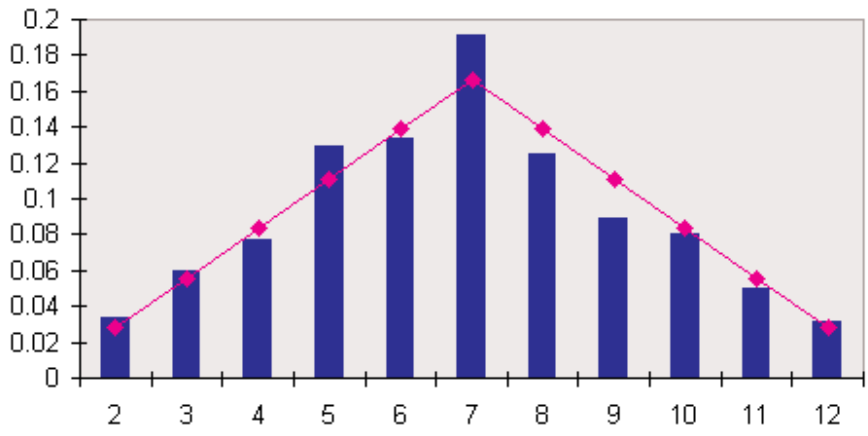

Figura 4 - Confronto tra i valori empirici delle frequenze relative prodotti da un programma di simulazione del lancio di due dadi ed i valori teorici di probabilità.

I grafici delle **figure 3 e 4** permettono di confrontare le frequenze relative - calcolate come rapporto tra il numero di volte che un evento specifico si verifica e il numero complessivo di prove effettuate - della simulazione di 1000 lanci ripetuti, rispettivamente di un dado e di due dadi, con le relative probabilità teoriche (la distribuzione teorica di probabilità è uniforme nel primo caso e nel secondo fornisce una prima e grossolana approssimazione - destinata a migliorare con il crescere del numero di dadi lanciati - della famosa curva a campana di Gauss).
In precedenza è stata introdotta la formula di Poisson che esprime la probabilità degli eventi rari:

$$P(n)=\frac{k^n}{n!}e^{-k}$$

Come si può osservare la formula permette di calcolare P(n) per un qualsiasi valore intero e positivo di n per il quale abbia senso il calcolo del fattoriale n!. La variabile casuale che descrive è quindi discreta, ma non è finita: i possibili valori che può assumere sono infiniti e non sono (superiormente) limitati, anche se un teorema di analisi matematica assicura che la somma delle (infinite) probabilità associate è comunque 1, nel rispetto della definizione di variabile aleatoria:

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

Come è possibile generare i valori di una simile variabile casuale? Semplicemente tenendo conto che i valori realmente significativi sono solo alcuni e sempre compresi in un intervallo limitato. Per esempio, nel caso della distribuzione di probabilità di Poisson per k=1 si ha che P(10)=0.0000001 e le probabilità P(n) per n maggiore di 10 sono ancora e progressivamente più piccole: dato che il generatore di numeri casuali della *standard library* del C ha una risoluzione di 1/32767=0.00003052, questi eventi non sono di fatto simulabili e possono essere in pratica ignorati considerando come valori assegnabili alla variabile casuale solo quelli compresi tra 1 e 10.

Generazione di variabili casuali continue

Nello studio dei fenomeni stocastici non tutte le grandezze sono discrete: una variabile casuale può infatti assumere valori in un sottoinsieme, limitato o illimitato, dei numeri reali (anche se, volendo ricorrere alla *computer simulation*, quest'ultimi saranno in definitiva sempre rappresentati come un insieme finito di valori discreti). Per esempio, prendendo nuovamente in considerazione l'esperimento simulato relativo alla radioattività atomica, se il numero di disintegrazioni osservate nel periodo di tempo unitario è una variabile aleatoria discreta (non superiormente limitata e avente una

distribuzione teorica di probabilità espressa dalla formula di Poisson per gli "eventi rari") allora i tempi che intercorrono tra una disintegrazione e la successiva ("intertempi" tra eventi successivi) sono rappresentati da una variabile aleatoria continua che assume valori reali positivi nell'intervallo che si estende da 0 all'infinito. La distribuzione di probabilità di quest'ultima variabile aleatoria (si veda in proposito il riquadro dedicato alle variabili aleatorie ed alle distribuzioni teoriche di probabilità) è la seguente funzione esponenziale negativa:

$$f(t) = ke^{-kt}$$

dove k è un parametro che esprime il reciproco dell'intertempo medio.

Per chiarire le idee prendiamo in esame l'esempio - proposto da A. K. Dewdney nella sua rubrica "(Ri)Creazioni al Calcolatore" [2] - della "porta degli *zombie*":

"La porta dello zombie è un'apertura di larghezza *w* in un muro altrimenti impenetrabile. Migliaia di zombie camminano con passo regolare verso il muro e ogni secondo uno zombie raggiunge un punto casuale di esso. Quelli fortunati arrivano all'apertura e oltrepassano il muro. Quelli sfortunati invece sbattono contro il muro. Se l'apertura del muro porta a una banca, i tempi casuali di arrivo degli zombie assomigliano a quelli di normali clienti. Se si vuole regolare il tempo medio di arrivo, si può stabilire un valore adeguato per la larghezza dell'apertura." [2]

Il seguente codice C simula una "porta degli *zombie*" di ampiezza *w* al centro di un muro di lunghezza unitaria per un periodo di *N* unità di tempo e registra negli elementi del vettore *freq[]* il numero di volte che si verifica un intertempo *interT* di durata pari al valore dell'indice dell'elemento stesso:

```
for (interT=0; interT<N; interT++)
    freq[interT]=0;
Narr=0;
for (t=1; t<=N; t++)
   {
      nc=((double)rand())/RAND_MAX;
      if ((nc>(.5-w/2))&&(nc<(.5+w/2)))
        {
          Narr++;
          interT=(t-Tarr);
          Tarr=t;
          freq[interT]++;
        }
   }
```

Nel grafico della **figura 5** la funzione esponenziale negativa di distribuzione della probabilità è confrontata con le frequenze relative (calcolate come rapporto *freq[interT]/Narr*) registrate in una simulazione di un periodo di 10000 unità temporali della "porta degli *zombie*" (in questo caso particolare gli intertempi sono rappresentati esclusivamente con valori interi).

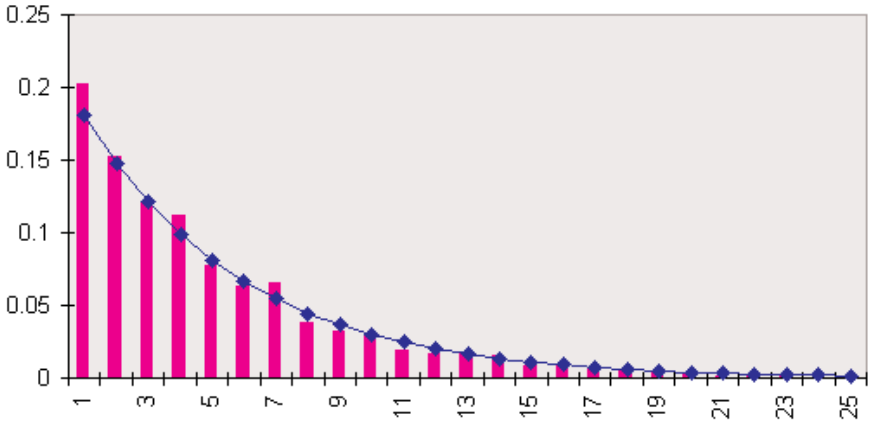

Figura 5 - Confronto tra i valori empirici delle frequenze relative prodotti da un programma di simulazione della "porta degli Zombie" (vedi testo) ed i valori della funzione di distribuzione della probabilità esponenziale negativa.

.

Come già nel caso discreto non è necessario eseguire una vera e propria microsimulazione per ottenere il valore da attribuire ad una variabile casuale di cui è nota la distribuzione teorica di probabilità. Anche nel caso continuo esiste infatti una tecnica standard per la generazione dei valori di una variabile aleatoria: se si generalizza la tecnica del ricorso ai valori di probabilità cumulata già impiegata nel caso discreto il valore *v* da generare può essere calcolato come ascissa corrispondente, secondo la funzione F(x) di ripartizione della probabilità, all'ordinata - sempre compresa tra 0 e 1 - del numero casuale *nc* (**figura 6**).

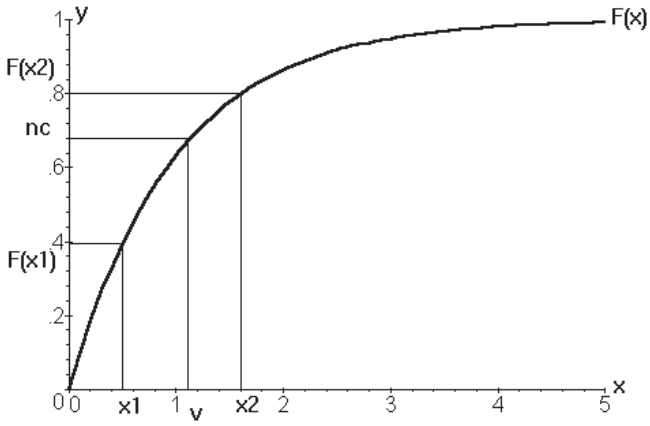

Figura 6 - Grafico della funzione di ripartizione della distribuzione di probabilità esponenziale negativa e tecnica della funzione di ripartizione inversa per la generazione dei valori della variabile aleatoria a partire da numeri casuali generati uniformemente.

La funzione di ripartizione F(x) di una variabile aleatoria continua X - si tratta, come si può leggere nel riquadro di testo dedicato alle variabili casuali, dell'analogo continuo della probabilità cumulata del caso discreto - esprime la

probabilità P(X<x) che il valore della variabile sia minore di x; per giustificare la tecnica illustrata in precedenza si osservi che ogni valore appartenente ad un qualsiasi intervallo (x_1,x_2) deve essere generato con probabilità $P(x_1<X<x_2)=F(x_2)-F(x_1)$ e che l'intervallo $[F(x_1),F(x_2)]$ - immagine di (x_1,x_2) per mezzo della funzione di ripartizione F(x) - ha effettivamente probabilità uguale alla sua ampiezza di contenere un numero casuale generato uniformemente nell'intervallo [0,1). Teoricamente le funzioni di ripartizione si ricavano dalle funzioni di distribuzione per integrazione, ma - a parte il fatto che è sempre possibile effettuare l'integrazione simbolica ricorrendo ad uno strumento software come *Mathematica* della *Wolfram Research* o *MapleV* della *Waterloo Maple Software* - nei manuali, per ogni variabile aleatoria, sono sempre riportate entrambe le funzioni.
Per fare un esempio prendiamo nuovamente in esame i tempi casuali che intercorrono tra due eventi stocastici successivi: la loro durata è regolata secondo la legge esponenziale negativa e la probabilità che un intertempo t qualsiasi sia minore di un periodo T è data dalla funzione di ripartizione

$$P(t \le T) = F(T) = 1 - e^{-kT}$$

In un programma di simulazione, per generare il valore casuale di un intertempo secondo la legge esponenziale negativa, anziché applicare direttamente la tecnica esposta in precedenza ed illustrata nella figura 6 è computazionalmente più semplice, ma matematicamente equivalente, calcolare *v* come risultato dell'applicazione dell'inversa F-1(y) della funzione di ripartizione F(x) al valore fornito dal generatore uniforme di numeri casuali compresi tra 0 e 1. Dato che

$$y = F(x) = 1 - e^{-kx}$$

si ha che

$$x = F^{-1}(y) = \frac{-\ln(1-y)}{k}$$

e, di conseguenza, la generazione del valore *v* di durata di un intertempo casuale può essere codificata in C come segue:

```
nc=((double)rand())/RAND_MAX;
v=-log(1-nc)/k;
```

Le proprietà caratteristiche delle funzioni di ripartizione della probabilità relative alle variabili casuali continue assicurano comunque l'esistenza della funzione inversa, ma quest'ultima non sempre è facilmente determinabile (anche in questo caso è comunque possibile ricorrere alle funzionalità di uno strumento software come *Mathematica* della *Wolfram Research* o *MapleV* della *Waterloo Maple Software*) e, in alcuni casi, non è esprimibile in termini di funzioni facilmente programmabili in un "normale" linguaggio di programmazione, spesso perché non lo è la stessa funzione di ripartizione: è questo il caso, in particolare, della variabile aleatoria normale standard distribuita secondo la nota legge di Gauss il cui grafico è rappresentato nella **figura 7**:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

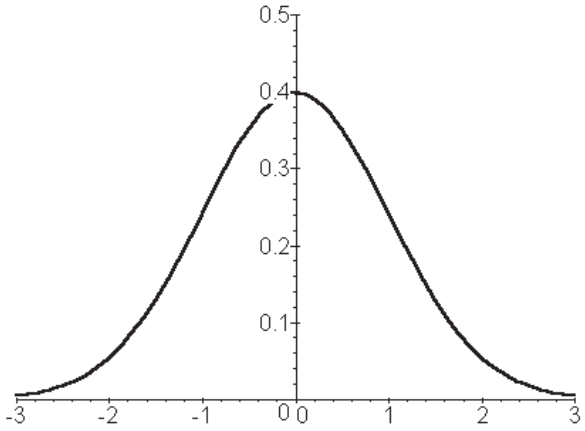

Figura 7 - Grafico della funzione di distribuzione della probabilità normale standard di Gauss.

In questo caso, come in altri analoghi, la funzione di ripartizione delle probabilità F(x) e la corrispondente inversa $F^{-1}(y)$ esistono, ma non sono esprimibili in termini di funzioni elementari ed è quindi possibile calcolare i valori corrispondenti a valori noti di x o di y solo per mezzo di formule di approssimazione (si veda in proposito il riquadro fuori testo) o ricorrendo a tabelle contenenti i valori delle funzioni stesse. In quest'ultimo caso è sufficiente invertire la modalità di consultazione della tabella dei valori della funzione di ripartizione y=F(x) effettuando la ricerca della "chiave" tra i valori y (necessariamente compresi tra 0 e 1) ed utilizzando come risultato della generazione il valore x corrispondente.
Sia che si ricorra a formule di approssimazione, sia che si utilizzino tabulazioni della funzione di ripartizione inversa i valori x* generati sono distribuiti secondo la variabile casuale normale standard: la loro media statistica è 0 e la loro varianza è 1. Per ottenere valori distribuiti normalmente secondo la legge di Gauss, ma non standardizzati è necessario moltiplicare il valore generato per la radice quadrata della varianza (scarto quadratico medio) della distribuzione simulata ed aggiungere la media:

$$x = \sigma x^* + \mu$$

Un esempio per il quale è facile realizzare un programma di simulazione (**listato 2**, *mm1*.c) è un modello di fila di attesa tecnicamente definito coda di tipo M|M|1. Si tratta di un particolare sistema di coda (per una introduzione generale ai sistemi di code si veda [3]) che ha le seguenti caratteristiche:

1. gli arrivi sono distribuiti nel tempo secondo la legge di Poisson di parametro kt, con k numero medio di arrivi nell'unità di tempo t, e di conseguenza, come si è già

detto, i tempi tra un arrivo e il successivo sono distribuiti con legge esponenziale negativa di parametro k e tempo medio tra un arrivo e il successivo 1/k;
2. la coda è disciplinata secondo l'ordine di arrivo (FIFO: *First In First Out*);
3. nel sistema esiste un solo punto di servizio: il tempo di durata del servizio è una variabile aleatoria distribuita con legge esponenziale negativa di parametro c, essendo 1/c il tempo medio di servizio.

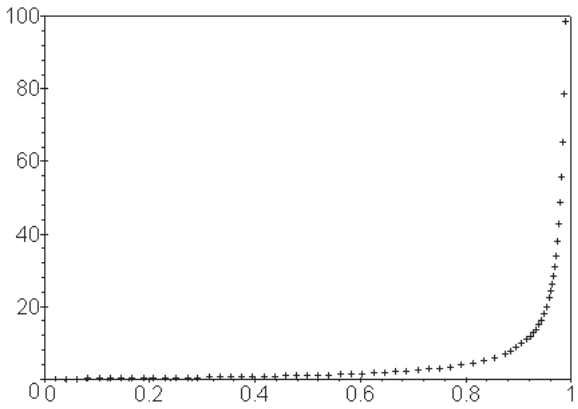

Figura 8 - Grafico della variazione della lunghezza media di una coda M/M/1 (vedi testo) in funzione del numero di arrivi per unità di tempo.

Nella "teoria delle code" si dimostra che la lunghezza media - inclusa l'unità servita - di una coda di questo tipo risulta essere k/(c-k): il grafico della **figura 8** rappresenta le lunghezze medie teoriche di una coda M|M|1 in funzione del numero medio k di arrivi per unità di tempo nell'ipotesi di un tempo medio di servizio unitario. E` interessante confrontare questo risultato teorico con i valori forniti dal programma di simulazione *mm1.c* (si noti che il "ciclo del tempo" del programma *mm1.c* è realizzato secondo la tecnica denominata "dell'evento critico": anziché incrementare la variabile *TempoSimulato* di un intervallo costante per ogni iterazione, in modo da rappresentare il regolare fluire del tempo, il suo valore viene "avanzato" dall'istante in cui avviene un nuovo arrivo in coda al momento dell'arrivo successivo).

La simulazione di una centralina per telecomunicazioni

Il programma del **listato 3** (*telecom.c*) è stato realizzato alcuni anni fa dai miei studenti dell'ITIS "Galilei" di Livorno e simula il sistema - discreto rispetto al tempo - costituito da una centralina per telecomunicazioni che serve, con un numero massimo di collegamenti contemporaneamente attivi, una stella di utenze telefoniche (**figura 9**). Le chiamate sono distribuite casualmente nel tempo secondo la legge di Poisson (i relativi intertempi sono quindi distribuiti secondo la legge esponenziale negativa) e la loro durata e` distribuita secondo la legge normale di Gauss.

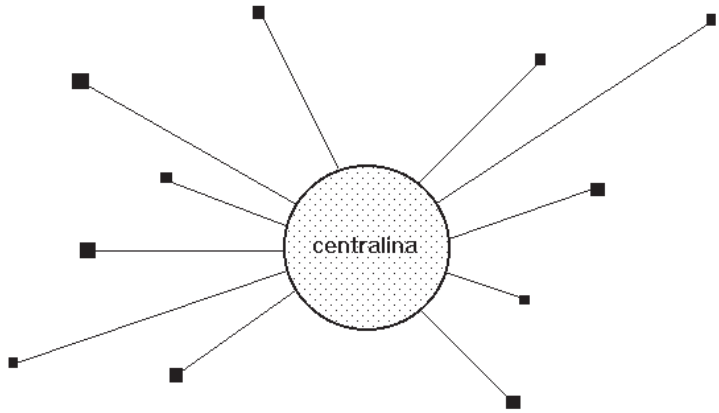

Figura 9 - Collegamento stellare delle utenze telefoniche a una centralina di interconnessione per telecomunicazioni.

La determinazione casuale delle utenze che di volta in volta effettuano una chiamata e delle utenze che sono chiamate segue la distribuzione uniforme, rispettivamente sull'insieme di tutte le utenze attualmente libere e sull'insieme di tutte le utenze diverse da quella che effettua la chiamata. La centralina è rappresentata nel programma di simulazione da un vettore, di dimensione pari al numero massimo di collegamenti attivabili contemporaneamente, avente come elementi strutture per la memorizzazione di tutte le informazioni necessarie a specificare l'attività di ogni singolo collegamento: *flag* libero/impegnato, identificativo delle due utenze in comunicazione, tempo simulato di fine comunicazione calcolato in base alla durata della comunicazione stessa. La funzione *Random()* implementa un generatore di numeri casuali compresi tra 0 e 1: viene impiegata dalle funzioni *Prossima()*, *Durata()*, *Chiamante()* e *Chiamata()* che restituiscono, rispettivamente, il tempo simulato della prossima chiamata (intertempo + tempo attuale), la durata della chiamata attuale, l'identificativo dell'utenza chiamante e l'identificativo dell'utenza chiamata. *Prossima()* genera una variabile casuale continua distribuita secondo la legge esponenziale negativa con il metodo della funzione di ripartizione inversa descritto nel paragrafo precedente. *Durata()* genera una variabile casuale continua distribuita secondo la legge normale di Gauss sempre con il metodo della funzione di ripartizione inversa, ma ricorrendo alle formule di approssimazione illustrate nello specifico riquadro di testo. *Chiamante()* e *Chiamata()* generano una variabile casuale discreta distribuita uniformemente, cioè una variabile che assume tutti gli N valori di un dato intervallo, ciascuno con la stessa probabilità 1/N. Se per generare una variabile casuale continua ed uniforme nell'intervallo [a,b) è sufficiente una trasformazione lineare del valore fornito dal generatore di numeri casuali nell'intervallo [0,1):

```
nc=((double)rand())/RAND_MAX;
v=nc*(b-a)+a;
```

allora una variabile casuale discreta ed uniforme con valori appartenenti al medesimo intervallo [a,b) dovrà essere generata in modo da produrre esclusivamente valori interi ed equiprobabili:

```
nc=((double)rand())/RAND_MAX;
v=(int)floor(nc*(b-a)+a);
```

La struttura del programma *telecom.c* è quella già descritta in generale per un qualsiasi programma di simulazione: un ciclo-tempo - realizzato in questo caso secondo la tecnica, esposta nel paragrafo precedente, dell'evento critico - ed un ciclo-spazio nidificato; in questo caso il ciclo-tempo itera, oltre che il ciclo-spazio, anche il codice di gestione dell'evento critico costituito dalla nuova chiamata in arrivo. Nonostante la semplicità della struttura e dello *output* prodotto (numero di telefonate simulate, numero di telefonate per cui l'utenza chiamata è risultata "occupata" e numero di telefonate respinte per "sovraccarico" della centralina) il programma esemplifica l'implementazione di molte delle tecniche fondamentali introdotte nell'articolo e permettere di risolvere empiricamente "per tentativi" il problema del dimensionamento di una centralina per telecomunicazioni.

Conclusioni

Dato che la simulazione mediante un programma per calcolatore è spesso realmente l'unico modo per prevedere l'evoluzione dei sistemi complessi ed in particolare dei sistemi non deterministici, credo che sia possibile applicare le tecniche descritte in questo articolo ad una vasta gamma di situazioni problematiche.

Riferimenti bibliografici

[1] S. Wolfram, "Software nella scienza e nella matematica", Le Scienze (ed. it. di *Scientific American*), Novembre 1985

[2] A. K. Dewdney, "Cinque pezzi facili per ciclo iterativo e generatore di numeri casuali", Le Scienze (ed. it. di *Scientific American*), Gennaio 1985

[3] M. Leibowitz, "Le Code", Le Scienze (ed. it. di *Scientific American*), Gennaio 1969

RIQUADRO 1
Determinazione delle probabilità di disintegrazione atomica

L'espressione calcolata dalla funzione *Prob()* del programma di simulazione *radio.c* (listato 1) è stata ricavata a partire dal classico modello empirico e deterministico della radioattività (dedotto dai dati osservati sperimentalmente) secondo il quale il numero N di atomi non ancora disintegrati all'istante t è:

$$N(t) = N_0 e^{-kt}$$

Il numero di atomi disintegrati nell'intervallo di tempo (t_0,t_1) è quindi dato da

$$N(t_0) - N(t_1) = N_0 \left(e^{-kt_0} - e^{-kt_1}\right)$$

e, calcolando la probabilità di disintegrazione p come rapporto tra numero di atomi disintegrati e numero totale di atomi, si ha

$$p = \frac{N(t_0) - N(t_1)}{N(t_0)} = \frac{N_0}{N(t_0)}\left(e^{-kt_0} - e^{-kt_1}\right)$$

che è esattamente la formula calcolata dalla funzione *Prob()*.

RIQUADRO 2
Il generatore di numeri casuali e il metodo congruenziale lineare

I numeri casuali non possono essere generati con un metodo a caso. Si deve ricorrere ad una teoria.
[Donald Knuth]

Come può un calcolatore - che è una macchina discreta, finita e rigidamente deterministica - generare una sequenza apparentemente infinita di numeri casuali distribuiti nell'intervallo reale compreso tra 0 e 1?
Per prima cosa il generatore di numeri casuali della *standard library* del C ANSI - la funzione *rand()* - non restituisce direttamente un numero reale compreso tra 0 e 1, ma un intero positivo compreso tra 0 e *RAND_MAX* (tipicamente 32767): è possibile ottenere un numero "reale" positivo e minore di 1 dividendo il valore restituito da *rand()* per *RAND_MAX*. Il generatore è quindi discreto e può ovviamente fornire soltanto un numero finito di valori distinti: resta il problema di produrne una sequenza casuale. In realtà la sequenza dei valori successivamente restituiti da *rand()* è rigidamente predeterminata e si ripete ciclicamente producendo tutti i diversi valori interi compresi tra 0 e *RAND_MAX* sempre nello stesso ordine ed il suo uso come generatore di numeri "pseudo-casuali" è giustificato solo dalla relativa uniformità dei valori all'interno dell'intervallo: anche una breve sequenza di valori successivi risulta infatti "equamente" distribuita tra 0 e *RAND_MAX*.
Come può dunque essere implementato un generatore di numeri pseudo-casuali?
La tecnica più diffusa - denominata "metodo congruenziale lineare" - è fondata sulla seguente formula che permette di generare il valore Ni a partire dal valore immediatamente precedente N_i-1:

$$\mathbf{N_i = (aN_{i-1} + c) \bmod m}$$

dove i e N_i sono interi positivi, a e c sono costanti reali positive ed m è tipicamente 2w con w *word-size* dell'architettura *hardware* ($m>N_0$, m>a, m>c). Nel secondo volume - "Seminumerical Algorithms" - dell'opera "The Art of Computer Programming" (disponibile anche presso il servizio libri delle Edizioni Infomedia) l'autore Donald Knuth elenca i seguenti requisiti per un "ottimo" generatore congruenziale lineare (cioè per un generatore che distribuisce in modo sufficientemente uniforme i valori successivi della sequenza):

- N_0 deve essere arbitrario (inizializzazione casuale);
- a mod 8 = 5 per garantire che la periodicità della sequenza sia effettivamente m;
- $a > \sqrt{m}$; ;
- a>m/100;
- $a < m - \sqrt{m}$;
- la configurazione di cifre di a deve essere irregolare;
- c deve essere dispari;
- $\frac{c}{m} \approx \frac{1}{2} - \frac{1}{6}\sqrt{3}$; ;
- le cifre meno significative di N_i devono essere ignorate;
- N_i /m deve essere compreso tra 0 (incluso) ed 1 (escluso);

Al programmatore non viene mai richiesta l'implementazione di un generatore di numeri pseudo-casuali e, di conseguenza, i precedenti requisiti hanno esclusivamente valore informativo. Ma come scegliere N_0 in modo da garantire una inizializzazione veramente casuale della sequenza di valori generata? La *standard library* del C ANSI permette al programmatore

di assegnare, per mezzo della funzione *srand()*, un qualsiasi valore iniziale N_0 e, in caso di omissione, N_0 viene automaticamente posto uguale a 1. Molto spesso nei programmi di simulazione il generatore di numeri pseudo-casuali viene inizializzato come segue:

```
srand((unsigned)time(NULL));
```

La funzione *time()* (standard ANSI) restituisce il valore dell'orologio di sistema come numero di secondi trascorsi dalla mezzanotte del primo gennaio 1970 e le cifre meno significative del valore restituito di fatto rappresentano, in fase di esecuzione del programma, un numero sufficientemente "arbitrario".

Resta il problema della "qualità" - cioè della casualità! - della sequenza pseudo-casuale così generata. Nel già citato libro di Knuth sono indicati alcuni "test di casualità" (test statistico di uniformità chi-quadro con stimatore di Pearson, test spettrale di Knuth) impiegabili per verificare la qualità di un generatore di numeri casuali. Per chi volesse approfondire gli aspetti teorici (matematici, ma anche informatici) delle sequenze casuali di numeri consiglio senz'altro la seguente lettura: G. Chaitin, "Casualità e dimostrazione matematica", Le Scienze (ed. it. di *Scientific American*), Settembre 1975.

RIQUADRO 3
Variabili casuali, distribuzioni teoriche di probabilità e processi stocastici

Secondo la "definizione soggettiva" di Bruno de Finetti una variabile casuale, o variabile aleatoria, è "un valore definito, ma non conosciuto o non ancora conosciuto". Di una variabile aleatoria si conoscono quindi i possibili valori che essa può assumere (appartenenti ad un insieme numerico discreto o continuo, limitato o illimitato) e le relative probabilità di essere, o divenire, il reale valore della variabile. Come già illustrato nel testo dell'articolo per una variabile casuale che assume valori in un insieme numerico discreto e finito (per esempio i valori interi compresi tra 1 e 10) è sufficiente specificare, mediante una tabella o una formula, la probabilità associata a ciascuno dei valori ed espressa con un numero compreso tra 0 e 1, in modo che la somma di tutte le probabilità sia esattamente 1:

$$\sum_{x=\min}^{\max} p(x) = 1$$

La funzione p(x) che associa ciascuno dei possibili valori x alla probabilità p(x) di essere o divenire il valore della variabile aleatoria X prende il nome di "funzione di distribuzione della probabilità". La "funzione di ripartizione della probabilità" P(x) rappresenta invece la probabilità che la variabile X assuma valore minore o uguale ad x; il nome di "probabilità cumulata" con il quale spesso si indica la funzione di ripartizione è dovuto alla sua relazione con la funzione di distribuzione:

$$P(x) = \sum_{n=X_{\min}}^{x} p(n)$$

Ovviamente si ha che

$$P(X_{\max}) = 1$$

Nel caso di variabili casuali discrete, ma illimitate (ad esempio superiormente) la funzione di distribuzione p(x) deve essere tale che

$$\sum_{x=X_{\min}}^{\infty} p(x) = 1$$

e si avrà che

$$\lim_{x\to\infty} P(x) = 1$$

Se una variabile aleatoria assume valori in un insieme continuo illimitato (superiormente e/o inferiormente) la funzione di distribuzione della probabilità f(x) - che non è immediatamente interpretabile in termini di probabilità dei valori assunti - deve soddisfare i due seguenti requisiti

$$0 \le f(x) \le 1$$

$$\int_{-\infty}^{+\infty} f(x)dx = 1$$

e la funzione di ripartizione F(x) - che esprime la probabilità che la variabile assuma un valore minore o uguale a x - è data dall'integrale

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

per cui f(x) è la derivata di F(x), che è una funzione ovviamente sempre crescente e quindi invertibile, e

$$\lim_{x\to-\infty} F(x) = 0$$

$$\lim_{x\to+\infty} F(x) = 1$$

Nel caso che una variabile casuale continua assuma valori in un intervallo limitato [a,b] tutte le relazioni indicate restano valide: è sufficiente infatti sostituire nelle formule a e b come estremi inferiore e superiore. In entrambi i casi la probabilità che la variabile assuma un valore compreso tra x_1 e x è data da:

$$F(x_2) - F(x_1) = \int_{x_1}^{x_2} f(t)dt$$

Le funzioni di distribuzione e di ripartizione della probabilità rappresentano in modo "analitico" una variabile aleatoria discreta o continua: una descrizione "sintetica" della stessa variabile aleatoria può comunque essere ricavata dai cosiddetti indici di concentrazione e di dispersione. Il classico indice di misura della concentrazione dei valori di una variabile casuale è il "valore atteso" (o valore medio):

$$\mu = \sum_{x=X_{\min}}^{X_{\max}} p(x) \cdot x$$

$$\mu = \int_{-\infty}^{+\infty} f(x) \cdot x dx$$

Il tradizionale indice per la misura della dispersione dei valori assunti da una variabile casuale è invece la "varianza" (è il quadrato della quantità nota come "deviazione standard"):

$$\sigma^2 = \sum_{x=X_{\min}}^{X_{\max}} f(x) \cdot (x-\mu)^2$$

$$\sigma^2 = \int_{-\infty}^{+\infty} f(x) \cdot (x-\mu)^2 dx$$

In molti casi concreti una variabile aleatoria può essere convenientemente approssimata per mezzo di una distribuzione di probabilità teorica, come la "distribuzione uniforme" - discreta o continua - di valori in un intervallo numerico che si estende da a (compreso) a b (escluso)

$$p(x) = \frac{1}{b-a}$$

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

la distribuzione discreta di "Poisson", dove il parametro k è uguale al valore atteso ed il suo quadrato k^2 alla varianza

$$p(x) = \frac{k^x}{x!} e^{-k}$$

la distribuzione di Gauss, continua ed illimitata sia inferiormente che superiormente, denominata "normale standard", per la quale il valore atteso è 0 e la varianza è 1

$$\mathrm{f(x)} = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{\frac{-x^2}{2}}$$

e come la distribuzione continua di valori positivi superiormente illimitati nota con il nome di "esponenziale negativa"

$$f(x) = ke^{-kx}$$

che ha come valore atteso il reciproco 1/k del parametro e come varianza il suo quadrato $1/k^2$.
Un processo stocastico è formalmente definito come un insieme di variabili aleatorie i cui valori sono dipendenti dal tempo. Un processo stocastico è detto "stazionario" se le distribuzioni di probabilità delle variabili che lo compongono non variano con il tempo ed è anche "ergodico" se l'evoluzione dei valori assunti dalle variabili non dipende dalle specifiche condizioni iniziali. Un tipico esempio di processo stocastico stazionario ed ergodico è rappresentato dalle "catene di Markov", un modello matriciale probabilistico al quale si ricorre spesso sia in campo teorico che applicativo. Il processo stocastico di Poisson degli eventi distribuiti casualmente nel tempo - di cui si parla anche nel testo dell'articolo come modello degli "arrivi" in una coda - è invece interessante perché presenta aspetti duali nel discreto (il numero di eventi casuali che si verificano nell'unità di tempo avviene secondo la distribuzione di Poisson) e nel continuo (la durata casuale del tempo che trascorre tra un evento ed il successivo segue la distribuzione esponenziale negativa).

RIQUADRO 4
Formule di approssimazione per la distribuzione normale standard di probabilità

La variabile casuale normale standard ha come funzione di distribuzione della probabilità la seguente "funzione di Gauss":

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

In base alla definizione valida per una qualsiasi variabile aleatoria continua, la relativa funzione di ripartizione della probabilità F(x) è data dall'integrale

$$\mathrm{F(x)} = \frac{1}{\sqrt{2\pi}} \int_{x}^{+\infty} \mathrm{e}^{-\frac{t^2}{2}} \mathrm{dt}$$

che risulta essere non esprimibile in termini di funzioni elementari, ma che può essere adeguatamente approssimato per mezzo delle seguenti formule [Abramowitz & Stegun, "Handbook of Mathematical Functions", National Bureau of Standards, 1964]:

$$F(x) \approx f(x) \cdot \left(c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5\right)$$

$$t = \frac{1}{1 + r \cdot |x|}$$

$$r = 0.2316419$$

$$c_1 = 0.31938153$$

$$c_2 = -0.356563782$$

$$c_3 = 1.781477937$$

$$c_4 = -1.821255978$$

$$c_5 = 1.330274429$$

Quest'ultime sono facilmente programmabili con gli operatori tipici di un linguaggio di programmazione *general-purpose* come il C:

```
double F(double x)
{
 const double r=.2316419;
```

```
 const double c1=.31938153;
 const double c2=-.356563782;
 const double c3=1.781477937;
 const double c4=-1.821255978;
 const double c5=1.330274429;
 const double pg=3.141592654;
 double f,t;

 f=(1/sqrt(2*pg))*exp(-pow(x,2)/2);
 t=1/(1+r*fabs(x));
return(f*(c1*t+c2*pow(t,2)+c3*pow(t,3)
+c4*pow(t,4)+c5*pow(t,5));
}
```

Per generare un valore casuale distribuito secondo la legge di Gauss è invece necessario calcolare il valore della funzione inversa di ripartizione $x=F^{-1}(y)$, il cui calcolo approssimato - per y compreso tra 0 e 1 - può essere effettuato mediante le seguenti formule [Abramowitz & Stegun, "Handbook of Mathematical Functions", National Bureau of Standards, 1964]:

$$z = t - \frac{c_0 + c_1 t + c_2 t^2}{1 + k_1 t + k_2 t^2 + k_3 t^3}$$

$$t = \begin{cases} \sqrt{\ln \frac{1}{y^2}}, y \le .5 \\ \sqrt{\ln \frac{1}{(1-y)^2}}, y > .5 \end{cases}$$

$$x = \begin{cases} z, y \le .5 \\ -z, y > .5 \end{cases}$$

$$c_0 = 2.515517$$

$$c_1 = 0.802853$$

$$c_2 = 0.010328$$

$$k_1 = 1.432788$$

$$k_2 = 0.189269$$

$$k_3 = 0.001308$$

Anche in questo caso è immediata l'implementazione in C:

```
double F_1(double y)
{
 const double c0=2.515517;
 const double c1=.802853;
 const double c2=.010328;
 const double k1=1.432788;
 const double k2=.189269;
 const double k3=.001308;
 double t,z;

 if (y<=.5) t=sqrt(log(1/pow(y,2)));
 else t=sqrt(log(1/pow((1-y),2)));
z=t((c0+c1*t+c2*pow(t,2))/(k1*t+k2*pow
t,2)+k3*pow(t,3)));
 if (y<=.5) return(z);
 else return (-z);
}
```

LISTATO 1
Uso del generatore di numeri casuali nei programmi di simulazione (file: radio.c)

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

#define FALSE 0
#define TRUE 1
#define N0 10000 // numero iniziale di atomi

const double k=0.0005; // costante di disintegrazione radioattiva
const unsigned Ti=0; // tempo iniziale esperimento simulato
const unsigned Tf=100; // tempo finale esperimento simulato
const double dt=1.; // intervallo di tempo per la simulazione discreta
const unsigned NumEsper=10; // numero di esperimenti completi da ripetere

double ProbDis(double t0, double t1, unsigned N);

void main (void)
{
 char Atomi[N0];
 unsigned Freq[N0+1];
 double t,t0,t1,nc;
 unsigned NumAtomi, NumDis, Atomo, i, Esper, NumMaxDis;

 for (i=0; i<=N0; i++) Freq[i]=0;
 NumMaxDis=0;
 for (Esper=1; Esper<=NumEsper; Esper++) // “ciclo degli esperimenti”
    {
     printf(“\nEsperimento %u”,Esper);
     srand((unsigned)time(NULL));
     for (Atomo=0; Atomo<N0; Atomo++)
        Atomi[Atomo]=TRUE; // “vivo”
     t=Ti;
     NumAtomi=N0;
     while (t<Tf) // “ciclo del tempo”
          {
           // simulazione intervallo di tempo [t,t+dt)
           t0=t; t1=t+dt;
           printf(“\n%f-%f: “,t0,t1);
           NumDis=0;
           for (Atomo=0; Atomo<N0; Atomo++) // “ciclo dello spazio”
              {
               if (Atomi[Atomo])
                 {
                  // generazione di un numero casuale 0<=nc<1
                  nc=((double)rand())/RAND_MAX;
                  if (nc<ProbDis(t0,t1,NumAtomi))
                    {
                     // disintegrazione
                     Atomi[Atomo]=FALSE; // “morto”
                     NumDis++;
                    }
                 }
              }
           printf(“%u”,NumDis);
           Freq[NumDis]++;
           if (NumDis>NumMaxDis) NumMaxDis=NumDis;
```

```
              NumAtomi=NumAtomi-NumDis;
            t=t+dt; // avanzamento discreto del tempo simulato
           }
    }
 printf("\nFrequenze assolute e relative");
 printf("\ndel numero di disintegrazioni per unita` di tempo:");
 for (i=0; i<=NumMaxDis; i++)
    {
     double FreqRel=((double)Freq[i])/(NumEsper*((Tf-Ti)/dt));
     printf("\n%u: %u - %f",i,Freq[i],FreqRel);
    }
}

double ProbDis(double t0, double t1, unsigned N)
{
 double prob=(((double)N0)/N)*(exp(-k*t0)-exp(-k*t1));
 return prob;
}
```

LISTATO 2
Uso del generatore di numeri casuali nei programmi di simulazione (file: mm1.c)

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

#define NumArrivi 1000
#define IntertempoMedioArrivi 3.
#define TempoMedioServizio 2.

const double k=1/IntertempoMedioArrivi;
const double c=1/TempoMedioServizio;

typedef double Tempo;

Tempo ProssimoArrivo(Tempo t)
{
 /*
   metodo di Monte Carlo e della funzione di ripartizione
   inversa per la generazione di una variabile casuale
   continua con distribuzione di probabilità esponenziale
   negativa
 */
 double nc,val;
 nc=((double)rand())/RAND_MAX;
 val=-log(1.-nc)/k;
 return (t+val);
}

Tempo Durata()
{
 /*
   metodo di Monte Carlo e della funzione di ripartizione
   inversa per la generazione di una variabile casuale
    continua con distribuzione di probabilità esponenziale
```

```
      negativa
 */
 double nc,val;
 nc=((double)rand())/RAND_MAX;
 val=-log(1.-nc)/c;
 return (val);
}

void main()
{
 long LunMedia;
 Tempo TempoSimulato, Prossimo, Delta;
 Tempo Coda[NumArrivi];
 /*
   vettore Coda[Tempo]: per ogni unità in coda memorizza il
   tempo di servizio
 */
 int Primo, Ultimo, LunCoda, Arrivi;

 srand((unsigned)time(NULL));
 LunMedia=0;
 Primo=Ultimo=0;
 LunCoda=0;
 TempoSimulato=0;
 for (Arrivi=1; Arrivi<=NumArrivi; Arrivi++)
    {
     /*
       simulazione intervallo di tempo tra un arrivo
       e il successivo
     */
     Prossimo=ProssimoArrivo(TempoSimulato);
     Delta=Prossimo-TempoSimulato;
     while ((LunCoda>0)&&(Delta>0))
          {
           /*
             simulazione servizio: estrazioni da coda
           */
           if (Delta>=Coda[Primo])
             {
              Delta=Delta-Coda[Primo];
              Primo=(Primo+1) % NumArrivi;
              LunCoda=LunCoda-1;
             }
           else
               {
                Coda[Primo]=Coda[Primo]-Delta;
                Delta=0;
               }
          }
     printf("\n%d",LunCoda);
     LunMedia=LunMedia+LunCoda;
     /*
       memorizzazione tempo di servizio nuovo arrivo:
     */
     Coda[Ultimo]=Durata();
     /*
       simulazione nuovo arrivo:
     */
     Ultimo=(Ultimo+1) % NumArrivi;
     LunCoda=LunCoda+1;
     TempoSimulato=Prossimo;
    }
 printf("\n\nLunghezza media della coda: %f",\
        (((double)LunMedia)/((double)NumArrivi)));
}
```

LISTATO 3
Uso del generatore di numeri casuali nei programmi di simulazione (file: telecom.c)

```
  /*
  Il programma 'telecom.c' simula il sistema tempo-discreto costituito da
  una centralina per telecomunicazioni servente - con un numero massimo di
  collegamenti contemporaneamente attivi - una stella di utenze telefoniche.
  Le chiamate sono distribuite nel tempo secondo la legge di Poisson (i
  relativi intertempi sono quindi distribuiti secondo la legge esponenziale
  negativa) e la loro durata e` distribuita secondo la legge normale di
  Gauss. La determinazione delle utenze di volta in volta chiamanti e
  chiamate avviene secondo la distribuzione uniforme, rispettivamente
  sull'insieme di tutte le utenze libere e sull'insieme di tutte le utenze
  diverse dalla chiamante.
*/

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

#define NumChiamate 1000 /* numero chiamate da simulare */
#define NumCollegamenti 10 /* numero massimo di collegamenti attivabili */
#define NumUtenze 1000 /* numero utenze servite */
#define IntertempoMedioChiamate 1. /* intervallo medio di tempo tra 2
                                      chiamate successive (min.) */
#define DurataMediaChiamate 5. /* durata media di una telefonata (min.) */
#define VarianzaDurataChiamate 2. /* varianza della durata di una tel. */

typedef double TEMPO;
enum BOOL {FALSE,TRUE};
enum STATO {LIBERO,OCCUPATO};
struct COLLEGAMENTO
       {
        enum BOOL Impegnato;
        int UtenzaChiamante;
        int UtenzaChiamata;
        TEMPO FineCom;
       };

const double k=1/IntertempoMedioChiamate;
const double c=1/DurataMediaChiamate;
TEMPO TempoSimulato,ProssimaChiamata;
int NumTel; /* numero di chiamate simulate */
int NumOcc; /* numero di chiamate respinte per 'utenza occupata' */
int NumRes; /* numero di chiamate respinte per assenza di collegamenti
                liberi */
int NumCom; /* numero di comunicazioni in corso */
int Conta;
enum Stato StatoUtenza[NumUtenze];
struct COLLEGAMENTO Centralina[NumCollegamenti];
int UtenzaRichiedente,UtenzaRichiesta,Collegamento;

void Randomize()
{
 /* inizializzazione casuale del generatore di numeri casuali */
 srand((unsigned)time(NULL));
}

double Random()
{
 /* generazione di un numero casuale nell'intervallo [0,1) */
  return(rand()/32768.);
```

```
  }

TEMPO Prossima(TEMPO t)
{
 /*
   tempo simulato di arrivo chiamata ->
   tempo simulato di arrivo prossima chiamata:
   distribuzione esponenziale negativa degli intertempi di arrivo
 */
 double Val;

 Val=-log(1.-Random())/k;
 return (t+Val);
}

TEMPO Durata()
{
 /*
   -> tempo simulato di durata della chiamata attuale:
   distribuzione normale
 */
 const double c0=2.515517;
 const double c1=.802853;
 const double c2=.010328;
 const double k1=1.432788;
 const double k2=.189269;
 const double k3=.001308;
 double nc,StdVal,Val,t;

 nc=Random();
 if (nc<=.5) t=sqrt(log(1/pow(nc,2)));
 else t=sqrt(log(1/pow((1-nc),2)));
 StdVal=t-((c0+c1*t+c2*pow(t,2))/(k1*t+k2*pow(t,2)+k3*pow(t,3)));
 if (nc>.5) StdVal=-StdVal;
 Val=sqrt(VarianzaDurataChiamate)*StdVal+DurataMediaChiamate;
 if (Val<0) Val=0;
 return (Val);
}

int Chiamante()
{
 /*
   -> utenza chiamante:
   distribuzione uniforme sull'insieme di tutte le utenze libere
 */
 int caso,conta,indice;

 caso=(int)floor(Random()*(NumUtenze-NumCom*2));
 conta=0;
 while (conta<=caso)
       {
        if (StatoUtenza[indice]==LIBERO)
          conta++;
        indice++;
       }
 return(indice);
}

int Chiamata(int chiamante)
{
 /*
   utenza chiamante -> utenza chiamata:
   distribuzione uniforme sull'insieme di tutte le utenze diverse dalla
   chiamante
 */
```

```
   int caso;

 caso=(int)floor(Random()*(NumUtenze-1));
 if (caso<chiamante) return(caso);
 else return (caso+1);
}

int CollegamentoLibero()
{
 /* -> collegamento libero, 0 se tutti attivi */
 int collegamento;

 collegamento=0;
 while (Centralina[collegamento].Impegnato)
      collegamento++;
 if (collegamento<=NumCollegamenti)
   return(collegamento);
 else
     return(0);
}

void main()
{
 /* inizializzazione */
 Randomize();
 for (Conta=0; Conta<NumCollegamenti; Conta++)
    Centralina[Conta].Impegnato=FALSE;
 for (Conta=0; Conta<NumUtenze; Conta++)
    StatoUtenza[Conta]=LIBERO;
 NumRes=NumOcc=0;
 TempoSimulato=0;
 ProssimaChiamata=Prossima(TempoSimulato);
 Centralina[0].Impegnato=FALSE;
 Centralina[0].UtenzaChiamante=Chiamante();
 Centralina[0].UtenzaChiamata=\
  Chiamata(Centralina[0].UtenzaChiamante);
 Centralina[0].FineCom=TempoSimulato+Durata();
 StatoUtenza[Centralina[0].UtenzaChiamante]=OCCUPATO;
 StatoUtenza[Centralina[0].UtenzaChiamata]=OCCUPATO;
 NumCom=1;
 /*
   ciclo di simulazione del susseguirsi degli eventi nel tempo; al suo
   interno si simulano gli eventi che avvengono nell'intervallo di tempo
   (TempoSimulato,ProssimaChiamata]:
 */
 for (NumTel=1; NumTel<NumChiamate; NumTel++)
    {
     /* eventuale terminazione di comunicazioni: */
     for (Conta=0; Conta<NumCollegamenti; Conta++)
        if ((Centralina[Conta].Impegnato)&&\
             (Centralina[Conta].FineCom<=ProssimaChiamata))
          {
           Centralina[Conta].Impegnato=FALSE;
           StatoUtenza[Centralina[Conta].UtenzaChiamante]=LIBERO;
           StatoUtenza[Centralina[Conta].UtenzaChiamata]=LIBERO;
           NumCom—;
          }
     /* nuova chiamata, eventualmente nuova comunicazione: */
     TempoSimulato=ProssimaChiamata;
     ProssimaChiamata=Prossima(TempoSimulato);
     if (NumCom==NumCollegamenti) NumRes++;
     else
         {
          UtenzaRichiedente=Chiamante();
          UtenzaRichiesta=Chiamata(UtenzaRichiedente);
```

```
            if (StatoUtenza[UtenzaRichiesta]==OCCUPATO) NumOcc++;
          else
              {
               Collegamento=CollegamentoLibero();
               Centralina[Collegamento].Impegnato=TRUE;
               Centralina[Collegamento].UtenzaChiamante=UtenzaRichiedente;
               Centralina[Collegamento].UtenzaChiamata=UtenzaRichiesta;
               Centralina[Collegamento].FineCom=TempoSimulato+Durata();
               StatoUtenza[UtenzaRichiedente]=OCCUPATO;
               StatoUtenza[UtenzaRichiesta]=OCCUPATO;
               NumCom++;
              }
          }
    }
 printf("\n#%d Telefonate",NumTel);
 printf("\n#%d 'Utenza occupata'",NumOcc);
 printf("\n#%d 'Chiamata respinta'",NumRes);
}
```