Generazione di numeri pseudo-casuali

Corso Ottimizzazione Combinatoria (IN440)

Prof. M. Liverani

Numeri casuali

- I computer sono degli strumenti **fortemente deterministici**: ogni azione/operazione eseguita da un computer è la *conseguenza prevedibile* e *ben determinata* dell'esecuzione di un'istruzione di un programma
- Tuttavia spesso è utile produrre sequenze di numeri **apparentemente** casuali
- Una **variabile aleatoria** X si dice **uniformemente distribuita** (*uniforme*) se per ognuno dei suoi possibili n valori $\{x_1, x_2, \ldots, x_n\}$ la probabilità corrispondente è $p = 1/n$
- Non tutte le variabili aleatorie hanno distribuzione uniforme
- Esempio: sia X il valore prodotto come somma del lancio di ***due*** dadi

$$
\begin{aligned}
2 &= 1+1 \\
3 &= 1+2 = 2+1 \\
4 &= 1+3 = 2+2 = 3+1 \\
5 &= 1+4 = 2+3 = 3+2 = 4+1 \\
6 &= 1+5 = 2+4 = 3+3 = 4+2 = 5+1 \\
7 &= 1+6 = 2+5 = 3+4 = 4+3 = 5+2 = 6+1 \\
8 &= 2+6 = 3+5 = 4+4 = 5+3 = 6+2 \\
9 &= 3+6 = 4+5 = 5+4 = 6+3 \\
10 &= 4+6 = 5+5 = 6+4 \\
11 &= 5+6 = 6+5 \\
12 &= 6+6
\end{aligned}
$$

Successioni casuali

- Consideriamo le seguenti successioni di numeri: sono casuali?
  - 1 1 1 1 1 1 1 1 1 1 1 … → n lanci di un dato
  - 1 4 1 5 9 2 6 5 3 5 8 9 … → le prime 12 cifre decimali del numero π
  - 640 231 100 91 1003 … → i numeri snocciolati da Leporello a Donna Elvira nell'aria «*Madamina, il catalogo è questo*» del Don Giovanni di Mozart

- Sono state date molte definizioni per circoscrivere il concetto di **successione casuale di numeri**:
  - **Richard von Mises**: una successione numerica è casuale se non è possibile alterarne la statistica eliminandone una sotto-successione
  - **Andrei Kolmogorov**: una successione numerica è casuale se non può essere prodotta da un programma per computer che sia sensibilmente più breve della successione stessa
  - **Andrei Kolmogorov**: una successione numerica è casuale se non può essere «compressa»
  - …

Successioni di numeri pseudo-casuali

- Un **generatore di numeri pseudo-casuali** è un algoritmo il cui schema generale è molto semplice: si parte dal primo valore (s) della sequenza pseudo-causale, che si chiama **seme** della sequenza, e, ogni volta che si deve generare un nuovo valore, si applica una stessa funzione al valore generato precedentemente:

$$x_0 = s$$
$$x_n = f(x_{n-1})$$

- La bontà del generatore di numeri pseudo-casuali nel produrre una sequenza che somigli effettivamente a una successione casuale sta nella funzione f, che andrà scelta in modo che:
  - I valori x_n siano valori **uniformemente distribuiti** in un certo intervallo: per esempio fra 0 e 1 se sono numeri decimali o fra 0 e un certo N fissato se sono interi
  - I valori x_n siano (o meglio comportarsi come se fossero) **mutuamente indipendenti**

La seconda condizione non è realmente possibile soddisfarla: va quindi intesa nel senso che il **periodo della successione** pseudo-casuale deve essere il più grande possibile

Generatore di von Neumann

- John von Neumann (anni ‘40) ha definito il seguente generatore:

  dato x_{n-1} calcolo $x_n = f(x_{n-1})$ prendendo le quattro cifre centrali del quadrato di x_{n-1}

  $x_0 = s$ è un numero qualsiasi di quattro cifre (compreso quindi tra 1.000 e 9.999)

- Esempio:
  - $s = 1234$
  - $x_1 = 5227$
  - $x_2 = 3215$
  - $x_3 = 3362$
  - $x_4 = 3030$
  - $x_5 = 1809$
  - $x_6 = 2724$
  - ...

Algoritmo 1

f(*x*):
  y = x^2
  y = *y* / 100
  y = *y* % 10.000
  return *y*

```
def f(x):
    y = int(x**2/100) % 10000
    return y

x = int(input("Seme: "))
while x > 0 and x % 100 > 0:
    x = f(x)
    print(x)
```

Generatore con successione di quadrati

- Si può ottenere un comportamento apparentemente caotico anche attraverso la reiterazione di una funzione semplice come $f(x_{n-1}) = (x_{n-1})^2 - 2$
- Esempio:
  - $s = 0{,}5$
  - $x_1 = -1.75$
  - $x_2 = 1.0625$
  - $x_3 = -0.871094$
  - $x_4 = -1.241196$
  - $x_5 = -0.459433$
  - $x_6 = -1.788921$
  - …

Algoritmo 2

f(*x*):
  $y = x^2 - 2$
return *y*

```
def f(x):
    y = x**2 - 2
    return y

c = 0
x = int(input("Seme: "))
while c < 100:
    x = f(x)
    c = c+1
    print(x)
```

Generatore con congruenze lineari

- Una tecnica molto diffusa fino ad alcuni anni fa (era nella libreria standard del linguaggio Fortran) si basa sulle congruenze lineari
- Fissati tre numeri interi a, b e m si può ottenere un comportamento apparentemente caotico attraverso la reiterazione della funzione $f(x_{n-1}) = a\, x_{n-1} + b \bmod m$
- Esempio:
  - Siano $a = 1.103.515.245$, $b = 12.345$ e $m = 32.768$
  - Sia $x_0 = 17$
  - $x_2 = 25974$
  - $x_3 = 22391$
  - $x_4 = 12260$
  - $x_5 = 3149$
  - $x_6 = 25346$
  - …

Algoritmo 3

f(*x*):
  y = (*ax* + *b*) mod *m*
return *y*

- Attenzione: questo generatore produce numeri non proprio casuali…!

```
def f(x):
    a = 1103515245
    b = 12345
    m = 32768
    y = (a * x + b) % m
    return y

c = 0
x = int(input("Seme: "))
while c < 100:
    x = f(x)
    c = c+1
    print(x)
```

Python: il modulo «random»

- Il modulo «**`random`**» rende disponibili funzioni per la generazione di numeri casuali con varie distribuzioni di probabilità
- Utilizza l'algoritmo «Marsenne Twister» come generatore di numeri pseudo-casuali; è un buon generatore, ma non è adatto ad ambiti crittografici
- Si carica il modulo «`random`» con la seguente istruzione:

  ```
  from random import *
  ```

  oppure con l'istruzione:

  ```
  import random as rnd
  ```

- La funzione principale del modulo è «**`random()`**» che genera numeri casuali x: $0 \leq x < 1$
- Il generatore dei numeri casuali viene inizializzato con la data e l'ora corrente se non diversamente specificato; altrimenti si può usare la funzione

  ```
  random.seed(x)
  ```

  con cui è possibile specificare un seme x per l'inizializzazione del generatore

Python: il modulo «random»

Altre funzioni utili:

- `random.randrange(`x_1`, `x_2`, `*step*`)`: genera un numero casuale nell'insieme $\{x_1, x_1 + step, x_1 + 2\ step, x_1 + 3\ step, \ldots, x_2 - step\}$
- `random.randint(`x_1`, `x_2`)`: genera un numero intero casuale nell'insieme $\{x_1, x_1 + 1, x_1 + 2, \ldots, x_2\}$
- `random.choice(`*lista*`)`: restituisce un elemento a caso tra quelli presenti nella *lista*
- `random.shuffle(`*lista*`)`: permuta in modo casuale gli elementi della *lista*

Python: il modulo «secrets»

- Il modulo **`secrets`** è utilizzato per generare numeri pseudo-casuali robusti, con un generatore adatto ad applicazioni crittografiche
- `secrets.SystemRandom(`*`s`*`)`: crea un oggetto per la generazione di numeri casuali x $(0 \leq x < 1)$ con seme s; si può usare l'oggetto con questa sintassi:

```
a = secrets.SystemRandom(s)
a.random()
```

- `secrets.choice(`*`list`*`)`: seleziona un elemento casuale dalla lista
- `secrets.randbelow(`*`x`*`)`: produce un numero intero casuale maggiore o uguale a 0 e minore di x