Performance Modeling of Computer Systems and Networks

Prof. Vittoria de Nitto Personè

Generating Discrete Random Variates

Università degli studi di Roma Tor Vergata
Department of Civil Engineering and Computer Science Engineering

1

Prerequisite

We assume the knowledge of discrete random variables (sect.6.1).
In particular:

- *Equilakely*(a,b)
- *Geometric*(p)
- *Bernoulli*(p)
- *Binomial*(n,p)
- *Pascal*(n,p)
- *Poisson*(μ)

2

in sis2.c equilikely fa chiamata random e poi calcola qualcosa.

sis2.c

```
#include <stdio.h>
#include "rng.h"

#define MINIMUM   20
#define MAXIMUM   80
#define STOP     100     /* 100 weeks = about 2 years*/
#define sqr(x)    ((x) * (x))

long Equilikely(long a, long b)
{  return (a + (long) ((b - a + 1) * Random()));}

long GetDemand(void)
{
        return (Equilikely(10, 50)); }
```

3

anche esponenziale fa qualcosa, manipolando un random.
uguale anche la uniform (il continuo dell'equilikely)

ssq2.c distribution-driven simulation

```
#include <stdio.h>
#include <math.h>
#include "rng.h"
#define LAST          10000L    /* number of jobs processed */
#define START         0.0

double Exponential(double m)                /* ------*
{return (-m * log(1.0 - Random())); }          m > 0.0
                                             ------- */
double Uniform(double a, double b)          /* ------*
{return (a + (b - a) * Random());    }         a < b
                                            * -------*/

          double GetArrival(void)
        {static double arrival = START;
          arrival += Exponential(2.0);
               return (arrival);}

          double GetService(void)
        {return (Uniform(1.0, 2.0));}
```

Prof. Vittoria de Nitto Personè 4

4

cdf = cumulativa.
uno F* perchè nel caso var. discrete non è una vera inversa, prende numero (0,1) che è probabilità e ci dà x che corrisponde alla cumulativa in x.
la cumulativa in x dovrebbe avere quel valore di probabilità.

Preliminary Definitions

X random variable, $F(\cdot)$ is the cdf of X

The *inverse distribution function* (idf) of X is the function

$F^* : (0, 1) \rightarrow \chi$, $\forall\, u \in (0, 1)$

$$F^*(u) = \min_x \{x : u < F(x)\}$$

that is, if $F^*(u)=x$, x is the smallest possible value of X for which $F(x)$ is greater than u

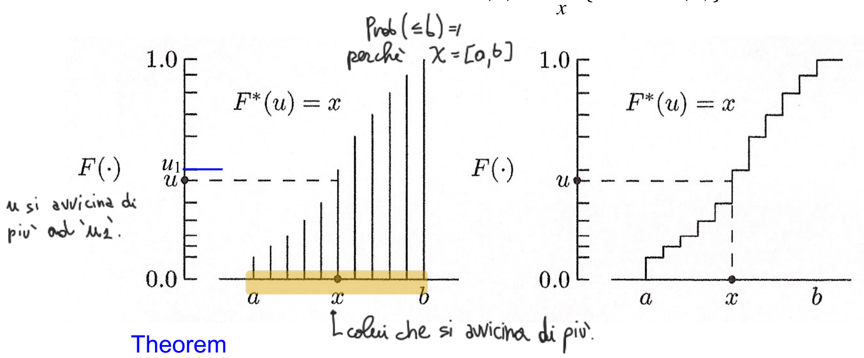
non è esattamente quell'x, ma il più piccolo u tale che F(x) sia minore di u.

5

il supporto χ assume valori discreti da "a" a "b". definisco anche prob X<=x = u1

l'inversa è definita come prima

- $\chi = \{a, a+1, \ldots, b\}$, where b may be ∞, $F(\cdot)$ is the cdf of X,
- $F(x)=\text{Prob}\{X \le x\}=u_1 > u$ $\qquad F^*(u) = \min_x \{x : u < F(x)\}$



Theorem

- if $u < F(a)$, $F^*(u) = a$
- else $F^*(u) = x$ where $x \in \chi$ is the unique possible value of X for which $F(x-1) \le u < F(x)$

6

come lo trovo? modo 1, parto dal minimo, incremento finchè rispetto la condizione. con molti valori è ricerca lineare lentissima.

Algorithm 1

```
x = a;
while (F(x) <= u)
        x++;
return x;            /*x is F*(u)*/
```

Average case analysis:
- let Y be the number of `while loop` passes
- $Y = X - a$
- $E[Y] = E[X - a] = E[X] - a = \mu - a$

Linear search algorithm !

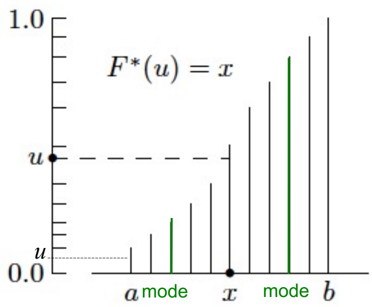
parto dalla moda (il piu probabile). da li mi muovo avanti o indietro.

Idea: start at a more likely point

For $\mathcal{X}=\{a, a+1, \ldots, b\}$, a more efficient linear search algorithm defines $F^*(u)$

Algorithm 2

```
x = mode;          /*initialize with the mode of X */
if (F(x) <= u)
    while (F(x) <= u)
        x++; (avanti)
else if (F(a) <= u)
        while (F(x-1) > u)
          x--; (indietro)
    else
        x = a;
return x;          /*x is F*(u)*/
```



For large $\mathcal{X}$, consider binary search con funzioni semplice si possono scrivere inverse esplicite.

Idf Examples

- In some cases $F^*(u)$ can be determined explicitly
- If X is *Bernoulli(p)* and $F(x) = u$, then x=0 iff $0 < u < 1$-p

$\rightarrow \begin{cases} 1 & u > p \\ 0 & 0 < u < 1-p \end{cases}$

$$F^*(u) = \begin{cases} 0 & 0 < u < 1-p \\ 1 & 1-p \leq u < 1 \end{cases}$$

9

posso fare questa cosa perchè:

Random Variate Generation By Inversion

- X is a discrete random variable with idf $F^*(\cdot)$ (esplicita)
- continuous random variable U is *Uniform*(0,1)
- Z is the discrete random variable defined by $Z = F^*(U)$

Theorem
Z and X are identically distributed

this Theorem allows any discrete random variable (with known idf) to be generated with one call to `Random()`

Algorithm 3

```
u = Random();
return F*(u);
```
→ l'inversa nel punto 'u'.

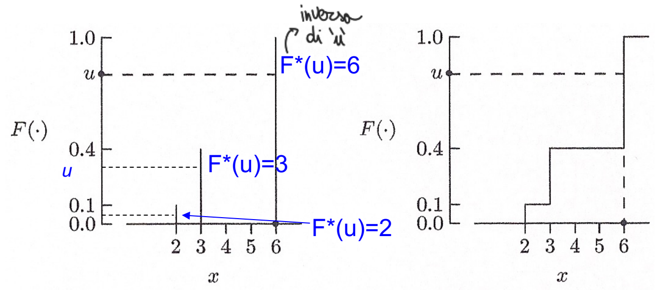
10

Inversion Examples

- Consider X with pdf

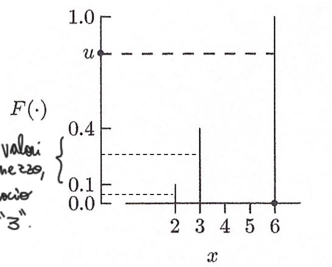
$$f(x) = \begin{cases} 0.1 & x = 2 \\ 0.3 & x = 3 \\ 0.6 & x = 6 \end{cases}$$

- The cdf for X is plotted using two formats




```
if (u < 0.1)
    return 2;
else if (u < 0.4)
        return 3;
    else
        return 6;
```

∃ ∞ valori in mezzo, a cui associo sempre "3".



This algorithm returns
2 with probability 0.1,
3 with probability 0.3 and
6 with probability 0.6.
This corresponds to the pdf of X.

more efficiency: check the ranges for *u* associated with $x = 6$ first (the mode), then $x = 3$, then $x = 2$

- problems may arise when $|\mathcal{X}|$ is large or infinite

More inversion examples

Generating a *Bernoulli*(*p*) random variate

```
u = Random();
if (u < 1-p)
        return 0;
else
        return 1;
```

Generating an *Equilikely*(*a*,*b*) random variate

```
u = Random();
return a + (long) (u * (b - a + 1));
```


Library `rvgs`

pseudo-rand by lehmer

- Includes 6 discrete random variate generators (as below) and 7 continuous random variate generators
  - `long Bernoulli(double p)`
  - `long Binomial(long n, double p)`
  - `long Equilikely(long a, long b)`
  - `long Geometric(double p)`
  - `long Pascal(long n, double p)`
  - `long Poisson(double μ)`
- Functions Bernoulli, Equilikely, Geometric use inversion; essentially ideal
- Functions Binomial, Pascal, Poisson do not use inversion

Library `rvms`

- Provides accurate pdf, cdf, idf functions for many random variates
- Idfs can be used to generate random variates by inversion
- Functions `idfBinomial`, `idfPascal`, `idfPoisson` may have high marginal execution times
- Not recommended when many observations are needed due to time inefficiency
- Array of cdf values with inversion may be preferred

a volte è meglio usare un sottoinsieme, sia per realismo che per efficienza.

Truncation

Sometimes, the realistic values of a variable are restricted to a subset

X random variable with possible values $\chi=\{0, 1, 2, \ldots\}$ and cdf $F(x)=\Pr(X \le x)$

voglio • want to restrict X to the finite range $0 \le a \le x \le b < \infty$

- if $a > 0$, taglio $\alpha = \Pr(X < a)$, $\beta = \Pr(X > b)$

coda sx: $\alpha = \Pr(X < a) = \Pr(X \le a\text{-}1) = F(a\text{-}1)$ (cumulativa)

coda dx: $\beta = \Pr(X > b) = 1\text{-}\Pr(X \le b) = 1\text{-}F(b)$ (1 - cumulativa)

$$\Pr(a \le X \le b) = \Pr(X \le b) - \Pr(X < a) = F(b) - F(a\text{-}1)$$

essentially, always true iff $F(b) \cong 1.0$ and $F(a\text{-}1) \cong 0.0$

Specifying truncation points

- if a and b are specified (punti limite)

  Left-tail, right-tail probabilities α and β obtained using cdf

  $$\alpha = \Pr(X < a) = F(a\text{-}1) \quad \text{and} \quad \beta = \Pr(X > b) = 1\text{-}F(b)$$

  transformation is exact

- if α and β are specified

  idf can be used to obtain a and b

  $$a = F^*(\alpha) \quad \text{and} \quad b = F^*(1\text{-}\beta)$$

  transformation is not exact because X is discrete

  $$\Pr(X < a) \leq \alpha \quad \text{and} \quad \Pr(X > b) < \beta$$

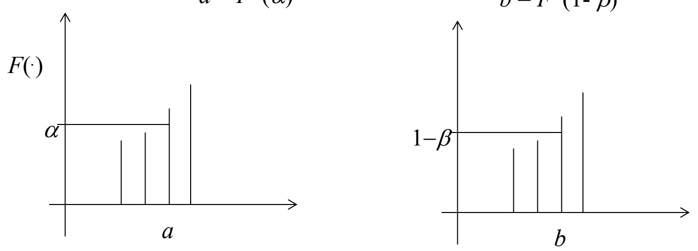
F* è approssimazione inversa, non è "esatta", perchè passo
17 dal continuo ad alcuni valori discreti.

$F(x-1) \leq u < F(x)$

Specifying truncation points

- if α and β are specified

$a = F^*(\alpha)$ $\qquad$ $b = F^*(1\text{-}\beta)$

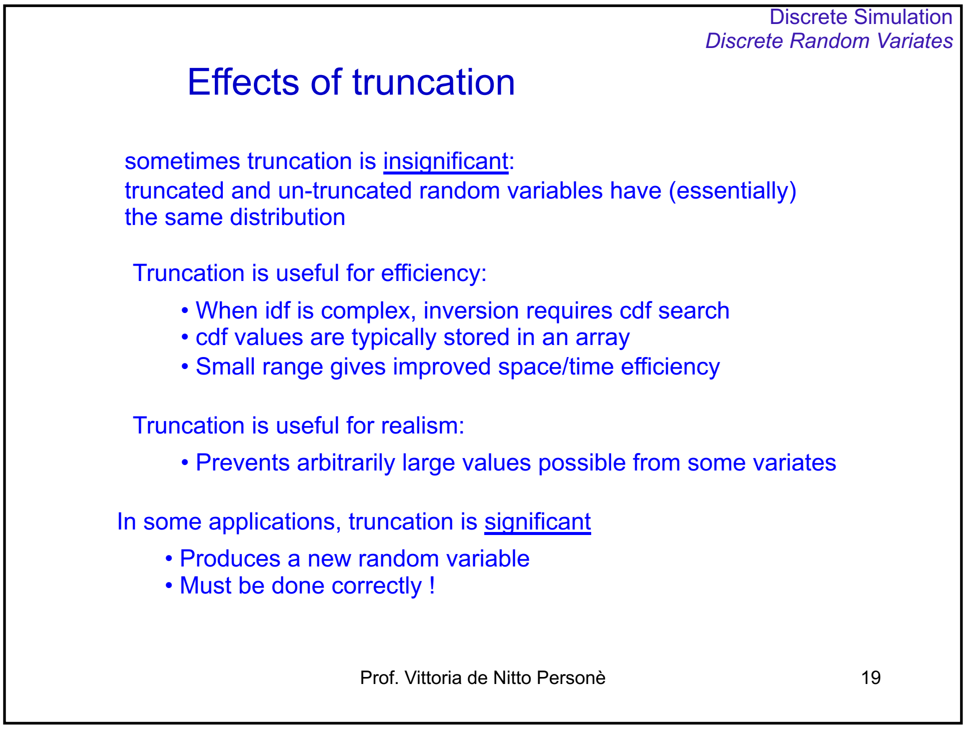


$$F(a) > \alpha$$
$$\Pr(X < a) \leq \alpha$$

$$F(b) > 1\text{-}\beta$$
$$\Pr(X \leq b) > 1\text{-}\beta$$
$$\text{-}\Pr(X \leq b) < \beta\text{-}1$$
$$1\text{-}\Pr(X \leq b) < \beta$$
$$\Pr(X > b) < \beta$$

18
qui vediamo come calcolare i vari passaggi.
ho una disuguaglianza, con un <=, mentre beta è <. quindi non sono "esatti".

gli effetti sono insignificanti se tolgo poco, ma può avere senso per efficienza, se la funzione è complessa, per realismo. Se la troncata è molto diversa, ho nuova var.random!



19