PAC - PROBABILITA' AL CALCOLATORE 2004
RUDIMENTI SULLA SIMULAZIONE DI VARIABILI ALEATORIE

Numeri pseudo–random. La simulazione di (sequenze di) numeri aleatori tramite calcolatore è un argomento molto studiato e con notevoli applicazioni. Per chi fosse interessato, al tema sono dedicati ampi spazi sul web, si veda per es. il sito `http://random.mat.sbg.ac.at`. Per gli scopi di questo corso ci accontentiamo delle funzioni standard incluse nei comuni linguaggi di programmazione.

Nel linguaggio `C` (l'unico a cui faremo esplicito riferimento in seguito) per esempio, si puo' utilizzare per esempio la funzione `drand48()`, che fornisce un numero reale pseudo-random nell'intervallo $[0,1]$. La funzione `drand48()` utilizza un algoritmo *lineare congruenziale*, si basa cioé sull'iterazione di una mappa del tipo

$$X_{n+1} = \big(aX_n + c\big)_{\mathrm{mod}\ m}\,, \qquad n = 0, 1, 2, \dots$$

dove m è un intero molto grande e a e c sono interi scelti in maniera opportuna. Il punto di partenza X_0 della iterazione è detto *seme*. Il valore del seme è fissato tramite il comando `srand48(seme)`.

Variabili uniformi in $[0,1]$**.** Vediamo alcuni esempi di uso concreto della funzione `drand48()`. Il semplice programma riportato qui sotto produce e mostra un numero r nell'intervallo $[0,1]$.

```
/* Una variabile in [0,1] con seme fissato */
# include   <stdlib.h>
int main(void)
{
      double r;
      srand48(127657);
      r = drand48();
      printf("r=%f", r);
      return 0;
}
```

Poiché il seme è fissato (pari a 127657 nell'esempio) il numero r sarà sempre lo stesso ogni volta che si fa girare il programma. Il seme va quindi cambiato a ogni giro per avere *sequenze* di numeri pseudo–random. Per comodità si può aggiornare il seme automaticamente tramite il timer interno della macchina, come illustrato dal seguente programma.

```
/* Una variabile in [0,1] con seme aggiornato */
# include   <stdlib.h>
# include   <time.h>  /* Notare l'inclusione della libreria <time.h> */
int main(void)
{
      double r;
      srand48((unsigned)time(NULL));
      r = drand48();
      printf("r=%f", r);
      return 0;
}
```

Note del corso di PAC, per informazioni: Pietro Caputo, `caputo@mat.uniroma3.it`.

Questa volta il programma restituisce un valore di r ogni volta differente, che per i nostri scopi può essere interpretato come una variabile "distribuita uniformemente" in $[0,1)$. Inoltre, il valore di r ad ogni esecuzione è con buona approssimazione *indipendente* dai valori ottenuti nelle esecuzioni precedenti.

Media empirica e scarto quadratico medio. La bontà del generatore cosi ottenuto verrà testata in seguito in diversi modi. Una prima verifica si può basare sul calcolo della *media empirica*

$$m_n = \frac{1}{n}\sum_{i=1}^{n} r_i\,, \tag{0.1}$$

dove $r_1, r_2, \ldots, r_n$ sono i numeri ottenuti attraverso n successive applicazioni del programma.

Un'altra verifica della bontà del generatore può essere basata sullo *scarto quadratico medio*

$$v_n = \frac{1}{n}\sum_{i=1}^{n}(r_i - m_n)^2 = \frac{1}{n}\sum_{i=1}^{n} r_i^2 - m_n^2\,. \tag{0.2}$$

La grandezza v_n verrà anche chiamata varianza empirica. Vediamo ora un primo programma che permette di calcolare media e varianza empirica per n variabili indipendenti, uniformi in $[0,1]$.

```
/*  n variabili indipendenti uniformi in [0,1] */

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

int main(void)
{
      double r,m,v;
      int   k,n;
      srand48((unsigned)time(NULL));

      printf("numero variabili da simulare:");
      scanf("%d",&n);

      m=0;
      v=0;
      for (k=0; k < n; k++){
         r = drand48();
         m+=r;
         v+=r*r;}
      m = ((double) m / n);
      v = ((double) v / n) - m*m;
      printf("media empirica = %12f", m);
      printf("varianza empirica = %12f", v);
      return 0;
}
```

Si osserverà che al crescere di n, le grandezze m_n e v_n si stabilizzano intorno ai valori teorici

$$m_n \sim \int_0^1 x\,dx = \frac{1}{2}\,, \qquad v_n \sim \int_0^1 \big(x - \frac{1}{2}\big)^2\,dx = \frac{1}{12}\,,$$

corrispondenti a media e varianza della variabile uniforme in $[0,1]$. Insistiamo che tutto questo, ovviamente, non garantisce la bontà del generatore, ma costituisce solo una prima verifica.