Processi random in natura

Esistono in natura processi *naturalmente* random

- decadimento radiativo
- agitazione termica
- moto di particelle in sospensione

Per descrivere questi fenomeni bisogna spesso ricorrere ad un metodo chiamato Monte Carlo perchè basato sull'uso di numeri random

Il Monte Carlo viene usato anche per il calcolo di integrali

Numeri random dal computer

Alcuni metodi numerici come il Monte Carlo hanno bisogno di usare sequenze di numeri random

Come si producono con il computer?

Una sequenza di numeri random

R1, R2, R3,.....

è una sequenza di numeri non correlati fra loro.
Esempio: numeri successivi ottenuti lanciando un dado 3,4,6,1,..... Abbiamo una sequenza random con distribuzione uniforme, ogni numero ha uguale propabilità di uscire.

Generatori di numeri random (1)

•Il calcolatore è una macchina deterministica, in principio quindi non potrebbe generare sequenze veramente random.

•Inoltre in genere si vorrebbe poter ripetere la stessa sequenza, nel caso si voglia ripetere o controllare un certo calcolo.

- **generare numeri random non correlati fra loro è impossibile**

•Si cerca quindi di generare numeri pseudo-random.

- **con ciclo indichiamo il numero di estrazioni dopo il quale appare la correlazione**
- **si può cercare di generare sequenze con cicli molto lunghi, dove *molto lungo* è misurato rispetto al problema da studiare**

Generatori di numeri random (2)

Funzione MOD:

$$MOD(X,Y) = X - INT(X/Y)*Y$$

Cioè ritorna come valore il resto del divisore di y per x.

Metodo semplice per generare numeri random

$$r_i = \mathrm{mod}(a * r_{i-1} + c, m)$$

Si generano sequenze di numeri random che si ripetono dopo m step.

Esempio di generazione di sequenze

$$r_i = \mathrm{mod}(a * r_{i-1} + c, m)$$

$c = 1 \quad a = 4 \quad m = 9$

$r1 = 3$

$r2 = (4 \times 3 + 1)/9 = 4$

$r3 = (4 \times 4 + 1)/9 = 8$

$r4 = (4 \times 8 + 1)/9 = 6$

$r5 = (4 \times 6 + 1)/9 = 7$

$r6 = (4 \times 7 + 1)/9 = 2$

$r7 = (4 \times 2 + 1)/9 = 0$

$r8 = (4 \times 0 + 1)/9 = 1$

$r9 = (4 \times 1 + 1)/9 = 5$

$r10 = r1 = (4 \times 5 + 1)/9 = 3$

Dopo 9 steps si ripete la sequenza

```
      program modulo
      implicit none
c
      real*8 r(0:100),x,m,c,a
      integer i
c
      print*,' r(i) viene da mod(a*r(i-1)+c,m)'
      print*,' valore per r(0),a,c,m'
      read*,r(0),a,c,m
      open(7,file='ran.dat',status='unknown')
      open(8,file='correl.dat',status='unknown')
c
      do i=1,100
         x=a*r(i-1)+c
         r(i)=mod(x,m)
         print*, i,r(i)/m
         write(7,'(i6,e15.6)') i,r(i)/m
      end do
      do i=1,50
         write(8,'(2e15.6)') r(2*i-1)/m,r(2*i)/m
      end do
      stop
      end
```

Provare con a=57, c=1, M=256 e r(0)=10 e determinare il periodo

```
      program gen_ran
      implicit none
c

      real*8 r(100)
      integer ix,i
c
c     seed
      ix=1357

      open(7,file='ran.dat',status='unknown')
      open(8,file='correl.dat',status='unknown')
c
      call random(ix,100,r)



      do i=1,100
         write(7,'(i6,e15.6)') i,r(i)
      end do
      do i=1,50
         write(8,'(2e15.6)') r(2*i-1),r(2*i)
      end do
      stop
      end
```

```
      subroutine random(ix,n,rno)
c*****Congruential random number generator, Forsythe's constants
      implicit none
      integer ix,iy,i,n
      real*8 rno(n)

      do 50 i=1,n
        iy=ix*314159269+453806245
c       Eliminate bits over 31
        iy=ibclr(iy,31)
        rno(i)=float(iy)/2.1474838e+09
c       rno(i)=float(iy)/2.1474836e+09
        ix=iy
50    continue
      return
      end
```

ibclr (i,pos) prende il numero i in binario e mette a zero il bit che si trova nella posizione pos

Proprietà di un buon generatore

1. **Efficienza**
2. **Sequenze con periodo lungo**
3. **Riproducibilità**

Modulo generator (Lehmer 1948)

$$r_i = \mathrm{modulo}(ar_{i-1} + c, m) \qquad r_0 > 0, m > r_0, a > 0, c > 0$$

r_0 è chiamata la "seed"

Generatore moltiplicativo

$$r_i = \mathrm{mod}(ar_{i-1}, m)$$

è noto dare risultati migliori del precedente

a e *m* vanno scelti in modo da massimizzare il periodo della sequenza

Con computer a n bit l'intero più grande è 2^n-1.

Con n=32 si può scegliere $m=2^{32}-1=2147483647$ (Mersenne number)

Ma anche a va scelto opportunamente !!

Una scelta appropriata risulta essere a=16807

Funzione ran0

```
      program test_ran0
c
      implicit none
      integer iseed,nm,i,nn
      real ran0,rx(100000)
c
      print*,' numero di punti?'
      read*,nm
c
      open(7,file='ran.dat',status='unknown')
      open(8,file='correl.dat',status='unknown
c
      iseed=198765
      do i=1,nm
         rx(i)=ran0(iseed)
      end do
      do i=1,nm
         write(7,'(i6,e15.6)') i,rx(i)
      end do
      nn=int(nm/2)
      do i=1,nn
         write(8,'(2e15.6)') rx(2*i-1),rx(2*i)
      end do
      stop
      end
```

```
      function ran0(iseed)
      implicit none
      integer iseed,ia,im,iq,ir,mask
      integer k
      real ran0,am
      parameter (ia=16807,im=2147483647)
      parameter (iq=127773,ir=2836,mask=123459876)
      parameter (am=1./im)
c
      iseed=ieor(iseed,mask)
      k=iseed/iq
      iseed=ia*(iseed-k*iq)-ir*k
      if (iseed .lt. 0) iseed=iseed+im
      ran0=am*iseed
      iseed=ieor(iseed,mask)
      return
      end
```

Distribuzione uniforme

Abbiamo considerato generatori di numeri random con distribuzione uniforme

$$p(x)dx = \begin{cases} dx & 0 < x < 1 \\ 0 & \text{altrimenti} \end{cases}$$

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

Test da effettuare: calcolare i momenti della distribuzione

$$< x^k >= \frac{1}{N} \sum_{i=1}^{N} x_i^k$$

Test della distribuzione

$$\int_0^1 dx\, x^k P(x) = \frac{1}{k+1}$$

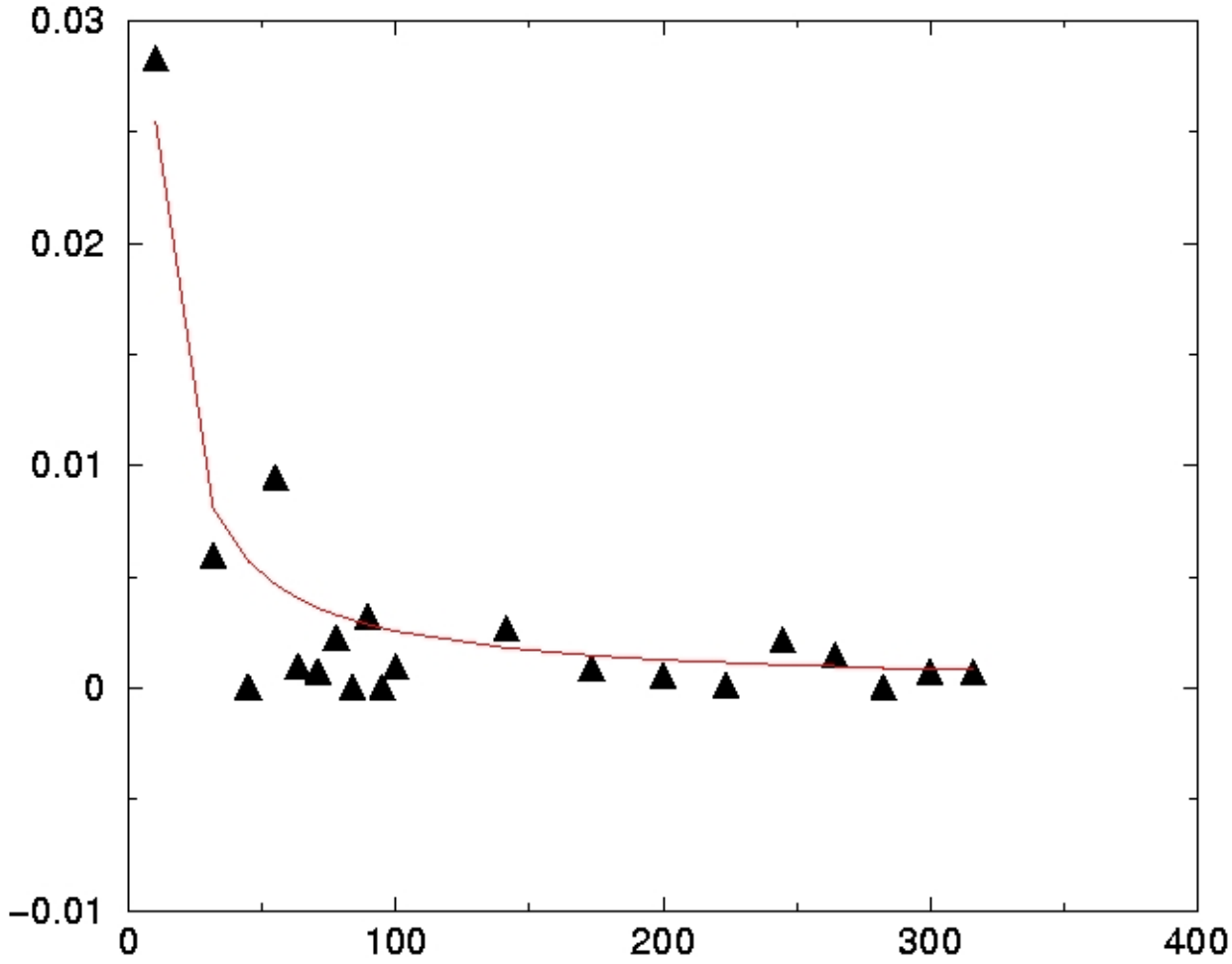
$$\frac{1}{N}\sum_{i=1}^{N} x_i^k \approx \int_0^1 dx\, x^k P(x) + O\left(1/\sqrt{N}\right) \approx \frac{1}{k+1} + O\left(1/\sqrt{N}\right)$$

Calcolare per k=1,2,4 N=100, 10000, 100000

Per esempio si può guardare alla quantità

$$\frac{1}{N}\sum_i x_i^k - \frac{1}{k+1} \quad \text{vs} \quad \sqrt{N}$$

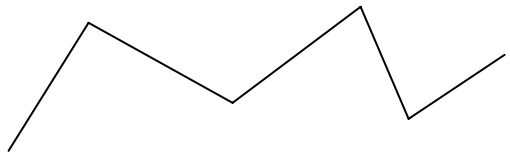
Momento k=1



Random walk (1)

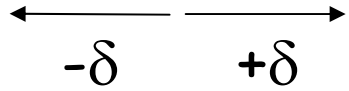
Moto casuale di particelle: Brown (1828)

detto anche moto browniano



- **moto libero fra due urti**
- **perdita di memoria dopo l'urto**

In 1 dimensione $x(t+\tau)=x(t)\pm\delta$



Prob. di spostamento $+\delta = 1/2$

Random walk (2)

Dopo N spostamenti di cui m positivi la distanza percorsa sarà

$$x=m\delta+(N-m)(-\delta)=(2m-N)\,\delta$$

La prob. di avere m spostamenti positivi

$$P(m)=\frac{N!}{(N-m)!m!}\,p^m(1-p)^{N-m}$$ con *p=1/2*

Distribuzione gaussiana

Nel limite di grande N

$$P(m) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{(m-\mu)^2}{2\sigma^2} \right)$$

$$\overline{m} = \mu = Np = \frac{N}{2} \qquad \sigma = \frac{\sqrt{N}}{2}$$

$x = (2m - N)\delta$ quindi il valor medio è dato da

$$\overline{x} = (2\overline{m} - N)\delta = 0$$

Distribuzione continua

Con $x = (2m - N)\delta$

$$P(x)dx = \frac{1}{\sqrt{2\pi N\delta^2}} \exp\left(-\frac{x^2}{2N\delta^2}\right)dx$$

da cui si vede che $< x >= 0$ $< x^2 > \propto N$

Se τ è il tempo fra le collisioni e t è il tempo di osservazione, abbiamo N=t/τ

$$P(x)dx = \frac{1}{\sqrt{2\pi \frac{t}{\tau}\delta^2}} \exp\left(-\frac{x^2}{2\frac{t}{\tau}\delta^2}\right)dx$$

Distribuzione statistica per il random walk

Si usa introdurre una costante D

$$\frac{\delta^2}{2\tau} \equiv D$$

$$P(x) = \frac{1}{\sqrt{4\pi D t}} \exp\left(-\frac{x^2}{4Dt}\right)$$

$$< x^2 >= 2Dt$$

relazione dovuta ad Einstein

in 2 dim. $< r^2 >= 4Dt$ in 3 dim. $< r^2 >= 6Dt$

Calcolo dello spostamento medio

Algoritmo per calcolare $$r^2(t) = \frac{1}{n}\sum_{i=1}^{n}\left(r_i(t) - r_i(0)\right)^2$$

Esempio in 2D

- do t=0,tmax
  - do i=1,n
  - x(i)=x(i-1)+(rx-0.5)
  - y(i)=y(i-1)+(ry-0.5)
  - sum=sum+x**2+y**2
- r2=r2+sum/n

numero di step del cammino per ogni t

o anche numero di particelle

Unità di tempo:

Monte Carlo Step (MCS)

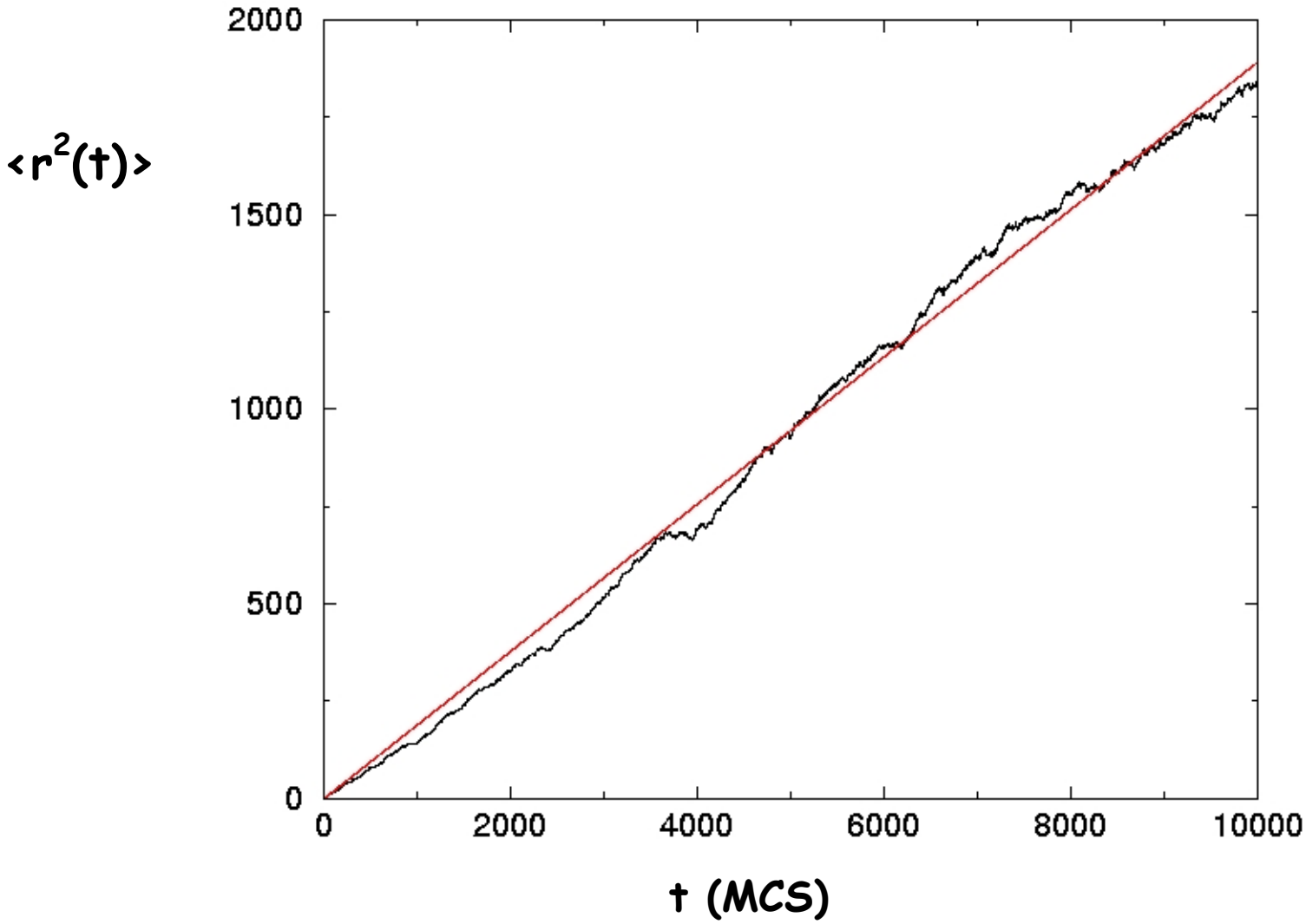
Programma per random walk

```
 call start(p,N,Nmax,ntrial,iseed)

      do 100 itrial = 1,ntrial      step "temporali"

         call walk(ix,p,N,iseed,trace)

c        raccogli i dati alla fine di ogni cammino di N steps
         call data(ix,xcum,x2cum,prob)

         write(10,'(2e15.6)') real(itrial),x2cum
         write(11,'(3e15.6)') real(itrial),trace(1),trace(2)
 100     continue

      calcola le medie

      call aver(N,Nmax,ntrial,xcum,x2cum,prob)
c
```

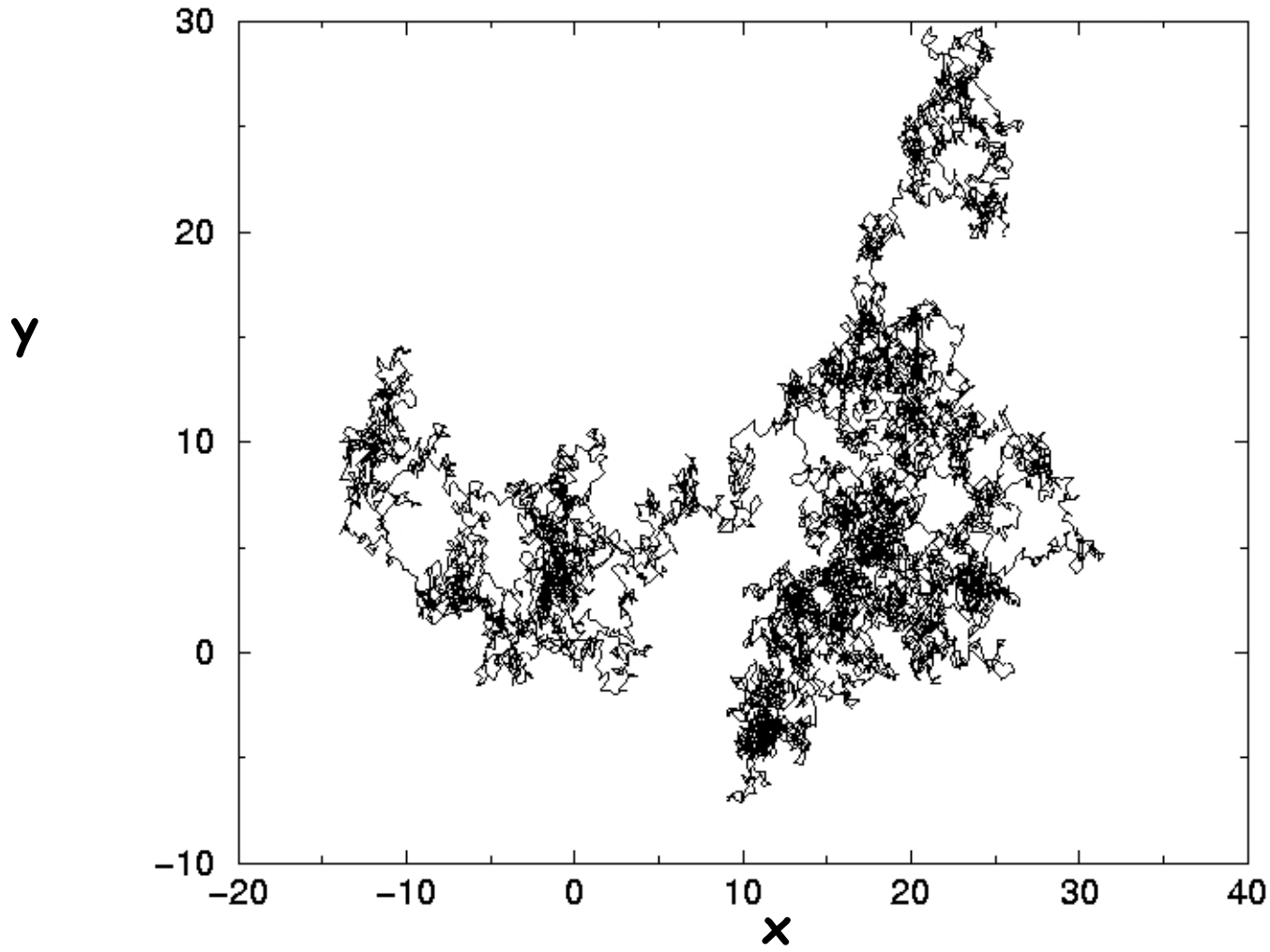
Routine walk

```
 subroutine  walk(ix,p,N,iseed,xtrace)
      implicit none
      integer*4 N,ntrial,iseed,ix,istep
      real*8 p,drand48,xtrace(N)

c  cammina per N steps
         ix = 0        !  posizione iniziale per ogni prova
         do 100 istep = 1,N
            if(drand48().le.p) then  !
               ix = ix + 1
               xtrace(istep)=xtrace(istep)+1
            else

               ix = ix - 1
               xtrace(istep)=xtrace(istep)-1
             end if
   100     continue
```

Spostamento quadratico



Traiettoria di una particella



Esercizi sul random walk

- scaricare il file es_random_walk.ps
- scaricare il file walk1d.f
- Usando drand48() compilare con

  g77 –o walk1d.x –fno-underscoring walk1d.f
- eseguire i calcoli in dimensione d=1
- modificare il programma per d=2 (walk2d.f)