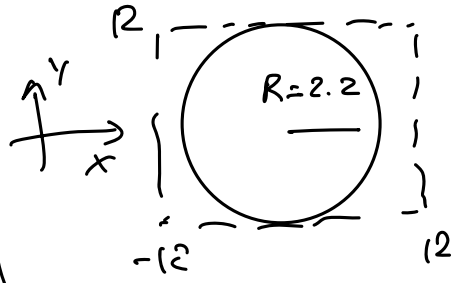
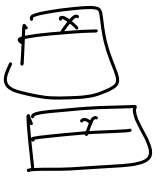
Array, Puntatori, Funzioni

```
void cerchio(double, double *, double *);
void cerchio2(double, double *);

int main() {
    double R = 2.2;
    double x,y;

    x,y = cerchio(3);
    cerchio(R, &x, &y);
    printf("x: %f, y: %f\n", x,y);
    printf("r: %f\n", sqrt(x*x + y*y));
```





Interfaccia funzione cerchio:

tipo: void

arg: 3

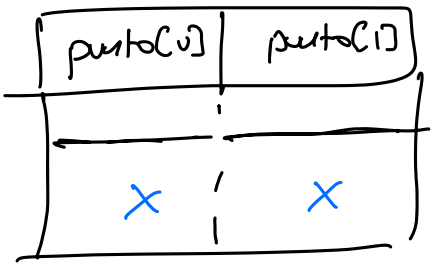
tipo argomenti:

raggio: double

x: double *

y: double *

```
double punto[2]; // array 1D lunghezza 2
```



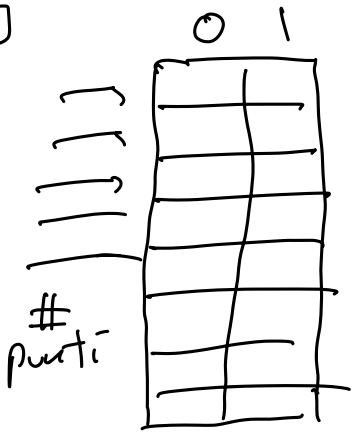

```
cerchio(R, &punto[0], &punto[1]);
```

```
cerchio(R, punto, punto+1);
```

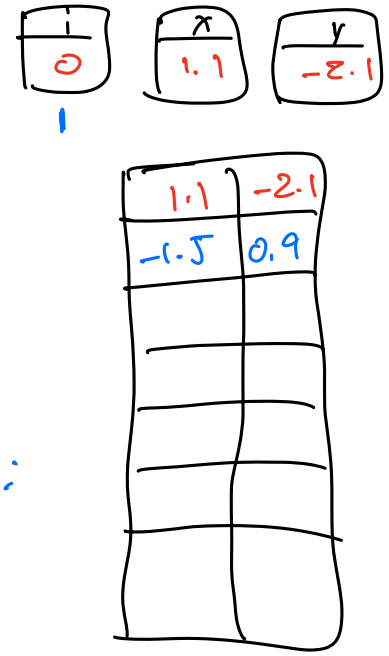
punto ≡ &punto[0]

punto+1 ≡ &punto[1]

```
double disco[1000][2];
```



```
for(int i=0; i<1000; i++) {
    cerchio( R, &x, &y );
    disco[i][0] = x;
    disco[i][1] = y;
```

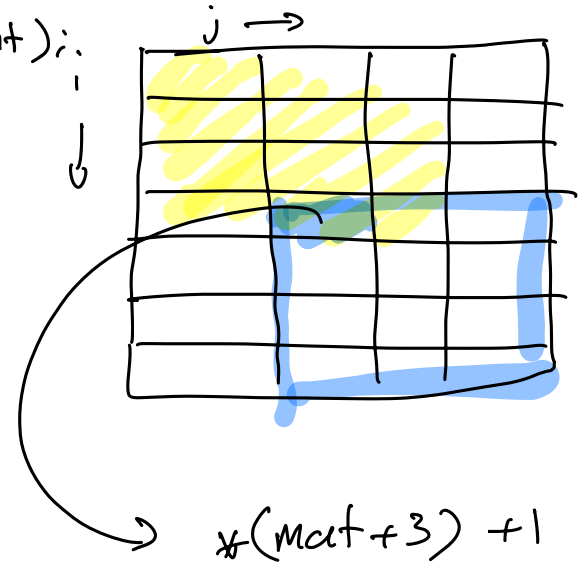



```
cerchio(R, &disco[i][0],
           &disco[i][1]);

cerchio(R, *(disco+i)+0,
           *(disco+i)+1);
```

```
void cerchio(double r, double* g, double* y) {
    do {
        *g = 2*r*drand48() - r;        x ∈ [-r, r]
        *y = 2*r*drand48() - r;
    } while( (*g * *g) + (*y * *y) >= r*r );
}
```

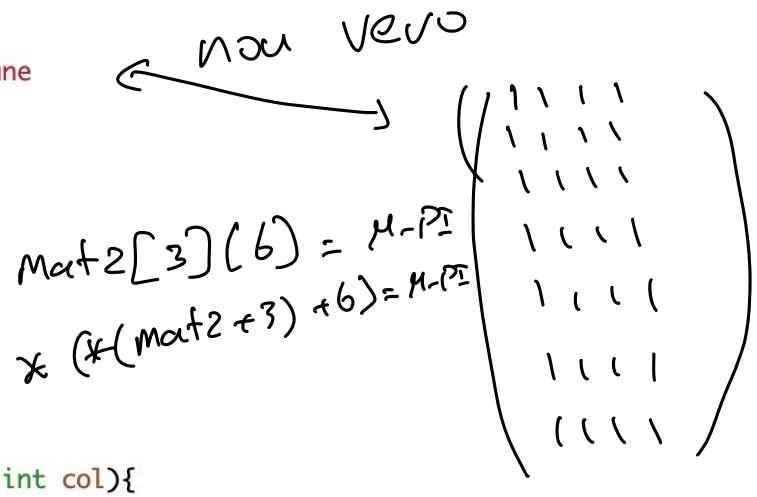
```
double mat[100][10];

mat[i][j] = 2

*(*(mat+i)+j) = 2
```

```
void printMat(double [7][4], int, int);

---

double mat[7][4];

printMat(mat, 7, 4);

printMat(mat, 3, 3);

printMat(*(mat+3)+1, 3, 3);
```




```
void printMat(double M[7][4], int NR, int NC) {

    // ciclo i sulle righe

        // ciclo j sulle colonne.

        // stampa

}
```

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

// stampa elementi array 2D
// bisogna specificare dimensione e lunghezza max di array
void printMat(double [7][4], int, int);

// assegna un valore a tutti gli elementi di un array
void assegna(double, double [7][4], int, int);


int main() {

  double mat1[7][4] = {1}; // errore comune
  printMat(mat1, 7, 4);

  double mat2[7][4]; //
  printMat(mat2, 7, 4);


  assegna( M_PI, mat1, 3, 3);
  printMat(mat1, 7, 4);

}

void printMat(double m[7][4], int righe, int col){
  printf("printMat:\n");
  for(int i=0; i<righe; i++){
    for(int j=0; j<col; j++) {
      printf("%.4f\t", m[i][j] );
    }
    printf("\n");
  }

}

void assegna(double val, double m[7][4], int nr, int nc) {
    printf("modificando valori matrice in assegna()\n");
    for(int i=0; i< nr; i++){
        for(int j=0; j<nc; j++){
            *(*(m+i)+j) = val;
        }
    }
}
```



```
Mac:material rahatlou$ gcc -o /tmp/app matrici.c
Mac:material rahatlou$ /tmp/app
printMat:
1.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
printMat:
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
modificando valori matrice in assegna()
printMat:
3.1416  3.1416  3.1416  0.0000
3.1416  3.1416  3.1416  0.0000
3.1416  3.1416  3.1416  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
0.0000  0.0000  0.0000  0.0000
```

double mat1[7][4] = {1};

double mat2[7][4];
printMat(mat2, 4, 4);

3.1416

int cifre[10] = {0};

3	1	4	1	6	0	0	0	0	0

cifre[0] = (int) M_PI;

cifre[1] = 1

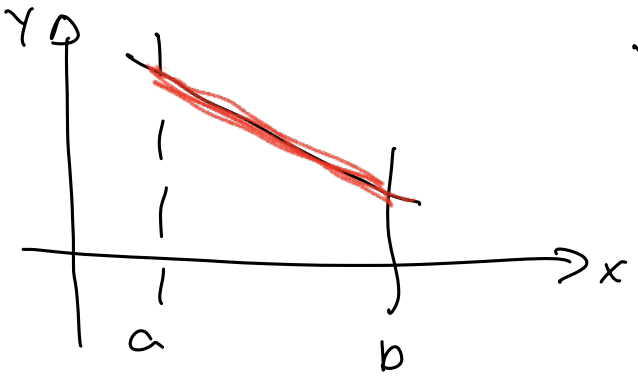
cifre[i] = (int) (M_PI - cifre[0]) * 10

cifre[i] = (int) (M_PI - cifre[i]) *

char nome[100];

scanf("%s", nome);

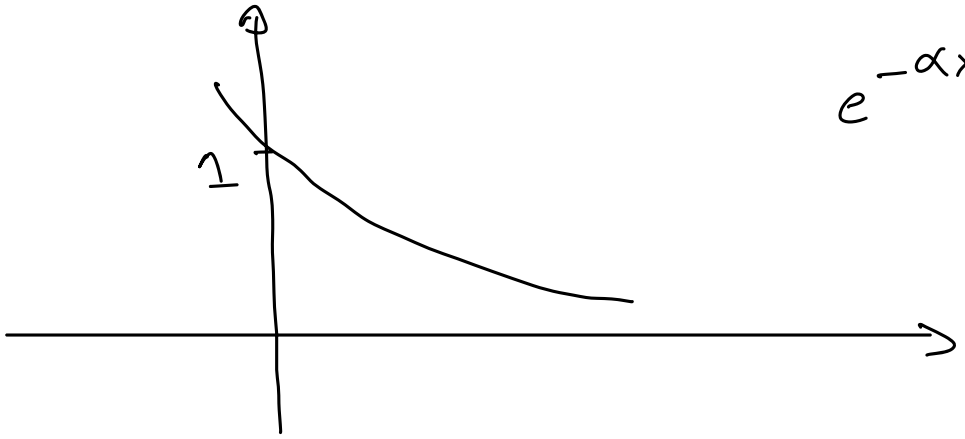
Generazione casuale secondo $f(x)$



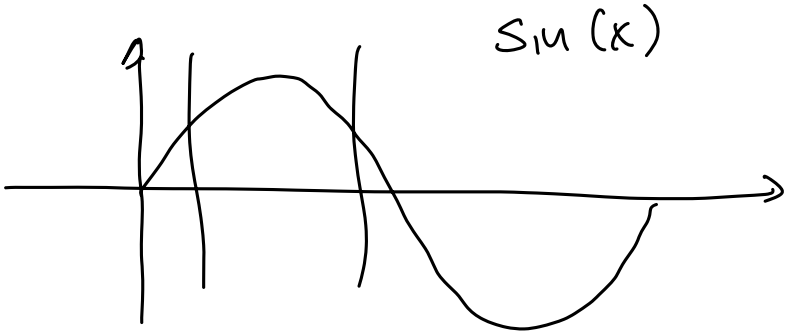
$y = mx + q$

$m < 0$

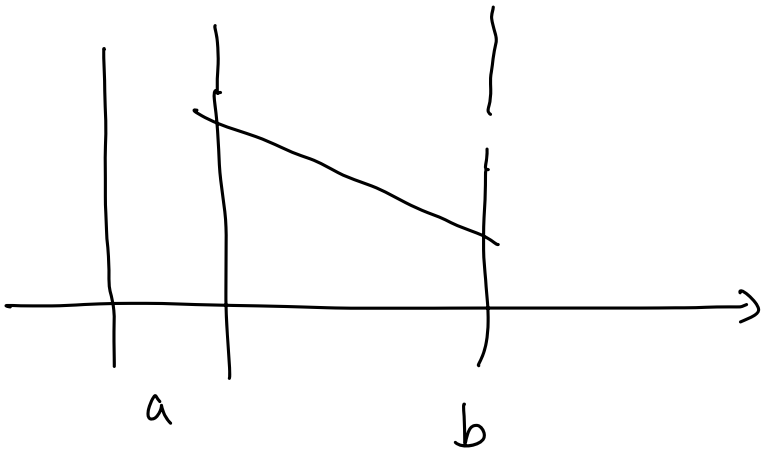
$q > 0$



$e^{-\alpha x}$



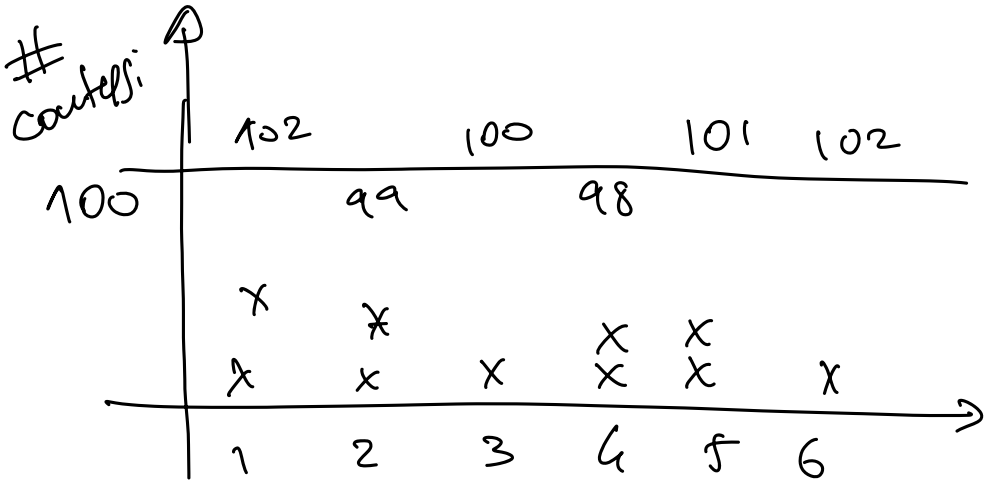
$\sin(x)$



$x \in [a, b]$

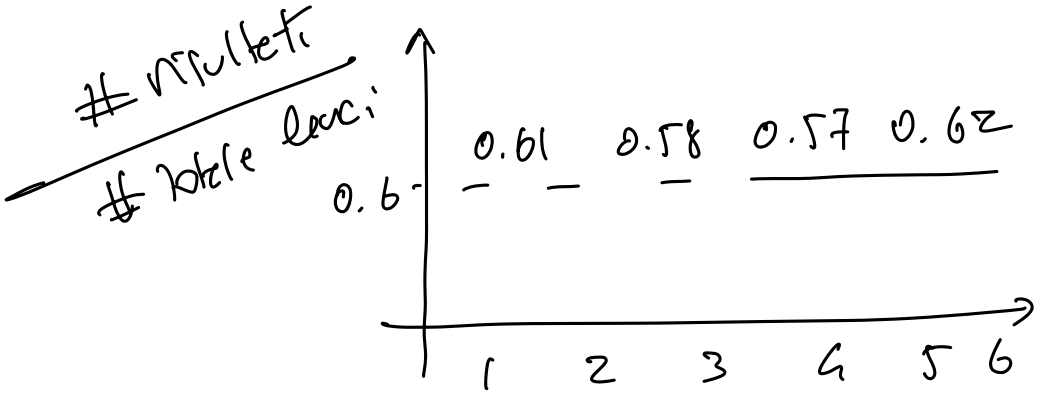
distribuiti secondo $f(x)$

dado

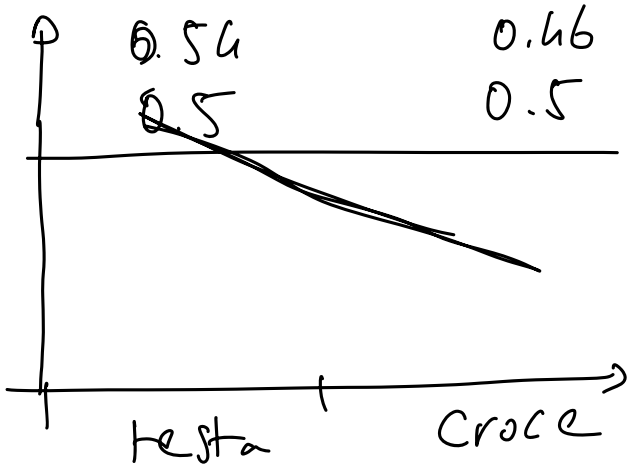


600 lanci:

int facce[6]



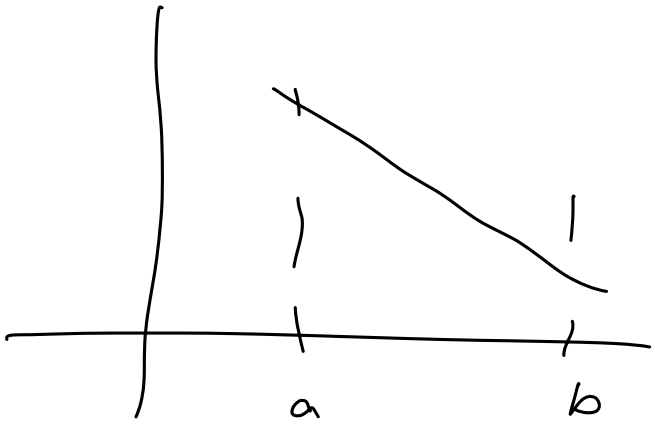
dado truccato



10^9 lanci

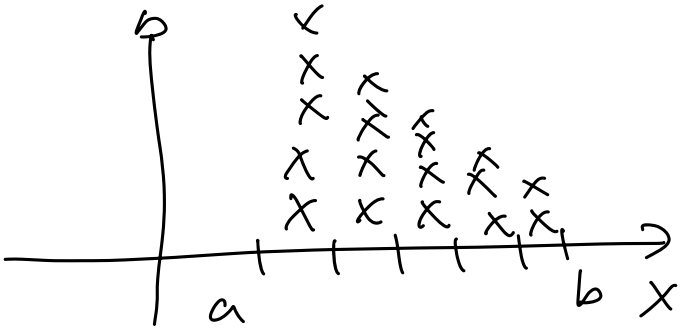
genererе secondo $f(x)$:

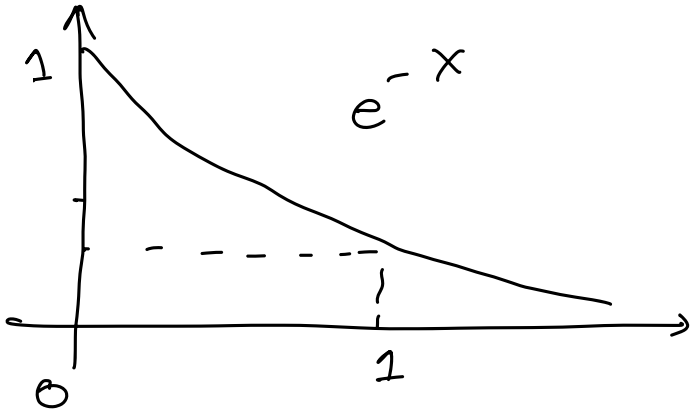
$x \in [a,b]$



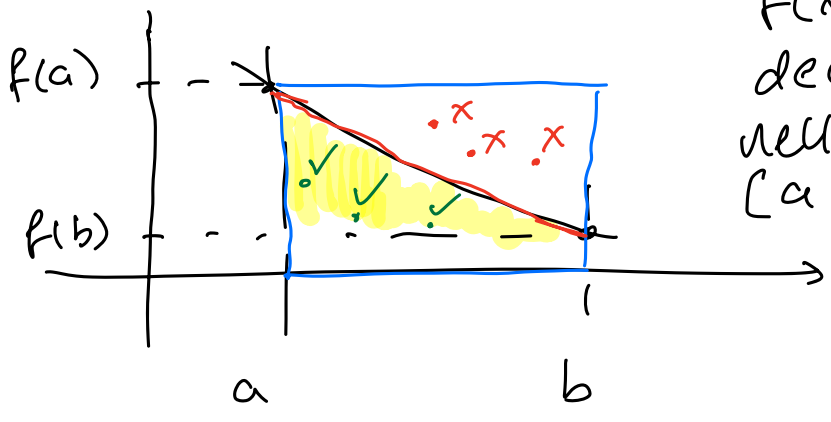
ciclo

$x =$ generaf()





hit & miss



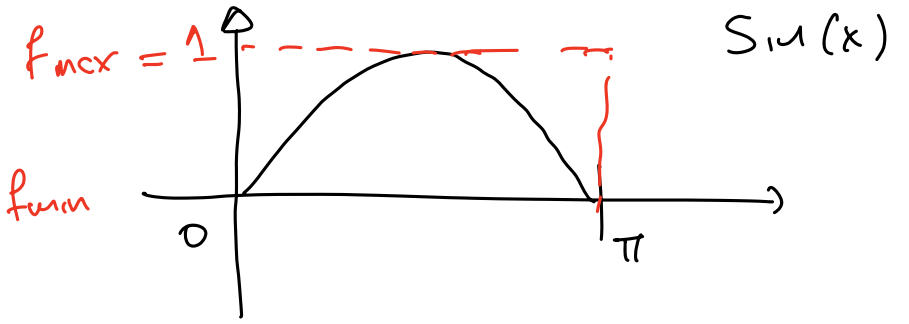
f(x) decrescente nell'intervallo [a,b]

$x \in [a,b]$ uniforme $\quad y \in [f(b), f(a)]$ uniforme.

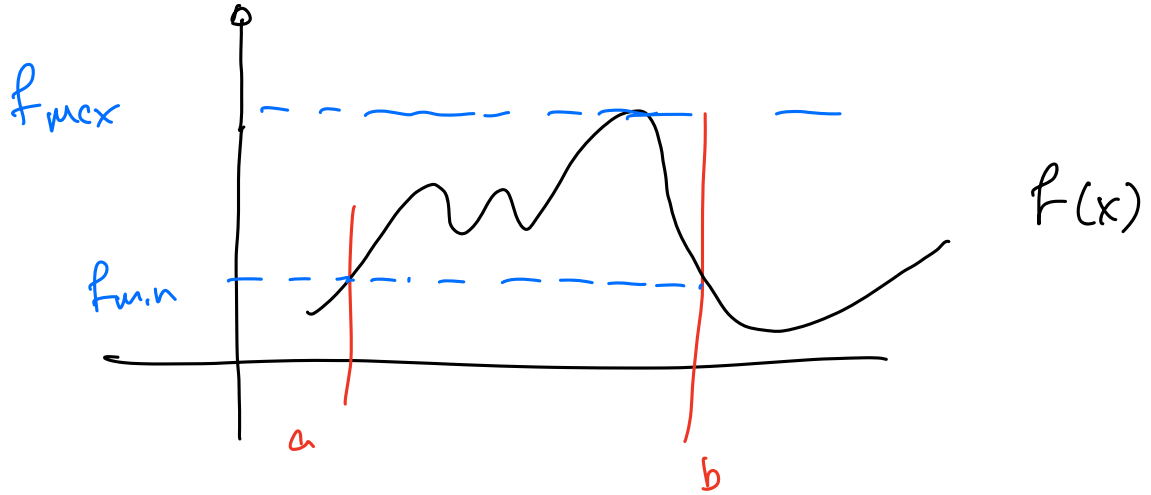
genero uniforme coppie x,y

```
do {
    x = uniforme(a, b);
    y = uniforme( f(b), f(a) )
} while( y > f(x) );
```

=> genero x secondo f(x) in [a,b]




```
do {
    x = uniforme( 0, M_PI );
    y = uniforme (0, 1) ;
} while( y > Sin(x) );
```



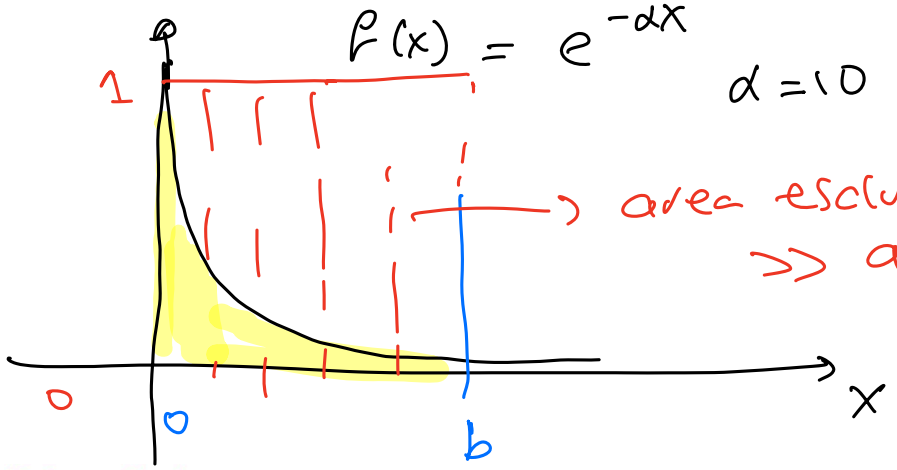
genero $x \in [a,b]$ uniforme

genero $y \in [f_{min}, f_{max}]$ uniforme

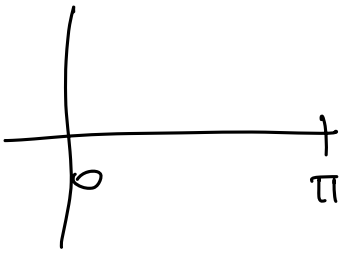
se $y > f(x)$ rigenero.

```
double myf(double x) {
    return -x + 2.3*x*x*x + 6.5 * x*x*x*x +
           -5.2 * pow(x,5) + 1.2 * pow(x,6);
}
```

Inefficiente se l'area escluso molto grande



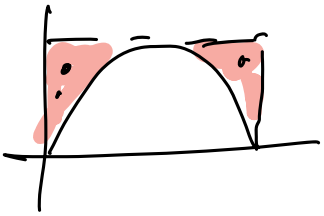

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define NMAX 100000

double uniforme(double,double);


int main() {

  srand48( time(0) );
  double a=0, b=M_PI;

  // max di sin(x) e e^(-x) per x in [0,PI]
  double ymax = 1.;

  // array per salvare i numeri generati  da utilizzare
  double piatto[NMAX];
  double sinx[NMAX];
  double expx[NMAX];
  double myfx[NMAX];

  // numero effettivo di numeri generatori per ottenere
  // NMAX numeri con la distribuzione giusta
  int npiatto=0, nsinx=0, nexpx=0, nmyfx=0;

  FILE* fp;
  char fname[] = "numeri.txt";
  fp = fopen(fname,"w+");
  if(!fp) {
    printf("errore a creare %s... exit\n", fname);
    exit(-1);
  }

  int i;
  double x,y;

```

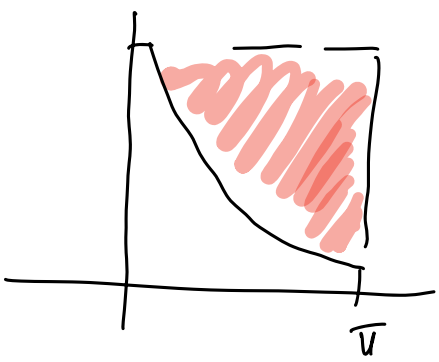


```
39    for(i=0; i<NMAX; i++) {
40
41      if(! ((i+1)%1000) ) printf("generazione: %3d\n", i+1);
42      // distribuzione uniforme
43      x = uniforme(a,b);
44      *(piatto+i) = x;
45      npiatto++;
46
47      do{
48        nsinx++;
49        x = uniforme(a,b);
50        y = uniforme(0,ymax);
51      } while ( y > sin(x) );
52      *(sinx+i) = x;
53
54      do{
55        nexpx++;
56        x = uniforme(a,b);
57        y = uniforme(0,ymax);
58      } while ( y > exp(-x ) );
59      *(expx+i) = x;
60
61
62      // scrivi output
63      fprintf(fp, "%.10f\t %.10f\t %.10f\n", piatto[i], sinx[i], expx[i]);
64
65    } // ciclo generazione
66    printf("numero generazioni per piatto: %6d \t sin(x): %6d \t exp(-x): %6d\n",
67     npiatto, nsinx, nexpx);
68
69
70    fclose(fp);
71
72  } // main
73
```

```
generazione: 10000
generazione: 20000
generazione: 30000
generazione: 40000
generazione: 50000
generazione: 60000
generazione: 70000
generazione: 80000
generazione: 90000
generazione: 100000
numero generazioni per piatto: 100000    sin(x): 157295          exp(-x): 328550
```

57295 punti nell'area rossa



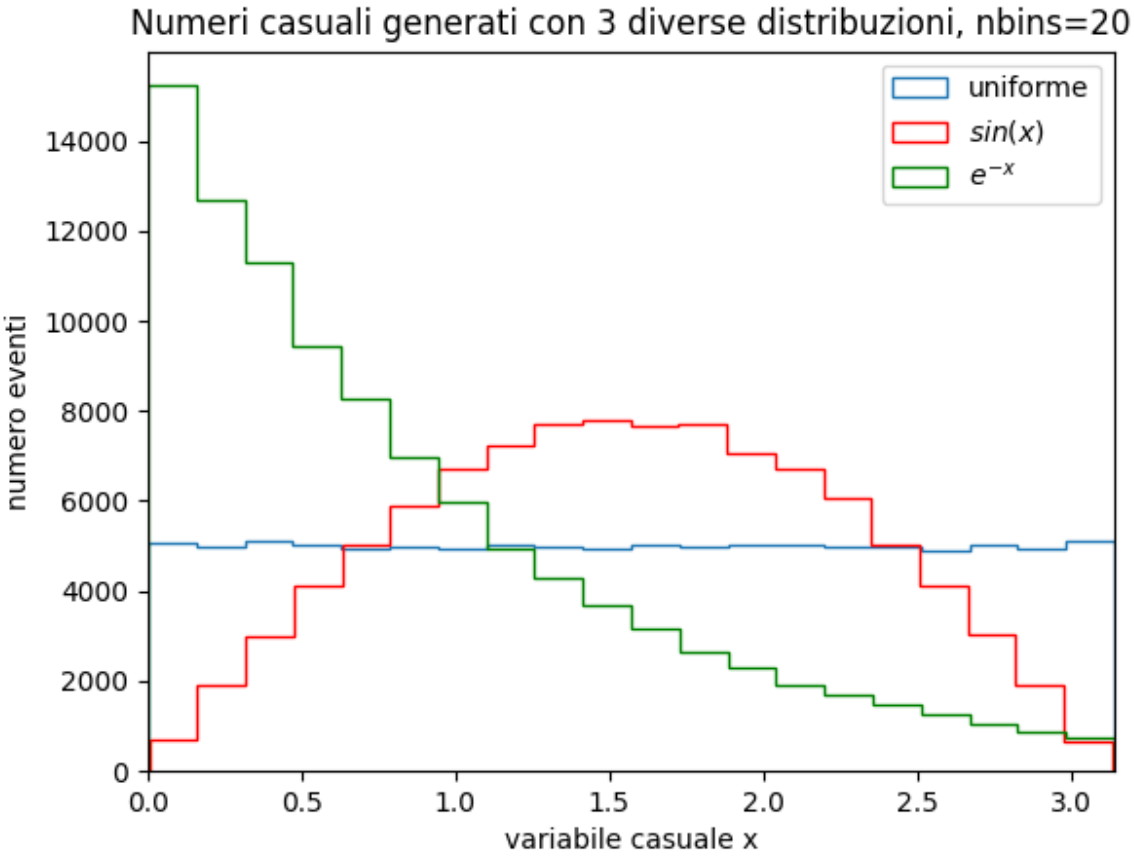


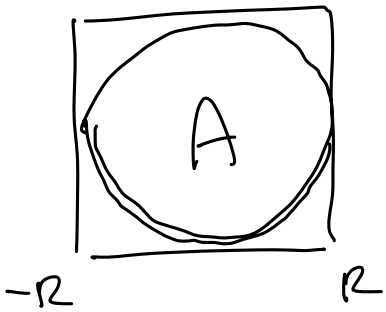
metodo hit & miss per generare secondo f(x)

```python
import matplotlib.pyplot as plt
import numpy as np
import math as m

piatto, sinx, expx = np.loadtxt("numeri.txt", unpack=True)

nbins=20

plt.title("Numeri casuali generati con 3 diverse distribuzioni, nbins=%d"%(nbins))

plt.xlim(0, m.pi)
plt.xlabel('variabile casuale x')
plt.ylabel('numero eventi')

plt.hist(piatto, bins=nbins, histtype='step', label='uniforme')

plt.hist(sinx, bins=nbins, histtype='step', color='red', label='$sin(x)$')

plt.hist(expx, bins=nbins, histtype='step', color='green', label="$e^{-x}$")

# legenda dei 3 grafici
plt.legend(loc="upper right")


plt.show()
```

$[0, \pi]$ diviso in 20 intervalli

Ciascun intervallo largo $\pi/20$





prob. dentro cerchio

$x \in [-R, R]$

$y \in [-R, R]$

$$\text{prob} = \frac{\text{area cerchio}}{\text{area quadrato}} = \frac{A}{4R^2} = \frac{\#\text{punti dentro}}{\#\text{totale}}$$

$$\pi R^2 = A = \frac{\#\text{punti dentro}}{\#\text{totale lanci}} \cdot 4R^2$$