European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

Mid level programming

20-21 April 2013

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013

This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

Uniform random number generator

Random number: a value drawn from a range [a,b] with a specified probability distribution.

For example, a number x, drawn in the range of [a,b] is said to be uniformly distributed if

$$p(x) = \begin{cases} 1/(b-a), & a \leqslant x \leqslant b \\ 0, & elsewhere \end{cases}$$

How can we produce random numbers using a computer program?

The simplest case is to use the routine provided by the compiler used. For ANSI C, this is:

```
#include <cstdlib>

void srand(unsigned seed);

int rand(void);
```

The ANSI C standard falls in the category of Linear Congruence Method

$$I_{j+1} = (aI_j + c)\ mod\ m$$

$$I, a, c, m \ \ integers$$

Range: [0,m-1]

The numbers are produced in a deterministic way, starting from an initial value called **seed**. The sequence of these numbers will be the same for every PC if we use the same seed value. However, they follow a uniform probability distribution, meaning that the probability of getting a number in the range [0,m-1] is 1/m. Below, we present 3 sets of 15 random numbers for different seed values.

seed = 5098	seed = 5099	seed = 5100
0.509232	0.509323	0.509445
0.457747	0.78576	0.113773
0.931181	0.476333	0.021485
0.811365	0.545701	0.280068
0.997436	0.312204	0.626972
0.275216	0.395032	0.514878
0.486099	0.581103	0.676077
0.563921	0.474593	0.385235
0.169225	0.673147	0.177007
0.426374	0.238746	0.0511185
0.167577	0.163091	0.158574
0.710288	0.711631	0.712973
0.370464	0.0460524	0.721702
0.263039	0.767998	0.272958
0.252602	0.929868	0.607135

Generally, a good random number generator should have the following characteristics:

- Fast
- Simple
- Desired Statistical Properties
(i.e. no significant correlations)
- Long period

The ANSI C standard is simple, fast (only few operations per call) but exhibits

POOR STATISTICAL PROPERTIES AND SMALL PERIOD

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

A simple program for measuring the period for ANSI C rand()

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <cstdio>
4  #include <cmath>
5
6  using namespace std;
7
8  int main(int argc, char *argv[]) {
9
10     double counter=0,totalCount=0;
11     int runs = 10000;
12     int X,Y;
13     int seed=5097;
14
15     for(int i=0;i<runs;i++)
16     {
17
18         counter=0;
19
20         srand(seed+i);
21
22         X=rand();
23         Y=rand();
24
25         while(X!=Y)
26         {
27
28             Y=rand();
29             counter++;
30
31         }
32
33         totalCount+=counter;
34         cout<<i<<"\n";
35     }
36
37     totalCount/=runs;
38     cout<<int(totalCount)<<"\n";
39
40     return 0;
41 }
```

Average period after 10000 runs: 32718
numbers ≈ RAND_MAX (= 32767)

Following the presentation of Wong's book, ***"Computational Methods in Physics and Engineering"***, we will implement 3 of the most common tests used:

- **Frequency test**
- **Serial correlation test**
- **Run up test**

It must be clear that the behavior of RNGs may differ with respect to the statistical or empirical test used. The RNG that performs well for the majority of the tests is considered to be the best.

A full set of statistical and other tests concerning the performance of random number generators (RNGs) can be found in Donald Knuth's ***"The art of computer programming"***, volume 2, 3rd edition.

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

FREQUENCY TEST

The procedure is the following:

1. Select the random number generator (RNG).
2. Set the number of data classes (N_{bin}). Specify bin range.
3. Iterate to produce N_{rand} random numbers. Increase the frequency of the appropriate bin.
4. Calculate the chi-square.
5. Output the frequencies and chi – square.

The chi – square is calculated in the following way:

$$\chi^2 = \sum_{i=1}^{N_{bin}} \frac{(M_j - nw)^2}{nw}$$

$M_j : number\ of\ RNs\ in\ j\ bin$

$nw :\ expected\ number\ of\ occurrences\ per\ bin$

$$w = \frac{1}{N_{bin}} \qquad \nu = N_{bin} - 1$$

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

A simple program performing frequency test

```
#include <iostream>
#include <cstdlib>
#include <cstdio>

using namespace std;

int main(int argc, char *argv[])
{

    int Nbin=50, i, j , numOfRN=50000, seed=5097;
    double step=1.0/Nbin,RNG;
    int *Bin=new int[Nbin];

    for(i=0;i<Nbin;i++) Bin[i]=0;

    srand(seed);

    for(i=1;i<=numOfRN;i++)
    {
        RNG=rand()/(RAND_MAX+1.0);
        j=int(RNG*Nbin);
        Bin[j]++;

    }

    //perfroming chi-square

    double chi2=0;
    int degOfFree=Nbin-1;

    for(i=0;i<Nbin;i++) chi2+=(Bin[i]- numOfRN*step)*(Bin[i]-numOfRN*step);

    chi2=chi2/(numOfRN*step);

    FILE *fp;
    fp=fopen("frequency test results.dat","w");
    fprintf(fp,"chi^2 = %lf , nu = %i\n",chi2,degOfFree);

    for(i=0;i<Nbin;i++) fprintf(fp,"%lf %i\n",(i*step+(i+1)*step)/2,Bin[i]);

    fclose(fp);

    return 0;
}
```

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

SERIAL CORRELATION TEST

The procedure is the following:

1. Select the random number generator (RNG).
2. Iterate to produce N_{rand} random numbers (RN). Calculate the correlation coefficient C.
4. Output the correlation coefficient.

The serial correlation coefficient is calculated by the following way:

$$C = \frac{n(\sum_{i=1}^{n} X_{i-1}X_i) - (\sum_{i=1}^{n} X_i)^2}{n(\sum_{i=1}^{n} X_i^2) - (\sum_{i=1}^{n} X_i)^2}$$

$$n: \ number\ of\ RNs$$

$$X_i : i^{th}\ RN$$

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

A simple program performing serial correlation test

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cmath>

using namespace std;

int main(int argc, char *argv[])
{

    int numOfRN=5000,i,seed=5097;
    double sum2=0,sum1=0,sumCorr=0;
    double current, previous, C, sigma,mu;

    srand(seed);

    previous = rand()/double(RAND_MAX);
    sum1+=previous;
    sum2+=previous*previous;

    for(i=2;i<=numOfRN;i++)
    {

        current=rand()/double(RAND_MAX);
        sumCorr+=previous*current;
        sum1+=previous;
        sum2+=previous*previous;
        previous=current;

    }

    sum1+=previous;
    sum2+=previous*previous;

    //Define correlation coefficient

    C=(numOfRN*sumCorr-sum1*sum1)/(numOfRN*sum2-sum1*sum1);

    printf("%lf\n",C);

    return 0;
}
```

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

RUN UP TEST

The procedure is the following:

1. Select the random number generator (RNG).
2. Set the number of possible ascending random number (RN) sequences (m).
3. Start producing RNs sequences. While the new RN is greater than the previous one, increase the length of the sequence by one. Else, store the result to an appropriate frequency counter and start a new sequence.
4. Output the frequency counter and the chi-square.

The chi – square is in the following way (for uncorrelated sequences as produced by the above algorithm:

$$\chi^2 = \sum_{l=1}^{L} \frac{(K_l - m p_l)^2}{m p_l}$$

p_l : probability of a sequence to have length l

$$p_l = \frac{1}{l!} - \frac{1}{(l+1)!}$$

K_l : counter storing the number of sequences of length l

m : total number of sequences

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

Performing run-up test

```cpp
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include "ran0.h"
#include "ran1.h"

using namespace std;

int main(int argc, char *argv[])
{

    int lmax=7, lup, Nttl=100000, seed=5097;
    double Xold, Xnew;
    int *Kup=new int[lmax+1];
    int counter=0;

    for(lup=0;lup<lmax+1;lup++) Kup[lup]=0;

    FILE *fp;

    while(counter < Nttl)
    {

        srand(seed+counter);
        Xold=rand()/double(RAND_MAX);
        lup=1;
        Xnew=rand()/double(RAND_MAX);

        while(Xnew>Xold)
        {
            lup++;
            Xold=Xnew;
            Xnew=rand()/double(RAND_MAX);

        }

        counter++;

        if(lup<=lmax) Kup[lup]++;
        else Kup[lmax]++;

    }

    char str[100];

    sprintf(str,"results_run_test_%d.dat",seed);

    fp=fopen(str,"w");

    //calculate chi^2
    double *pl=new double[lmax+1];

    double fact=1;

    double chi2=0;
    pl[0]=1;
    for(lup=1;lup<=lmax;lup++)
    {

        fact=fact*lup;
        pl[lup]=(1/fact)*(lup/(lup+1.0));

    }

    for(lup=1;lup<lmax+1;lup++) fprintf(fp,"%i %i %i\n",lup,Kup[lup],int(Nttl*pl[lup]));

    fclose(fp);

    for(lup=1;lup<=lmax;lup++)
    {
        if(Kup[lup]!=0) chi2+=((Kup[lup]-Nttl*pl[lup])*(Kup[lup]-Nttl*pl[lup]))/(Nttl*pl[lup]);

    }
    cout<<chi2<<"\n";

    return 0;
}
```

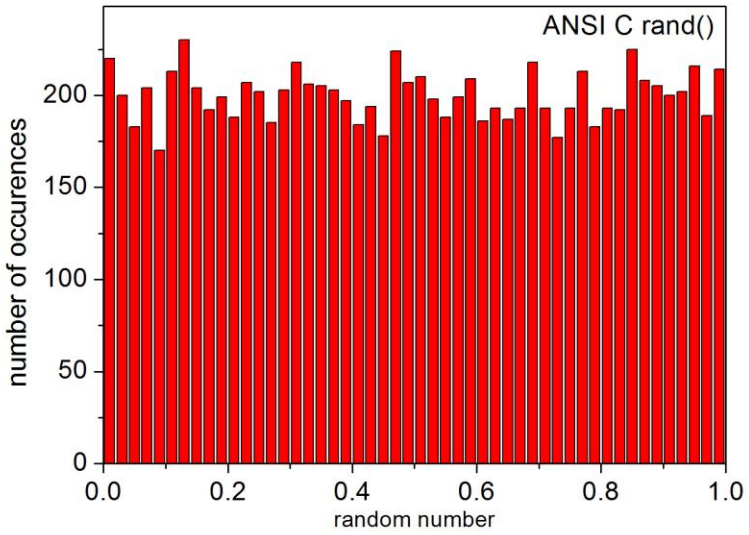
European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

The results of the tests for the case of the ANSI C standard are the following:

Frequency test



$$\left.\begin{array}{c}\chi^2 = 42.98 \\ \nu = N_{bin} - 1 = 49\end{array}\right\} \Rightarrow \chi^2_\nu = 0.877 \sim 1$$

Serial correlation test

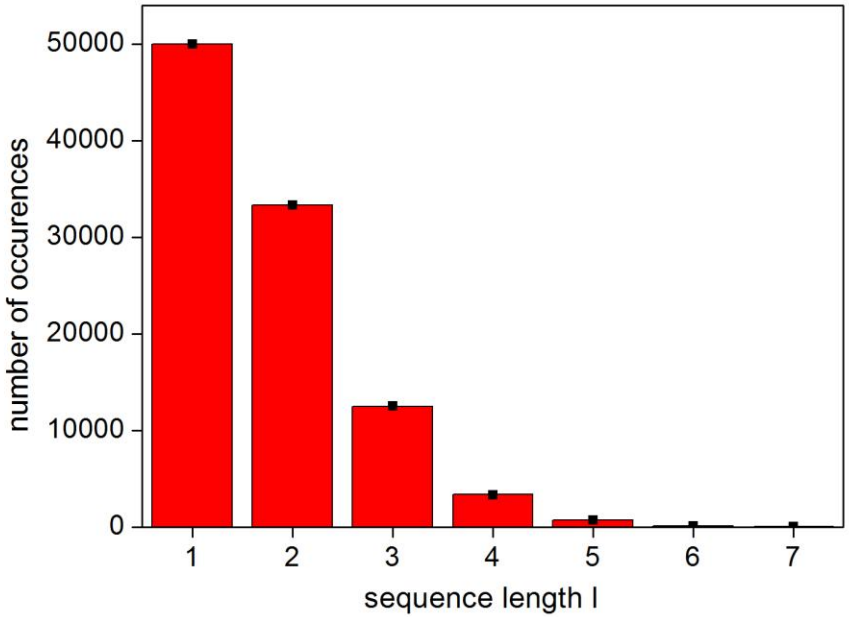
$$C = 0.00797$$

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

Run up test



$$\left.\begin{array}{c}\chi^2 = 1.67 \\ \nu = 7\end{array}\right\} \Rightarrow \chi^2_\nu = \frac{1.67}{7} = 0.238 < 1$$

The ANSI C standard is proven to be a poor random number generator (meaning that it performs worse than other RNGs in the majority of the tests known so far)

Guidelines for the implementation of efficient random number generators can be found in ***"Numerical Recipes in C++"***, 2nd edition, by Press, Vetterling, Teukolsky, Flannery.

The selection of an appropriate random number generator depends on the problem under study.

E.g., if anyone wants to have a site on a 2D lattice, he/she must be very careful to choose an RNG that has the minimal correlations between successive calls.

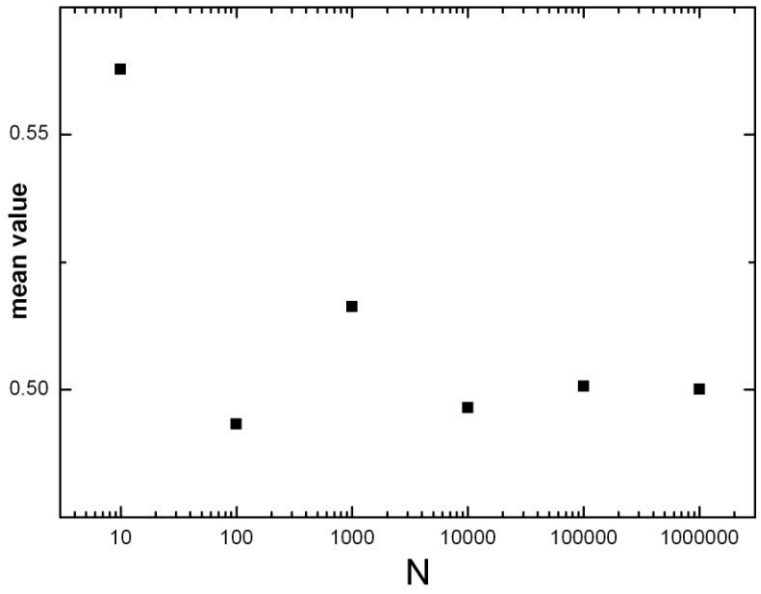
For the rest of the presentation (and just for illustrative purposes) we will use the RNG specified by the compiler. You can use any other RNG at the part of the code where rand() is being used.

It is strictly recommended **TO TEST THE RNG BEFORE ANY USAGE** (or to consult any relative documentation) to decide if it is suitable for the problem you indent to simulate.

In the following slides, we will present a simple program which calculates the mean value of a sequence of N RNs, uniformly distributed and will try to sketch the way of producing RNs with obeying to different distributions.

Exercise

Create a program which calculates the average of N random numbers taken from a uniform random number distribution. The program must run for N=10, 100, 1000, 10000,100000,1000000 random numbers. Plot the mean value as a function of N (it's preferable that the axis of N is logarithmic). Describe your conclusions from the results.



ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

Indicative program

```
#include <iostream>
#include <cstdlib>
#include <cstdio>

using namespace std;

int main(int argc, char *argv[])
{

    int seed=5097,i,j,order=6,N=10;
    double sum;

    FILE *fp=fopen("results.dat","w");

    for(i=1;i<=order;i++)
    {

        sum=0.0;
        srand(seed+i);

        for(j=1;j<=N;j++)  sum+=rand()/double(RAND_MAX);

        sum/=N;
        fprintf(fp,"%i %lf\n",N,sum);
        N*=10;

    }

    fclose(fp);

    return 0;
}
```

Exercise

Create an exponential RNG based on the uniform RNG.

It is easy to show that if x follows a uniform distribution, then we can produce a series of numbers following an exponential distribution.

$$Pr(Y \leq y) = PrX \leq x \Rightarrow F_Y(y) = F_X(x)$$
$$\Rightarrow 1 - e^{-\lambda y} = x \Rightarrow y(x) = -\frac{1}{\lambda}\ln(1-x), x \in [0,1]$$

Because, if x is a uniform random number, it follows that 1-x is of the same distribution, we can write instead

$$y(x) = -\frac{1}{\lambda}\ln(x)$$

For more information refer to the book: **“Numerical recipes in C++”**

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

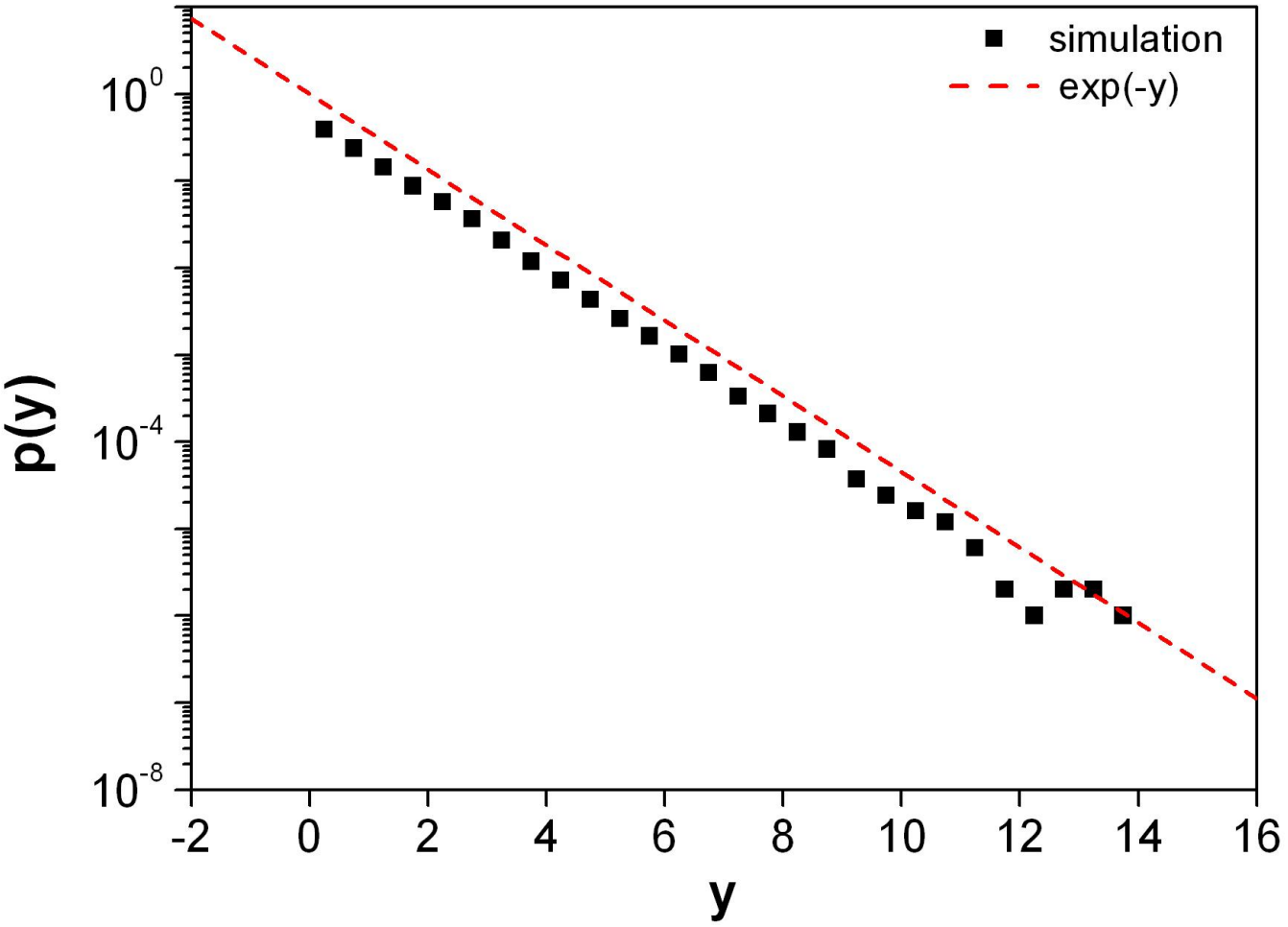
European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cmath>

using namespace std;

int main(int argc, char *argv[])
{

    double x,y;

    int N=atoi(argv[1]);

    int seed=atoi(argv[2]);

    double lambda=atof(argv[3]);

    cout<<N<<" "<<seed<<" "<<lambda<<"\n";

    char str[100];

    sprintf(str,"unif2expo_N%d_l%5.2lf.dat",N,lambda);

    FILE *fp=fopen(str,"w");

    srand(seed);

    while(N)
    {

        x=rand()/(RAND_MAX+0.0);

        y=-log(x)/lambda;

        fprintf(fp,"%lf\n",y);

        N--;

    }

    fclose(fp);

    return 0;
}
```

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

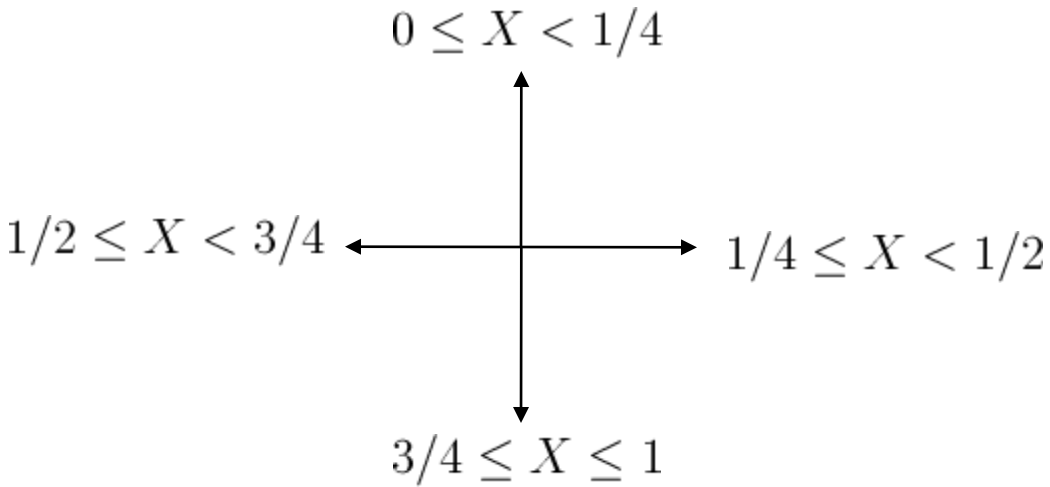
European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria



Random Walk in 2-D

One of the most common problems in the Monte Carlo simulations is the RANDOM WALK.

The basic part is the movement: in a 2-D lattice, a particle moves to a neighboring site according to a simple if – else process. We restrict the random values produced by the RNG in the range [0,1]. We divide this range in four segments of equal width (generally,, in 2d segments, with d being the dimension). The RNs are uniformly distributed, so there is no bias to the resulting values. Then, depending on the segment into which the RN falls, we choose in which direction to move. A possible rule of movement is given below.



European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

Exercise

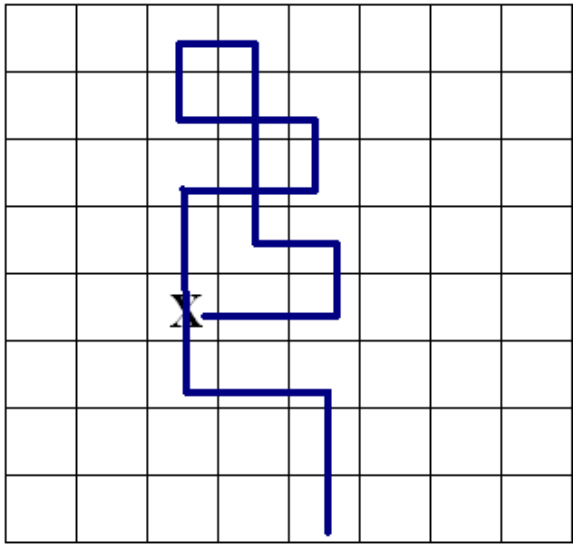
Create a program which simulates the random walk of a particle on a 2D square lattice. The particle performs N=1000 steps. Calculate for 10000 random walkers:

- the average square displacement, $\langle R^2 \rangle$ and
- the average number of distinct sites the random walker has visited

keeping a record for every 100 steps.

Plot the results as a function of time steps.

A random walk may have the following form



European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

The program

```
 0 #include <cstdlib>
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 int main(int argc, char *argv[])
 6 {
 7
 8     int step=atoi(argv[1]);
 9     int numOfSteps=atoi(argv[2]);
10     int numOfRuns=atoi(argv[3]);
11     int seed=atoi(argv[4]);
12     int Ldim=atoi(argv[5]);
13     int numOfVal=numOfSteps/step;
14
15     double r;
16     int i,j,x,y,counterS;
17
18     double *R2=new double[numOfVal];
19     int *S=new int[numOfVal];
20
21     for(i=0;i<numOfVal;i++)
22     {
23             R2[i]=0;
24             S[i]=0;
25
26     }
27
28     int **lattice=new int*[Ldim];
29
30     for(i=0;i<Ldim;i++) lattice[i]=new int[Ldim];
31
32     for(i=0;i<numOfRuns;i++)
33     {
34
35         srand(seed+i);
36
37         for(x=0;x<Ldim;x++) for(y=0;y<Ldim;y++) lattice[x][y]=0;
38
39         x=Ldim/2; y=Ldim/2;
40
41         counterS=0;
42
43         lattice[x][y]=1; counterS++;
44
45         for(j=1;j<=numOfSteps;j++)
46         {
47
48             r=rand()/double(RAND_MAX);
49
50             if(r<0.25) x++;
51             else if(r>=0.25 && r<0.5) x--;
52             else if(r>=0.5 && r<0.75) y++;
53             else y--;
54
55             if(x<0) x=0;
56             else if(x==Ldim) x=Ldim-1;
57             else if(x<0) y=0;
58             else if(y==Ldim) y=Ldim-1;
```

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

The program (cont'd)

```
59
60              if(lattice[x][y]==0)
61              {
62                      lattice[x][y]=1;
63
64                      counterS++;
65
66              }
67
68              if((j%step) == 0)
69              {
70
71                      R2[(j/step)-1]+=(x-Ldim/2)*(x-Ldim/2)+(y-Ldim/2)*(y-Ldim/2);
72
73                      S[(j/step)-1]+=counterS;
74
75              }
76          }
77
78      }
79
80      for(i=0;i<numOfVal;i++)
81      {
82            R2[i]=R2[i]/numOfRuns;
83
84            S[i]=S[i]/numOfRuns;
85
86
87      }
```

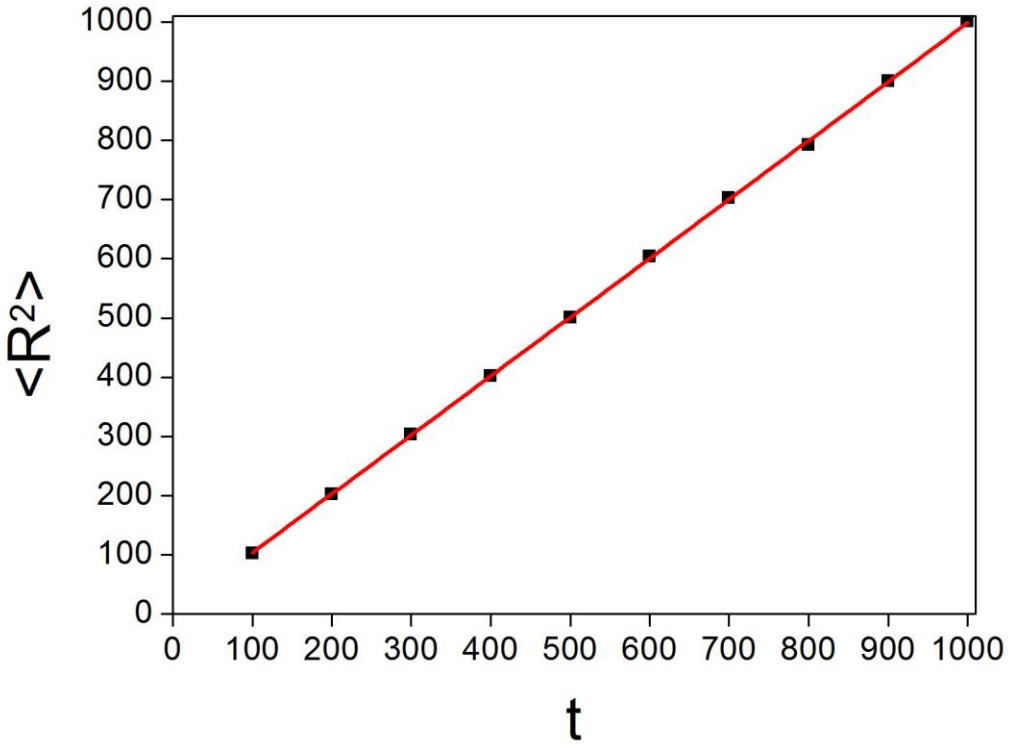
```
88
89      FILE *fp = fopen("RW_R2_results.dat","w");
90
91      for(i=0;i<numOfVal;i++) fprintf(fp,"%i %lf\n", (i+1)*step, R2[i]);
92
93      fclose(fp);
94
95      fp = fopen("RW_S_results.dat","w");
96
97      for(i=0;i<numOfVal;i++) fprintf(fp,"%i %i\n", (i+1)*step, S[i]);
98
99      fclose(fp);
100
101     system("PAUSE");
102     return EXIT_SUCCESS;
103 }
```

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

Results

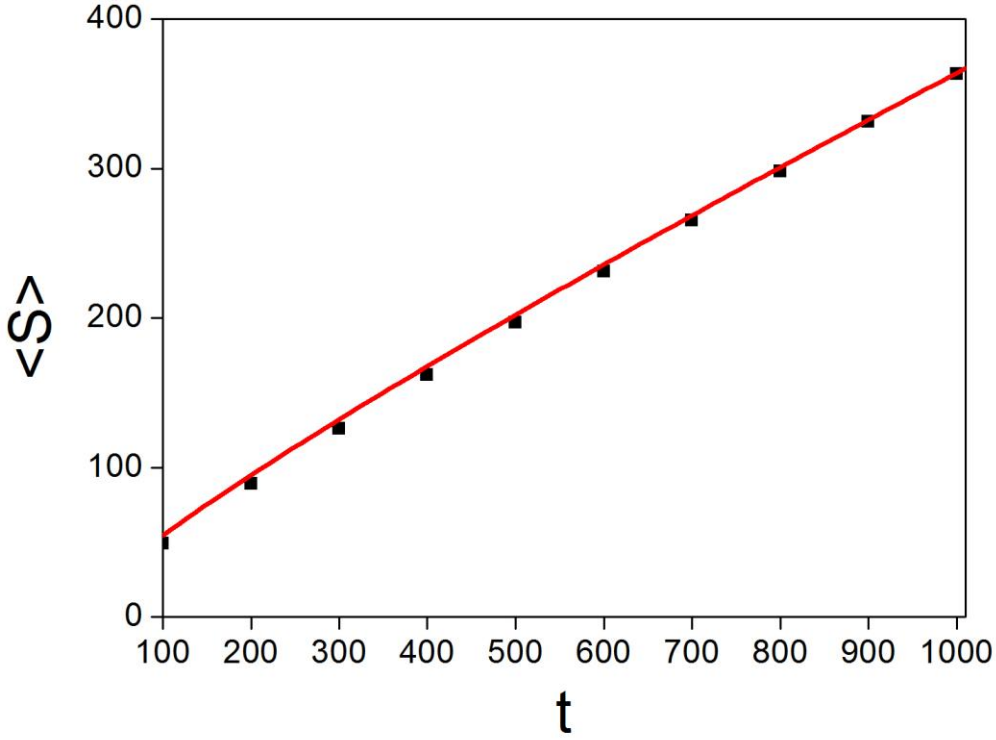


$$\langle R^2 \rangle = 4.04 + 0.99t \sim t$$

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria



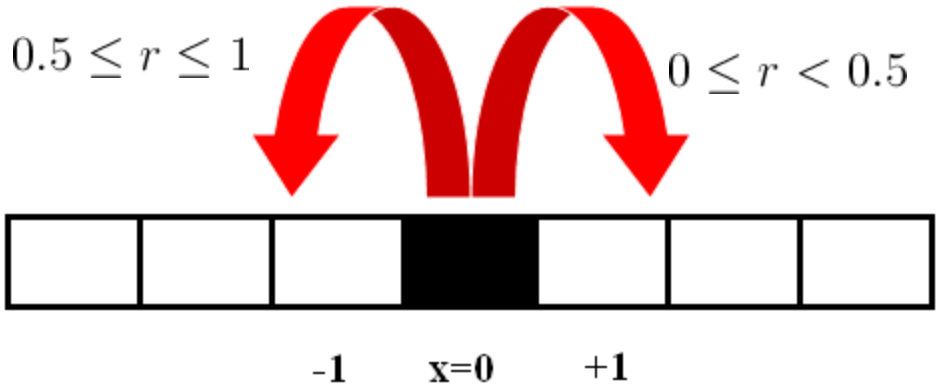
$$\langle S \rangle_{2D} \sim \pi t / \log(t)$$

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

Distribution of <R> in 1-D



European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

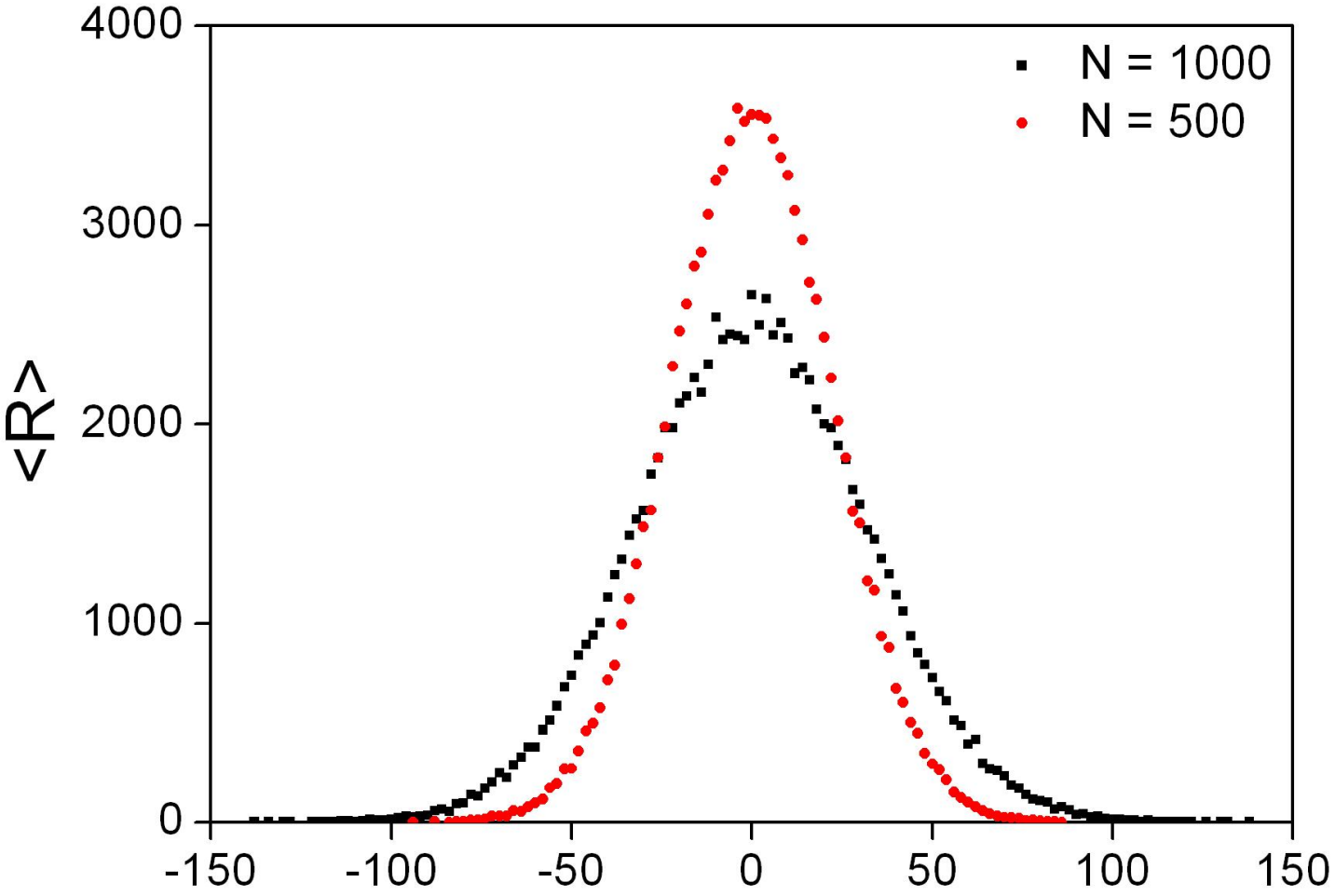
European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

```cpp
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <cstdio>

using namespace std;


int main(int argc, char *argv[]) {


    int runs=atoi(argv[1]);

    int steps=atoi(argv[2]);

    int Ldim=5*int(sqrt(1.0*steps));

    int x;

    double r;

    int i,j;

    int *R_left=new int[Ldim];

    int *R_right=new int[Ldim];

    for(i=0;i<Ldim;i++)
    {

        R_right[i]=0;

        R_left[i]=0;

    }


    for(i=0;i<runs;i++)
    {

        x=0;

        srand(5097+i);

        for(j=0;j<steps;j++)
        {

            r=rand()/(RAND_MAX+0.0);

            if(r>0.5) x++;
            else x--;

        }

        if(x>=0) R_right[x]++;
        else R_left[-x]++;


    }

    char str[100];

    sprintf(str,"results_1-D_RW_%d.dat",steps);

    FILE *fp=fopen(str,"w");

    for(i=Ldim-1;i>0;i--) if(R_left[i]) fprintf(fp,"%d\t%d\n",-i,R_left[i]);
    for(i=0;i<Ldim;i++) if(R_right[i]) fprintf(fp,"%d\t%d\n",i,R_right[i]);

    fclose(fp);

    return 0;
}
```

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria



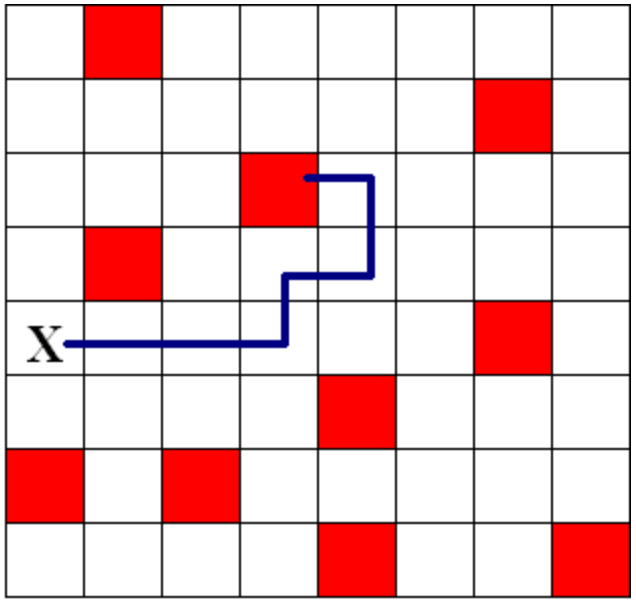
European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

Trapping

Create a program which constructs a a two (2) dimensional lattice of size 400x400. In this lattice, put at random positions a number of trap molecules, which will have concentration c. Put 1 particle in a random position on the lattice and let it perform a random walk. In this walk you will not place a time restriction, namely you will not declare a specific number of steps. The walk will stop when the particle falls on a trap. The time needed for this to happen is the trapping time. Perform 10000 runs, save the trapping times and make the distribution of these times for c=0.02 and 0.2. From this distribution and for c=0.2 calculate the survival probability Φ(c,n) and compare it with the Rosenstock approximation. Describe your conclusions. Beware of boundary conditions. When the particle reaches the borders of the lattice it shouldn't be allowed to escape from it but to remain in the lattice, either by returning on it former position or by being placed in the opposite site of the lattice.



European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

```cpp
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cmath>

using namespace std;


int main(int argc, char *argv[]) {

    int Ldim=atoi(argv[1]);

    double traps=atof(argv[2]);

    int Nwalk=atoi(argv[3]);

    char str[100];

    sprintf(str,"results_trapping_%4.2lf.dat",traps);

    double r;

    int i,j;

    int x,y;

    int **lattice=new int*[Ldim];

    for(i=0;i<Ldim;i++)
    {

        lattice[i]=new int[Ldim];

        for(j=0;j<Ldim;j++) lattice[i][j]=0;

    }

    //putting traps

    traps=floor(traps*Ldim*Ldim);

    cout<<traps<<"\n";
```

```cpp
    while(traps)
    {

        i=int((rand()/(RAND_MAX+1.0))*Ldim);

        j=int((rand()/(RAND_MAX+1.0))*Ldim);

        while(lattice[i][j]==1)
        {

            i=int((rand()/(RAND_MAX+1.0))*Ldim);

            j=int((rand()/(RAND_MAX+1.0))*Ldim);

        }

        lattice[i][j]=1;

        traps--;


    }

    //performing random walk

    int *countTraps=new int[Nwalk];

    int max=0;

    for(i=0;i<Nwalk;i++)
    {

        srand(5097+i);

        countTraps[i]=0;

        x=int((rand()/(RAND_MAX+1.0))*Ldim);

        y=int((rand()/(RAND_MAX+1.0))*Ldim);
```

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

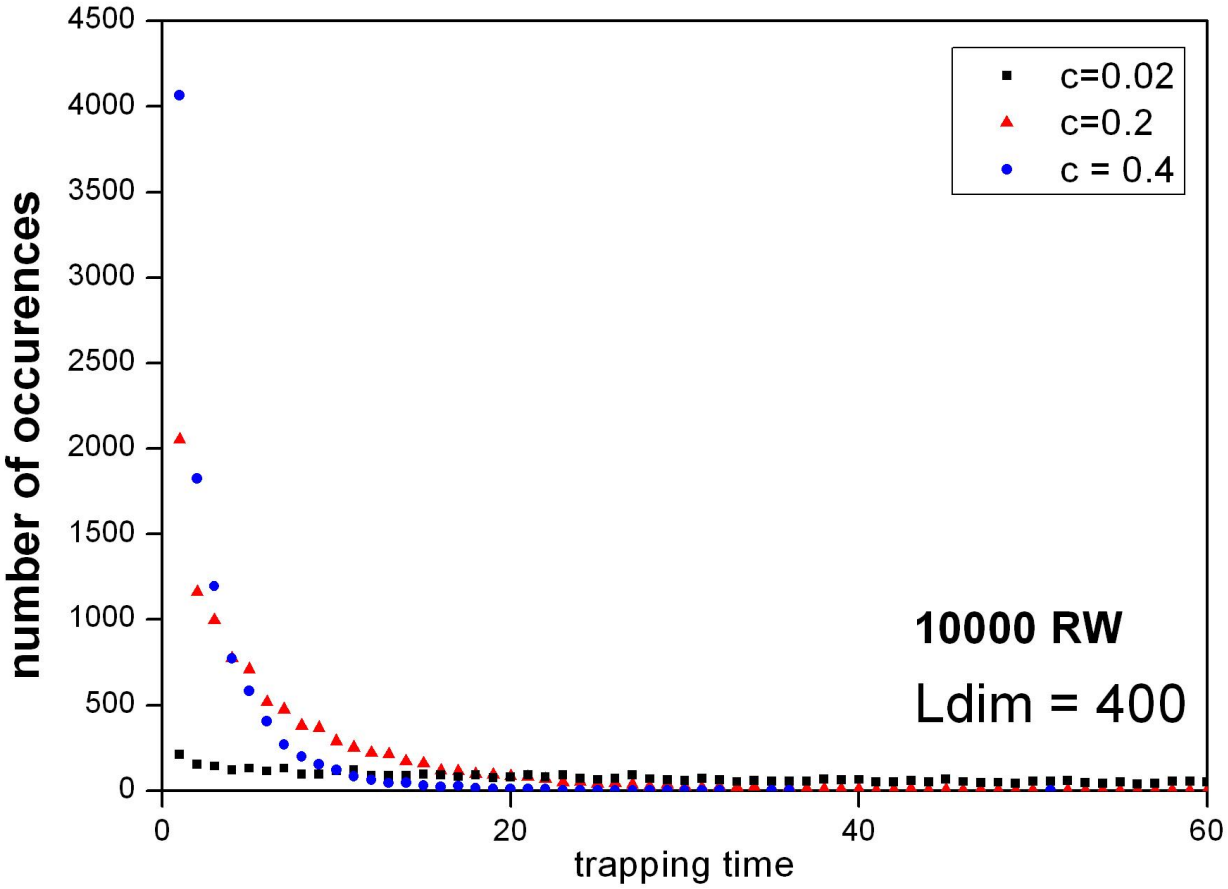
```
83
84          while(lattice[x][y]==1)
85          {
86
87              x=int((rand()/(RAND_MAX+1.0))*Ldim);
88
89              y=int((rand()/(RAND_MAX+1.0))*Ldim);
90
91          }
92
93          //performing RW
94
95          while(lattice[x][y]==0)
96          {
97
98              r=rand()/double(RAND_MAX);
99
100             if(r<0.25) x++;
101             else if(r>=0.25 && r<0.5) x--;
102             else if(r>=0.5 && r<0.75) y++;
103             else y--;
104
105             if(x<0) x=Ldim-1;
106             else if(x==Ldim) x=0;
107             else if(y<0) y=Ldim-1;
108             else if(y==Ldim) y=0;
109
110             countTraps[i]++;
111
112
113         }
114
115         if(max<countTraps[i]) max=countTraps[i];
116
117     }
118
119     int *freq=new int[max+1];
120
121     for(i=0;i<max+1;i++) freq[i]=0;
122
123     for(i=0;i<Nwalk;i++) freq[countTraps[i]]++;
```

```
124
125     FILE *fp=fopen(str,"w");
126
127     for(i=0;i<max+1;i++) if(freq[i]) fprintf(fp,"%i %i\n",i,freq[i]);
128
129     fclose(fp);
130
131
132     return 0;
133 }
```

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria



European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF) and National Funds of Greece and Bulgaria

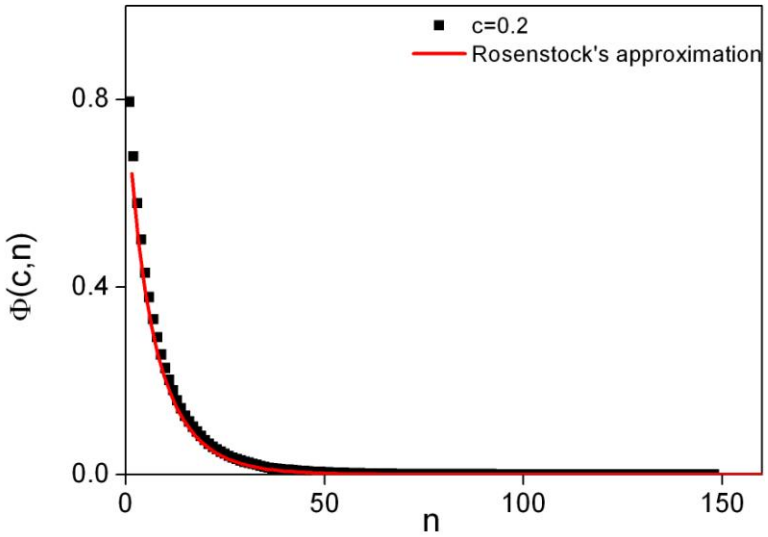
Survival probability

It can be proven that the probability of a particle to survive from trapping is given for 2D by the relation:

$$\Phi(c,n) \simeq (1-c)^{<S_n>}$$

c : *concentration of traps per site*

$$< S_n >_{2D} \sim \pi n / \log(8n)$$



Network growth

The investigation of real world systems led Barabasi and Albert[1] to propose a mechanism for the evolution of networks, based on 2 main features:

1. *Growth*: Starting with a small number of nodes (m_0), at every time step, we add a new node with $m(\leq m_0)$ edges that link the new node to m different nodes already present in the system.
2. *Preferential attachment*: When choosing the nodes to which the new node connects, we assume that the probability Π that a new node will be connected to node i depends on the degree k_i of node i, such that:

$$\Pi(k_i) = k_i / \sum_{j<i} k_j$$

[1] Barabasi and Albert, *"Emergence of Scaling in Random Networks", Science, vol 286, 1999*

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

```cpp
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <vector>

using namespace std;

int main(int argc, char *argv[])
{

    int initNodes=atoi(argv[1]);

    int totalNodes=atoi(argv[2]);

    double p=atof(argv[3]);

    int m=atoi(argv[4]);

    double r;

    vector<int> *network=new vector<int> [totalNodes+1];

    int i,j;

    for(i=0;i<initNodes-1;i++)
    {

        for(j=i+1;j<initNodes;j++)
        {

            if(rand()/double(RAND_MAX) < p )
            {

                network[i].push_back(j);

                network[j].push_back(i);

            }

        }


    }

    int nodeSum=0;

    int node;

    char str[100];

    FILE *fp;

    int k,l;

    vector<int> currNodeSize;

    for(i=0;i<initNodes;i++)
    {

        k=network[i].size();

        currNodeSize.push_back(k);

    }

    for(i=initNodes;i<totalNodes+1;i++)
    {

        srand(5097+i);

        for(j=0;j<m;j++)
        {

            r=rand()/(RAND_MAX+1.0);

            node=int(i*r);

            k = currNodeSize[node];

            l = network[node].size();

            r=rand()/(RAND_MAX+0.0);

            while( r > k/(nodeSum+0.0) || network[node][l-1] == i)
            {

                    r=rand()/(RAND_MAX+1.0);

                    node=int(i*r);

                    k = currNodeSize[node];
```

```
92
93                      l = network[node].size();
94
95                      r=rand()/(RAND_MAX+0.0);
96
97                  }
98
99                  network[node].push_back(i);
100
101                 network[i].push_back(node);
102
103             }
104
105             for(j=0;j<i;j++)
106             {
107
108                 k=network[j].size();
109
110                 currNodeSize[j]=k;
111
112             }
113
114             k=network[i].size();
115
116             currNodeSize.push_back(k);
117
118             nodeSum+=2*m;
119
120             cout<<i<<"\n";
121
122         }
123
124         sprintf(str,"network_nodes%d.dat",i);
125
126         fp=fopen(str,"w");
127
128         for(i=0;i<=totalNodes;i++)
129         {
130
131             fprintf(fp,"%d  %d ",i,network[i].size());
132
133             if(network[i].size())
134             {
135
136                 for(node=0;node<network[i].size();node++) fprintf(fp,"%d ",network[i][node]);
```

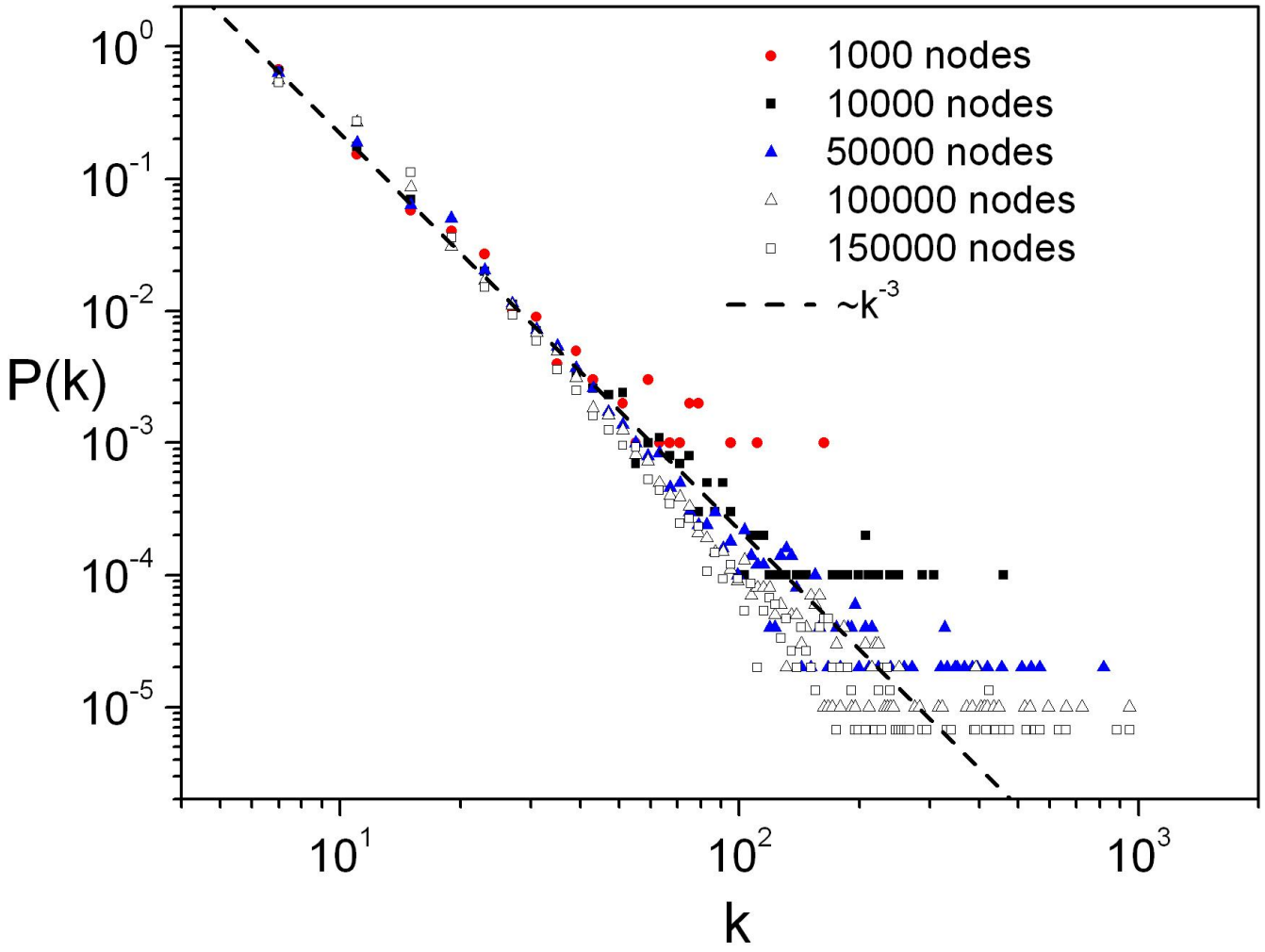
```
137
138             }
139
140             fprintf(fp,"\n");
141
142         }
143
144         fclose(fp);
145
146         sprintf(str,"network_nodes%d_distr.dat",totalNodes);
147
148         fp=fopen(str,"w");
149
150         for(i=0;i<=totalNodes;i++) fprintf(fp,"%d\n",network[i].size());
151
152         fclose(fp);
153
154
155         return 0;
156     }
```

European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013
This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

Results



European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013

This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

Diffusion – Limited Aggregation (DLA)

DLA is a simple model which was initially used to describe the aggregation process of solid particles by Sander and Witten[1].

The procedure is the following:

1. Initially, we position a particle at the origin of the lattice.
2. A second particle is introduced at some random site, a large distance from the origin (generally from the aggregate), which performs random walk.
3. If the particle becomes a nearest neighbor of the aggregate, it becomes a part of it and the walk stops. Else, if it goes at the borders of the lattice (or far from the aggregate) , it disappears.
4. A new particle is introduced, repeating steps 2 and 3.

[1] T.A.Witten, L.M.Sander, ***"Diffusion – Limited Aggregation, a Kinetic Critical Phenomenon"***, PRL vol 47, no 19, 1981

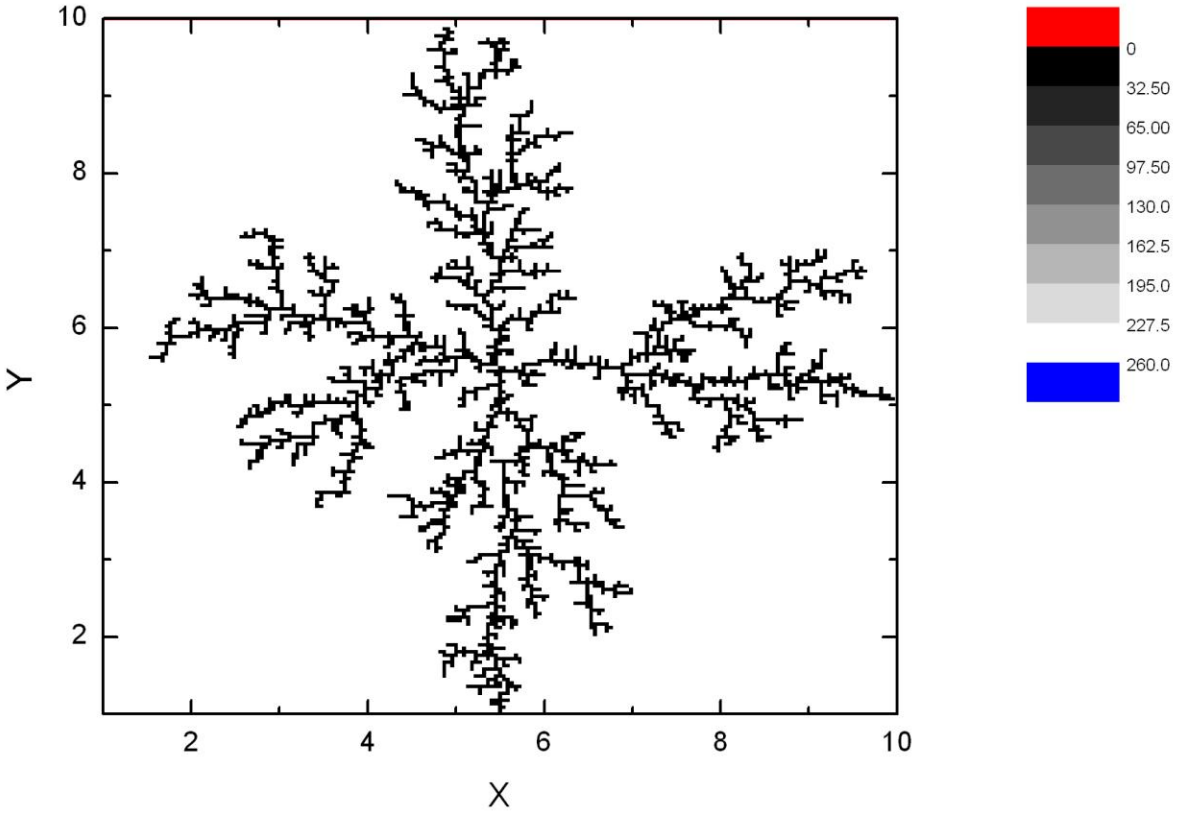
European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013

This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria

Making a simple implementation of the algorithm, for a 201 x 201 square lattice, we get the following result:



European Territorial Cooperation Programme
Greece-Bulgaria 2007-2013
INVESTING IN OUR FUTURE

ICoSCIS

European Territorial Cooperation Progamme
Greece - Bulgaria 2007 - 2013

This Project is co-funded by the European Union (ERDF)
and National Funds of Greece and Bulgaria