UAEM | Universidad Autónoma del Estado de México

Universidad Autónoma del Estado de México

Centro Universitario UAEM Atlacomulco

"Generación de números caóticos y su evaluación de distribución normal con el test de Kolmogorov"

Presenta:

Segundo Antonio Elias Edgardo

Unidad de Aprendizaje:

Estructuras de datos

Fecha de entrega:

7 de septiembre de 2017

1 Introducción

La generación de números genuinamente aleatorios es prácticamente imposible, sin embargo, eso no impide poder obtener números pseudo aleatorios generados por software o hardware con muy buen margen de comportamiento.

En esta práctica se desarrollará un programa que trate de acercarse a lo anterior con el uso de las funciones aleatorias definidas en C y el seteo de márgenes de generación superior e inferior, con esto se pretende crear pares ordenados aleatorios que se evalúen sobre la siguiente función:

$$F(x,y)= pow(\ (x*x*y)+ (3*x*y*y)+ (2*x*y)+ 1\ ,-1\ )$$

Con los valores generados se desplegará la gráfica de los puntos ordenados y adicionalmente la gráfica de la función mostrada, después se realizará el método estadístico de Kolmogorov para realizar el análisis de distribución de los números.

Para las gráficas se hará uso de la librería GNU Plot, sin embargo, para el cálculo estadístico se realizó el algoritmo de cero,

Como último requisito de la práctica se ordenarán los valores de x,y,z en la tabulación de menor a mayor, respecto a una comparación en la variable z que afectara a sus puntos de evaluación.

Finalmente, lo que se busca es comprobar que efectivamente de acuerdo con los resultados arrojados por el test de Kolmogorov se rechazara la hipótesis si se obtiene un valor menor a 0.05 o no se tendrán pruebas para refutarla en caso de ser mayor.

Sin más se presenta el marco teórico que presenta algunos conceptos importantes de los números aleatorios y sus funciones en C.

2 Marco teórico

La función rand() que posee el lenguaje C no genera números aleatorios genuinos sino más bien números pseudo aleatorios sujetos a una semilla (comúnmente el tiempo), existen más funciones útiles descritas por Joyanes Aguilar en este fragmento:

> "Las funciones usuales de la biblioteca estándar de C son: rand, random, randomize y srand. Estas funciones se encuentran en el archivo stdlib.h"

(Joyanes Aguilar, 2005)

En este caso solo se utilizará la función rand en conjunto con la semilla srand() para generar los números de esta práctica, Joyanes también describe el funcionamiento de rand:

> "La función rand genera un número aleatorio. El número calculado por rand varía en el rango entero de O a RAND-MAX."

(Joyanes Aguilar, 2005)

Como se mencionó el uso plano de rand puede no ser muy útil para muchas ejecuciones del programa, por lo que se usara también la función srand() para sembrar una semilla en rand y mejorar su nivel de aleatoriedad, el concepto de semilla en este contexto se muestra a continuación:

> "La función srand inicializa el generador de números aleatorios. Se utiliza para fijar el punto de comienzo para la generación de series de números aleatorios; este valor se denomina semilla ."

(Joyanes Aguilar, 2005)

Sin embargo, aparte de la semilla se utilizarán métodos extras para mejorar la salida de aleatorios, estos cambios pueden apreciarse en el código y en los diagramas.

En esta práctica se implementaron muchas de las técnicas vistas en las anteriores, se utilizaron estructuras de datos para encapsular variables, también se usaron los métodos de graficacion de la práctica pasada.

Respecto al teste de Kolmogorov se usaron los siguientes pasos:

- Se crea una tabla con cinco columnas, las columnas contendrán lo siguiente:
- La primera estará conformada por los valores muestra a evaluar

- La segunda tendrá los valores de las equi-probabilidades de acuerdo con la formula I/N

- La tercera columna estará rellena con la tipificación de los datos, para lo cual se requerirá del promedio y la desviación estándar.

- Para la cuarta columna se requerirá asignar un valor de acuerdo con la tabla de la normal estándar respecto al valor de la columna de tipificación.

- La columna siguiente tendrá la diferencia de las equiprobabilidades y el valor de la cuarta columna.

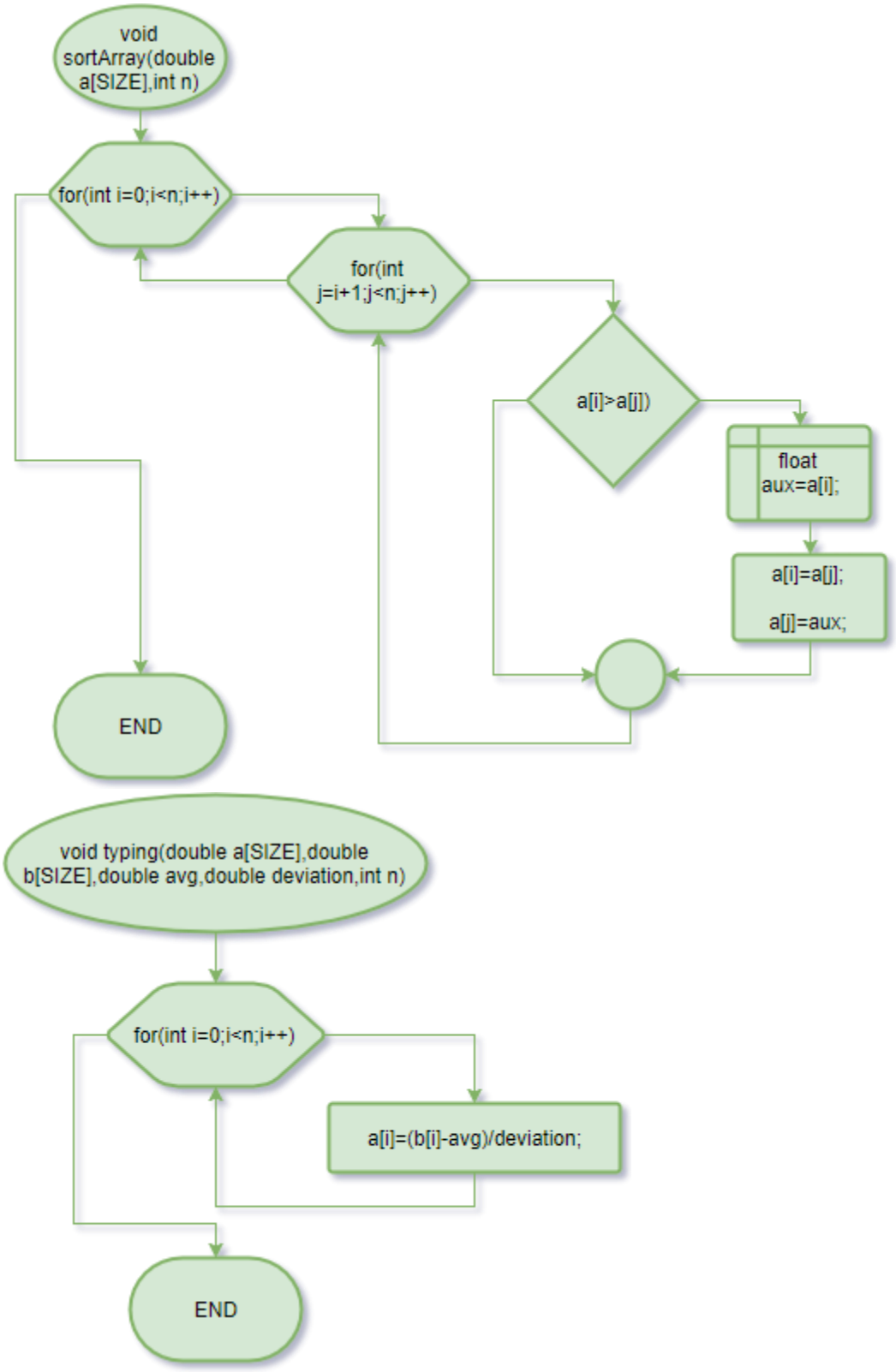
- La última columna será parecida a la anterior, solo que en este caso se sacará el valor absoluto de la diferencia de la equiprobabilidad anterior y el valor de la cuarta columna.

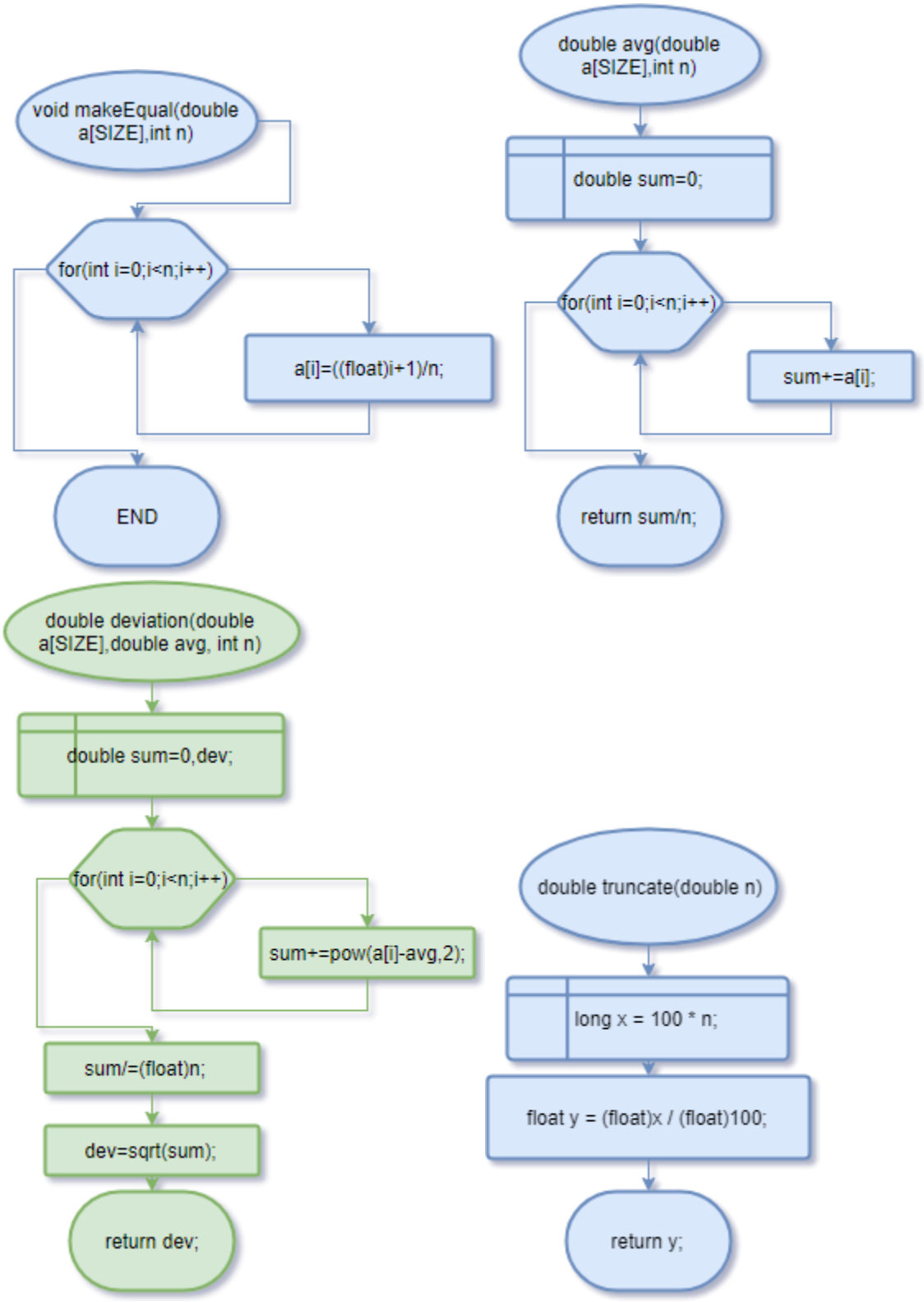
Finalmente se buscará el valor más alto de la 5ta y 6ta columnas y se determinara que ese valor es el coeficiente buscado.

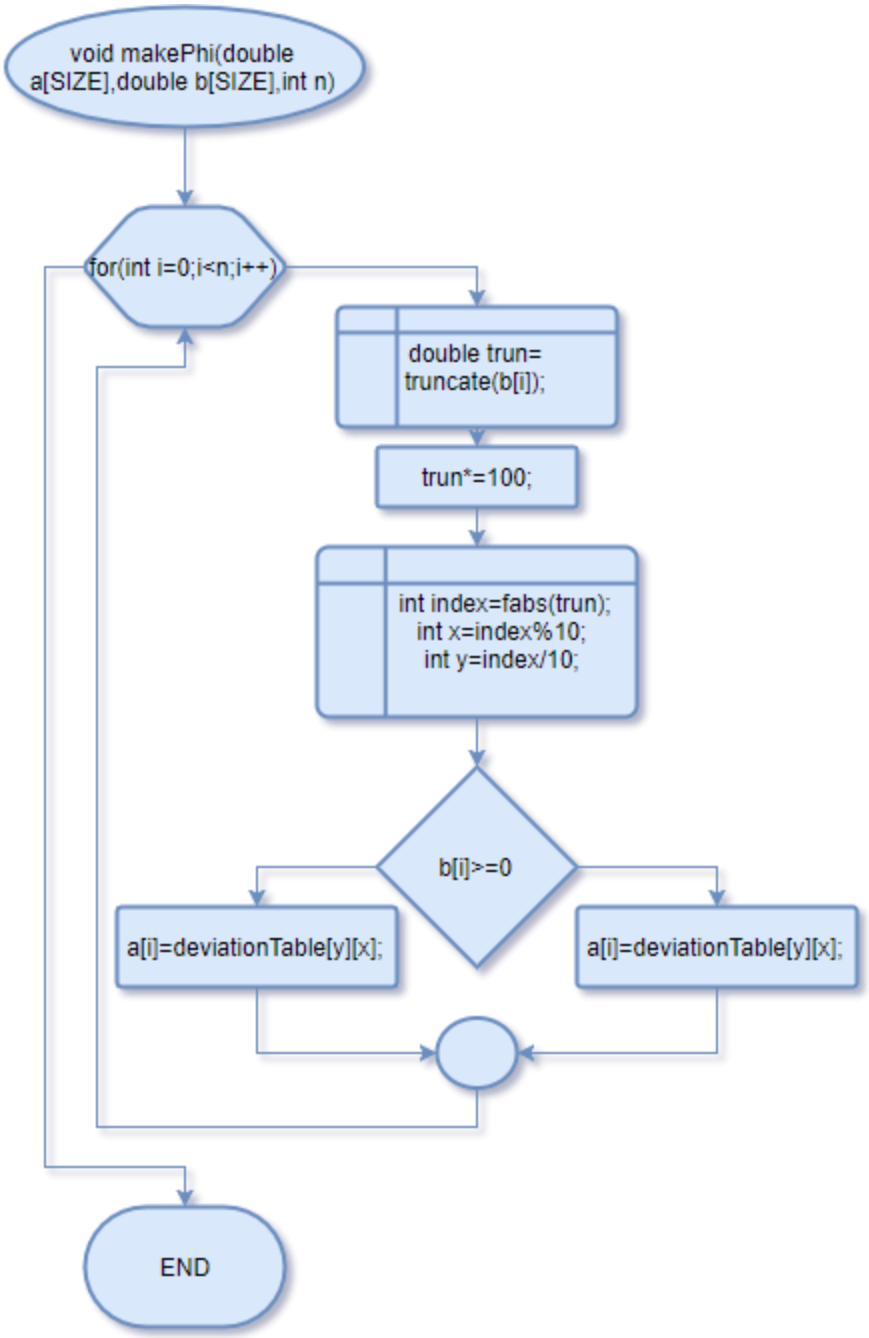
Ahora que ya se ha mostrado el fondo teórico de lo implementado se mostrara el código y los diagramas correspondientes.

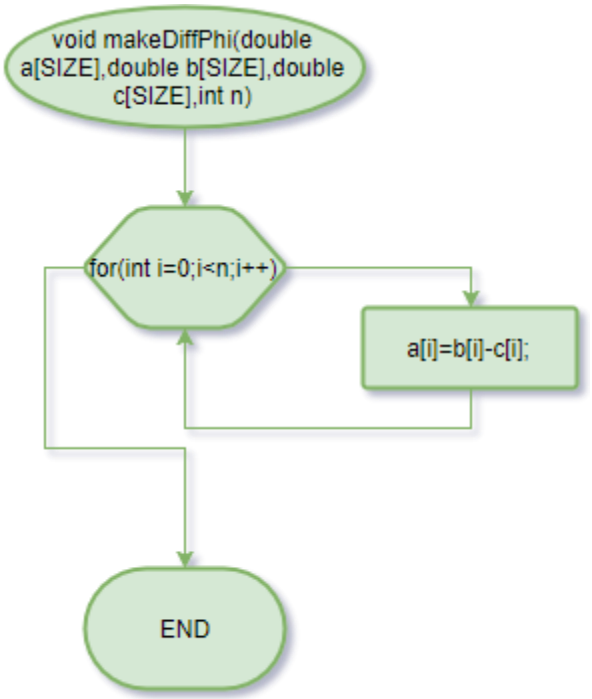
3 Desarrollo

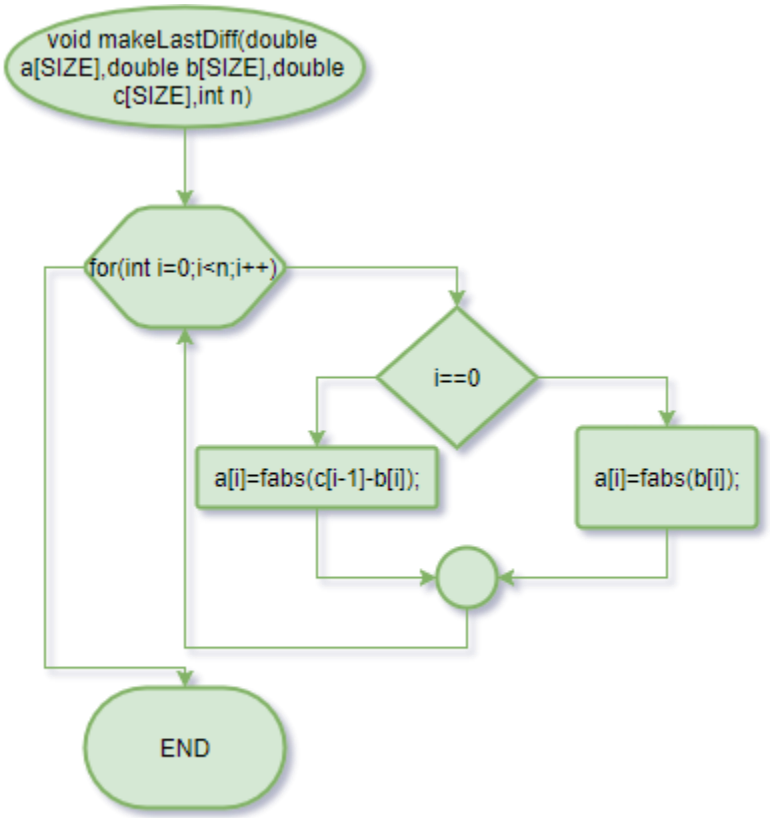
A continuación, se mostrarán los diagramas de flujo de la práctica realizada, esta se dividió en un gran número de funciones para mantener el orden dentro del programa y separar la lógica del test estadístico del de la generación de números:

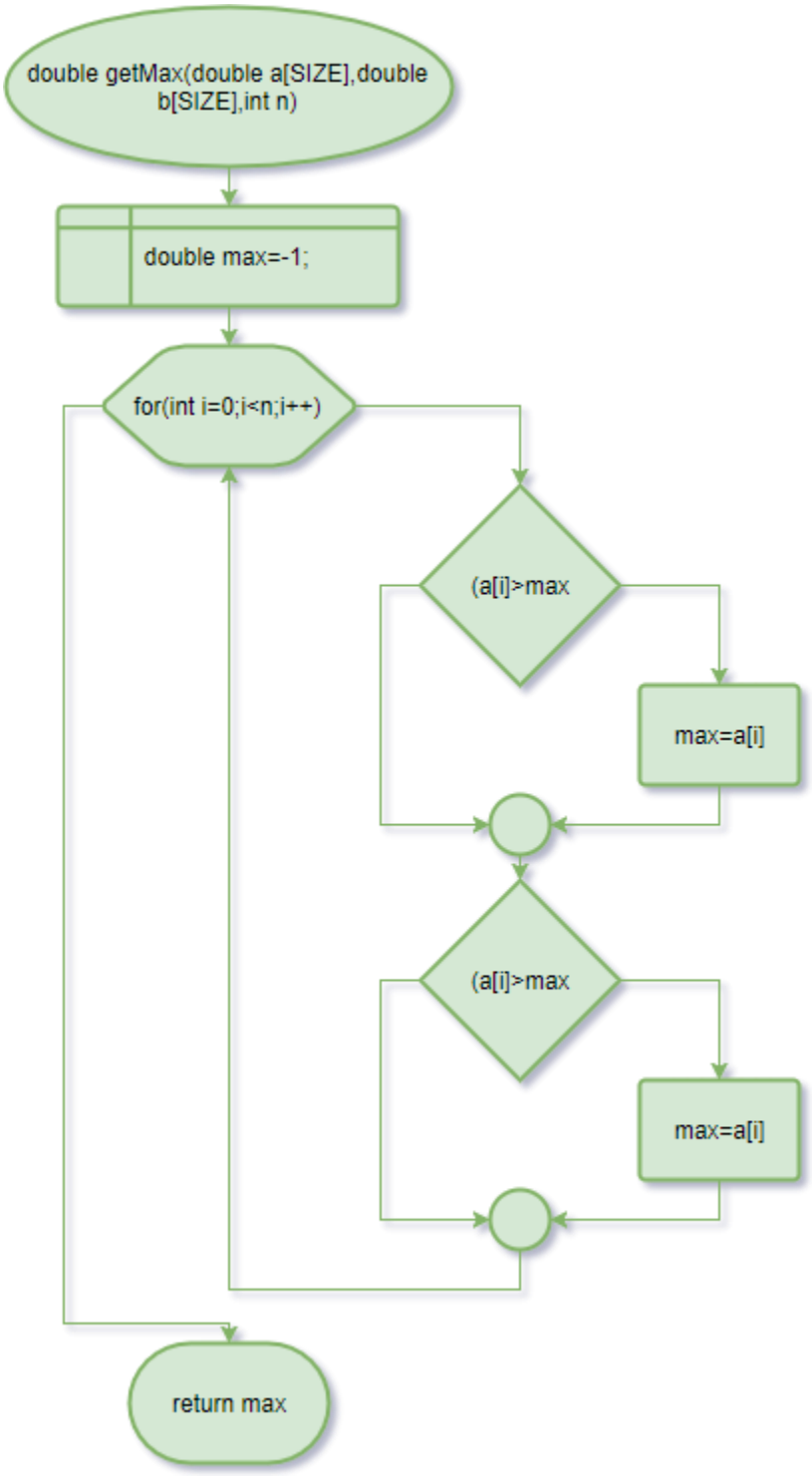

void sortArray(double a[SIZE],int n)
for(int i=0;i<n;i++)
for(int j=i+1;j<n;j++)
a[i]>a[j])
float aux=a[i];
a[i]=a[j];
a[j]=aux;
END
void typing(double a[SIZE],double b[SIZE],double avg,double deviation,int n)
for(int i=0;i<n;i++)
a[i]=(b[i]-avg)/deviation;
END

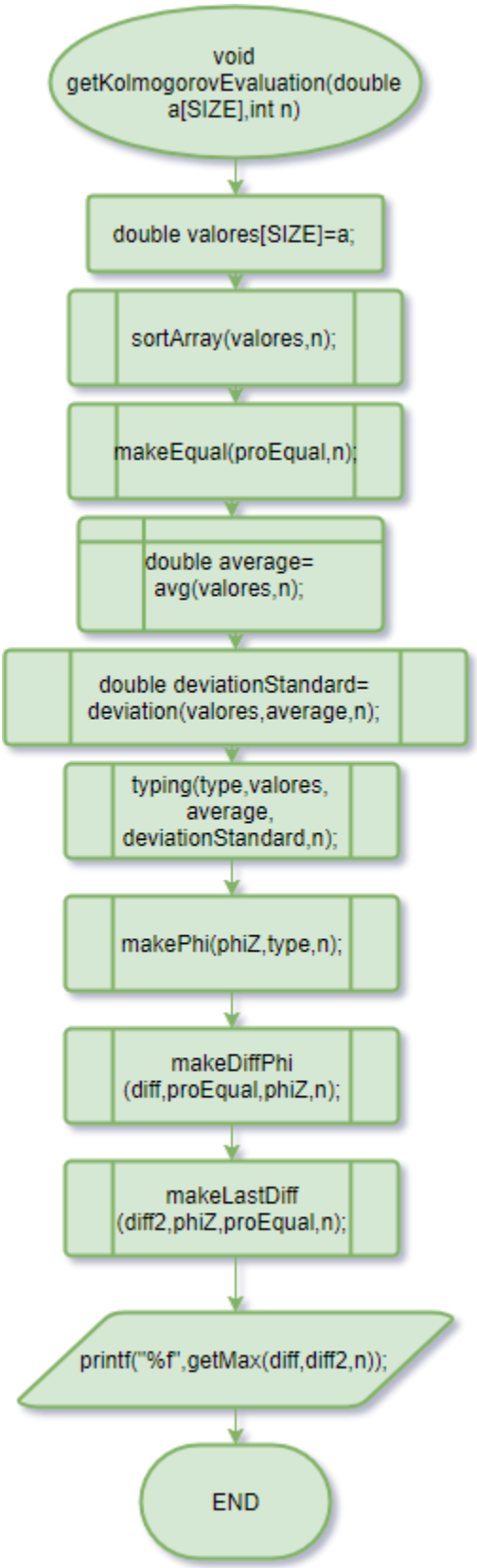

void makeEqual(double a[SIZE],int n)
for(int i=0;i<n;i++)
a[i]=((float)i+1)/n;
END
double avg(double a[SIZE],int n)
double sum=0;
for(int i=0;i<n;i++)
sum+=a[i];
return sum/n;
double deviation(double a[SIZE],double avg, int n)
double sum=0,dev;
for(int i=0;i<n;i++)
sum+=pow(a[i]-avg,2);
sum/=(float)n;
dev=sqrt(sum);
return dev;
double truncate(double n)
long x = 100 * n;
float y = (float)x / (float)100;
return y;

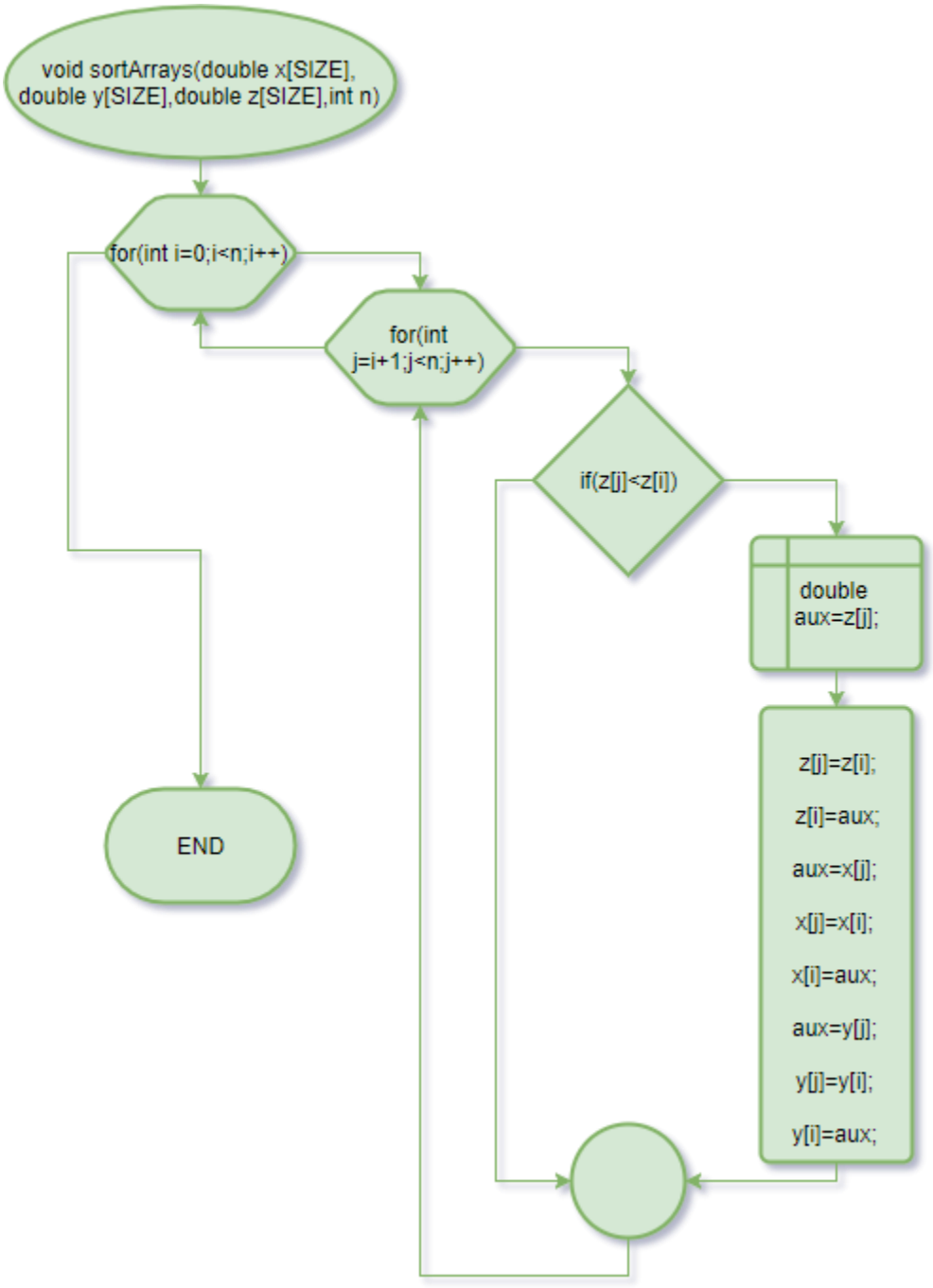

void makePhi(double a[SIZE],double b[SIZE],int n)
for(int i=0;i<n;i++)
double trun= truncate(b[i]);
trun*=100;
int index=fabs(trun);
int x=index%10;
int y=index/10;
b[i]>=0
a[i]=deviationTable[y][x];
a[i]=deviationTable[y][x];
END

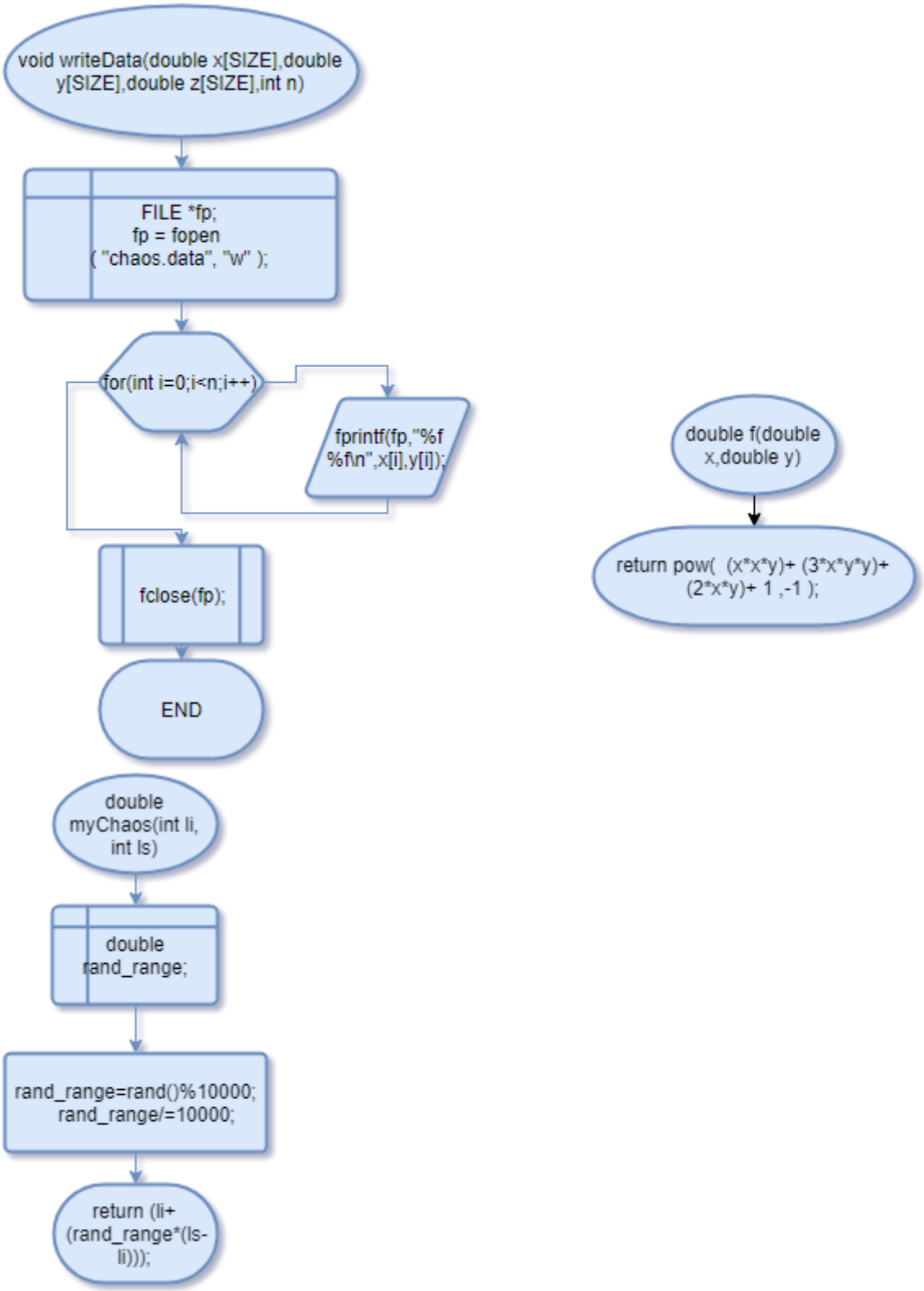

void makeDiffPhi(double a[SIZE],double b[SIZE],double c[SIZE],int n)
for(int i=0;i<n;i++)
a[i]=b[i]-c[i];
END

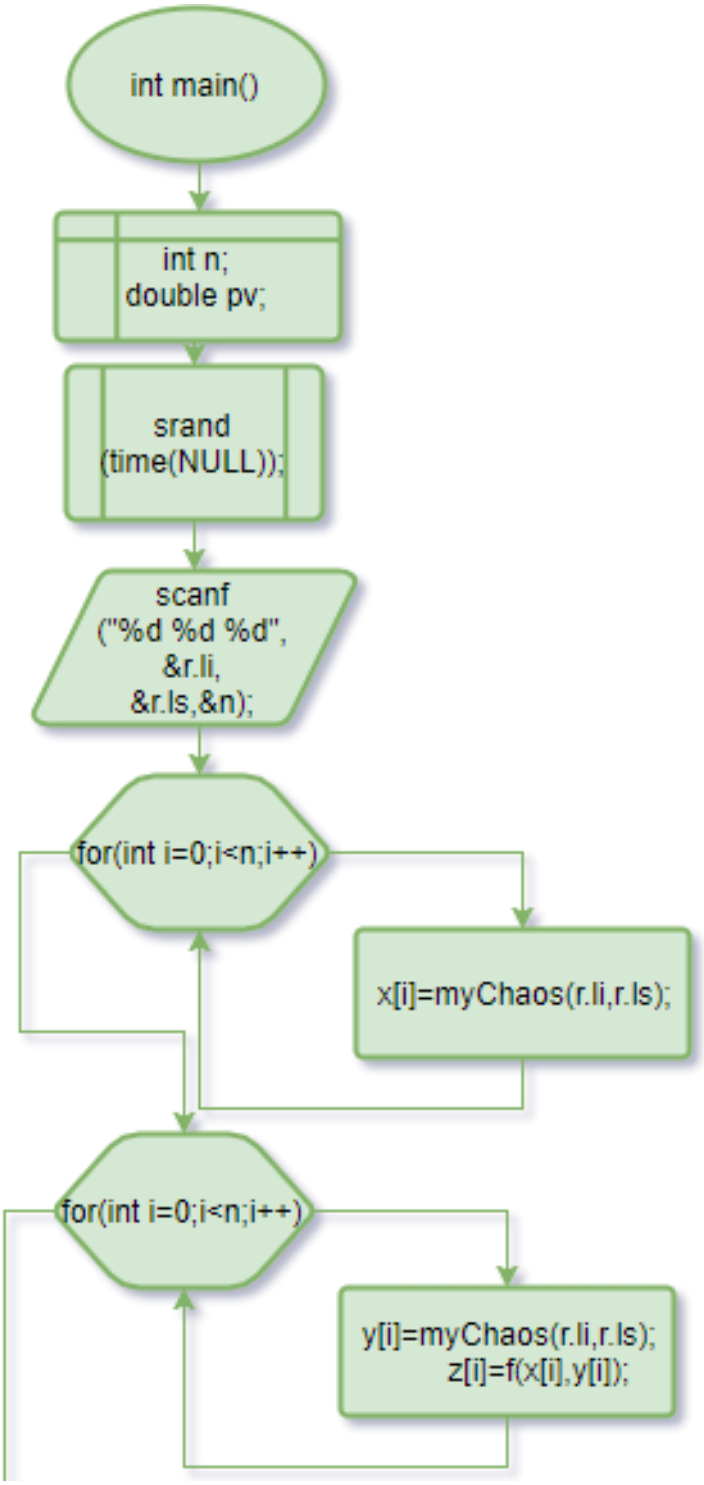

void makeLastDiff(double a[SIZE],double b[SIZE],double c[SIZE],int n)
for(int i=0;i<n;i++)
i==0
a[i]=fabs(c[i-1]-b[i]);
a[i]=fabs(b[i]);
END

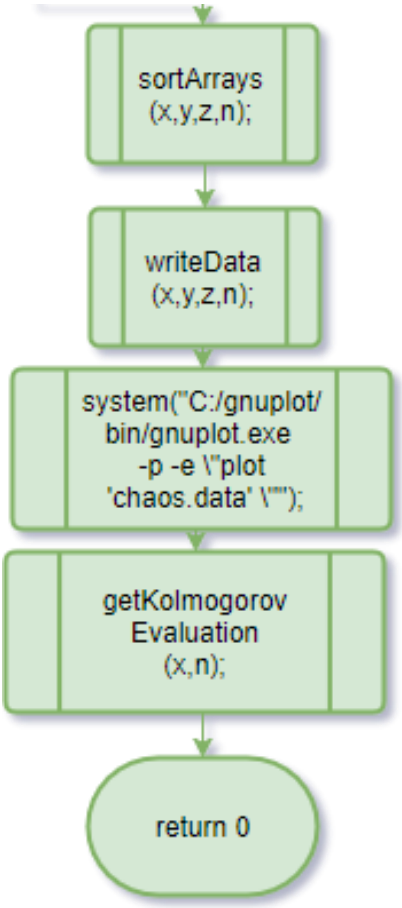

double getMax(double a[SIZE],double b[SIZE],int n)
double max=-1;
for(int i=0;i<n;i++)
(a[i]>max
max=a[i]
(a[i]>max
max=a[i]
return max


void getKolmogorovEvaluation(double a[SIZE],int n)
double valores[SIZE]=a;
sortArray(valores,n);
makeEqual(proEqual,n);
double average= avg(valores,n);
double deviationStandard= deviation(valores,average,n);
typing(type,valores, average, deviationStandard,n);
makePhi(phiZ,type,n);
makeDiffPhi (diff,proEqual,phiZ,n);
makeLastDiff (diff2,phiZ,proEqual,n);
printf("%f",getMax(diff,diff2,n));
END


void sortArrays(double x[SIZE], double y[SIZE],double z[SIZE],int n)
for(int i=0;i<n;i++)
for(int j=i+1;j<n;j++)
if(z[j]<z[i])
double aux=z[j];
z[j]=z[i];
z[i]=aux;
aux=x[j];
x[j]=x[i];
x[i]=aux;
aux=y[j];
y[j]=y[i];
y[i]=aux;
END


void writeData(double x[SIZE],double y[SIZE],double z[SIZE],int n)
FILE *fp;
fp = fopen
("chaos.data", "w");
for(int i=0;i<n;i++)
fprintf(fp,"%f %f\n",x[i],y[i]);
fclose(fp);
END
double myChaos(int li, int ls)
double rand_range;
rand_range=rand()%10000;
rand_range/=10000;
return (li+ (rand_range*(ls-li)));
double f(double x,double y)
return pow((x*x*y)+ (3*x*y*y)+ (2*x*y)+ 1 ,-1);


int main()
int n;
double pv;
srand
(time(NULL));
scanf
("%d %d %d",
&r.li,
&r.ls,&n);
for(int i=0;i<n;i++)
x[i]=myChaos(r.li,r.ls);
for(int i=0;i<n;i++)
y[i]=myChaos(r.li,r.ls);
z[i]=f(x[i],y[i]);



Caos.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define SIZE 50000


/*Codigo propio del calculo de Kolmogorov Smirnov*/
double proEqual[SIZE];
```

```
double valores[SIZE]={1.8,1.628,1.352,1.420,1.594,2.132,1.614,1.924,1.692};
double type[SIZE];
double phiZ[SIZE];
double diff[SIZE];
double diff2[SIZE];
double deviationTable[35][10]=
{
 {.5000, .5040, .5080, .5120, .5160, .5199, .5239, .5279, .5319, .5359},
 {.5398, .5438, .5478, .5517, .5557, .5596, .5636, .5675, .5714, .5753},
 {.5793, .5832, .5871, .5910, .5948, .5987, .6026, .6064, .6103, .6141},
 {.6179, .6217, .6255, .6293, .6331, .6368, .6406, .6443, .6480, .6517},
 {.6554, .6591, .6628, .6664, .6700, .6736, .6772, .6808, .6844, .6879},
 {.6915, .6950, .6985, .7019, .7054, .7088, .7123, .7157, .7190, .7224},
 {.7257, .7291, .7324, .7357, .7389, .7422, .7454, .7486, .7517, .7549},
 {.7580, .7611, .7642, .7673, .7704, .7734, .7764, .7794, .7823, .7852},
 {.7881, .7910, .7939, .7967, .7995, .8023, .8051, .8078, .8106, .8133},
 {.8159, .8186, .8212, .8238, .8264, .8289, .8315, .8340, .8365, .8389},
 {.8413, .8438, .8461, .8485, .8508, .8531, .8554, .8577, .8599, .8621},
 {.8643, .8665, .8686, .8708, .8729, .8749, .8770, .8790, .8810, .8830},
 {.8849, .8869, .8888, .8907, .8925, .8944, .8962, .8980, .8997, .9015},
 {.9032, .9049, .9066, .9082, .9099, .9115, .9131, .9147, .9162, .9177},
 {.9192, .9207, .9222, .9236, .9251, .9265, .9279, .9292, .9306, .9319},
 {.9332, .9345, .9357, .9370, .9382, .9394, .9406, .9418, .9429, .9441},
 {.9452, .9463, .9474, .9484, .9495, .9505, .9515, .9525, .9535, .9545},
 {.9554, .9564, .9573, .9582, .9591, .9599, .9608, .9616, .9625, .9633},
 {.9641, .9649, .9656, .9664, .9671, .9678, .9686, .9693, .9699, .9706},
 {.9713, .9719, .9726, .9732, .9738, .9744, .9750, .9756, .9761, .9767},
 {.9772, .9778, .9783, .9788, .9793, .9798, .9803, .9808, .9812, .9817},
 {.9821, .9826, .9830, .9834, .9838, .9842, .9846, .9850, .9854, .9857},
 {.9861, .9864, .9868, .9871, .9875, .9878, .9881, .9884, .9887, .9890},
 {.9893, .9896, .9898, .9901, .9904, .9906, .9909, .9911, .9913, .9916},
 {.9918, .9920, .9922, .9925, .9927, .9929, .9931, .9932, .9934, .9936},
 {.9938, .9940, .9941, .9943, .9945, .9946, .9948, .9949, .9951, .9952},
 {.9953, .9955, .9956, .9957, .9959, .9960, .9961, .9962, .9963, .9964},
 {.9965, .9966, .9967, .9968, .9969, .9970, .9971, .9972, .9973, .9974},
 {.9974, .9975, .9976, .9977, .9977, .9978, .9979, .9979, .9980, .9981},
 {.9981, .9982, .9982, .9983, .9984, .9984, .9985, .9985, .9986, .9986},
 {.9987, .9987, .9987, .9988, .9988, .9989, .9989, .9989, .9990, .9990},
 {.9990, .9991, .9991, .9991, .9992, .9992, .9992, .9992, .9993, .9993},
 {.9993, .9993, .9994, .9994, .9994, .9994, .9994, .9995, .9995, .9995},
 {.9995, .9995, .9995, .9996, .9996, .9996, .9996, .9996, .9996, .9997},
 {.9997, .9997, .9997, .9997, .9997, .9997, .9997, .9997, .9997, .9998}
};
double deviationTableForNegatives[35][10]={
   {.5000, .4960, .4920, .4880, .4840, .4801, .4761, .4721, .4681 ,.4641},
   {.4602, .4562, .4522, .4483, .4443, .4404, .4364 ,.4325 ,.4286, .4247},
   {.4207, .4168, .4129 ,.4090, .4052, .4013, .3974 ,.3936 ,.3897, .3859},
   {.3821, .3783 ,.3745 ,.3707 ,.3669 ,.3632 ,.3594 ,.3557 ,.3520 ,.3483},
   {.3446 ,.3409 ,.3372 ,.3336 ,.3300 ,.3264 ,.3228 ,.3192 ,.3156 ,.3121},
   {.3085 ,.3050 ,.3015 ,.2981 ,.2946 ,.2912 ,.2877 ,.2843 ,.2810 ,.2776},
   {.2743 ,.2709 ,.2676 ,.2643 ,.2611 ,.2578 ,.2546 ,.2514 ,.2483 ,.2451},
   {.2420, .2389, .2358 ,.2327 ,.2296 ,.2266 ,.2236 ,.2206 ,.2177 ,.2148},
   {.2119 ,.2090, .2061 ,.2033 ,.2005 ,.1977, .1949 ,.1922 ,.1894, .1867},
```

```
    {.1841, .1814 ,.1788 ,.1762, .1736 ,.1711 ,.1685 ,.1660 ,.1635 ,.1611},
    {.1587, .1562 ,.1539, .1515 ,.1492 ,.1469 ,.1446 ,.1423 ,.1401 ,.1379},
    {.1357, .1335, .1314, .1292 ,.1271, .1251 ,.1230 ,.1210 ,.1190 ,.1170},
    {.1151, .1131, .1112, .1093 ,.1075, .1056 ,.1038 ,.1020 ,.1003 ,.0985},
    {.0968, .0951 ,.0934 ,.0918 ,.0901 ,.0885 ,.0869 ,.0853 ,.0838 ,.0823},
    {.0808 ,.0793 ,.0778 ,.0764 ,.0749 ,.0735 ,.0721 ,.0708 ,.0694 ,.0681},
    {.0668 ,.0655 ,.0643 ,.0630, .0618, .0606 ,.0594 ,.0582 ,.0571 ,.0559},
    {.0548 ,.0537 ,.0526 ,.0516 ,.0505 ,.0495 ,.0485 ,.0475 ,.0465 ,.0455},
    {.0446 ,.0436, .0427, .0418 ,.0409 ,.0401 ,.0392 ,.0384 ,.0375 ,.0367},
    {.0359 ,.0351, .0344 ,.0336 ,.0329 ,.0322 ,.0314 ,.0307 ,.0301 ,.0294},
    {.0287, .0281 ,.0274 ,.0268 ,.0262 ,.0256 ,.0250 ,.0244 ,.0239 ,.0233},
    {.0228, .0222 ,.0217 ,.0212 ,.0207 ,.0202 ,.0197, .0192 ,.0188 ,.0183},
    {.0179 ,.0174 ,.0170 ,.0166 ,.0162 ,.0158 ,.0154 ,.0150 ,.0146 ,.0143},
    {.0139 ,.0136, .0132 ,.0129 ,.0125 ,.0122 ,.0119 ,.0116 ,.0113 ,.0110},
    {.0107 ,.0104 ,.0102 ,.0099, .0096, .0094 ,.0091 ,.0089 ,.0087 ,.0084},
    {.0082 ,.0080 ,.0078 ,.0075 ,.0073 ,.0071 ,.0069, .0068 ,.0066 ,.0064},
    {.0062 ,.0060 ,.0059 ,.0057 ,.0055 ,.0054 ,.0052 ,.0051 ,.0049 ,.0048},
    {.0047 ,.0045 ,.0044 ,.0043 ,.0041, .0040 ,.0039 ,.0038 ,.0037 ,.0036},
    {.0035 ,.0034 ,.0033 ,.0032 ,.0031 ,.0030 ,.0029 ,.0028 ,.0027 ,.0026},
    {.0026 ,.0025 ,.0024 ,.0023 ,.0023 ,.0022 ,.0021 ,.0021 ,.0020 ,.0019},
    {.0019 ,.0018, .0018, .0017 ,.0016 ,.0016 ,.0015 ,.0015, .0014 ,.0014},
    {.0013 ,.0013 ,.0013 ,.0012 ,.0012 ,.0011 ,.0011 ,.0011 ,.0010 ,.0010},
    {.0010 ,.0009 ,.0009, .0009 ,.0008, .0008 ,.0008 ,.0008, .0007 ,.0007},
    {.0007 ,.0007 ,.0006 ,.0006 ,.0006 ,.0006 ,.0006 ,.0005 ,.0005 ,.0005},
    {.0005, .0005 ,.0005 ,.0004 ,.0004 ,.0004 ,.0004 ,.0004 ,.0004 ,.0003},
    {.0003, .0003 ,.0003, .0003 ,.0003 ,.0003 ,.0003 ,.0003 ,.0003 ,.0002}
};

void sortArray(double a[SIZE],int n)
{
   for(int i=0;i<n;i++)
   {
      for(int j=i+1;j<n;j++)
      {
         if(a[i]>a[jj])
         {
            float aux=a[i];
            a[i]=a[j];
            a[j]=aux;
         }
      }
   }
}

void printArray(double a[SIZE],int n)
{
   for(int i=0;i<n;i++) printf("%.4f\t",a[i]);
   printf("\n\n");
}

void typing(double a[SIZE],double b[SIZE],double avg,double deviation,int n)
{
   for(int i=0;i<n;i++)
```

```
    {
        a[i]=(b[i]-avg)/deviation;
    }
}


void makeEqual(double a[SIZE],int n)
{
    for(int i=0;i<n;i++)
    {
        a[i]=((float)i+1)/n;
    }
}



double avg(double a[SIZE],int n){
    double sum=0;
    for(int i=0;i<n;i++)
    {
        sum+=a[i];
    }
    return sum/n;
}

double deviation(double a[SIZE],double avg, int n)
{
    double sum=0;
    double dev;
    for(int i=0;i<n;i++)
    {
        sum+=pow(a[i]-avg,2);
    }
    sum/=(float)n;
    dev=sqrt(sum);
    return dev;
}

 double truncate(double n){
     long x = 100 * n;
     float y = (float)x / (float)100;
     return y;
}
void makePhi(double a[SIZE],double b[SIZE],int n)
{
    for(int i=0;i<n;i++)
    {
        double trun=truncate(b[i]);
        trun*=100;
        int index=fabs(trun);
        int x=index%10;
        int y=index/10;
        if(b[i]>=0)
```

```
        {
            a[i]=deviationTable[y][x];
        }
        else a[i]=deviationTableForNegatives[y][x];
    }
}

void makeDiffPhi(double a[SIZE],double b[SIZE],double c[SIZE],int n)
{
    for(int i=0;i<n;i++)
    {
        a[i]=b[i]-c[i];
    }
}

void makeLastDiff(double a[SIZE],double b[SIZE],double c[SIZE],int n)
{
    for(int i=0;i<n;i++)
    {
        if(i==0)
        {
            a[i]=fabs(b[i]);
        }
        else{
            a[i]=fabs(c[i-1]-b[i]);
        }
    }
}
double getMax(double a[SIZE],double b[SIZE],int n)
{
    double max=-1;
    for(int i=0;i<n;i++)
    {
        if(a[i]>max)max=a[i];
        if(b[i]>max)max=b[i];
    }
    return max;
}

void getKolmogorovEvaluation(double a[SIZE],int n)
{
    double valores[SIZE];
    for(int i=0;i<n;i++) valores[i]=a[i];
    sortArray(valores,n);
    makeEqual(proEqual,n);
    double average=avg(valores,n);
    printf("%f\n",average);
    double deviationStandard=deviation(valores,average,n);
    printf("%f\n",deviationStandard);
    typing(type,valores,average,deviationStandard,n);
    makePhi(phiZ,type,n);
    makeDiffPhi(diff,proEqual,phiZ,n);
    makeLastDiff(diff2,phiZ,proEqual,n);
```

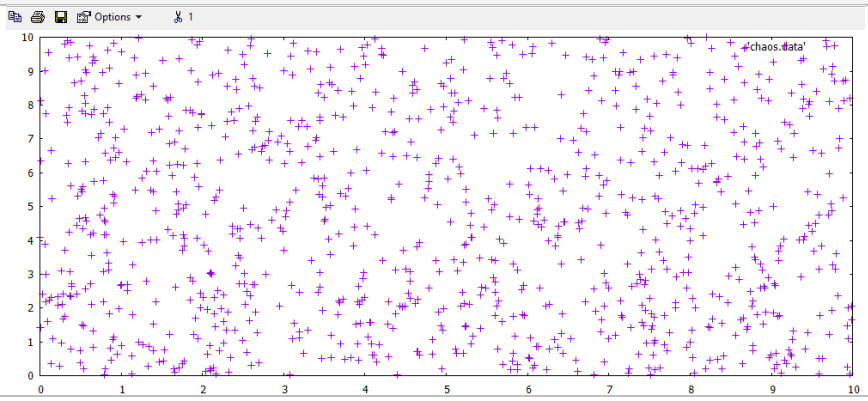
```
    printf("Values\n");
    printArray(valores,n);
    printf("ProEqual\n");
    printArray(proEqual,n);
    printf("Typing\n");
    printArray(type,n);
    printf("PhiZ\n");
    printArray(phiZ,n);
    printf("DiffPhi\n");
    printArray(diff,n);
    printf("Diff 2\n");
    printArray(diff2,n);
    printf("%f",getMax(diff,diff2,n));
}

/*Inicio del codigo propio del generador aleatorio y el ordenamiento y graficacion*/
double x[SIZE]={0};
double y[SIZE]={0};
double z[SIZE]={0};

void sortArrays(double x[SIZE],double y[SIZE],double z[SIZE],int n)
{
    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(z[j]<z[i])
            {
                double aux=z[j];
                z[j]=z[i];
                z[i]=aux;

                aux=x[j];
                x[j]=x[i];
                x[i]=aux;

                aux=y[j];
                y[j]=y[i];
                y[i]=aux;
            }
        }
    }
}

void writeData(double x[SIZE],double y[SIZE],double z[SIZE],int n)
{
    FILE *fp;
          fp = fopen ( "chaos.data", "w" );
          //fprintf(fp,"set isosamples 100,100\n");
    for(int i=0;i<n;i++)
    {
        fprintf(fp,"%f %f\n",x[i],y[i]);
    }
```

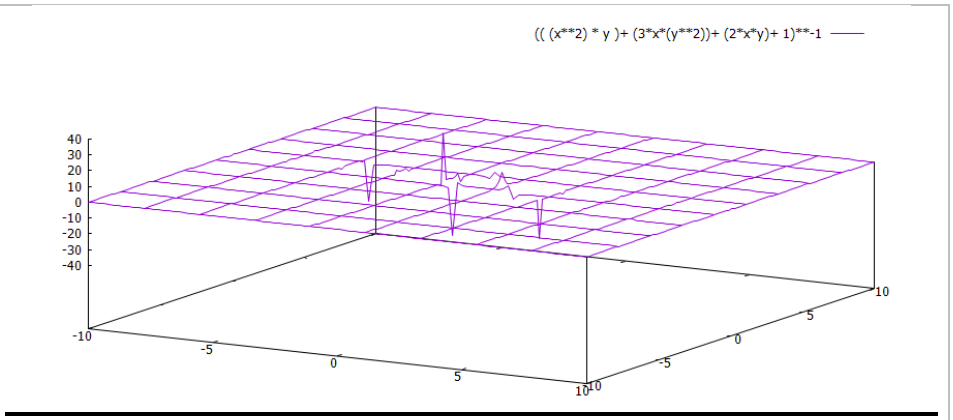
```
    fclose(fp);
}

struct range
{
    int li;
    int ls;
}r;

double filterChaos(double pv,double B)
{
    double a=(B*pv)-B*pow(pv,2);
    return a;
}

double myChaos(int li, int ls)
{
    double rand_range;
    rand_range=rand()%10000;
    rand_range/=10000;
    return (li+  (rand_range*(ls-li)));
}

void printArrays(double x[SIZE],double y[SIZE],double z[SIZE],int n)
{
    printf("x\t\t y\t\t f(x,y)\n\n");
    for(int i=0;i<n;i++)
    {
        printf("%f\t%f\t%f\n",x[i],y[i],z[i]);
    }
}
double f(double x,double y)
{
    return pow(  (x*x*y)+ (3*x*y*y)+ (2*x*y)+ 1 ,-1 );
}
int main()
{
    int n;
    srand(time(NULL));
    double pv;
    //pv=rand()%10000;
    //pv/=10000;

    scanf("%d %d %d",&r.li,&r.ls,&n);

    for(int i=0;i<n;i++)
    {
        //double pv=myChaos(r.li,r.ls);
        x[i]=myChaos(r.li,r.ls);
       // x[i]=filterChaos(pv,x[i])*10;
        //x[i]=filterChaos(pv,x[i]);
    }
    for(int i=0;i<n;i++)
```

```
    {
        y[i]=myChaos(r.li,r.ls);
        z[i]=f(x[i],y[i]);
    }
    sortArrays(x,y,z,n);

    //printArray(x,y,z,n);
    writeData(x,y,z,n);
    system("C:/gnuplot/bin/gnuplot.exe -p -e \"plot 'chaos.data' \"");
    system("C:/gnuplot/bin/gnuplot.exe -p -e \"splot  (( (x**2) * y )+ (3*x*(y**2))+ (2*x*y)+ 1)**-1  \"");
    getKolmogorovEvaluation(x,n);
}
```

Estructura range		
Tipo de dato	Nombre	Uso
Entero	Li	Límite inferior
Entero	Ls	Límite superior de generación

4 Resultados

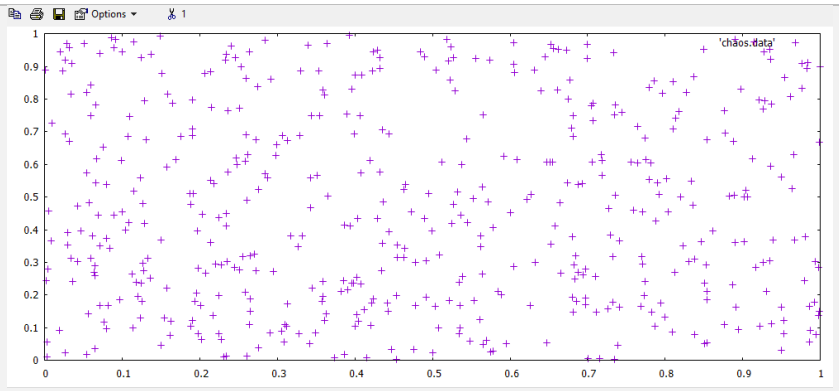
Resultados obtenidos	
RANGO-Muestras	Resultados
0 -10,1000	

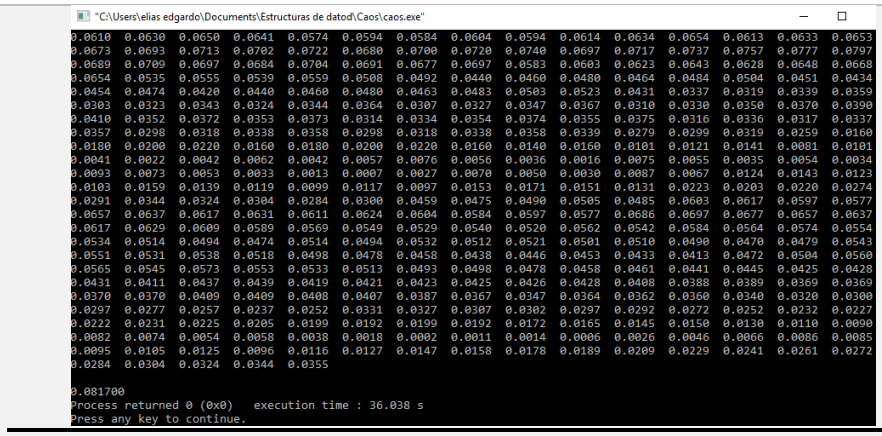



```
0.0364  0.0354  0.0380  0.0442  0.0432  0.0422  0.0412  0.0402  0.0392  0.0418  0.0479  0.0469  0.0495  0.0485  0.0475
0.0465  0.0455  0.0480  0.0470  0.0460  0.0450  0.0475  0.0465  0.0455  0.0445  0.0504  0.0494  0.0518  0.0543  0.0533
0.0557  0.0580  0.0570  0.0560  0.0617  0.0607  0.0631  0.0621  0.0611  0.0634  0.0624  0.0614  0.0604  0.0594  0.0584
0.0574  0.0597  0.0619  0.0674  0.0664  0.0654  0.0644  0.0666  0.0687  0.0677  0.0667  0.0657  0.0647  0.0637  0.0627
0.0649  0.0639  0.0629  0.0619  0.0609  0.0599  0.0620  0.0610  0.0600  0.0590  0.0642  0.0632  0.0714  0.0704  0.0694
0.0684  0.0674  0.0694  0.0684  0.0674  0.0664  0.0654  0.0644  0.0634  0.0654  0.0702  0.0721  0.0711  0.0701  0.0691
0.0710  0.0700  0.0690  0.0680  0.0670  0.0660  0.0679  0.0669  0.0687  0.0677  0.0723  0.0713  0.0703  0.0693  0.0711
0.0701  0.0691  0.0681  0.0671  0.0661  0.0651  0.0668  0.0658  0.0648  0.0638  0.0628  0.0618  0.0636  0.0626  0.0616
0.0606  0.0596  0.0613  0.0603  0.0593  0.0583  0.0573  0.0616  0.0606  0.0596  0.0586  0.0576  0.0566  0.0556  0.0572
0.0562  0.0578  0.0568  0.0558  0.0548  0.0538  0.0554  0.0544  0.0534  0.0524  0.0539  0.0529  0.0519  0.0509  0.0499
0.0540  0.0555  0.0569  0.0559  0.0549  0.0563  0.0553  0.0543  0.0533  0.0523  0.0513  0.0528  0.0518  0.0508  0.0498
0.0511  0.0501  0.0515  0.0505  0.0518  0.0508  0.0498  0.0488  0.0478  0.0491  0.0481  0.0471  0.0461  0.0497  0.0487
0.0477  0.0467  0.0457  0.0469  0.0459  0.0449  0.0439  0.0451  0.0463  0.0475  0.0465  0.0455  0.0466  0.0456  0.0446
0.0436  0.0448  0.0438  0.0428  0.0418  0.0408  0.0419  0.0409  0.0399  0.0389  0.0420  0.0410  0.0400  0.0390  0.0380
0.0390  0.0380  0.0410  0.0400  0.0390  0.0399  0.0389  0.0379  0.0408  0.0398  0.0388  0.0397  0.0405  0.0395  0.0404
0.0394  0.0384  0.0374  0.0364  0.0354  0.0344  0.0352  0.0360  0.0350  0.0340  0.0330  0.0320  0.0310  0.0300  0.0307
0.0297  0.0305  0.0295  0.0285  0.0292  0.0282  0.0306  0.0296  0.0286  0.0276  0.0266  0.0272  0.0262  0.0252  0.0242
0.0232  0.0222  0.0229  0.0219  0.0225  0.0215  0.0205  0.0195  0.0185  0.0175  0.0181  0.0171  0.0161  0.0151  0.0157
0.0147  0.0137  0.0127  0.0132  0.0122  0.0112  0.0117  0.0107  0.0112  0.0102  0.0092  0.0082  0.0072  0.0077  0.0067
0.0057  0.0062  0.0052  0.0042  0.0032  0.0022  0.0041  0.0031  0.0035  0.0025  0.0015  0.0005  0.0009  0.0012  0.0002
0.0006  0.0004  0.0014  0.0011  0.0021  0.0031  0.0041  0.0038  0.0048  0.0045  0.0055  0.0053  0.0063  0.0073  0.0083
0.0093  0.0103  0.0100  0.0098  0.0108  0.0118  0.0128  0.0138  0.0136  0.0122  0.0132  0.0142  0.0141  0.0139  0.0149
0.0159  0.0169  0.0168  0.0178  0.0188  0.0187  0.0197  0.0207  0.0217  0.0216  0.0226  0.0226  0.0236  0.0246  0.0245
0.0255  0.0265  0.0275  0.0285  0.0295  0.0295  0.0305  0.0315  0.0315  0.0315  0.0325  0.0335  0.0345  0.0326  0.0336
0.0346  0.0346  0.0356  0.0366  0.0367  0.0377  0.0387  0.0397  0.0398  0.0408

0.079300
Process returned 0 (0x0)   execution time : 91.209 s
```

Coeficiente de 0.0793

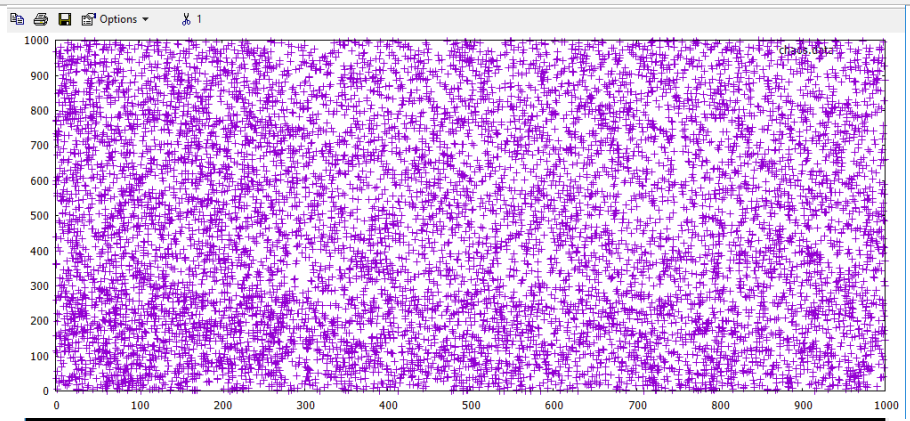
0-1,500

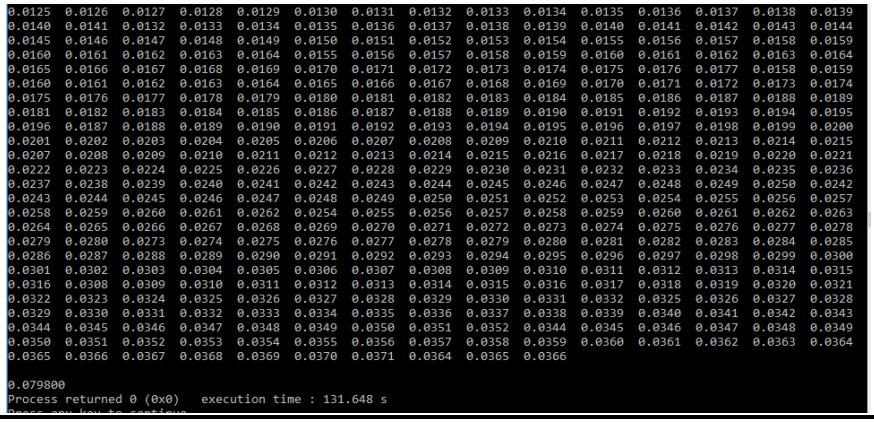




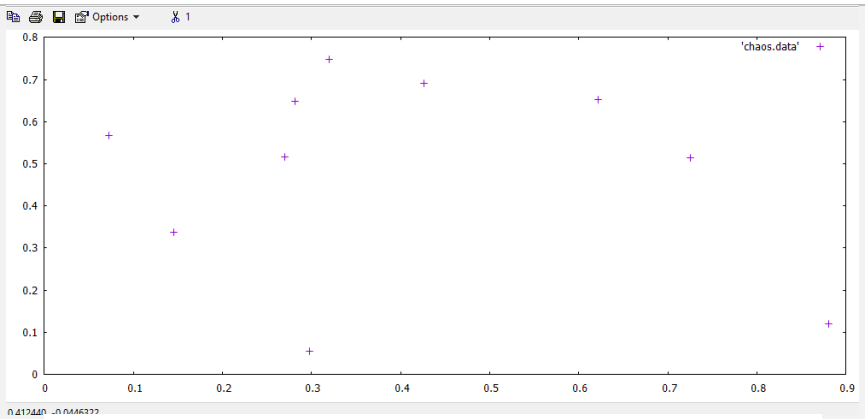
Coeficiente de 0.081700

0 1000, 10000





Coeficiente de 0.079

0 1 ,10	 0 1 10 0.404130 0.246252 Values 0.0730 0.1453 0.2700 0.2813 0.2978 0.3195 0.4265 0.6218 0.7254 0.8807 ProEqual 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1.0000 Typing -1.3447 -1.0511 -0.5447 -0.4988 -0.4318 -0.3437 0.0908 0.8839 1.3046 1.9353 PhiZ 0.0901 0.1492 0.2946 0.3121 0.3336 0.3669 0.5359 0.8078 0.9015 0.9726 DiffPhi 0.0099 0.0508 0.0054 0.0879 0.1664 0.2331 0.1641 -0.0078 -0.0015 0.0274 Diff 2 0.0901 0.0492 0.0946 0.0121 0.0664 0.1331 0.0641 0.1078 0.1015 0.0726 0.233100 Process returned 0 (0x0) execution time : 28.268 s Press any key to continue. Coeficiente de 0.233

5 Conclusiones

Dado que ninguno de los coeficientes infligió la regla del 0.05 se puede asumir que la evaluación del test del Kolmogorov fue satisfactoria, de igual manera los puntos aleatorios tienen una distribución aceptable de acuerdo con las gráficas.

Un punto que falto en esta práctica es la graficacion de los puntos x,y,z sobre la superficie mostrada en los resultados, esta superficie corresponde a la función a evaluar, si bien se graficaron los puntos x e y en un plano 2d lo ideal hubiera sido también situar los puntos de 3 coordenadas sobre su respectiva superficie, sin embargo, debido a las limitaciones de GNU Plot esto no fue posible.

En conclusión, el test de Kolmogorov fue afrontado con éxito para los valores que se probaron, sin embargo, existen variantes interesantes que se dejaron pendientes, por ejemplo, la inclusión de rangos negativos, se espera que esta práctica tenga una segunda parte para poder atacar estas areas de oportunidad.

6 Referencias

Joyanes Aguilar, L. (2005). *Programacion en C.* Madrid: Mc Graw Hill.

Langsam, Y., Augenstein, M., & Tenenbaum, A. (1997). *Estructuras de datos con C y C++.* Mexico: Prentice Hall.

Marquez Fausto, T. G., Osorio Angel, S., & Olvera Perez, E. N. (2011). *Introduccion a la programacion estructurada en C.* Mexico: Prentice Hall.

Tenebaum, A., Langsam, Y., & Augenstein, M. (1993). *Estructuras de datos en C.* Mexico: Prentice Hall.