Resuelto por: Javier Osca Cotarelo

2. Repetir el problema anterior con el método de muestreo uniforme. ¿Cuál es la relación entre las varianzas de uno y otro método? ¿Cuál es la relación de tiempo de ordenador de uno y otro método? ¿Qué método es más eficiente? ¿Cuántos segundos de CPU harían falta para calcular la integral dada con un error absoluto menor que 10^{-6}?

A continuación puede verse la relación entre las varianzas σ^2 para el método de Montecarlo H&M y el método de muestreo uniforme MU. La relación entre ambas (MU σ^2)/(HM σ^2) tiende a 0.3 lo que representa que es menor para el método de muestreo uniforme que para el método de Montecarlo pero que disminuyen con N de la misma manera porque esta relación se mantiene constante.

N	H&M σ^2	MU σ^2	(MU σ^2)/(HM σ^2)
10	1.60E-02	9.60E-03	0.6001
100	1.72E-03	5.98E-04	0.3484
1000	1.77E-04	5.37E-05	0.3041
10000	1.71E-05	5.04E-06	0.2936
100000	1.69E-06	4.94E-07	0.2923
1000000	1.68E-07	4.99E-08	0.2961
10000000	1.68E-08	4.98E-09	0.2955
100000000	1.69E-09	4.98E-10	0.2955

En la siguiente tabla pueden verse los tiempos de ejecución de uno y otro algoritmo t_{total}, el tiempo que tarda por bucle t_{bucle} y la relación entre uno y otro. El algoritmo de muestreo uniforme tarda la mitad de tiempo que el de Montecarlo.

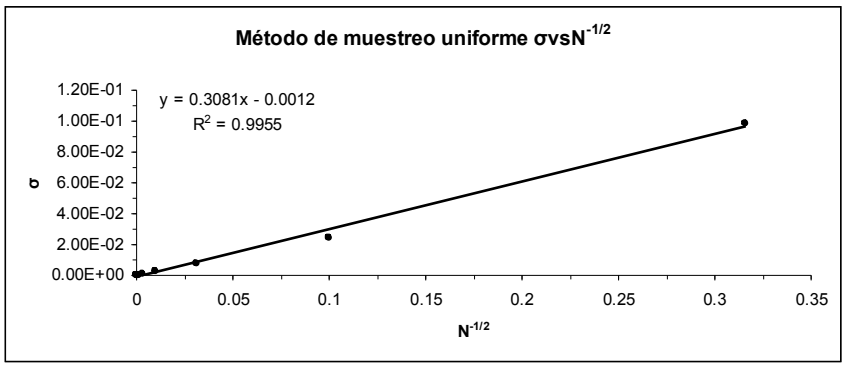
N	H&M (ttotal)	MU (ttotal)	H&M (tbucle)	MU (tbucle)	MU/HM(tbucle)
10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-
100	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-
1000	0.00E+00	0.00E+00	0.00E+00	0.00E+00	-
10000	1.50E-02	0.00E+00	1.50E-06	0.00E+00	0.00
100000	4.70E-02	3.10E-02	4.70E-07	3.10E-07	0.66
1000000	8.27E-01	4.21E-01	8.27E-07	4.21E-07	0.51
10000000	8.30E+00	4.29E+00	8.30E-07	4.29E-07	0.52
100000000	8.06E+01	4.21E+01	8.06E-07	4.21E-07	0.52

Tarda menos tiempo y con una dispersión menor, el método de muestreo uniforme es el más eficiente. Podemos estimar el tiempo total con este método a partir de las siguientes ecuaciones, dada una desviación típica esta es inversamente proporcional a la raíz de N. A su vez el tiempo total es proporcional a N por el tiempo que tarda en ejecutarse un bucle.

$$\sigma = \frac{\sigma_o}{\sqrt{N}} \Rightarrow N = \left(\frac{\sigma_o}{\sigma}\right)^2$$

$$t_{total} = N t_{bucle}$$

La constante σo la podemos calcular a partir de la regresión lineal de la desviación estándar frente a la inversa de la raíz cuadrada de N. Con este resultado y las anteriores ecuaciones se calcula el tiempo total.



σ	N	ttotal(s)	ttotal(h)
1.00E-06	9.49E+10	39953	11.1

Código del algoritmo de muestreo uniforme.

```
program ejer2
  !Librerias
  use util
  implicit none
  !Parametros
  integer,parameter :: nexperiment=8;
  real (kind=double),parameter :: a=0,b=1;
  !Funciones
  real (kind=double) :: f
  !indices
  integer :: i,j
  !variables
  integer :: na,nloops
  real :: total,etime,starttime(2),endtime(2)
  real (kind=double) :: u,v,p,Area,Error,muestra

  nloops=1;
  do i=1,nexperiment
    total=etime(starttime)
    Area=0.0_double;
    Error=0.0_double;
    nloops=nloops*10;
    do j=1,nloops
      call random_number(u)

      muestra=f(a+(b-a)*u)
      Area=Area+muestra
      Error=Error+muestra*muestra
    end do

    Area=Area/dble(nloops)
    Error=sqrt((Error/dble(nloops)-Area*Area)/dble(nloops))

    Area=(b-a)*Area
    Error=(b-a)*Error

    total=etime(endtime)
    write(*,*), nloops,Area,Error,endtime(1)-starttime(1)
  end do
end program ejer2
```

```
function f(x)
  !Librerias.
  use util
        implicit none
  !in/out
  real (kind=double) :: x,f

  f=sqrt(1-(x*x))
end function
```

Código del algoritmo de Montecarlo (mismo que el ejercicio nº 1)

```
program ejer1
  !Librerias
  use util
  implicit none
  !Parametros
  integer,parameter :: nexperiment=8;
  real (kind=double),parameter :: a=0,b=1,c=1;
  !Funciones
  real (kind=double) :: f
  !indices
  integer :: i,j
  !variables
  integer :: na,nloops
  real :: total,etime,starttime(2),endtime(2)
  real (kind=double) :: u,v,p,Area,Error

  nloops=1;
  do i=1,nexperiment
    total=etime(starttime)
    na=0.0_double;
    nloops=nloops*10;

    do j=1,nloops
      call random_number(u)
      call random_number(v)

      if ( f(a+(b-a)*u)> c*v ) na=na+1
    end do

    p=dble(na)/dble(nloops)
    Area=c*(b-a)*p
    Error=sqrt(p*(1-p)/dble(nloops))*c*(b-a)
    total=etime(endtime)
    write(*,*), nloops,Area,Error,endtime(1)-starttime(1)
  end do
end program ejer1

function f(x)
  !Librerias.
  use util
  implicit none
  !in/out
  real (kind=double) :: x,f

  f=sqrt(1-(x*x))
end function f
```