Informe de Laboratorio N°3

Alumno: Méndez Martín
Docentes: Dra. Marconi Verónica I.; Dr. Banchio Adolfo
Universidad Nacional de Córdoba (UNC)
Facultad de Matemática, Astronomía, Física y Computación (FaMAF)
Curso de Física Computacional: Problema N°2

I. Introducción

I-1. Problema 2 - Caminatas al azar

En una red cuadrada bidimensional implementar un algoritmo que realice una caminata al azar de n pasos. Iniciar la caminata en el sitio central de la red $R(0) = (0, 0)$.

- a) Hallar el desplazamiento cuadrático medio $\langle [R(n) - R(0)]^2 \rangle$ en función del número de pasos n, promediando $(\langle \dots \rangle)$ sobe $N = 10^6$ realizaciones de la caminata. ¿Se verifica la ley $\langle R^2 \rangle \sim n$?
- b) Subdividir la red en cuatro cuadrantes y contabilizar la cantidad de veces que el caminante termina en un dado cuadrante, comparar con el valor esperado $N/4$. Grafique estos resultados en función de n, comparando los resultados de distintos generadores (emplear como generadores de números al azar, el **ran2** del Numerical Recipes, el **mzran** y el **mt199337** Mersenne Twister).
- c) Ídem a) pero para una partícula real en 3 dimensiones (por ejemplo una molécula aromática que se difunde en una sala).

I-2. Generadores de números pseudo-aleatorios

El algoritmo **Random number generator Mersenne Twister (MT19937)** tiene un periodo primo tremendamente largo, $219937 - 1$. La implementación del algoritmo genera un número aleatorio de 32 bits de longitud (precisión simple). Las ventajas de este algoritmo es que ha pasado numerosas pruebas de aleatoriedad estadística y, al momento de realizar las optimizaciones del compilador, tiene mayor capacidad de mejorar su performance que la función **ran0** o **ran2** del Numerical Recipes. Su mayor desventaja, es que, al trabajar con arreglos, donde se almacenan los valores aleatorios generados, se requiere un mayor uso de memoria, por lo que al aumentar la cantidad de pasos del caminante se deberá requerir mayor tiempo de ejecución del programa. Por otro lado, **mzran** ofrece muy buenos resultados, aunque se tuvo la desventaja de que no fue posible modificar las semillas preestablecidas en la subrutina generadora, cosa que limita la repetibilidad de los resultados. Finalmente, los métodos congruenciales como **ran0** o **ran2** poseen la desventaja de que tienen correlaciones que pueden ser significativas, si uno observa en detalle la distribución de números es posible detectar hiper-planos paralelos donde los números se sitúan.

Cabe aclarar que existen procesos físicos capaces de generar variables estocásticas, como por ejemplo la radiación térmica, emisión de radioisótopos α o β, ruletas de casinos, dados, etc., sin embargo, estos procesos no cuentan con suficiente velocidad computacional que se requiere en los problemas de física. Por ello, fue necesario utilizar generadores de números pseudo-aleatorios (y quasi-aleatorios, que son menos aleatorios que los pseudo y están acotados en un dominio particular, y que son útiles en problemas específicos). La denominación de números "pseudo"se debe a que estos son representados por procesos no estocásticos, sino deterministas (como es el caso de las computadoras), entonces, es necesario utilizar generadores con un gran periodo para obtener ventajas computacionales. También, se buscan tres características fundamentales en los generadores, portabilidad (para no depender de las arquitecturas de las computadoras), velocidad computacional (para obtener mayor performance) y repetibilidad (para comparar resultados entre distintas simulaciones y testear los programas).

II. Resultados y Discusiones

II-A. Random Walk

Se recolectaron las posiciones del caminante para tres caminatas aleatorias con un mismo generador de números aleatorios (mzran), para una longitud de paso fija de 1, y una cantidad de pasos fija de 1 millón, los resultados se muestran en la figura (ver 1). Aquí se puede observar la forma típica de las

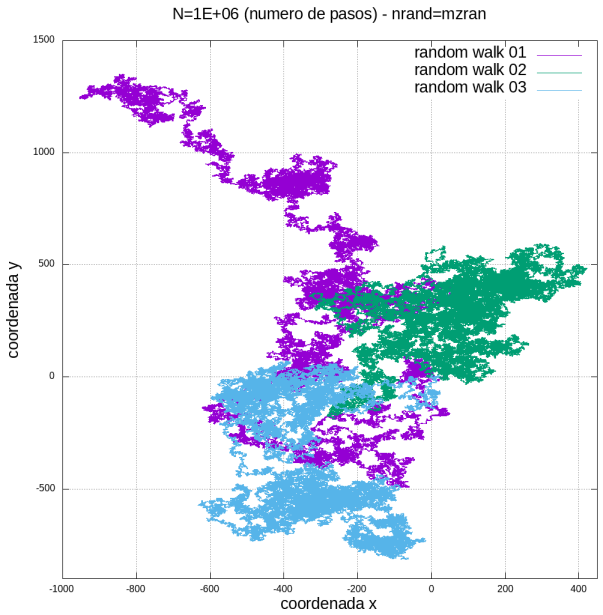

Fig. 1. caminatas aleatorias (muestreo del espacio de configuraciones)

caminatas aleatorias, y además nos permite corroborar que, a

pesar de haber simulado un millón de pasos por cada caminata el espacio de configuraciones no se abarca completamente, es decir, que existen correlaciones o tendencias del caminante a moverse en determinados lugares del espacio. Es por ello, que se plantea, resolver el problema con un cambio de enfoque, es decir, no realizar una única caminata muy larga, sino muchas caminatas cortas para barrer todo el espacio de fases y tener una mejor estadística. Estos dos enfoques son totalmente equivalentes por tratarse de sistemas ergódicos, es decir, que el valor medio temporal de cualquier variable dinámica del sistema es igual al promedio de esa variable sobre el ensamble.

II-B. Desplazamiento cuadrático medio (dcm)

Para computar el desplazamiento cuadrático medio se utilizó la siguiente expresión

$$dcm = \left\langle |\overrightarrow{x}(t) - \overrightarrow{x}(0)|^2 \right\rangle = \frac{1}{N}\sum_{i=1}^{N} |\overrightarrow{x}_i(t) - \overrightarrow{x}_i(0)| \quad (1)$$

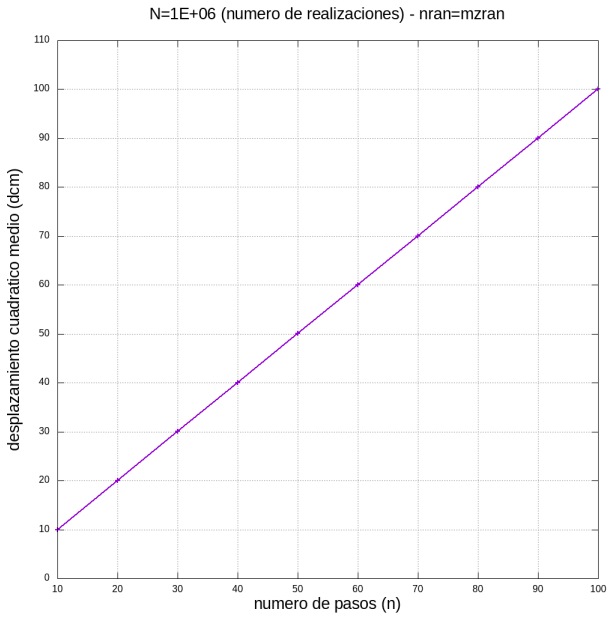

Fig. 2. desplazamiento cuadrático medio vs número de realizaciones

Notemos que se cumple la dependencia lineal (ver 2) entre dcm y el número de realizaciones N, específicamente, debería cumplirse que $\left\langle (\Delta\overrightarrow{x})^2 \right\rangle = N \left\langle (\Delta\overrightarrow{s})^2 \right\rangle$, donde $\left\langle (\Delta\overrightarrow{s})^2 \right\rangle$ sería la dispersión del desplazamiento por paso (en nuestro caso sería uno, pues cada longitud de paso es fija) y $\left\langle (\Delta\overrightarrow{x})^2 \right\rangle$ sería una medida del cuadrado del ancho de la distribución del desplazamiento neto, respecto de la media. Al ser lineal esta dependencia vemos que a medida que aumenta el número de realizaciones N, la distribución de probabilidades se va ensanchando, análogo a un problema de difusión de una partícula en un medio, a medida que transcurre el tiempo.

II-C. Contabilidad de cuadrantes

Para determinar en qué cuadrante el "borracho"termina luego de una caminata aleatoria, con condicionales se evaluó la posición en el plano y en función de está se pesó el contador de cada cuadrante específico es decir, si el caminante estaba un cuadrante sin ambiguedad el contador de ese cuadrante se incrementaba en 1, sin embargo, en los casos en que el caminante estaba en un semieje los contadores de los dos cuadrantes adyacentes se incrementaban en $1/2$ y si el caminante estaba en el origen los contadores de los cuatros cuadrante se incrementaban en $1/4$, el bloque de código sería de la forma,

```
! Determinamos el cuadrante de la part cula al
    final de la caminata
cond2: if (x>0._dp.and.y>0._dp) then
            count_cuad_01=count_cuad_01+1._dp
            exit cond2
    else if (x<0._dp.and.y>0._dp) then
        count_cuad_02=count_cuad_02+1._dp
        exit cond2
    else if (x<0._dp.and.y<0._dp) then
        count_cuad_03=count_cuad_03+1._dp
        exit cond2
    else if (x>0._dp.and.y<0._dp) then
        count_cuad_04=count_cuad_04+1._dp
        exit cond2
    else if (x==0._dp.and.y==0._dp) then !
        origen de coordenadas
        count_cuad_01=count_cuad_01+0.25_dp
        count_cuad_02=count_cuad_02+0.25_dp
        count_cuad_03=count_cuad_03+0.25_dp
        count_cuad_04=count_cuad_04+0.25_dp
        exit cond2
    else if (x>0._dp.and.y==0._dp) then ! semi-
        eje x positivo
        count_cuad_01=count_cuad_01+0.5_dp
        count_cuad_04=count_cuad_04+0.5_dp
        exit cond2
    else if (x<0._dp.and.y==0._dp) then ! semi-
        eje x negativo
        count_cuad_02=count_cuad_02+0.5_dp
        count_cuad_03=count_cuad_03+0.5_dp
        exit cond2
    else if (x==0._dp.and.y>0._dp) then ! semi-
        eje y positivo
        count_cuad_01=count_cuad_01+0.5_dp
        count_cuad_02=count_cuad_02+0.5_dp
        exit cond2
    else if (x==0._dp.and.y<0._dp) then ! semi-
        eje y negativo
        count_cuad_03=count_cuad_03+0.5_dp
        count_cuad_04=count_cuad_04+0.5_dp
        exit cond2
end if cond2
```

Los resultados (ver 3,4,5,6) muestran que la cantidad de veces que el caminante termina en un determinado cuadrante oscila alrededor de la media correcta $N/4$, los cual nos dice que los resultados se podrían considerar çorrectos", sin embargo, se obtienen variaciones significativas con todos los generadores, lo cual podría deberse a la correlación que existen entre los números generados, o poca cantidad de caminatas, es decir, a se esperaría que, a media que aumenta el número de realizaciones se mejora la estadística y se obtiene una oscilación entorno a la media menos significativa.

III. Códigos

Repositorio de GitHub

- https://github.com/mendzmartin/fiscomp2022.git

Repositorio GitHub del problema

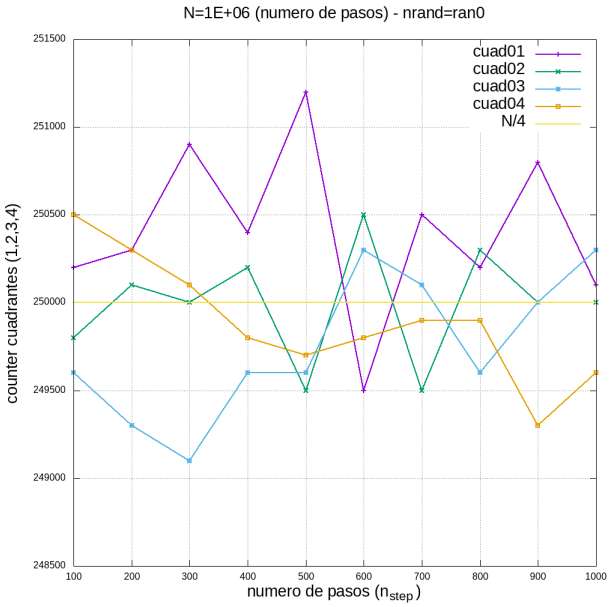

Fig. 3. contador cuadrante vs número de pasos

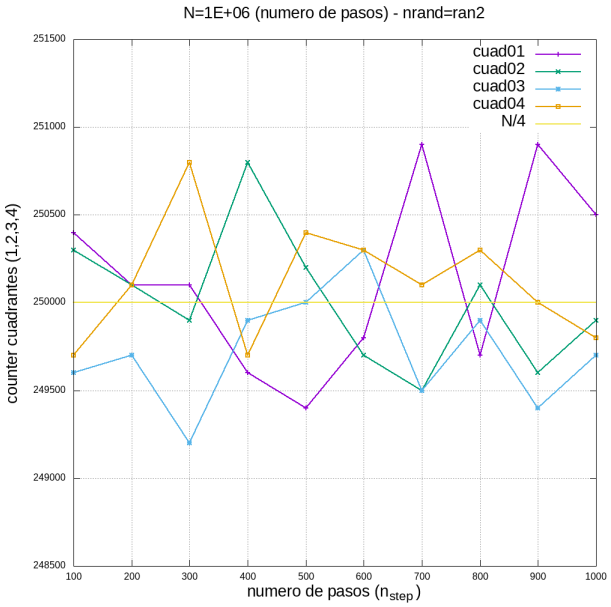

Fig. 4. contador cuadrante vs número de pasos

- https://github.com/mendzmartin/fiscomp2022/tree/main/lab03/prob02

Códigos principales y Makefile

- https://github.com/mendzmartin/fiscomp2022/blob/main/lab03/prob02/code/random_walk_2d.f90
- https://github.com/mendzmartin/fiscomp2022/blob/main/lab03/prob02/code/Makefile

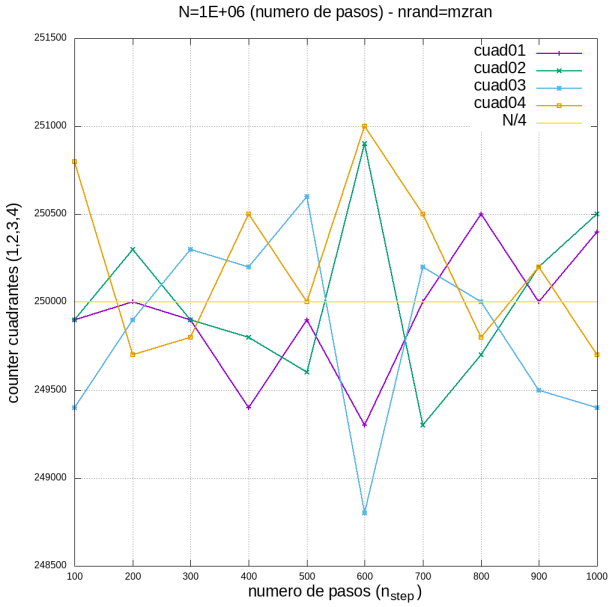

Fig. 5. contador cuadrante vs número de pasos

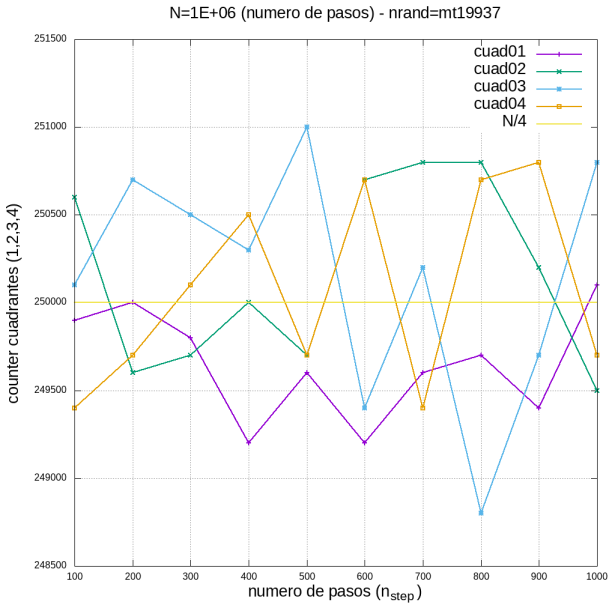

Fig. 6. contador cuadrante vs número de pasos

Informe de Laboratorio N°3

Alumno: Méndez Martín
Docentes: Dra. Marconi Verónica I.; Dr. Banchio Adolfo
Universidad Nacional de Córdoba (UNC)
Facultad de Matemática, Astronomía, Física y Computación (FaMAF)
Curso de Física Computacional: Problema N°3

I. Introducción

I-1. *Problema 3 - Integración de Monte Carlo*

$f(x) = x^n$

- a) Escriba un programa que estime la integral de la función $f(x) = x^n$ en el intervalo $[0, 1]$ usando el método de Monte Carlo. Para el caso $n = 3$, estime la integral usando $N = 10 \cdot i; i \in N$ evaluaciones de la función. Calcule la diferencia entre el valor exacto de la integral, $I = 1/(n + 1)$, y el obtenido usando el método de Monte Carlo, y grafique el valor absoluto de ésta función de N (use escala logarítmica). El error debe .escalearçomo $N^{-1/2}$.
- b) Modifique el programa del inciso anterior para calcular la misma integral, pero ahora utilizando ïmportance sampling", con la distribución de probabilidad $p(x) = (k + 1) \cdot x^k$, con $k < n$. Recuerde que puede obtener números aleatorios distribuidos de acuerdo a una ley de potencia, transformando números aleatorios, x, distribuciones uniformemente en el intervalo $[0, 1]$, de la siguiente manera:

$$y = x^{1/(k+1)} \tag{1}$$

Calcule la diferencia entre el valor exacto de integral, $I = 1/(n + 1)$, y el obtenido el método de Monte Carlo con ïmportance sampling"para el caso $k = 2, 3$ y grafique el absoluto de ésta (i.e el error) en función de N. Compare con los resultados del inciso anterior (en la misma gráfica).

II. Resultados y Discusiones

II-A. *Método de Monte Carlo*

Tanto para la integración estándar del método de Monte Carlo, como para la integración incorporando ïmportance sampling"se calculo el error relativo definido como:

$$\epsilon_{rel} = \frac{exact - aprox}{exact} \tag{2}$$

Los resultados obtenidos fueron, para $f(x) = x^2$ (ver figura 1) y para $f(x) = x^3$ (ver figura 2).

Notemos que para la integración estándar en las dos integraciones se cumple la ley de potencias para la variación del error relativo $1/\sqrt{N}$, sin embargo, para la integración con ïmportance sampling.el error relativo escala como debería sólo para $f(x) = x^2$, esto se debe a que, al aplicar el método de función inversa para "mapear"la distribución uniforme en una distribución como ley de potencias, el valor de la potencia k debe cumplir que $k < 3$, sin embargo, no existe una restricción explícita en las cuentas donde se pueda deducir este resultado, la única restricción explícita que existe sería $k > -2$ (suponiendo que $k \in Z$ para que la distribución de probabilidad $p(x)$ sea definida positiva.

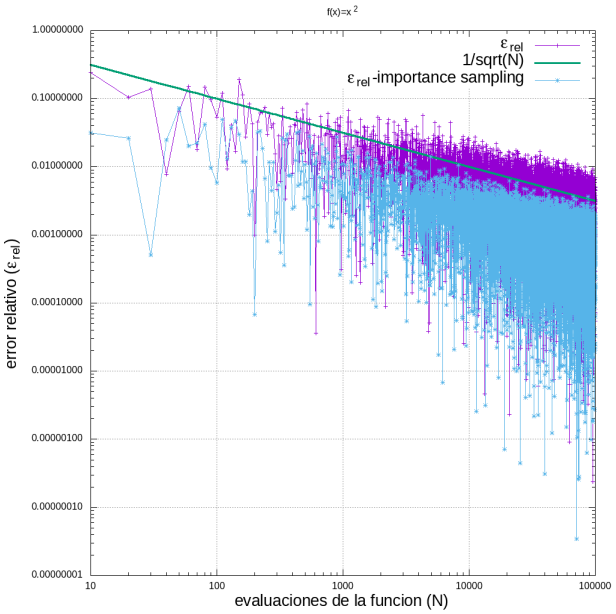

Fig. 1. error relativo $f(x) = x^2$ vs evaluaciones de la función

Por otro lado, notemos que, al utilizar ïmportance sampling.el error se reduce en un orden de magnitud respecto al método estándar usando distribución de probabilidad uniforme.

II-B. *Test generadores de números aleatorios*

Para utilizar el método de ïmportance sampling.en este problema partícular (ley de potencia) como en los problemas subsiguientes se testearon los generadores *ran0*, *ran2*, *mzran* de Marsaglia y Zaman y *mt19937* Mersenne Twister.

Se mapearon las distribuciones uniformes de cada uno de los generados en las distribuciones **ley de potencias**, **gaussiana** y **exponencial**. Los histogramas obtenidos se muestran en las figuras 3, 4, 5.

III. Códigos

Repositorio de GitHub

- https://github.com/mendzmartin/fiscomp2022.git

Repositorio GitHub del problema

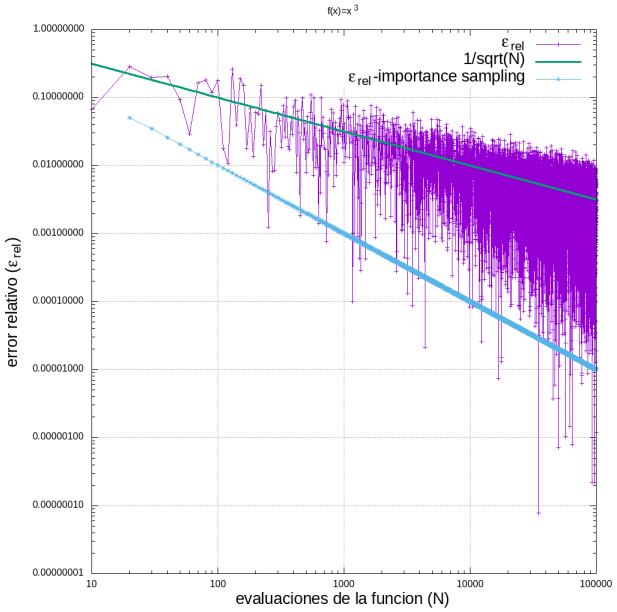

Fig. 2. error relativo $f(x) = x^3$ vs evaluaciones de la función

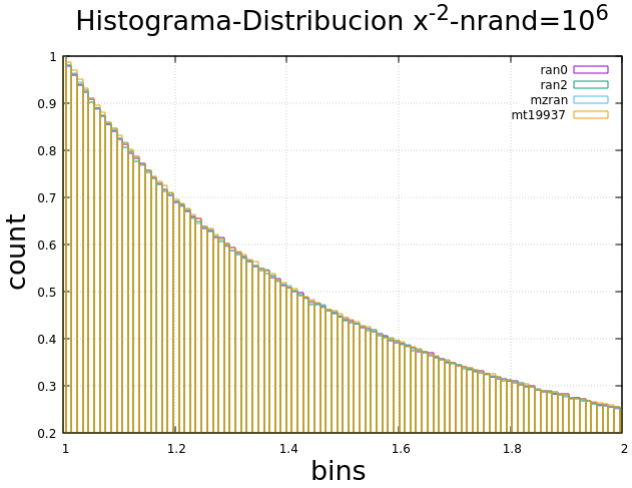

Fig. 3. ley de potencia $f(x) = x^{-2}, x \in [1, \infty]$

- https://github.com/mendzmartin/fiscomp2022/tree/main/lab03/prob03

Códigos principales y Makefile

- https://github.com/mendzmartin/fiscomp2022/blob/main/lab03/prob03/code/mc_integration.f90
- https://github.com/mendzmartin/fiscomp2022/blob/main/lab03/prob03/code/mc_integration_imp_sampling.f90
- https://github.com/mendzmartin/fiscomp2022/blob/main/lab03/prob03/code/Makefile

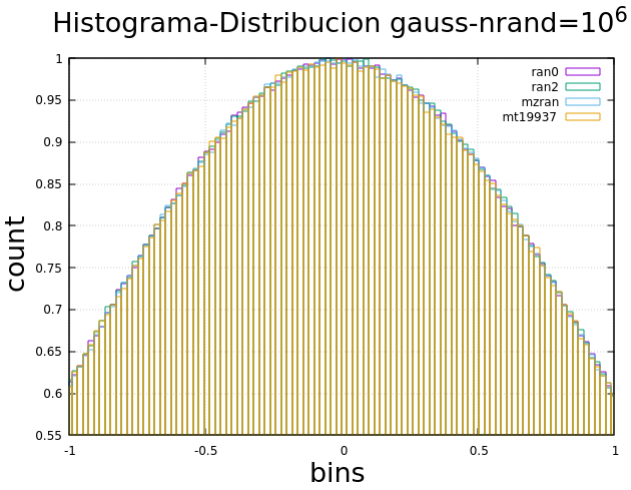

Fig. 4. gaussiana $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], x \in [-\infty, \infty]$

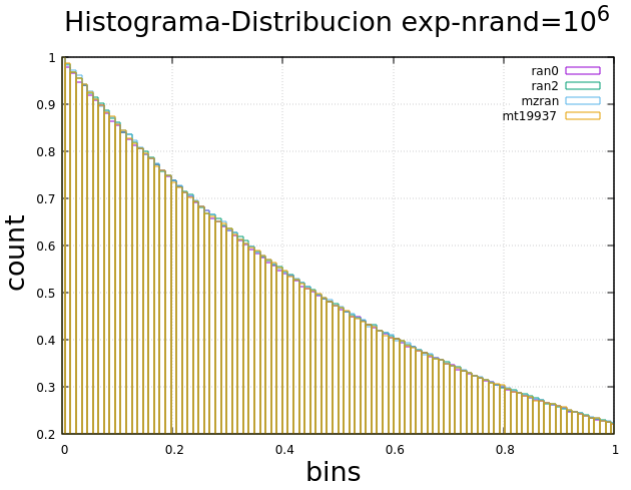

Fig. 5. exponencial $f(x) = \frac{1}{\lambda} \exp(-x/\lambda), x \in [0, \infty]$

Informe de Laboratorio N°3

Alumno: Méndez Martín
Docentes: Dra. Marconi Verónica I.; Dr. Banchio Adolfo
Universidad Nacional de Córdoba (UNC)
Facultad de Matemática, Astronomía, Física y Computación (FaMAF)
Curso de Física Computacional: Problema N°4

I. Introducción

I-1. Problema 4 - La Hiperesféra

Calcular numéricamente el volumen $V(n)$ de una esfera n-dimensional de radio 1, integrando la función $f(x_1,\ldots,x_{(n-1)}) = 2\cdot\sqrt{1-\sum_{i=1}^{(n-1)}(x_i)^2}$, sobre la esfera $(n-1)$-dimensional de los siguientes modos:

- a) Utilizando, para $n = 2, 3, 4$, el método del trapecio, generalizado a n dimensiones. Elija una cantidad de puntos, $N_p = 2^{24}$, y manteniéndolo fijo, grafique el error relativo de la integral versus n. Encuentre la dependencia esperada.
- b) Utilizando el método de Monte Carlo. Éste debe funcionar hasta al menos $n = 100$ (dando el valor correcto al 1 por mil de error en unos pocos minutos de CPU). Observe que el volumen de una hiperesféra rápidamente se hace mucho menor que el del hipercubo (una distribución uniforme muestrearía el hipercubo, por lo tanto, arrojaría casi todos los puntos fuera del dominio de integración). Además tenga en cuenta que para la distribución Gaussiana unidimensional $\langle x^2\rangle = \sigma^2$.

Nota: el resultado analítico para n par es $V(n) = \frac{\pi^{n/2}}{(n/2)!}$ y para n impar es $V(n) = \frac{\pi^{(n-1)/2}\cdot 2^n\cdot[(n-1)/2]!}{n!}$. Para $n = 100$ da $V(100) = 2{,}3682\cdot\cdots\cdot 10^{-40}$

I-2. Método del trapecio

Para computar el cálculo del volumen de la hiper-esféra con el método del trapecio se utilizó la fórmula recursiva que nos permite relacionar el volumen de una esféra n-dimensional con una esféra $(n-1)$-dimensional, a través de una integración de la forma

$$V_n(R) = V_{n-1}(R)\int_{-R}^{R}\left[1-\left(\frac{x}{R}\right)^2\right]^{(n-1)/2} dx \qquad (1)$$

En nuestro caso particular, el radio $R = 1$ y cómo se trata de una geometría completamente simétrica sólo integramos desde 0 hasta 1, con la salvedad de multiplicar el volumen n-dimensional obtenido por un factor de 2^n.

II. Resultados y Discusiones

II-A. Método del trapecio

Teniendo en cuenta que se debe mantener fijo el número de puntos ($N_p^{nD} = 2^{24}$) utilizados para la integración de un volumen n-dimensional. A medida que se aumenta la dimensión cada integral unidimensional debe utilizar ($N_p^{1D} = (N_p^{nD})^{1/n}$) de esta forma logramos recuperar la dependencia correcta del error relativo $\epsilon_{rel} = \frac{\epsilon_{exact}-\epsilon_{aprox}}{\epsilon_{exact}}$ con el número de dimensiones. Las simulaciones con este método calcularon para $n = 1, 2, 3, 4$ y los resultados obtenidos fueron,

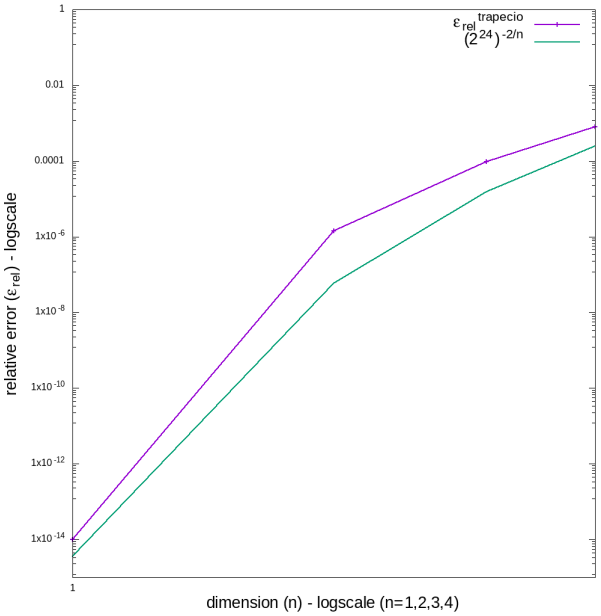

Fig. 1. error relativo (regla trapecio) vs dimensiones

Notemos que el *fitteo* no es exacto, lo cual podría deberse a algún error en el algoritmo. Sin embargo, las pendientes tienen bastante coincidencia. Este resultado, nos permite concluir que, como el método de Monte Carlo es independiente del número de dimensiones para dimensiones mayores a 4 el método de Monte Carlo saca ventaja sustancial con respecto al método del trapecio y para dimensiones mayores a 6 respecto al método de Simpson.

II-A1. Determinación de la desviación estandar σ

Teniendo en cuenta que la razón entre los volumenes n-dimensionales de la hiper-esféra y el del hiper-cubo disminuye a medida que el número de dimensiones n aumenta, y recordando que una distribución uniforme muestrearía completamente el hiper-cubo, se utilizó el método de *importance sampling*, con distribución gaussiana, para ser más eficiente con el método de MC, es decir, lograr que la mayor cantidad de puntos aleatorios caigan dentro del volumen a calcular (volumen de la hiper-esféra), sin embargo, a

medida que aumentamos las dimensiones de la hiper-esfera se debe modificar la desviación estándar de la distribución para lograr una precisión fija admisible. Lo primero que se hizo fue correr el programa para una cantidad de números aleatorios fija $n_{random} = 10^6$, definir el rango de variación de la desviación estándar ($\sigma_{máx} = 1; \sigma_{mín} = 0{,}01$) y el numero de valores para σ distintos que quiero que se calcule $n_\sigma = 1000$. Con estos valores se define un paso $\Delta\sigma$ y se comienza la simulación partiendo con un $\sigma = \sigma_{mín}$, al obtener el resultado del volumen se computa el error relativo $\epsilon_{rel} = \frac{\epsilon_{exact} - \epsilon_{aprox}}{\epsilon_{exact}}$ y nos preguntamos si este error es menor a 0,001 (correcto en uno por mil) en caso de ser afirmativo se termina la simulación y en caso de ser negativo se aumenta en uno el paso ($\sigma = \sigma + \Delta\sigma$) y se vuelve a computar el volumen. Estas corridas se hacen para un numero de dimensiones reducido, es decir, $1 \leq n \leq 7$. Los resultados obtenidos se resumen en la siguiente tabla I.

CUADRO I
PRIMER TESTEO DE σ

Dimensión	Elapsed time [s]	ϵ_{rel}	σ
1	0.2470E+02	0.3298E-03	0.2250
2	0.4426E+02	0.4368E-03	0.2141
3	0.5253E+02	0.8824E-03	0.1686
4	0.5659E+02	0.1947E-03	0.1408
5	0.1130E+03	0.8072E-03	0.2181
6	0.1105E+03	0.4670E-04	0.1765
7	0.1412E+03	0.4518E-03	0.1953

Notar que también se computó el tiempo de CPU transcurrido, es decir, el tiempo en que tarda el algoritmo en encontrar un $\sigma_{óptimo}$ tal que verifique el error propuesto, estos tiempos van incrementando con el número de dimensiones. Además, notemos que el $\sigma_{óptimo}$ para dimensiones mayores a 1 siempre es menor al $\sigma_{óptimo}$ de una dimensión, pero de forma general $\sigma_{óptimo}$ no disminuye con el numero de dimensiones (este tema se trata más adelante). Estos dos resultados, nos permiten asegurarnos que el mayor valor de σ se obtiene para $n = 1$, lo cual nos permitiría reducir significativamente los tiempos de CPU acotando el rango de variación de la desviación estándar, es decir, para más dimensiones consideraremos ($\sigma_{máx} = 1; \sigma_{mín} = 0{,}1$).

Por otro lado, como nosotros sabemos que a medida que aumenta el número de dimensiones la razón entre el volumen del la hiper-esféra y el del hiper-cubo se reduce, la distribución gaussiana debe disminuir su desviación estándar para mejorar la precisión del método y lograr que la mayor cantidad de números computados caigan dentro del volumen a calcular, el valor $\sigma = 0$ será una especie de *atractor* y a medida que aumenta el número de dimensiones el valor de $\sigma_{óptimo}$ tratará de desviarse lo menos posible del *atractor* tendiendo, en el límite de $n \to \infty$, a cero y la distribución gaussiana tenderá a una delta de Dirac. Es por ello que, para reducir significativamente los tiempos de CPU se propuso una "ley"(ad-hoc) de variación de $\Delta\sigma$ con el número n de dimensiones de la siguiente manera

$$n \in N \Rightarrow n_\sigma = 100 \cdot n \Rightarrow \Delta\sigma = \frac{\sigma_{máx} - \sigma_{mín}}{(n_\sigma - 1)}$$

$$\Rightarrow \Delta\sigma = \frac{\sigma_{máx} - \sigma_{mín}}{(100 \cdot n - 1)} \quad (2)$$

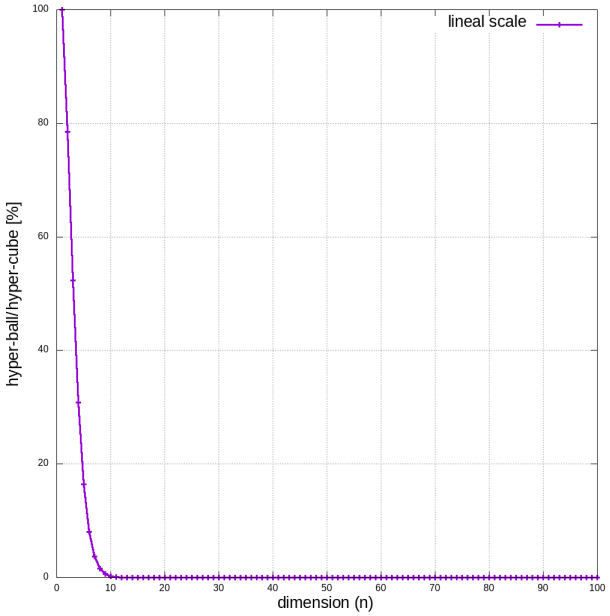

Fig. 2. razon entre volumenes (ball/cube) vs dimensiones

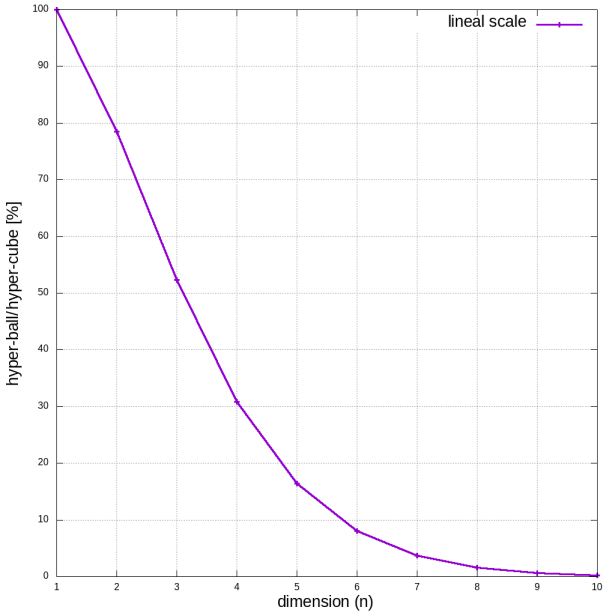

Fig. 3. razon entre volumenes (ball/cube) vs dimensiones

De esta forma, a medida que aumenta el número de dimensiones se computa muy cerca del valor mínimo propuesto para la desviación estándar y se itera hasta conseguir el error admisible. Con esta propuesta se calcularon los volúmenes de la hiper-esféra hasta 10 dimensiones obteniendo los siguientes resultados

Notemos que el tiempo de CPU que se tarda en encontrar un $\sigma_{óptimo}$ que cumpla con el error admisible no aumenta siempre, aunque, se muestra cierta tendencia a aumentar con el número de dimensiones. Esto se puede deber a

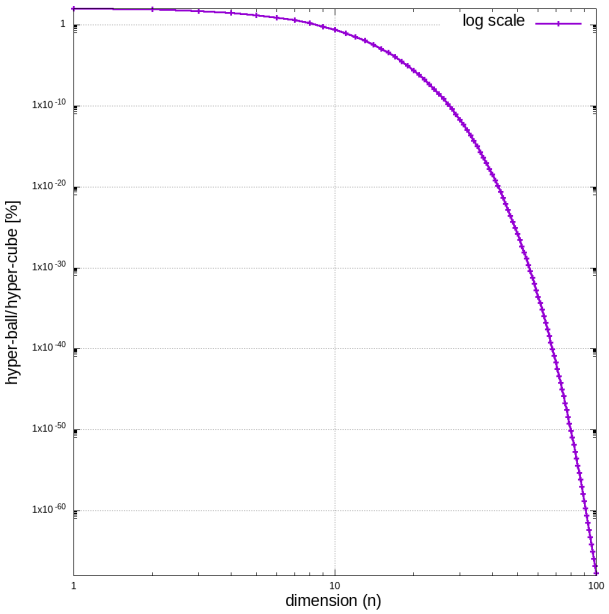

Fig. 4. razon entre volumenes (ball/cube) vs dimensiones

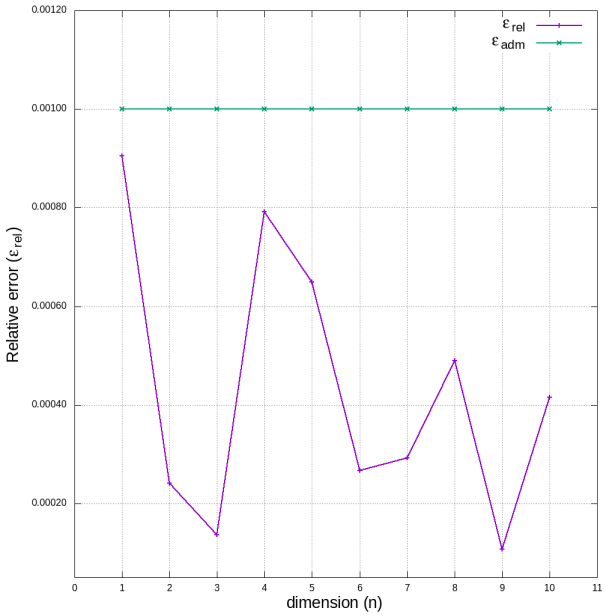

Fig. 6. error relativo vs dimensiones

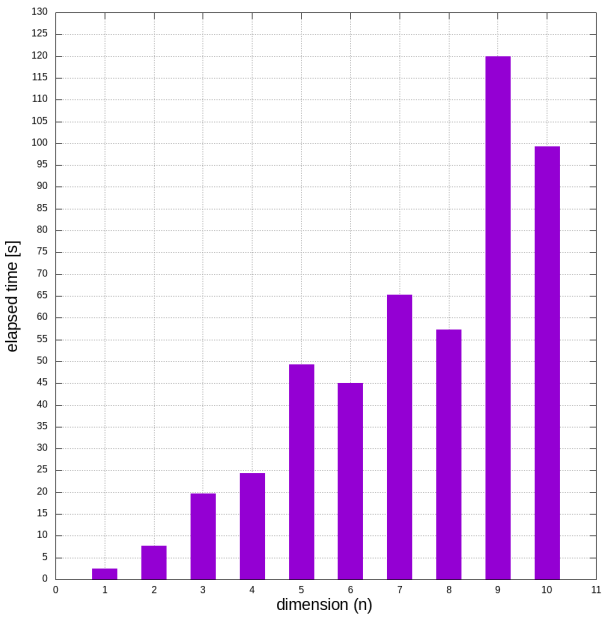

Fig. 5. tiempo de CPU vs dimensiones

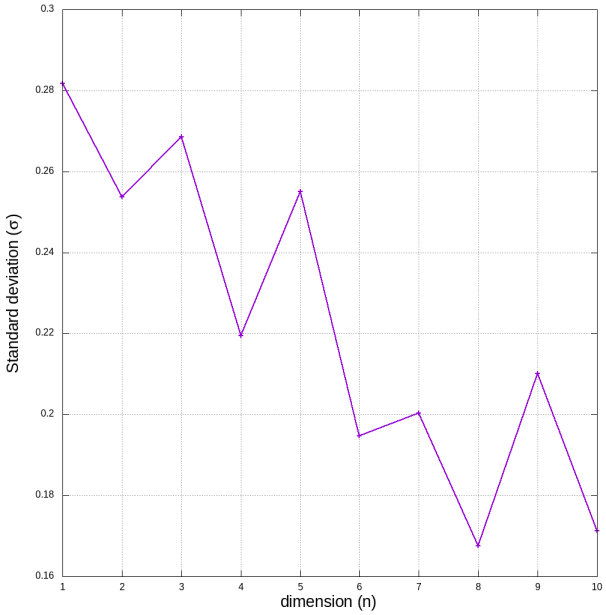

Fig. 7. desviación estandar vs dimensiones

muchos factor, por un lado, notemos que el error relativo es muy variable, si bien, siempre estamos por debajo del error admisible, el $\sigma_{óptimo}$ encontrado no es el mínimo que realmente verificaría, esto es debido a la *ley* de variación del $\Delta\sigma$ con el número de dimensiones que no es exacta. También observando como varía el $\sigma_{óptimo}$ encontrado en función de las dimensiones no vemos una disminución continua como esperaríamos, sino que hay picos y valles, lo cual nos muestra que debemos encontrar una *ley* de variación más precisa para disminuir continuamente el tiempo de CPU, obtener un error relativo más estable y disminuir continuamente el $\sigma_{óptimo}$ al aumentar el número de dimensiones. Cabe aclara que los tiempos de CPU encontrados con la *ley* ad-hoc propuesta se vieron disminuidos notablemente lo que nos pone en evidencia que vamos por buen camino.

Además, para mayor comprensión de lo rápido que disminuye la magnitud del volumen de la hiper-esféra a medida que aumentamos el tamaño de dimensiones se graficó lo siguiente,

donde podemos notar que la función volumen cuenta con un maximal en $n = 5$ y luego disminuye continuamente.

III. Códigos

Repositorio de GitHub

- https://github.com/mendzmartin/fiscomp2022.git

Repositorio GitHub del problema

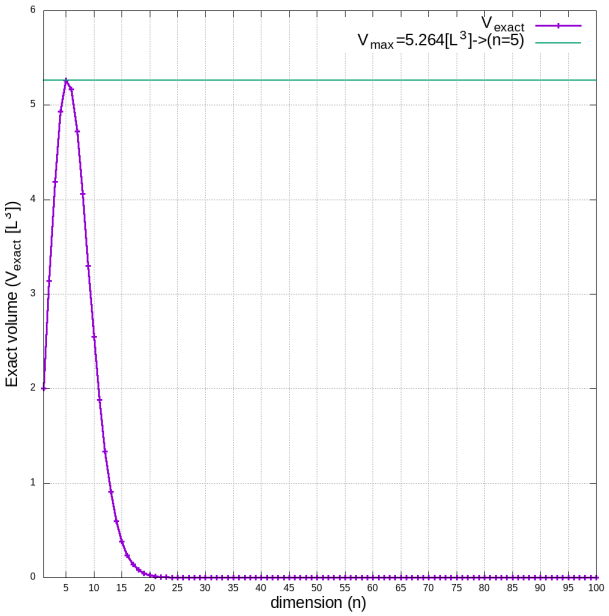

Fig. 8. Volumen exacto vs dimensiones

- https://github.com/mendzmartin/fiscomp2022/tree/main/lab03/prob04

Códigos principales y Makefile

- https://github.com/mendzmartin/fiscomp2022/blob/main/lab03/prob04/code/hyper_sphere.f90
- https://github.com/mendzmartin/fiscomp2022/blob/main/lab03/prob04/code/Makefile

Problema 02 - Códigos

random_walk_2d.f90

```
! Problema 02
program random_walk_2d
    use module_precision

    implicit none
    integer(sp)            :: seed,seed_val(8)                          ! semilla
    integer(sp), parameter :: n_step_total=1000000_sp                   ! numero total de random walks
    integer(sp), parameter :: switch=3_sp                               ! cambiar si se quieren escribir los datos
    integer(sp), parameter :: n_step=1000_sp                            ! numero de pasos de c/ random walk
    real(dp)               :: x,y,x_old,y_old,dcm,dcm_tot               ! pasos y desplazamiento cuadrático medio
    real(dp)               :: cuad01,cuad02,cuad03,cuad04               ! contadores en c/ cuadrante
    real(dp)               :: cuad01_tot,cuad02_tot,cuad03_tot,cuad04_tot ! contadores en c/ cuadrante
    integer(sp)            :: i,j,istat
    integer(sp)            :: rnd_type                                  ! tipo de random generator
    real(dp)               :: suma                                      ! variable de control

    open(10,file='../results/result.dat',status='replace',action='write',iostat=istat)
    select case(switch)
        case(1) ! mapa random walk
            open(10,file='../results/result_01.dat',status='replace',action='write',iostat=istat)
            21 format(A12,x,A12); write(10,21) 'x-coord','y-coord'
        case(2) ! inciso a
            open(10,file='../results/result_02.dat',status='replace',action='write',iostat=istat)
            22 format(A12,x,A12); write(10,22) 'n_step','dcm'
            23 format(I12,x,E12.4)
        case(3) ! inciso b
            open(10,file='../results/result_03.dat',status='replace',action='write',iostat=istat)
            24 format(5(A12,x),A12); write(10,24) 'j','N/4','cuad01','cuad02','cuad03','cuad04'
            25 format(I12,x,5(E12.4,x),E12.4)
    end select
    if (istat /= 0_sp) write(*,*) 'istat_error=', istat

    rnd_type=4_sp ! elegir random generator
    do j=100_sp,n_step,100_sp
        write(*,*) j
        ! generamos la semilla
        call date_and_time(values=seed_val)
        seed=seed_val(8)*seed_val(7)*seed_val(6)+seed_val(5)
        x_old=0._dp;y_old=0._dp;dcm_tot=0._dp
        cuad01_tot=0._dp;cuad02_tot=0._dp;cuad03_tot=0._dp;cuad04_tot=0._dp
        do i=1,n_step_total
            call walk_2d(switch,10,rnd_type,seed,j,x_old,y_old,x,y,dcm,cuad01,cuad02,cuad03,cuad04)
            cuad01_tot=cuad01_tot+cuad01;cuad02_tot=cuad02_tot+cuad02
            cuad03_tot=cuad03_tot+cuad03;cuad04_tot=cuad04_tot+cuad04
            dcm_tot=dcm_tot+dcm
            if (switch==1_sp) then;x_old=x;y_old=y; end if
        end do

        if (switch==2_sp) write(10,23) j,dcm_tot*(1._dp/n_step_total)

        cuad01=cuad01_tot;cuad02=cuad02_tot;cuad03=cuad03_tot;cuad04=cuad04_tot
        suma=(cuad01+cuad02+cuad03+cuad04)*0.25_dp

        if (switch==3_sp) write(10,25) j,real(n_step_total)*0.25_dp,cuad01,cuad02,cuad03,cuad04,suma
    end do
    close(10)
end program random_walk_2d

! subrutina para realizar caminata aleatoria de n_step pasos específicos
! partiendo de cierto origen especificoe, elegir tipo pseudo generador random,
! devolver, semilla y contar cuantos pasos cayeron en determinado cuadrante
subroutine walk_2d(switch,n_file,rnd_type,seed,n_step,x0,y0,x,y,dcm,&
    count_cuad_01,count_cuad_02,count_cuad_03,count_cuad_04)
    use module_precision;use module_random_generator
    use module_mzran;use module_mt19937

    implicit none
    integer(sp), intent(in)    :: switch,n_file ! prender o apagar escritura de datos
    integer(sp), intent(in)    :: n_step    ! numero de pasos totales
    real(dp),    intent(in)    :: x0,y0     ! cordenadas iniciales
    integer(sp), intent(inout) :: rnd_type,seed
    real(dp),    intent(out)   :: x,y      ! coordenadas
```

```
    real(dp),    intent(out)   :: count_cuad_01,count_cuad_02,&
                                  count_cuad_03,count_cuad_04
    real(dp),   intent(out)    :: dcm       ! desplazamiento cuadrático medio

    !real(dp),    parameter :: px=0.5_dp,py=0.5_dp ! probabilidades de pasos
    real(dp),    parameter :: step=1._dp          ! longitud de paso (fija)
    integer(sp)            :: i
    real(dp)               :: nrand               ! numero pseudo-aleatorio

    if (switch==1_sp) then;20 format(E12.4,x,E12.4);write(n_file,20) x,y;end if
    ! posición inicial
    x=x0;y=y0;count_cuad_01=0._dp;count_cuad_02=0._dp;count_cuad_03=0._dp;count_cuad_04=0._dp
    !if (rnd_type==4_sp) call sgrnd(seed)
    do i=1,n_step
        select case(rnd_type)
                case(1);nrand=ran0(seed)                        ! ran0 random generator
                case(2);nrand=ran2(seed)                        ! ran2 random generator
                case(3);nrand=rmzran()                          ! mzran random generator
                case(4);nrand=real(grnd(),dp) ! mt19937 random generator
                case default; write(*,*) 'Invalid random generator type'
        end select
        cond1:   if (0._dp<=nrand.and.nrand<0.25_dp)  then; x=x+step; exit cond1
            else if (0.25_dp<=nrand.and.nrand<0.5_dp) then; y=y+step; exit cond1
            else if (0.5_dp<=nrand.and.nrand<0.75_dp) then; x=x-step; exit cond1
            else if (0.75_dp<=nrand.and.nrand<1._dp)  then; y=y-step; exit cond1
        end if cond1
        if (switch==1_sp) write(n_file,20) x,y
    end do
    ! Determinamos el cuadrante de la partícula al final de la caminata
    cond2:   if (x>0._dp.and.y>0._dp) then; count_cuad_01=count_cuad_01+1._dp; exit cond2
        else if (x<0._dp.and.y>0._dp) then; count_cuad_02=count_cuad_02+1._dp; exit cond2
        else if (x<0._dp.and.y<0._dp) then; count_cuad_03=count_cuad_03+1._dp; exit cond2
        else if (x>0._dp.and.y<0._dp) then; count_cuad_04=count_cuad_04+1._dp; exit cond2
        else if (x==0._dp.and.y==0._dp) then     ! origen de coordenadas
        count_cuad_01=count_cuad_01+0.25_dp;count_cuad_02=count_cuad_02+0.25_dp
        count_cuad_03=count_cuad_03+0.25_dp;count_cuad_04=count_cuad_04+0.25_dp; exit cond2
        else if (x>0._dp.and.y==0._dp) then ! semi-eje x positivo
        count_cuad_01=count_cuad_01+0.5_dp;count_cuad_04=count_cuad_04+0.5_dp; exit cond2
        else if (x<0._dp.and.y==0._dp) then ! semi-eje x negativo
        count_cuad_02=count_cuad_02+0.5_dp;count_cuad_03=count_cuad_03+0.5_dp; exit cond2
        else if (x==0._dp.and.y>0._dp) then ! semi-eje y positivo
        count_cuad_01=count_cuad_01+0.5_dp;count_cuad_02=count_cuad_02+0.5_dp; exit cond2
        else if (x==0._dp.and.y<0._dp) then ! semi-eje y negativo
        count_cuad_03=count_cuad_03+0.5_dp;count_cuad_04=count_cuad_04+0.5_dp; exit cond2
    end if cond2
    if (switch==2_sp) dcm=(x-x0)*(x-x0)+(y-y0)*(y-y0)
end subroutine walk_2d
```

Problema 03 - Códigos

mc_integration.f90

```
! problema 03
program mc_integration
    use module_precision
    use module_random_generator
    implicit none

    integer(sp), parameter :: pot=2_sp   ! potencia
    integer(sp)            :: n ! cantidad de evaluaciones de la función
    real(dp),    parameter :: x_start=0._dp,x_end=1._dp
    real(dp),    parameter :: exact_integ=1._dp/(real(pot,dp)+1._dp)
    integer(sp)            :: seed,seed_val(8),i,j,k,istat
    real(dp)               :: nrand,x_rand,f_rand,integ

    call date_and_time(values=seed_val)
    seed=seed_val(8)*seed_val(7)*seed_val(6)+seed_val(5)

    integ=0._dp
    open(10,file='../results/result_mc_integrator_pot2.dat',status='replace',action='write',iostat=istat) ! p/ pot=2
    !open(10,file='../results/result_mc_integrator_pot3.dat',status='replace',action='write',iostat=istat) ! p/ pot=3
    if (istat /= 0_sp) write(*,*) 'istat_error=', istat
    20 format(2(A12,x),A12); 21 format(I12,x,E12.4,x,E12.4)
    write(10,20) 'n','Iaprox','E_rel'
    do i=1,1E+04
```

```
            n=10_sp*i
        do j=1,n
            nrand=ran2(seed)
            x_rand=nrand*(x_end-x_start)+x_start ! genero un x random en el intervalo de integración
            f_rand=1._dp;do k=1,pot;f_rand=f_rand*x_rand;end do
            integ=integ+f_rand
        end do
        integ=(x_end-x_start)*(1._dp/real(n))*integ
        write(10,21) n,integ,abs((exact_integ-integ)*(1._dp/exact_integ))
    end do
    close(10)

    ! control de resultados
    write(*,'(A10,E12.4)') 'Iaprox=',integ
    write(*,'(A10,E12.4)') 'Iexact=',1._dp/(real(pot,dp)+1._dp)

end program mc_integration
```

mc_integration_imp_sampling.f90

```
! problema 03.b
! ojo la distribución debe normalizarse segun estos valores
! este programa sólo vale para cuando x_start=0; x_end=1
program mc_integration_imp_sampling
    use module_precision;use module_random_generator
    implicit none

    integer(sp), parameter :: pot=3_sp   ! potencia (debe ser mayor a -2)
    integer(sp), parameter :: potk=3_sp  ! usar valores 2 y 3
    integer(sp)            :: n ! cantidad de evaluaciones de la función
    real(dp),    parameter :: x_start=0._dp,x_end=1._dp
    real(dp),    parameter :: exact_integ=1._dp/(real(pot,dp)+1._dp)
    integer(sp)            :: seed,seed_val(8),i,j,k,istat
    real(dp)               :: nrand,x_rand,f_rand,g_rand,integ
    real(dp)               :: factor

    call date_and_time(values=seed_val)
    seed=seed_val(8)*seed_val(7)*seed_val(6)+seed_val(5)

    integ=0._dp
    !open(10,file='../results/result_P03b_01.dat',status='replace',action='write',iostat=istat) ! p/ potk=2
    open(10,file='../results/result_P03b_02.dat',status='replace',action='write',iostat=istat)  ! p/ potk=3
    if (istat /= 0_sp) write(*,*) 'istat_error=', istat
    20 format(2(A12,x),A12);21 format(I12,x,E12.4,x,E12.4)
    write(10,20) 'n','Iaprox','E_rel'
    do i=1,1E+04
            n=10_sp*i
        do j=1,n
            nrand=ran0(seed)
            ! x^k distribution (x \in {1,Infinity})
            x_rand=nrand**(1_dp/real(potk+1_dp,dp))
            f_rand=1._dp;do k=1,pot;f_rand=f_rand*x_rand;end do  ! x_rand**pot
            factor=1._dp;do k=1,potk;factor=factor*x_rand;end do ! x_rand**potk
            g_rand=(real(potk,dp)+1_dp)*factor
            integ=integ+f_rand*(1._dp/g_rand)
        end do
        integ=(x_end-x_start)*(1._dp/real(n))*integ
        write(10,21) n,integ,abs((exact_integ-integ)*(1._dp/exact_integ))
    end do
    close(10)

    ! controlamos valores de la integral en el último paso
    write(*,'(A10,E12.4)') 'Iaprox=',integ
    write(*,'(A10,E12.4)') 'Iexact=',1._dp/(real(pot,dp)+1._dp)

end program mc_integration_imp_sampling
```

Problema 04 - Códigos

hyper_sphere.f90

```
!Problema 04
```

```
program hyper_sphere
    use module_precision

    implicit none
    ! variables generales
    real(dp),    parameter :: x_end=1._dp,x_start=0._dp
    integer(sp)            :: n ! dimensiones
    integer(sp)            :: i,j,k,l,istat
    real(dp)               :: exact_volume,volumen
    real(dp)               :: t_start,t_end
    ! variables para método del trapecio
    real(dp)               :: Iaprox,factor
    ! variables para método de monte carlo
    integer(sp), parameter :: n_random=10**6_sp
    real(dp),    parameter :: x_med=0._dp,sigma_max=1._dp,sigma_min=0.1_dp
    integer(sp)            :: seed,seed_val(8),sigma_n
    real(dp)               :: x_rand,f_rand,integ,r
    real(dp)               :: g_inv_rand,sigma,sigma_step,rel_err
    real(dp)               :: gauss_dist,heaviside,gaussdev

    open(10,file='../results/result_P04a_01.dat',status='replace',action='write',iostat=istat)
    20 format(I14,x,2(E14.6,x),E14.6); 21 format(3(A14,x),A14)
    write(10,21) 'n-dimension','tr_volumen','ex_volumen','rel_err'
    if (istat /= 0_sp) write(*,*) 'istat_error=', istat

    !trapez_integral(m,a,b,n,Iaprox)
    volumen=1._dp
    do j=1_sp,4_sp
        n=j ! n={1,2,3,4}
        volumen=1._dp
        do i=1,n-1
            !call trapez_integral(((2**24)/n),x_start,x_end,i,Iaprox)
            call trapez_integral(int((2._dp**(24._dp/real(n,dp))),sp)+1_sp,x_start,x_end,i,Iaprox)
            write(*,*) n,int((2._dp**(24._dp/real(n,dp))),sp)+1_sp
            volumen=volumen*Iaprox
        end do
        factor=1._dp;do i=1,n;factor=factor*2._dp;end do ! 2**n
        volumen=volumen*factor
        rel_err=abs(exact_volume(n)-volumen)*(1._dp/exact_volume(n))
        write(10,20) n,volumen,exact_volume(n),rel_err
    end do
    close(10)

    open(11,file='../results/result_P04b_01.dat',status='replace',action='write',iostat=istat)
    30 format(I12,x,5(E12.4,x),E12.4)); 31 format(6(A12,x),A12)
    write(11,31) 'n-dimension','elapsed time','mc_volumen','rel_err','sigma','ex_volumen','n-ball/n-cube'
    if (istat /= 0_sp) write(*,*) 'istat_error=', istat

    open(12,file='../results/result_P04b_02.dat',status='replace',action='write',iostat=istat)
    32 format(I12,x,E12.4,x,E12.4); 33 format(2(A12,x),A12)
    write(12,33) 'n-dimension','ex_volumen','n-ball/n-cube'
    if (istat /= 0_sp) write(*,*) 'istat_error=', istat

    ! monte carlo
    call date_and_time(values=seed_val)
    seed=seed_val(8)*seed_val(7)*seed_val(6)+seed_val(5)

    n=100_sp ! numero máximo de dimensiones
    do l=1_sp,n
        sigma_n=100_sp*l
        sigma_step=abs(sigma_max-sigma_min)*(1._dp/(real(sigma_n,dp)-1._dp))
        call cpu_time(t_start)
        do1: do k=1,sigma_n
            sigma=sigma_min+sigma_step*(real(k,dp)-1._dp)
            integ=0._dp
            do j=1,n_random
                r=0._dp;g_inv_rand=1._dp
                do i=1,l
                    x_rand=gaussdev(seed,1_sp)
```

```
                    x_rand=(sigma*x_rand+x_med)
                    r=r+x_rand*x_rand
                    g_inv_rand=g_inv_rand*(1._dp/gauss_dist(x_rand,x_med,sigma))
                end do
                r=sqrt(r) ! (x1^2+x2^2+...+xn^2)^(1/2)
                f_rand=heaviside(r)
                integ=integ+f_rand*g_inv_rand
            end do
            integ=(x_end-x_start)*(1._dp/real(n_random,dp))*integ
            volumen=integ
            rel_err=abs(exact_volume(l)-volumen)*(1._dp/exact_volume(l))
            if (rel_err<=0.001) then;write(*,*), 'VERIFICA';exit do1;end if
        end do do1
        call cpu_time(t_end)
        write(11,30) l,(t_end-t_start),volumen,rel_err,sigma,exact_volume(l),exact_volume(l)*
(1._dp/(2_dp**real(l,dp)))
        write(12,32) l,exact_volume(l),exact_volume(l)*(1._dp/(2_dp**real(l,dp)))
    end do
    close(11)

end program hyper_sphere

subroutine trapez_integral(m,a,b,n,Iaprox)
    use module_precision
    implicit none
    ! Data dictionary: declare calling parameter types & definitions
    integer(sp), intent(in)     :: m      ! cantidad puntos => m = n + 1, n intervals number
    real(dp),    intent(in)     :: a,b    ! límites de integración
    real(dp),    intent(out)    :: Iaprox ! numerical integration with trapezoidal method
    integer(sp), intent(in)     :: n         ! dimension hyper-sphere
    ! Data dictionary: declare local variables types & definitions
    integer(sp) :: i ! index loop
    real(dp)    :: h,x_current
    real(dp)    :: function_vector(1,m),coeff_vector(m,1),Iaprox_aux(1,1)

    h=abs((b-a))*(1._dp/(real(m,dp)-1._dp)) ! paso de integración
    x_current=a
    coeff_vector(:,1)=2._dp
    do i=2,m-1
        x_current=x_current+h
        function_vector(1,i)=sqrt((1._dp-x_current*x_current)**real(n,dp))
    end do
    coeff_vector(1,1)=1._dp;coeff_vector(m,1)=1._dp
    function_vector(1,1)=sqrt((1._dp-a*a)**real(n,dp))
    function_vector(1,m)=sqrt((1._dp-b*b)**real(n,dp))
    Iaprox_aux=h*matmul(function_vector,coeff_vector)*0.5_dp
    Iaprox=Iaprox_aux(1,1)
end subroutine trapez_integral

function heaviside(r)
    use module_precision
    implicit none
    real(dp), intent(in) :: r
    real(dp)             :: heaviside
    if (r<=1._dp) heaviside=1._dp
    if (r>1._dp) heaviside=0._dp
end function heaviside

! to calculate 1D gauss distribution
function gauss_dist(x,x_med,sigma)
    use module_precision
    implicit none
    real(dp), intent(in) :: x,x_med,sigma
    real(dp),  parameter :: pi=4._dp*atan(1._dp)
    real(dp)             :: gauss_dist,factor_01,factor_02
    factor_01=1._dp/(sqrt(2._dp*pi)*sigma)
    factor_02=(x-x_med)*(1._dp/sigma)
    gauss_dist=factor_01*exp(-0.5_dp*factor_02*factor_02)
end function gauss_dist
```

```
! To calculate de exact expresion for hyper-sphere's volume
function exact_volume(n)
    use module_precision
    implicit none
    integer(sp), intent(in) :: n
    real(dp),    parameter  :: pi=4._dp*atan(1._dp)
    real(dp)                :: exact_volume,factor_02,factorial
    integer(sp)             :: factor_01,i
    cond1: if (mod(n,2_sp)==0_sp) then ! n pares
        factor_01=n/2_sp
        exact_volume=(pi**real(factor_01,dp))*(1._dp/factorial(factor_01))
        exit cond1
    else ! n impares
        factor_01=(n-1_sp)/2_sp
        factor_02=1._dp;do i=1,n;factor_02=factor_02*2._dp;end do ! factor_02=2**n
        exact_volume=(pi**real(factor_01,dp))*factor_02*factorial(factor_01)*(1._dp/factorial(n))
        exit cond1
    end if cond1
end function exact_volume

! To calculate the factorial function
recursive function factorial(n) result(fact)
    use module_precision
    implicit none
    integer(sp), intent(in) :: n
    real(dp)                :: fact
    if (n>=1_sp) fact=real(n,dp)*factorial(n-1_sp)
    if (n==0_sp) fact=1_dp
end function factorial

function gaussdev(seed,rnd_type)
    use module_precision;use module_random_generator
    use module_mzran;use module_mt19937
    implicit none
    integer(sp), intent(in) :: seed, rnd_type
    integer(sp)             :: iset=0_sp
    real(dp),    parameter  :: pi=4._dp*atan(1._dp)
    real(dp) :: gset,gaussdev,nrand_01,nrand_02
    save iset,gset
    ! queda pendiente agregar más rnd_type para incluir
    !  distintos generadores
    select case(rnd_type)
    case(1);nrand_01=ran2(seed);nrand_02=ran2(seed)
    case(2);nrand_01=ran0(seed);nrand_02=ran0(seed)
    case(3);nrand_01=rmzran();nrand_02=rmzran()
    case(4);call sgrnd(seed);nrand_01=real(grnd(),dp);nrand_02=real(grnd(),dp)
    end select
    if (iset==0_sp) then
        gset=sqrt(-2*log(nrand_01))*cos(2*pi*nrand_02)
        gaussdev=sqrt(-2*log(nrand_01))*sin(2*pi*nrand_02)
        iset=1_dp
    else;gaussdev=gset;iset=0_sp;end if
end function gaussdev
```