Capítulo 3

Integración Numérica

Muchos valores de interés para las ciencias se expresan a través de integrales, y algunas de esas integraes son difíciles de resolver analíticamente. Esto puede deberse a la complejidad de la función a integrar, o al dominio de integración o a ambos. Un ejemplo se encuentra en Parker (1998) usando teoría estadística

$$P(q) = \int_{-\infty}^{\infty} \frac{e^{-\alpha\sqrt{p^2+q^2}}\left(1 - e^{-\beta\sqrt{p^2+q^2}}\right)^2}{q^2 + \epsilon^2 p^2} dp, \tag{3.1}$$

En algunos casos las integrales pueden ser aún mas complejas en estadística, magnetismo, sismología, mecánica cuántica, biología, etc. Otro ejemplo:

$$T = \sqrt{\frac{8l}{g}} \int_{0}^{\theta_m} \frac{d\theta}{\sqrt{cos\theta - cos\theta_m}} \tag{3.2}$$

para un péndulo. Este tipo de integrales también puede ser difícil de resolver con base en funciones elementales. Entonces, que se puede hacer al enfrentarse a este tipo de integrales?

Existen dos posibles pasos a seguir si no se puede llegar a una solución analítica de alguna integral.

1. Hacer algunas aproximaciones, haciendo algún parametetro pequeño, análisis asintótico, etc.

2. Hacer la integración numérica

Ambos pasos requieren de un poco de arte, pero en este texto estamos interesados en el segundo punto. Por qué decimos arte? Que pasa si se quiere hacer una integración numérica en Matlab? Existen dos funciones (`quad` y `quad8`, pero éstas solo pueden trabajar con integración sobre intervalos finitos y además requieren que la función se comporte de cierta manera (que no haya singularidades, que la función se diferenciable, etc.). En este texto se discutirá este tipo

de situaciones e incluso como realizar integrales numéricas similares a 3.1, que tengan límites infinitos

Que tan buena es la aproximación de una integral con suma? Este es el tipo de pregunta que el lector tiene que hacer ya que los computadores son realmente buenos sumando, pero malos para integrar. Y a veces, sumar no es eficiente y se requiere *ayudar* al computador para obtener un resultado preciso y rápido.

3.1. Cuadratura cómo un conteo de bloques

La integración numérica se conocía anteriormente como cuadratura numérica o simplemente cuadratura (Quadrature en inglés). La integración numérica de una función puede requerir de habilidad matemática, pero en principio es fácil en un computador. Una forma tradicional de integración numérica es usar un papel milimetrado y contar el número de cajones (cuadrados) que estan por debajo de la curva a integrar. Es por esto que en principio se llama cuadratura (aunque el método sea más avanzado que contar cajones).

Siguiendo la definición de la integral de Riemman (tal vez la primera definición rigurosa de la integral de una función)

$$\int_a^b f(x)dx = \lim_{h\to 0}\left[h\sum_{i=1}^{(b-a)/h} f(x_i)\right] \tag{3.3}$$

la integral de una función $f(x)$ entre a y b. es el límite de la suma sobre cajones a medida que el espesor h de los cajones aproxima cero.

La integración numérica de una función $f(x)$ se aproxima entonces como la suma *finita* sobre cajones de altura $f(x)$ y espesor w_i,

$$\int_a^b f(x)dx \approx \sum_{i=1}^{N} f(x_i)w_i, \tag{3.4}$$

lo cual se asemeja a la definición de Riemann, excepto que no hay un límite.

La ecuación 3.4 es la forma standard to todos los algoritmos de integración; la función $f(x)$ se evalua en N puntos en el intervalo $[a, b]$ a los valores de la función $f_i = f(x_i)$ se suman con una ponderación o peso w_i. Esta sumatoria da la integral exacta **sólamente** cuando $N = \infty$, y puede ser exacta aún si N es finito si la función integrable es un polinomio.

Los distintos algoritmos de integración equivalen a diferente selección de los pesos w_i y de los puntos x_i. A medida que N aumenta, asi mismo aumenta la precisión (excepto por errores de redondeo o *round-off*). La *mejor* aproximación depende en buena parte de la función $f(x)$.

- Si la función presenta una singularidad, el usuario debe removerla o partir el intervalo en subintervalos para evitar sumar la singularidad.

- Algunas funciones varian muy lentamente en algunas regiones y es necesario *acelerar* el decaimiento para sumar más rápido. Esto se puede hacer con cambios de variables de tal forma que comprima la región de interés y sean necesarios menos puntos N.

3.2. La Regla Trapezoidal

La primera aproximación para la integración numérica es la regla trapezoidal, ilustrada gráficamente en la Figura 3.1. La curva a integrar se aproxima con trapezoides. Cada trapezoide posee un area proporcional al promedio de la altura de los lados multiplicado por la base (espesor).

$$\begin{aligned} T_h[f;a,b] &= \frac{h}{2}(f(a)+f(a+h)) + \frac{h}{2}(f(a+h)+f(a+2h)) + \qquad (3.5)\\ &\quad \cdots + \frac{h}{2}(f(b-h)+f(b)) \\ &= \frac{h}{2}(f(a)+f(b)) + h\sum_{j=0}^{N} f(a+jh) \qquad (3.6) \end{aligned}$$

donde el espaciamiento entre puntos es $h = (b-a)/N$.

En términos de la ecuación 3.4, los pesos w_i siguen la siguiente regla

$$\boxed{w_i = \left\{\frac{h}{2}, h, h, \cdots, h, \frac{h}{2}\right\}} \quad \text{Regla Trapezoidal} \qquad (3.7)$$

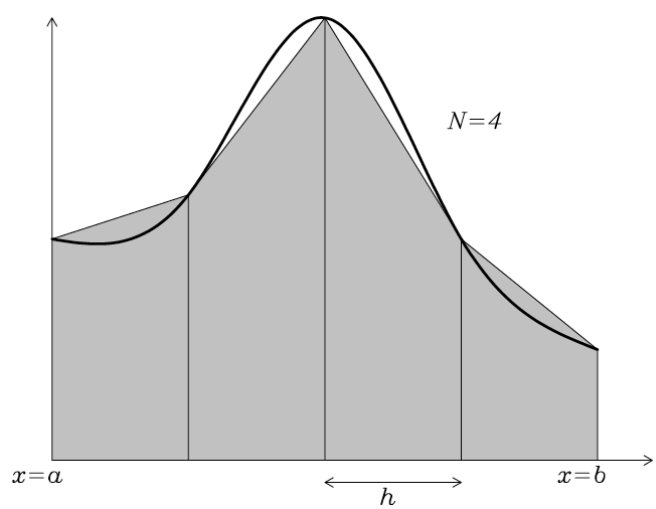

Figura 3.1: Regla trapezoidal

Programa 3.1 Ejemplo de subrutina de $\mathbb{F}90$ para integración numérica de función `fun` con la regla Trapezoidal.

```
(trapz.f90)
subroutine trapz(fun,a,b,N,I)
! Numerical integration using the trapezoidal rule
!
! INPUTS:
!        fun      real(8) function routine that needs to be called
!                 by the trapezoidal rule.
!        a, b     upper and lower limit of integration, real(8)
!        N        number of points used to subdivide the integration
!                 limits. Integer
! OUTPUT:
!        I        real(8) value with result of the integration
!
! uses function "fun"
!******
   implicit none
   integer, intent(in)  :: N
   real(8), intent(in)  :: a, b
   real(8), intent(out) :: I
   integer :: j
   real(8) :: x, h
   real(8) :: fun

!  Initialize integral to zero
   I = 0.d0

!  Divide the region of integration, using sampling h
   h = (b-a)/dble(N-1)

!  Start integrating
   x = a
   I = I + fun(x)/2.d0
   do j = 1,N-2
      x = x+h
      I = I + fun(x)
   enddo
   x = b
   I = I + fun(x)/2.d0

!  Normalize by the sampling h
   I = I*h

end subroutine trapz
```

3.3. La regla de Simpson

Para cada intervalo, la regla de Simpson aproxima la función a integrar $f(x)$ con una parabola (ver figura 3.2):

$$f(x) \approx \alpha x^2 + \beta x + \gamma \tag{3.8}$$

donde todos los intervales tienen un muestreo homogeneo. Sin la intención de derivar esta relación, para el intervalo -1 and $+1$ la función $f(x)$ se puede expresar como la suma ponderada de valores de la función en tres puntos, así:

$$\int_{-1}^{+1} \alpha x^2 + \beta x + \gamma = \frac{f(-1)}{3} + \frac{4f(0)}{3} + \frac{f(1)}{3} \tag{3.9}$$

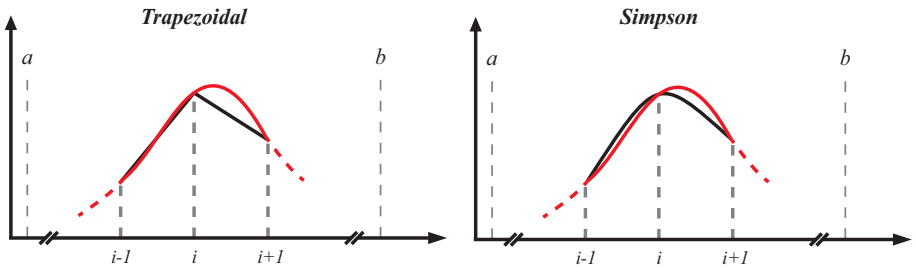

Figura 3.2: Aproximación de una función $f(x)$ a ser integrada (curva roja) usando la regla trapezoidal (lineas rectas) y la regla de Simpson (interpolante cuadrático), ambos con lineas negras. Note que la función $f(x)$ es dividida en N puntos, y la aproximación se hace en sub-intervalos $(i \pm 1)$ en la figura.

Este última aproximación se puede generalizar evaluando la función en dos intervalos adyacentes, de tal forma que la función $f(x)$ se evalua en los dos extremos $(i \pm 1)$ y un punto central (i)

$$\int_{x_i-h}^{x_i+h} f(x) \approx \frac{h}{3} f_{i-1} + \frac{4h}{3} f_i + \frac{h}{3} f_{i+1} \tag{3.10}$$

En términos de nuestra regla de integración standard (ecuación 3.4), se obtienen los pesos

$$\boxed{w_i = \left\{ \frac{h}{3}, \frac{4h}{3}, \frac{2h}{3}, \frac{4h}{3}, \cdots, \frac{4h}{3}, \frac{h}{3} \right\}} \quad \text{Regla de Simpson} \tag{3.11}$$

Sin embargo, es importante notar que **el número de puntos N TIENE que ser impar** para la regla de Simpson. Esto puede no ser una dificultad para funciones que se pueden evaluar en cualquier punto, pero si puede ser una limitación si se tienen datos de campo o de laboratorio.

Programa 3.2 Ejemplo de subrutina de $\mathbb{F}90$ para integración numérica de función `fun` con la regla de Simpson.

```
(simps.f90)
subroutine simps(fun,a,b,N,I)
! Numerical integration using the Simpson's rule
...
!**************
   implicit none
   integer, intent(in)  :: N
   real(8), intent(in)  :: a, b
   real(8), intent(out) :: I
   integer :: j, M
   real(8) :: x, h
   real(8) :: fun

   if (mod(N,2)==0) then
      write(6,*) 'N must be odd, one point added'
      M = N+1
   else
      M = N
   endif

!  Initialize integral to zero
   I = 0.d0

!  Divide the region of integration, using sampling h
   h = (b-a)/dble(M)

!  Start integrating
   x = a
   I = I + fun(x)
   do j = 1,(M)/2-1
      x = x+h
      I = I + 4.d0*fun(x)
      x = x+h
      I = I + 2.d0*fun(x)
   enddo
   x = x+h
   I = I + 4.d0*fun(x)
   x = x+h
   I = I + fun(x)

! Normalize by the sampling h
   I = I * h/3.d0

end subroutine simps
```

3.4. Error de integración

En general, el usuario elige una regla de integración que resulte en una respuesta precisa realizando la menor cantidad de operaciones, es decir con un valor N tan pequeño como sea posible. Un estimado crudo de la aproximación o error del algoritmo ε y de su error relativo ϵ expandiendo la función $f(x)$ como una serie de Taylor alrededor del punto central del intervalo de integración. Posteriormente se multiplica el error por el número de intervalos N que se ha utilizado para dividir el intervalo total $[a, b]$.

Cuadro 3.1: Errores de algoritmos de integración.

Integración	**Aproximación** ε	**Error Rel.** ϵ
Trapezoidal	$\varepsilon_t = \left(\frac{[b-a]^3}{N^2}\right) f^{(2)}$	$\epsilon_s = \frac{\varepsilon_t}{f}$
Simpson	$\varepsilon_s = \left(\frac{[b-a]^5}{N^4}\right) f^{(4)}$	$\epsilon_s = \frac{\varepsilon_s}{f}$

La tabla 3.1 muestra cono una pequeña modificación en l aintegración puede llevar a una proporcionalidad a una derivada de f de mayor orden. Tanbién que, como es de esperarse, a un mayor número N, un descenso del error.

En general, la conclusión fundamental es que la regla de Simpson es una mejora sobre la regla Trapezoidal, con una pequeña adición en la dificultad de la implementación del algoritmo.

3.5. Cuadratura Gaussiana

Hasta el momento se ha discutido dos métodos de integración, el trapezoidal y el de Simpson. Ambos métodos usan un muestreo homogéneo de la función $f(x)$, por lo cual pueden ser muy lentos cuando de obtener cierto nivel de precisión se habla.

En algunos casos es útil re-escribir la formula básica de integración (3.4) de tal forma que se separe la función de ponderación $W(x)$ de la función a integrar

$$\int_a^b f(x)dx = \int_a^b W(x)g(x)dx \approx \sum_{i=1}^{N} w_i g(x_i) \tag{3.12}$$

En los métodos de cuadratura gaussiana, los N puntos y pesos w_i son elegidos de tal forma que el error de aproximación desaparezca si $g(x)$ es un polinomio de grado $(2N-1)$. Para obtener este resultado tan increible los puntos x_i no tienen

un muestreo uniforme y al contrario tienen una distribución más compleja en el intervalo $[a, b]$.

Los algoritmos de cuadratura gaussiana producen resultados con mayor precisión que los métodos de Simpson y Trapezoidal. La selección de puntos x_i no es simple y no entraremos en detalle al respecto. Existen distintas cuadraturas gaussianas y para cada una de ellas es posible encontrar algoritmos en $\mathbb{C}$, $\mathbb{F}90$ e incluso Matlab si el lector lo cree necesario. Es importante que el lector sepa de su existencia si en algún momento requiere de su uso.

3.6. Cómo integrar más rápido

Volviendo a la pregunta de como realizar cálculos en un computador cuando la integral es sobre un intervalo infinito, es necesario mejorar la taza a la cual se reduce el error en las reglas trapezoidal o Simpson (que básicamente son proporcionales a N^2).

Una sugerencia es el uso de *Integración de Romberg*, donde se aplica la ecuación (3.6) a un número diferente de valores de h, por ejemplo δ, $\delta/2$, $\delta/4$, y así eliminar tantas derivadas como sea posible. Este método es bastante popular cuando el intervalo es (a, b).

La anterior es una opción que dejamos abierta al lector, pero queremos presentar un resultado extraordinario. Primero, un ejemplo para convencer al lector que continuar leyendo esta sección realmente vale la pena. Una integral cuyo resultado conocemos es:

$$\int_{-\infty}^{\infty} \frac{dx}{1+x^2} = \pi \tag{3.13}$$

Para este caso, aplicando la regla Trapezoidal con un espaciamiento h, da

$$T_h = h + \sum_{j=1}^{\infty} \frac{2h}{1+(jh)^2} = \pi + \varepsilon \tag{3.14}$$

Aplicando el algoritmo trapezoidal con diferentes valores de h, se obtiene el resultado de la tabla 3.2. Estos resultados son estupendos, en particular si se

Cuadro 3.2: default

h	T_h	ε
2	2.85...	0.28
1	3.109 ...	0.032
0.5	3.14159 ...	$3{,}5 \times 10^{-08}$
0.25	3.1415926535897953 ...	$5{,}3 \times 10^{-15}$

tiene en cuenta que la reducción del error de tener un $h = 1$ a un $h = 0{,}25$

debería ser de un factor de 0,08 y no de 10^{-15}. Estos resultados muestran que por cada vez que se divide en 2 el valor de h, el número del decimal se dobla.

Qué truco se utilizó para acelerar esta integración? El truco utilizado no es siempre aplicable, pero puede ser útil en multiples casos. Utilizando la regla trapezoidal en la ecuación 3.14, se requeriría de 7×10^{10} términos para obtener la misma aproximación que el aplicado acá con un $h = 0{,}2$. Para evitar hacer demasiados cálculos (por lo menos en el caso de funciones monotonicas) la idea es *comprimir* los bordes de la función a través de *mapeo* de la linea real sobre si misma. Un buen *mapa* para este ejemplo es cambiar las variables $x = sinhu$, con derivada $dx = coshudu$ para obtener una nueva función

$$I = \int_{-\infty}^{\infty} f(x)dx = \int_{-\infty}^{\infty} f(\sinh u)\cosh u\, du \tag{3.15}$$

donde se puede observar [Figura 3.5] que la función cae como $exp(|u|)$. Después de un poco de algebra se obtiene

$$\int_{-\infty}^{\infty} \frac{dx}{1+x^2} = \int_{-\infty}^{\infty} \frac{du}{\cosh u} \tag{3.16}$$

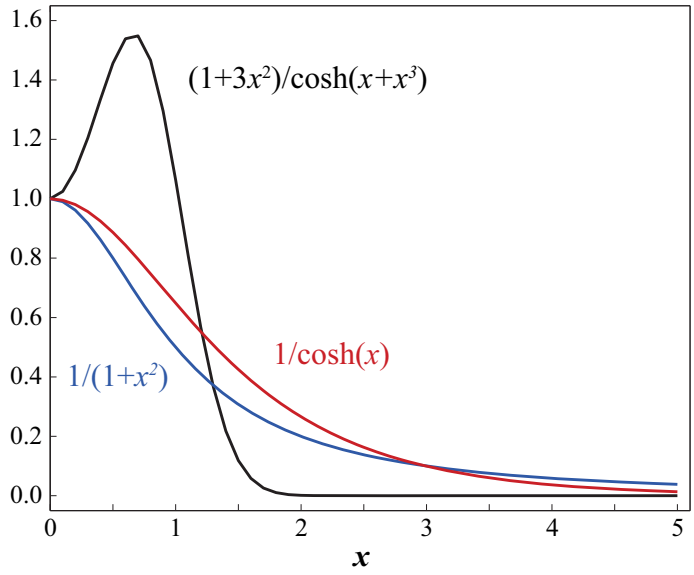

Figura 3.3: Acelerando la integración

Decaimiento exponencial puede no ser lo suficientemente bueno, y si el lector esta de afán, puede intentar

$$x = \sinh(t+t^3), \quad dx = \cosh(t+t^3)(1+3t^2)dt \tag{3.17}$$

para obtener

$$\int_{-\infty}^{\infty} \frac{dx}{1+x^2} = \int_{-\infty}^{\infty} \frac{(1+3x^2)}{\cosh(u+u^3)} du \tag{3.18}$$

La función ahora decae rápidamente (e^{t^3}) lo que quiere decir que se requieren pocos términos y los límites de integración no tienen que ir hasta ∞, sino hasta valores pequeños.

Una observación final, en el caso que los límites de integración sean por ejemplo (a, ∞), se busca un *mapeo* de tal manera que el punto $x = a$ sea $-\infty$. Esto se puede lograr con la siguiente relación $x = a + exp(\gamma u)$, $dx = \gamma exp(\gamma u) du$, para un valor apropiado de $\gamma > 0$.

Cuando la función $f(x)$ posee complejidades en algún intervalo finito antes de entrar en un decaimiento monotónico, se recomienda realizar la integración en dos partes; la porción finita usando un método standard, y la *cola* del intervalo semi.infinito a través de un *mapeo* descrito arriba.

3.7. Integración de Monte Carlo

En esta sección se introduce brevemente la idea de la integración de Monte Carlo. En algunas aplicaciones, como por ejemplo mecánica estadística y teoría de campos cuánticos, la integración puede ser sobre espacios de múltiples dimensiones. E integrar sobre múltiples dimensiones puede costar mucho tiempo de computador si se utilizan los métodos discutidos hast ahora.

La idea básica es generar un muestreo aleatorio del espacio N-dimensional a través de un generador de números aleatorios. Idealmente al usar un muestreo aleatorio será posible tener una idea aproximada del área por debajo de la función de interés.

PRECAUCIÓN: Tener en cuenta que tanto en $\mathbb{C}$, $\mathbb{F}90$ y Matlab los algoritmos generadores de números aleatorios generalmente SIEMPRE inician con los mismos números aleatorios. Esto quiere decir, que si se corre un programa de $\mathbb{F}90$ con números aleatorios, la siguiente vez que se corra, los números aleatorios son idénticos. Esto es porque los generadores son *pseudo-random* y en muchos casos se requiere generar una semilla *seed* para que estos valores cambien. (En Matlab, cada vez que se reinicia el programa, el generardor de números aleatorios comienza con la misma secuencia de números).

3.7.1. Implementación tipo Stone Throwing

Este ejemplo muestra el uso del método de Monte Carlo para aproximar π. Considera un lago circular encerrado en un cuadrado con lado 2 (el lago tiene un radio de $r = 1$). Sabemos que el área del lago es igual a π. Si tuvieramos un avión que volara sobre el lago y lanzara al azar N piedras, la cantidad de priedras que caerían en el lago (comparado por fuera del lago y dentro del cuadrado) sería proporcional al área del lago.

Programa 3.3 Ejemplo de programa de integración de Monte Carlo por Stone Throwing con $N = 10000$ puntos.

```
program pond_mc
!
! Example program to do the MonteCarlo integration
! using the Stone Throwind implementation.
!
   implicit none
   integer :: n, i, j, Npond, Nbox
   real(4) :: Apond, Abox, x, y, rad
   real(4), parameter :: pi = 3.14159265358979

!  Set everything to zero
   Npond = 0
   Nbox  = 0
   Apond = 0.0
   Abox  = 4.0

   n = 10000
   do j = 1,n
      call random_number(x)
      call random_number(y)
      ! Make x and y from -1 to 1
      x = x*2.0 - 1.0
      y = y*2.0 - 1.0
      ! Calculate radius
      rad = sqrt(x**2 + y**2)
      ! Check if in or out
      if (rad > 1.0) then
         Nbox = Nbox + 1
      else
         Npond = Npond + 1
      endif
   enddo
!  Calculate area
   Apond = real(Npond)/(real(Npond+Nbox)) * Abox
   print *, 'Area of pond = ', Apond
...
end program pond_mc
```

Este experimento lo podemos realizar en el computador. Se generan dos grupos de números aleatorios (*pseudo-random*) que representan las dos dimensiones del problema. Tienen límites $-1 \leqslant x_i \leqslant +1$ y $-1 \leqslant y_i \leqslant +1$. Para cualquier $i = 1, N$ se valida si el par de números cae dentro del lago con la relación $x_i^2 + y_i^2 < 1$ [ver programa 3.3].

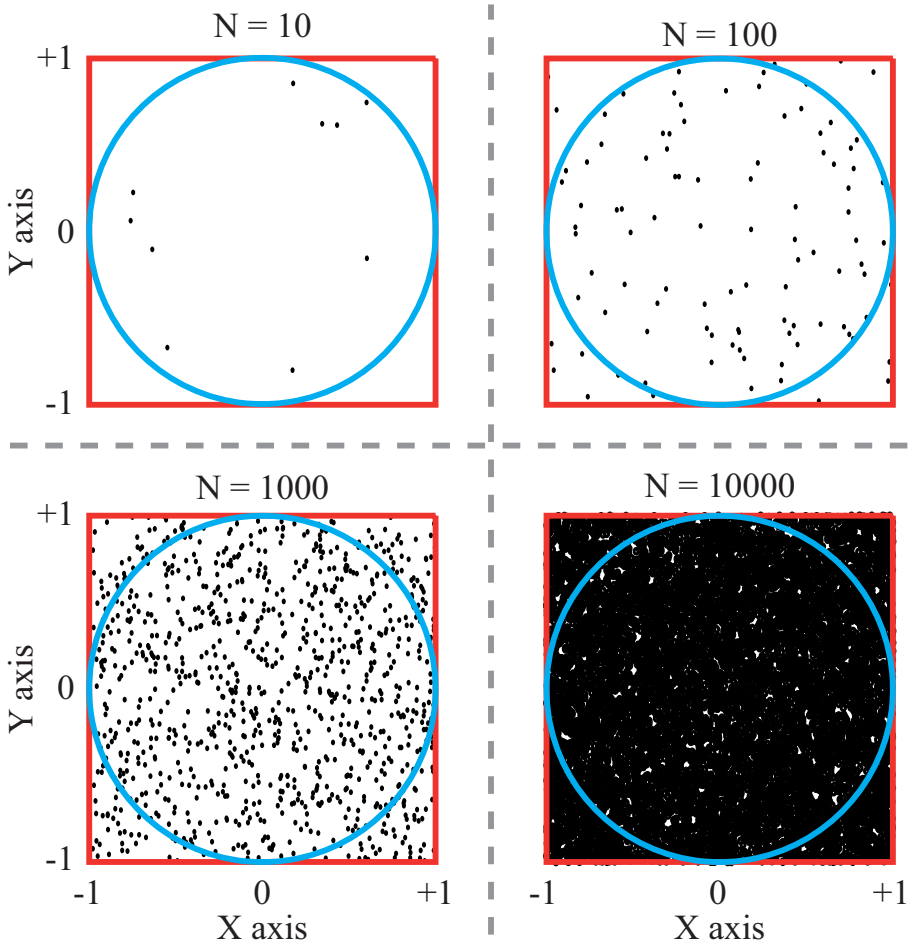

Figura 3.4: Integración de Monte Carlo utilizando la implementación *Stone Throwing*.

Si cae adentro, se adiciona un valor de 1 a la variable $N_{pond} = N_{pond} + 1$, y si no se hace lo mismo con la variable $N_{box} = N_{box} + 1$. Al terminar la generación de N experimentos, se calcula el área

$$A_{pond} = \frac{N_{pond}}{(N_{pond} + N_{box})} A_{box} \tag{3.19}$$

Obviamente a un mayor valor de N, mayor la precisión. Cuál es una buena

elección de N?

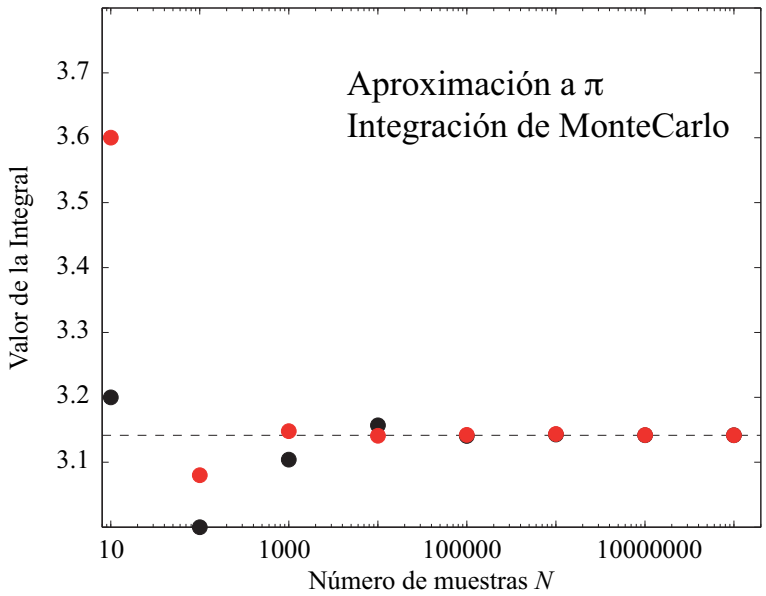

Figura 3.5: Aumento en la precisión de la integración MC, en dos experimentos independientes. El resultado deseado es π (linea punteada).

3.7.2. Integración por valor promedio

A pesar de parecer bastante simple la implementación anterior, no es muy eficiente. (1) se requiere generar números aleatorios para cada dimensión, lo cual puede tardar mucho tiempo. (2) se debe evaluar para cada par de puntos si se encuentran dentro o fuera del área de interés. En el caso de un círculo este es fácil, pero para casos o formas más complejas, ya no es necesariamente fácil saber si se encuentra dentro o fuera.

Ya el lector se habrá dado cuenta que el cálculo de una integral de $f(x)$ equivale a estimar el valor promedio de la función sobre el intervalo específico. El promedio puede ser calculado evaluando la función $f(x)$ en una serie de puntos al azar.

Este método de integración de Monte Carlo se puede implementar así:

$$I = \int_a^b f(x)dx \approx \frac{b-a}{N} \sum_{i=1}^{N} f(x_i) \tag{3.20}$$

donde x_i son los N puntos aleatorios en el intervalo $[a, b]$.

La gran ventaja de la integración de Monte Carlo es evidente cuando el problema es sobre multiples dimensiones (o múltiples variables). Abajo presentamos

la desviación standard para la integración MC (bajo condiciones de distribución normal) sin intentar su derivación

$$\sigma_I \approx \frac{1}{\sqrt{N}}\sigma_f \tag{3.21}$$

así que para valores grandes de N, el error del valor obtenido decrece proporcional a $1/\sqrt{N}$. Lo realmente interesante es que cuando se realiza integración sobre múltiples dimensiones, el error de la integración de Monte Carlo (siendo estadístico), dececе proporcional a $1/\sqrt{N}$, independiente de las dimensiones del problema D.

En el caso de las otras reglas de integración (Trapezoidal, Simpson e incluso Gaussiana), se tienen D integraciones de 1-Dimensión, por lo que para cada dimensión se tienen N/D puntos. Eso quiere decir que a medida que D aumenta, así mismo aumenta el error. Para 3 - 4 dimensiones el error de la integración de MC es similar a los métodos convencionales, pero cuando se tienen 8 o más dimensiones, MC siempre es más preciso.

Como ejemplo de la aplicación de este método de Integración Monte Carlo por valor promedio, se tiene la siguiente integral en 2D

$$I = \int_0^1 \int_0^1 e^{(x+y)^2} dx\, dy \tag{3.22}$$

cuyo valor verdadero es 4,89916. El algoritmo para solucionar esta integral numéricamente se muestra en el Programa 3.4.

Cuadro 3.3: Resultados de integración Monte Carlo para la ecuación 3.22 realizando 100 realizaciones para cada uno de los valores N que se muestran en la tabla. Se muestra el valor estimado de la integral promedio y el resultado de máximo y mínimo de las 100 realizaciones.

N	**min**	**promedio**	**max**
10	2.271930	4.959330	9.283634
100	3.485747	4.821831	6.425066
1000	4.497928	4.877829	5.315516
10000	4.740265	4.892518	5.067602
100000	4.845586	4.899458	4.944800
1000000	4.881807	4.898822	4.913147
10000000	4.894374	4.899156	4.902969
100000000	4.897932	4.899141	4.900718

El algoritmo es relativamente sencillo y la tabla 3.3 muestra los resultados de 100 realizaciones de la integración de Monte Carlo por promedio. En 2D algoritmos e integración como el Trapezoidal o Simpson pueden ser más rápidos, pero si se incluyen más dimensiones, el método de Monte Carlo puede ser una elección apropiada.

Programa 3.4 Ejemplo de programa de integración de Monte Carlo por promedio con $N = 10000$ puntos. Note que la función a integrar se encuentra en una función propia.

```
program mc_2d

   implicit none
   integer :: n, j
   real(8) :: x, y
   real(8) :: integral
   real(8), external :: fun1

   n = 10000
   integral = 0.d0

   do j = 1,n
      call random_number(x)
      call random_number(y)
      integral = integral + fun1(x,y)
   enddo
   integral = integral/dble(n)

   print *, n, integral

end program mc_2d

real(8) function fun1(x,y) result(z)

   implicit none
   real(8)  :: x, y
   real(8)  :: z

   z = exp( (x+y)**2 )

end function fun1
```