# Parte B2: Integración Monte Carlo

Generated by Doxygen 1.9.4

# Chapter 1

# README.md Actualizado

## 1.1 Parte B2: Integración de e^(-x^2) por Monte Carlo

Este directorio contiene el código y los artefactos para la Parte B2 del proyecto final de Física Computacional II, enfocada en la aplicación del método de Monte Carlo para la integración numérica.

### 1.1.1 Descripción

Se implementa un programa para calcular la integral definida de la función $f(x) = e^{-x^2}$ en el intervalo $[0, 1]$ utilizando el método de Monte Carlo por muestreo simple. Se analiza la convergencia del valor estimado de la integral y su error en función del número de muestras utilizadas.

El valor teórico de esta integral es $\frac{\sqrt{\pi}}{2}\text{erf}(1) \approx 0.74682413$.

#### 1.1.1.1 Clases Principales:

- `IntegradorMonteCarlo`: Clase que encapsula la lógica para realizar la integración por Monte Carlo de una función univariable dada.

### 1.1.2 Estructura del Directorio <tt>ParteB2</tt>

```
ParteB2/
 bin/
    integrador_montecarlo    # Ejecutable compilado
 include/
    IntegradorMonteCarlo.h
 src/
    main_montecarlo_integral.cpp
    IntegradorMonteCarlo.cpp
 scripts/
    plot_integral_error.gp   # Script Gnuplot para visualización
 results/
    integral_error_Nmax_1e7.dat # Datos de convergencia (valor y error vs N)
    (*.png)                     # Gráficas generadas
 documents/
    integral_mc_informe.tex  # Informe LaTeX específico para esta parte
    (html_integral_mc/ y latex_integral_mc/ de Doxygen)
 Makefile                    # Makefile para compilar esta parte
 Doxyfile                    # Configuración de Doxygen para esta parte
 README.md                   # Este archivo
```

Adicionalmente, el documento `../../documents/montecarlo_fisica_estadistica.tex` (en el directorio `documents` raíz del proyecto) contiene la investigación teórica asociada a esta parte.

### 1.1.3 Flujo de Trabajo Recomendado

#### 1.1.3.1 1\. Compilación

Desde el directorio `ParteB2/`, compila el proyecto:
```
make
```
Esto generará el ejecutable `bin/integrador_montecarlo`.

- **Nota**: El `Makefile` utiliza el estándar C++11. El código hace uso de `std::erf`, que es estándar a partir de C++17 pero suele estar disponible en compiladores modernos como una extensión.

Para limpiar los archivos compilados:
```
make clean
```

#### 1.1.3.2 2\. Ejecución de la Simulación

La forma recomendada de ejecutar la simulación es usando el `Makefile`:
```
make run
```
Este comando se encarga de:

1. Crear el directorio `results/` si no existe.
2. Ejecutar la simulación con una semilla predeterminada.
3. Guardar los datos de salida en `results/integral_error_Nmax_1e7.dat`.

#### 1.1.3.3 3\. Visualización de Resultados

Una vez generados los datos, crea las gráficas con:
```
make plot
```
Este comando invoca a Gnuplot con los parámetros correctos y guarda las tres gráficas (`.png`) en el directorio `results/`.

### 1.1.4 Opciones Avanzadas

#### 1.1.4.1 Ejecución Manual

Si deseas utilizar una semilla de generador de números aleatorios diferente, puedes ejecutar el programa manualmente. Desde el directorio `ParteB2/`:
```
# Asegúrate de que el directorio de resultados exista
mkdir -p results
# Ejecuta el programa con la semilla que elijas
./bin/integrador_montecarlo [semilla]
```
Ejemplo:
```
./bin/integrador_montecarlo 42
```

#### 1.1.4.2 Documentación

- **Informe LaTeX:** El informe detallado de esta parte se encuentra en `ParteB2/documents/integral_mc_informe.tex`. Para compilarlo: ```bash cd ParteB2/documents/ pdflatex integral_mc_informe.tex pdflatex integral_mc_informe.tex cd ../.. ```
- **Documentación Doxygen:** Para generar la documentación del código: Desde el directorio `ParteB2/`: ```bash doxygen Doxyfile ``La salida HTML estará enParteB2/documents/html_integral_mc/`.

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 IntegradorMonteCarlo Class Reference

`#include <IntegradorMonteCarlo.h>`

### Public Types

- using FuncionUnivariable = std::function< double(double)>

### Public Member Functions

- IntegradorMonteCarlo (FuncionUnivariable func, double a, double b, unsigned int semilla=std::random_device{}())
- double CalcularIntegralSimple (long long numero_muestras, double &error_estimado)

### Private Attributes

- FuncionUnivariable func_a_integrar
- double limite_inferior
- double limite_superior
- std::mt19937 gen
- std::uniform_real_distribution distribucion_uniforme

### 4.1.1 Detailed Description

Definition at line 9 of file IntegradorMonteCarlo.h.

### 4.1.2 Member Typedef Documentation

#### 4.1.2.1 FuncionUnivariable

```
using IntegradorMonteCarlo::FuncionUnivariable = std::function<double(double)>
```

Definition at line 12 of file IntegradorMonteCarlo.h.

### 4.1.3 Constructor & Destructor Documentation

#### 4.1.3.1 IntegradorMonteCarlo()

```
IntegradorMonteCarlo::IntegradorMonteCarlo (
            FuncionUnivariable func,
            double a,
            double b,
            unsigned int semilla = std::random_device{}() )
```

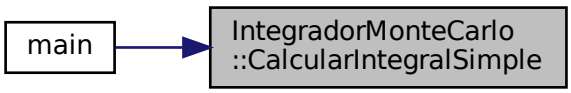
Definition at line 8 of file IntegradorMonteCarlo.cpp.

### 4.1.4 Member Function Documentation

#### 4.1.4.1 CalcularIntegralSimple()

```
double IntegradorMonteCarlo::CalcularIntegralSimple (
            long long numero_muestras,
            double & error_estimado )
```

Definition at line 16 of file IntegradorMonteCarlo.cpp.
Here is the caller graph for this function:




### 4.1.5 Member Data Documentation

#### 4.1.5.1 distribucion_uniforme

```
std::uniform_real_distribution IntegradorMonteCarlo::distribucion_uniforme  [private]
```

Definition at line 20 of file IntegradorMonteCarlo.h.

#### 4.1.5.2 func_a_integrar

```
FuncionUnivariable IntegradorMonteCarlo::func_a_integrar  [private]
```

Definition at line 15 of file IntegradorMonteCarlo.h.

#### 4.1.5.3 gen

```
std::mt19937 IntegradorMonteCarlo::gen  [private]
```

Definition at line 19 of file IntegradorMonteCarlo.h.

#### 4.1.5.4 limite_inferior

```
double IntegradorMonteCarlo::limite_inferior  [private]
```

Definition at line 16 of file IntegradorMonteCarlo.h.

#### 4.1.5.5 limite_superior

```
double IntegradorMonteCarlo::limite_superior  [private]
```

Definition at line 17 of file IntegradorMonteCarlo.h.

The documentation for this class was generated from the following files:

- IntegradorMonteCarlo.h
- IntegradorMonteCarlo.cpp

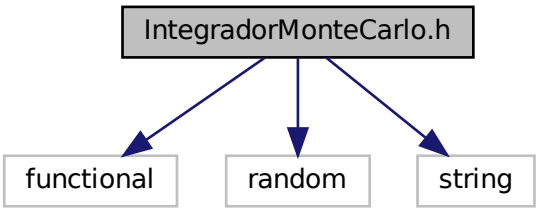
# Chapter 5

# File Documentation

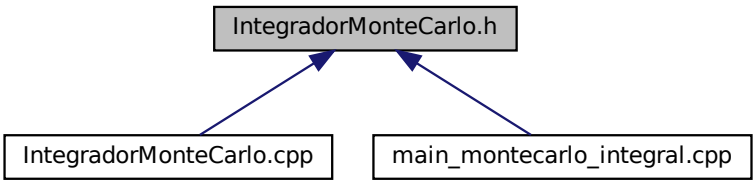
## 5.1 IntegradorMonteCarlo.h File Reference

```
#include <functional>
#include <random>
#include <string>
```

Include dependency graph for IntegradorMonteCarlo.h:




This graph shows which files directly or indirectly include this file:




### Classes

- class IntegradorMonteCarlo

## 5.2 IntegradorMonteCarlo.h

Go to the documentation of this file.

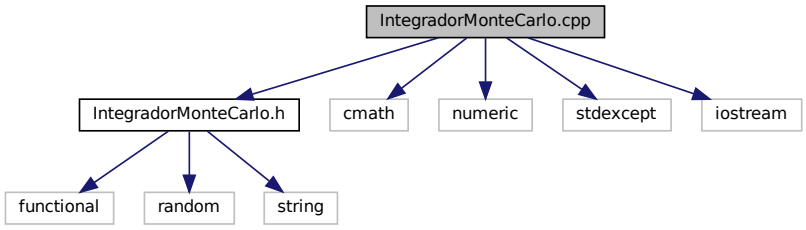
```
00001 // IntegradorMonteCarlo.h
00002 #ifndef INTEGRADOR_MONTE_CARLO_H
00003 #define INTEGRADOR_MONTE_CARLO_H
00004
00005 #include <functional> // Para std::function
00006 #include <random>     // Para generadores de números aleatorios
00007 #include <string>     // Para std::string (aunque no se usa directamente en la interfaz aquí)
00008
00009 class IntegradorMonteCarlo {
00010 public:
00011     // Typedef para la función a integrar (debe tomar un double y devolver un double)
00012     using FuncionUnivariable = std::function<double(double)>;
00013
00014 private:
00015     FuncionUnivariable func_a_integrar; // La función f(x)
00016     double limite_inferior;             // Límite inferior de integración 'a'
00017     double limite_superior;             // Límite superior de integración 'b'
00018
00019     std::mt19937 gen; // Generador de números aleatorios Mersenne Twister
00020     std::uniform_real_distribution<> distribucion_uniforme; // Para generar x en [a, b)
00021
00022 public:
00023     // Constructor
00024     // func:  la función f(x) a integrar.
00025     // a:  límite inferior de integración.
00026     // b:  límite superior de integración.
00027     // semilla:  semilla para el generador de números aleatorios.
00028     IntegradorMonteCarlo(FuncionUnivariable func, double a, double b, unsigned int semilla =
      std::random_device{}());
00029
00030     // Calcula la integral de func_a_integrar desde limite_inferior hasta limite_superior
00031     // utilizando el método de muestreo simple de Monte Carlo.
00032     // numero_muestras:  el número de puntos aleatorios (N) a generar.
00033     // error_estimado:  referencia a una variable donde se almacenará el error estándar de la media.
00034     // Devuelve el valor estimado de la integral.
00035     double CalcularIntegralSimple(long long numero_muestras, double& error_estimado);
00036
00037     // Podrían añadirse otros métodos de Monte Carlo aquí si fuera necesario,
00038     // como muestreo por importancia (importance sampling).
00039 };
00040
00041 #endif // INTEGRADOR_MONTE_CARLO_H
```

## 5.3 README.md File Reference

## 5.4 IntegradorMonteCarlo.cpp File Reference

```
#include "IntegradorMonteCarlo.h"
#include <cmath>
#include <numeric>
#include <stdexcept>
#include <iostream>
```

Include dependency graph for IntegradorMonteCarlo.cpp:



## 5.5 IntegradorMonteCarlo.cpp

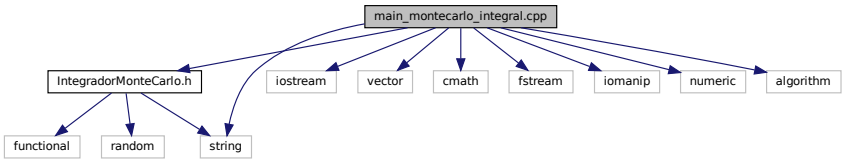
Go to the documentation of this file.

```
00001 // IntegradorMonteCarlo.cpp
00002 #include "IntegradorMonteCarlo.h" // Se espera que el Makefile configure la ruta de inclusión
00003 #include <cmath>     // Para std::sqrt, std::pow
00004 #include <numeric>   // Para std::accumulate (útil si se guardaran los f_xi en un vector)
00005 #include <stdexcept> // Para std::runtime_error
00006 #include <iostream>  // Para posibles mensajes de depuración o error
00007
00008 IntegradorMonteCarlo::IntegradorMonteCarlo(FuncionUnivariable func, double a, double b, unsigned int
      semilla)
00009     :  func_a_integrar(func), limite_inferior(a), limite_superior(b), gen(semilla),
      distribucion_uniforme(a, b) {
00010     if (a >= b) {
00011         // Lanzar una excepción o manejar el error de alguna manera
00012         throw std::runtime_error("Error en IntegradorMonteCarlo:  El límite inferior 'a' debe ser
      estrictamente menor que el límite superior 'b'.");
00013     }
00014 }
00015
00016 double IntegradorMonteCarlo::CalcularIntegralSimple(long long numero_muestras, double& error_estimado)
      {
00017     if (numero_muestras <= 0) {
00018         error_estimado = 0.0; // O un valor indicativo de error como NaN o infinito
00019         // std::cerr « "Advertencia:  numero_muestras debe ser positivo." « std::endl;
00020         return 0.0; // O NaN
00021     }
00022
00023     double suma_f = 0.0;
00024     double suma_f_cuadrado = 0.0;
00025
00026     for (long long i = 0; i < numero_muestras; ++i) {
00027         double x_aleatorio = distribucion_uniforme(gen); // Genera x_i uniformemente en [a, b)
00028         double valor_f_xi = func_a_integrar(x_aleatorio);
00029         suma_f += valor_f_xi;
00030         suma_f_cuadrado += valor_f_xi * valor_f_xi;
00031     }
00032
00033     // Estimación de la integral I = (b-a) * <f>
00034     // donde <f> es el promedio de f(x_i)
00035     double promedio_f = suma_f / static_cast<double>(numero_muestras);
00036     double valor_integral = (limite_superior - limite_inferior) * promedio_f;
00037
00038     // Estimación del error (error estándar de la media de la integral)
00039     // Error = (b-a) * sqrt( ( <f^2> - <f>^2 ) / N )
00040     //       = (b-a) * sigma_f / sqrt(N)
00041     if (numero_muestras > 1) {
00042         double promedio_f_cuadrado = suma_f_cuadrado / static_cast<double>(numero_muestras);
00043         double varianza_f = promedio_f_cuadrado - (promedio_f * promedio_f);
00044
00045         // La varianza no puede ser negativa, esto puede ocurrir por errores de precisión numérica.
00046         if (varianza_f < 0) {
00047             varianza_f = 0.0;
00048         }
00049         error_estimado = (limite_superior - limite_inferior) * std::sqrt(varianza_f /
      static_cast<double>(numero_muestras));
00050     } else {
00051         // No se puede estimar la varianza (y por lo tanto el error) con una sola muestra.
00052         // Se podría devolver NaN o un valor grande para indicar esto.
00053         error_estimado = 0.0; // O std::numeric_limits<double>::quiet_NaN(); si se incluye <limits>
00054     }
00055
00056     return valor_integral;
00057 }
```

## 5.6 main_montecarlo_integral.cpp File Reference

```
#include "IntegradorMonteCarlo.h"
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <iomanip>
#include <string>
#include <numeric>
#include <algorithm>
```

Include dependency graph for main_montecarlo_integral.cpp:




## Macros

- #define M_PI 3.14159265358979323846

## Functions

- double funcion_a_integrar_exp_neg_x_cuadrado (double x)
- int main (int argc, char ∗argv[ ])

### 5.6.1 Macro Definition Documentation

#### 5.6.1.1 M_PI

```
#define M_PI 3.14159265358979323846
```

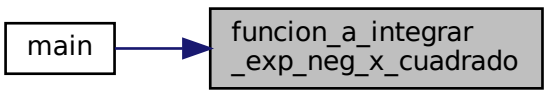
Definition at line 15 of file main_montecarlo_integral.cpp.

### 5.6.2 Function Documentation

#### 5.6.2.1 funcion_a_integrar_exp_neg_x_cuadrado()

```
double funcion_a_integrar_exp_neg_x_cuadrado (
            double x )
```

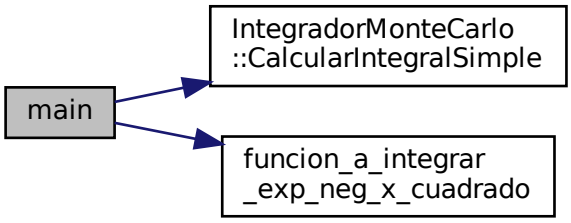
Definition at line 19 of file main_montecarlo_integral.cpp.
Here is the caller graph for this function:




#### 5.6.2.2 main()

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

Definition at line 23 of file main_montecarlo_integral.cpp.
Here is the call graph for this function:




## 5.7 main_montecarlo_integral.cpp

Go to the documentation of this file.

```
00001 // main_montecarlo_integral.cpp
00002 // Programa principal para la Parte B2:  Integración de e^(-x^2) por Monte Carlo
00003 #include "IntegradorMonteCarlo.h" // Se espera que el Makefile configure la ruta
00004 #include <iostream>
00005 #include <vector>
00006 #include <cmath>     // Para std::exp, std::pow, M_PI (puede necesitar -D_USE_MATH_DEFINES en Windows)
00007 #include <fstream>   // Para std::ofstream
00008 #include <iomanip>   // Para std::setprecision
00009 #include <string>    // Para std::string
00010 #include <numeric>   // Para std::accumulate (si se necesita para algo más)
00011 #include <algorithm> // Para std::sort (si se usa para ordenar lista_n_muestras)
00012
00013 // Definir M_PI si no está disponible (común en MSVC sin _USE_MATH_DEFINES)
00014 #ifndef M_PI
00015 #define M_PI 3.14159265358979323846
00016 #endif
00017
00018 // Función a integrar:  f(x) = e^(-x^2)
00019 double funcion_a_integrar_exp_neg_x_cuadrado(double x) {
00020     return std::exp(-x * x);
00021 }
00022
00023 int main(int argc, char *argv[]) {
00024     double limite_inferior = 0.0;
00025     double limite_superior = 1.0;
00026     unsigned int semilla = std::random_device{}(); // Semilla aleatoria por defecto
00027
00028     // Permitir cambiar la semilla desde la línea de comandos (opcional)
00029     if (argc > 1) {
00030         try {
00031             semilla = std::stoul(argv[1]);
00032         } catch (const std::exception& e) {
00033             std::cerr << "Advertencia:  No se pudo parsear la semilla '" << argv[1] << "'.  Usando
      semilla aleatoria por defecto." << std::endl;
00034         }
00035     }
00036
00037     std::cout << "Problema B2.4:  Cálculo de la integral de e^(-x^2) de "
00038               << limite_inferior << " a " << limite_superior
00039               << " usando Monte Carlo." << std::endl;
00040     std::cout << "  Usando semilla:  " << semilla << std::endl;
00041
00042     IntegradorMonteCarlo integrador(funcion_a_integrar_exp_neg_x_cuadrado, limite_inferior,
      limite_superior, semilla);
00043
00044     std::string nombre_archivo_salida = "results/integral_error_Nmax_1e7.dat";
00045     std::ofstream archivo_resultados(nombre_archivo_salida);
00046
00047     if (!archivo_resultados.is_open()) {
00048         std::cerr << "Error fatal:  No se pudo abrir el archivo de salida:  " << nombre_archivo_salida <<
      std::endl;
00049         // Intentar en el directorio actual como fallback para pruebas locales.
00050         nombre_archivo_salida = "integral_error_Nmax_1e7.dat";
00051         archivo_resultados.open(nombre_archivo_salida);
00052         if (!archivo_resultados.is_open()) {
```

```
00053             std::cerr « "Error fatal:  No se pudo abrir el archivo de salida en la raíz tampoco:  " «
      nombre_archivo_salida « std::endl;
00054             return 1;
00055         }
00056         std::cout « "Advertencia:  Guardando resultados en " « nombre_archivo_salida « " (directorio
      actual)." « std::endl;
00057     }
00058
00059     archivo_resultados « "# N_muestras Valor_Integral_Estimado Error_Estimado Valor_Teorico
      Diferencia_Absoluta" « std::endl;
00060
00061     // Configurar precisión para la salida
00062     std::cout « std::fixed « std::setprecision(8);
00063     archivo_resultados « std::fixed « std::setprecision(8);
00064
00065     // Valor de referencia (Analítico/Tabla) para la integral de 0 a 1 de e^(-x^2) dx = (sqrt(pi)/2) *
      erf(1)
00066     double valor_teorico = (std::sqrt(M_PI) / 2.0) * std::erf(1.0); // erf() está en <cmath> desde
      C++17
00067                                                                     // Si erf no está disponible, usar
      valor precalculado.
00068     // double valor_teorico_precalculado = 0.7468241328124271; // Usar si erf(1) no compila
00069
00070     std::cout « "Valor de referencia (Teórico) para la integral:  " « valor_teorico « std::endl «
      std::endl;
00071     std::cout « "N_muestras | Valor Estimado | Error Estimado | Diferencia Abs." « std::endl;
00072     std::cout « "-----------------------------------------------------------------" « std::endl;
00073
00074     std::vector<long long> lista_n_muestras;
00075     for (int i = 1; i <= 7; ++i) { // Desde 10^1 (10) hasta 10^7 (10,000,000)
00076         lista_n_muestras.push_back(static_cast<long long>(std::pow(10, i)));
00077     }
00078     // Podrían añadirse más puntos para una gráfica más suave si se desea:
00079     // long long N_actual = 10;
00080     // while (N_actual <= 10000000) {
00081     //     lista_n_muestras.push_back(N_actual);
00082     //     if (N_actual < 100) N_actual += 10;
00083     //     else if (N_actual < 1000) N_actual += 100;
00084     //     else if (N_actual < 100000) N_actual += 1000;
00085     //     else N_actual *= 2; // o N_actual *= 10 para menos puntos
00086     // }
00087
00088
00089     for (long long n_muestras :  lista_n_muestras) {
00090         double error_calc = 0.0;
00091         double valor_integral_calc = integrador.CalcularIntegralSimple(n_muestras, error_calc);
00092         double diferencia_abs = std::abs(valor_integral_calc - valor_teorico);
00093
00094         std::cout « std::setw(10) « n_muestras « " | "
00095                   « std::setw(14) « valor_integral_calc « " | "
00096                   « std::setw(14) « error_calc « " | "
00097                   « std::setw(14) « diferencia_abs « std::endl;
00098
00099         archivo_resultados « n_muestras « " "
00100                            « valor_integral_calc « " "
00101                            « error_calc « " "
00102                            « valor_teorico « " "
00103                            « diferencia_abs « std::endl;
00104     }
00105
00106     archivo_resultados.close();
00107     std::cout « "\nResultados de la convergencia guardados en:  " « nombre_archivo_salida « std::endl;
00108
00109     std::cout « "\n--- Fin del Problema B2.4 (Integración de e^-x^2) ---" « std::endl;
00110     std::cout « "El documento 'montecarlo_fisica_estadistica.tex' contendrá la discusión teórica y el
      análisis de estos resultados." « std::endl;
00111
00112     return 0;
00113 }
```