INFORME LAB 07 – FC

Link Codigo:
https://colab.research.google.com/drive/1Lu0y9QQFCkYbwVs_X2gGpkCCPxLQeKea?usp=sharing

Introducción

El presente informe describe la implementación y aplicación del método de integración de Monte Carlo para aproximar valores de integrales definidas, incluyendo funciones que presentan desafíos computacionales por su complejidad analítica. Este método es ampliamente utilizado en simulaciones numéricas y proporciona una herramienta útil para resolver problemas donde los métodos tradicionales son difíciles de aplicar. Además, se realiza una visualización gráfica del proceso, mostrando la distribución de los puntos generados y su relación con la función a integrar.

Marco Teórico

Método de Integración de Monte Carlo:
El método de Monte Carlo utiliza la generación de números aleatorios para aproximar soluciones a problemas matemáticos y físicos. En el caso de la integración, el método evalúa la función en puntos generados aleatoriamente dentro de un dominio, y la aproximación se obtiene como el promedio de los valores evaluados multiplicado por la longitud del intervalo de integración.

Visualización del método:
El proceso de Monte Carlo puede representarse gráficamente para ilustrar cómo los puntos generados aleatoriamente se distribuyen bajo y sobre la curva de la función. Esta representación permite validar visualmente la correspondencia entre los puntos bajo la curva y la integral calculada.

Ventajas:

- Maneja eficientemente funciones complejas o dominios de integración multidimensionales.
- Permite una representación gráfica intuitiva del proceso de integración.

Limitaciones:

- Depende del número de puntos generados, lo que afecta la precisión.
- Puede ser computacionalmente costoso para problemas en dimensiones altas.

Desarrollo

A continuación, se presentan las partes más relevantes del código implementado y su explicación:

1. **Importación de librerías:**
   Se utilizan librerías especializadas como numpy para cálculos numéricos, random para generación de puntos aleatorios, seaborn y matplotlib para visualización.

```python
import random
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

2. **Definición del método de Monte Carlo:**
   Este bloque calcula la integral aproximada evaluando puntos aleatorios en el intervalo [a,b][a, b][a,b]. La función es evaluada en los puntos generados, y su promedio se multiplica por la longitud del intervalo.

```python
# Función para realizar la integración de Monte Carlo
def monte_carlo_integration(func, a, b, num_points=10000):
    # Generar puntos aleatorios
    x_random = np.random.uniform(a, b, num_points)
    # Evaluar la función en los puntos aleatorios
    y_values = func(x_random)
    # Calcular la integral aproximada
    integral = (b - a) * np.mean(y_values)
    return integral
```

3. **Visualización del método:**
   Este código incluye gráficos que muestran cómo se generan y clasifican los puntos aleatorios en el intervalo. Los puntos debajo de la curva (en verde) representan las contribuciones a la integral, mientras que los puntos por encima de la curva (en rojo) no aportan al cálculo.

```python
def monte_carlo_integration_with_points(func, a, b, num_points=5000):
    # Generar puntos aleatorios
    x_random = np.random.uniform(a, b, num_points)
    y_max = max(func(x) for x in np.linspace(a, b, 1000))
    y_random = np.random.uniform(0, y_max, num_points)

    # Identificar puntos bajo y sobre la curva
    under_curve = y_random <= func(x_random)
    over_curve = ~under_curve

    # Calcular el área bajo la curva
    area_rectangle = (b - a) * y_max
    integral = (under_curve.sum() / num_points) * area_rectangle

    # Graficar
    sns.set(style="whitegrid")
    plt.figure(figsize=(10, 6))

    # Curva de la función
    x = np.linspace(a, b, 1000)
    y = func(x)
    sns.lineplot(x=x, y=y, label='Función', color='blue', linewidth=2.5)

    # Puntos bajo la curva
    sns.scatterplot(x=x_random[under_curve], y=y_random[under_curve],
                    color='green', s=5, label='Puntos bajo la curva', alpha=0.6)

    # Puntos sobre la curva
    sns.scatterplot(x=x_random[over_curve], y=y_random[over_curve],
                    color='red', s=5, label='Puntos sobre la curva', alpha=0.6)

    # Configuración del gráfico
    plt.title('Integración Monte Carlo', fontsize=14)
    plt.xlabel('x', fontsize=12)
    plt.ylabel('f(x)', fontsize=12)
    plt.legend(fontsize=10)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

    return integral
```

4. **Resultados generados:**
   Se definieron varias funciones y sus límites de integración, representadas mediante diccionarios, y se calcularon las aproximaciones numéricas.

```python
# Funciones definidas en la imagen
functions = {
    "Integral 1": lambda x: np.exp(x**2),
    "Integral 2": lambda x: np.exp(x) * 4,
    "Integral 3": lambda x: (1 - np.exp(x**2))**(1/2),
    "Integral 4": lambda x: x * (1 + x**2)**(-2),
    "Integral 5": lambda x: np.exp(x + x**2),
    "Integral 6": lambda x: np.exp(-x),
    "Integral 7": lambda x: (1 - x**2)**(3/2)
}

# Límites de integración para cada caso
limits = {
    "Integral 1": (0, 1),
    "Integral 2": (-1, 1),
    "Integral 3": (0, 1),
    "Integral 4": (0, 10),  # Para infinito, tomaremos un límite razonable
    "Integral 5": (0, 1),
    "Integral 6": (0, 10),  # Para infinito, tomaremos un límite razonable
    "Integral 7": (0, 1)
}

# Resultados de las integrales
results = {}
```

Resultados

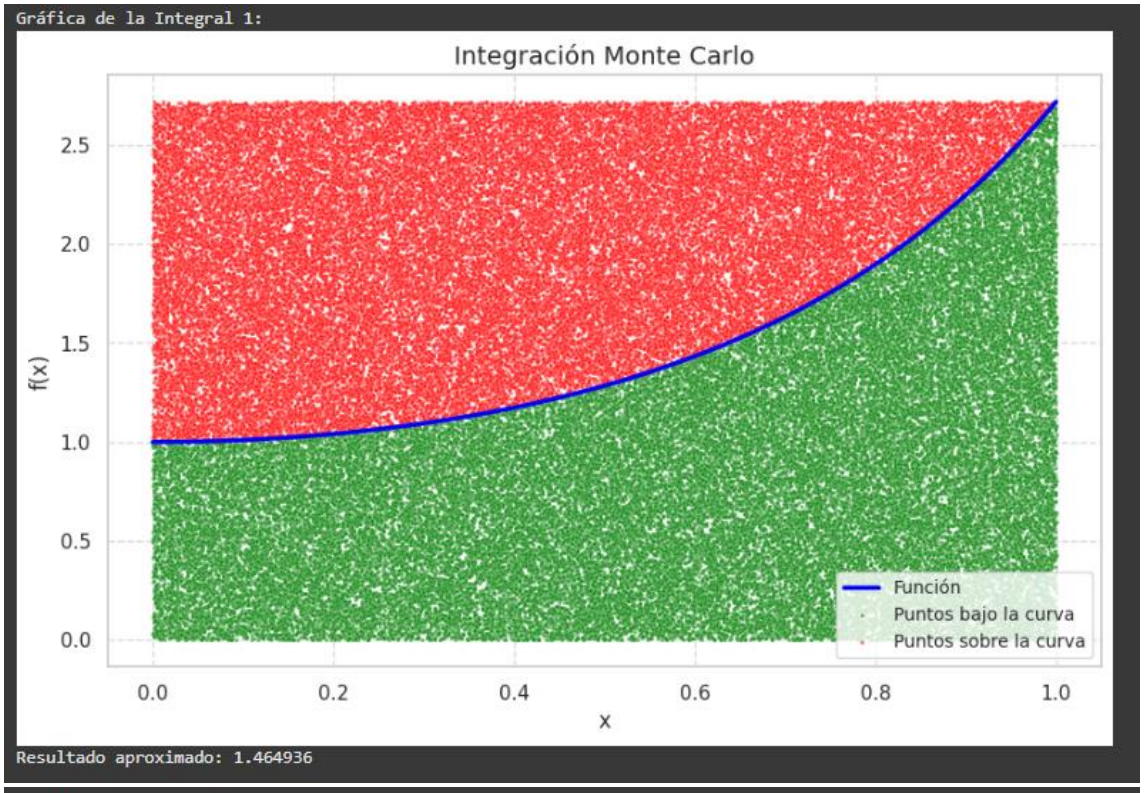
Aproximaciones calculadas:
Se presenta una tabla que resume los resultados obtenidos:

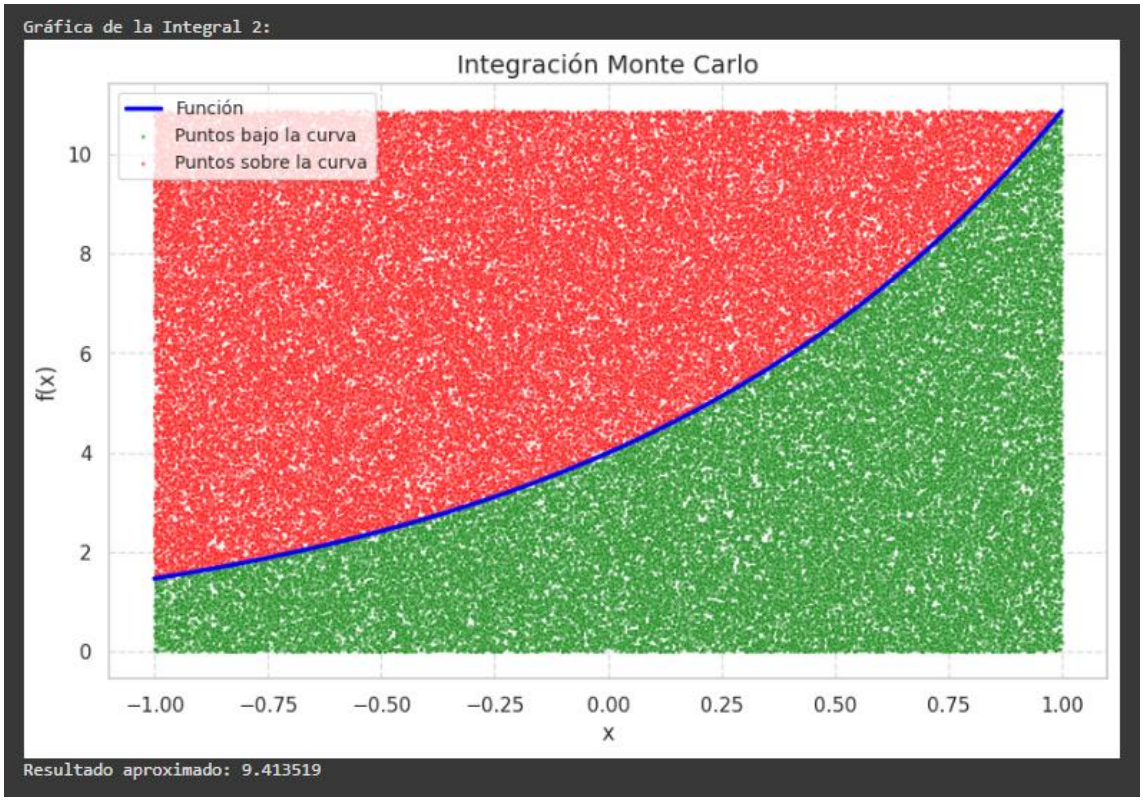
Integral	Aproximación
$\int_0^1 e^{x^2}\,dx$	1.461585
$\int_{-1}^1 e^x 4\,dx$	9.421685
$\int_0^1 \sqrt{1-e^{x^2}}\,dx$	NaN
$\int_0^\infty x(1+x^2)^{-2}\,dx$	0.494494
$\int_0^1 e^{x+x^2}\,dx$	2.742349
$\int_0^\infty e^{-x}\,dx$	1.001125
$\int_0^\infty (1-x^2)^{3/2}\,dx$	0.588601

Visualización gráfica:
Los gráficos muestran:

- **La curva de la función:** Representada por una línea azul.
- **Puntos aleatorios bajo la curva:** En verde, representando las contribuciones a la integral.
- **Puntos aleatorios sobre la curva:** En rojo, que no afectan el cálculo.

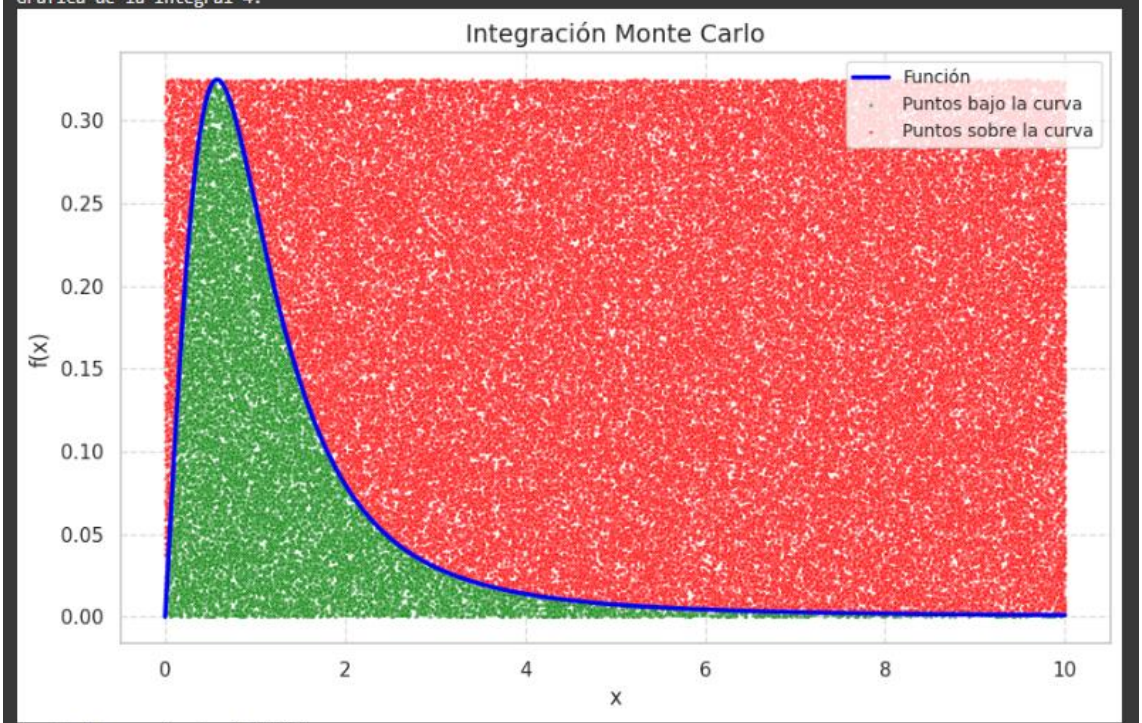
Esta visualización permite identificar cómo se distribuyen los puntos y cómo contribuyen al área bajo la curva.





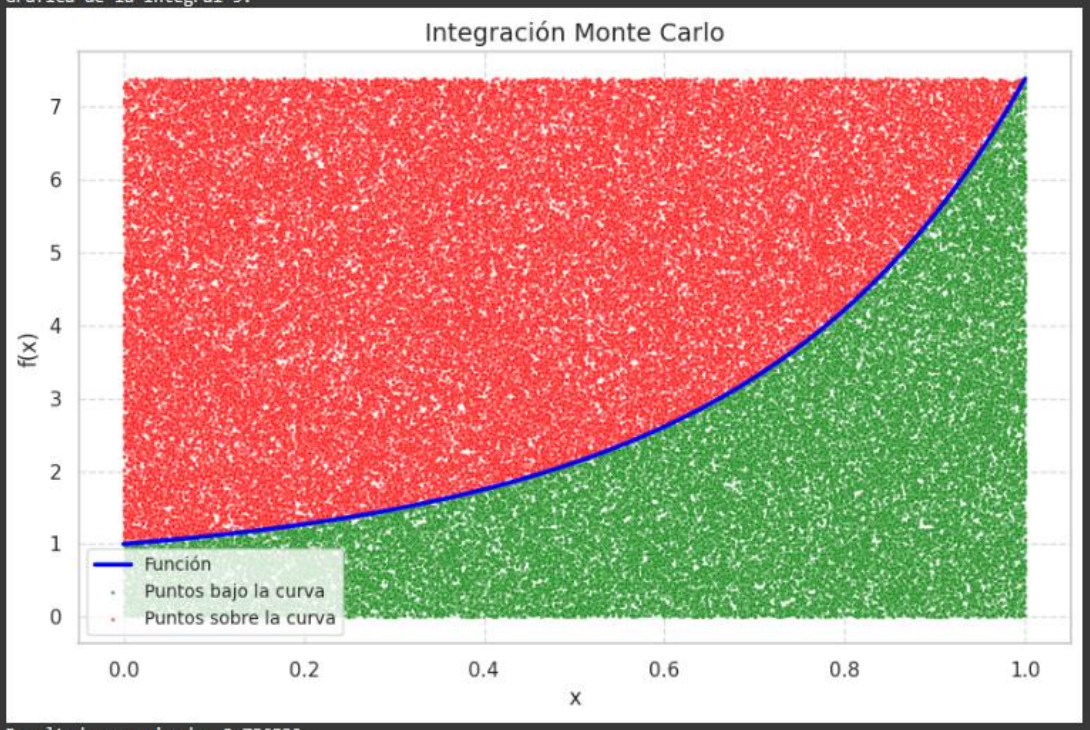
```
Gráfica de la Integral 3:
<ipython-input-30-a1d45a290cb2>:17: RuntimeWarning: invalid value encountered in scalar power
  "Integral 3": lambda x: (1 - np.exp(x**2))**(1/2),
<ipython-input-30-a1d45a290cb2>:17: RuntimeWarning: invalid value encountered in sqrt
  "Integral 3": lambda x: (1 - np.exp(x**2))**(1/2),
```


```
Resultado aproximado: 0.000000
Gráfica de la Integral 4:
```



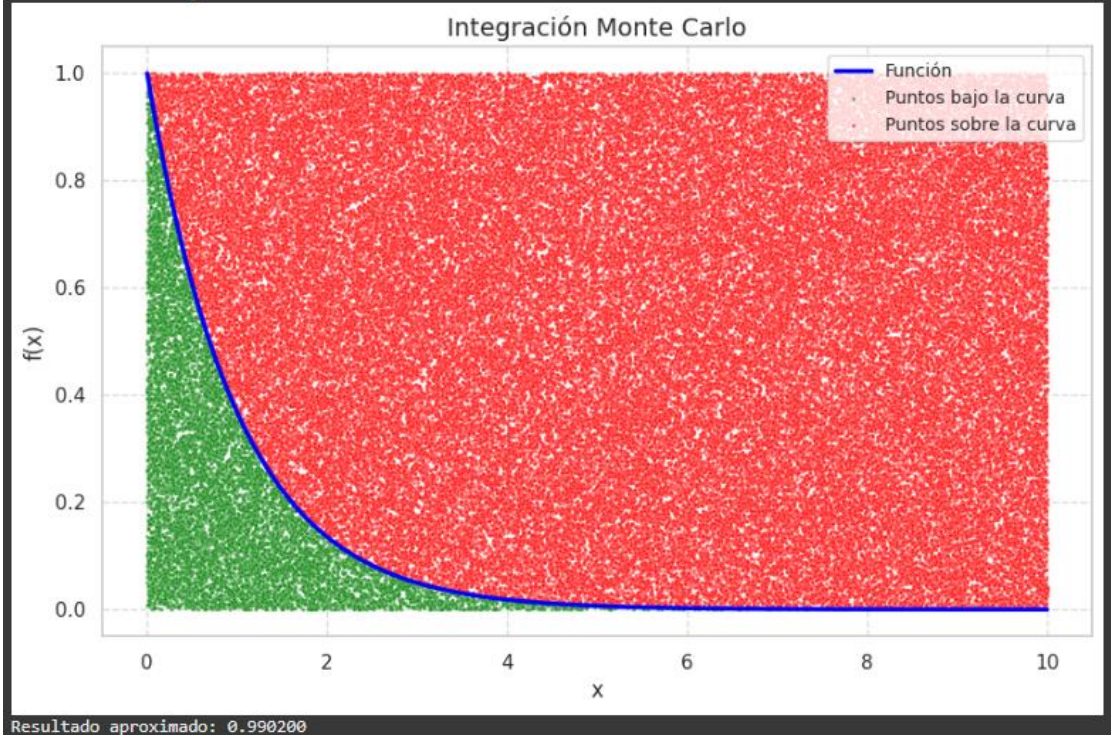

```
Resultado aproximado: 0.499371
```

Gráfica de la Integral 5:

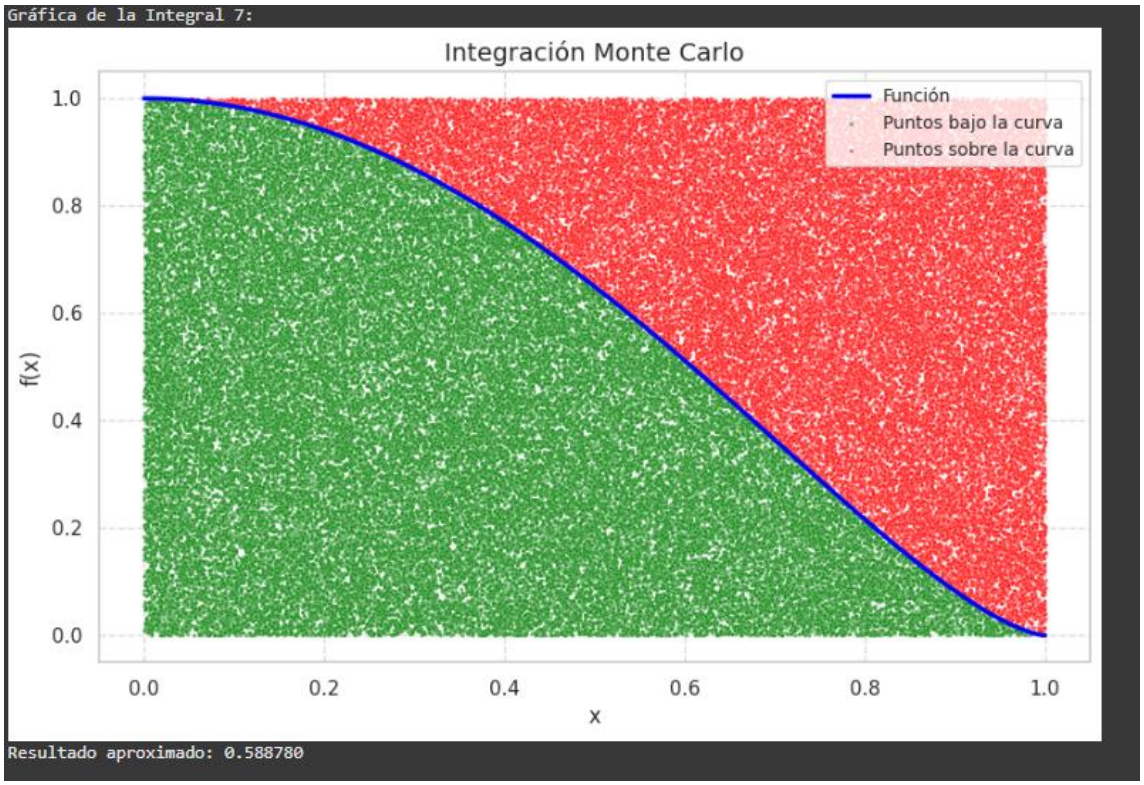


Resultado aproximado: 2.736832

Gráfica de la Integral 6:



Resultado aproximado: 0.990200



Conclusiones

- El método de Monte Carlo es efectivo para aproximar integrales de funciones complejas, y su precisión aumenta con el número de puntos generados.
- La representación gráfica complementa el entendimiento del método, mostrando cómo los puntos contribuyen a la aproximación de la integral.
- Aunque algunas funciones con límites infinitos pueden ser difíciles de aproximar, ajustar los límites a valores finitos razonables permite obtener resultados consistentes.