# Tarea2 SSF

Luis Gerardo Hernández Román

Septiembre 2024

## 1 Integrar numéricamente

$$\int_{-1}^{1} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx \tag{1}$$

### 1.1 Regla del trapecio

```
# Primera integral con 6 subintervalos
from numpy import *
import numpy as np

def func(x):
    return (e**((-x**2)/2))/np.sqrt(2*pi)
def trapezoid(A, B, N):
    h=(B-A)/(N-1)
    sum = (func(A)+ func(B))/2
    for i in range(1, N-1):
        sum += func(A+i*h)
    return h* sum

# Se define los limites de integracion y el numero de subintervalos
A=-1
B=1
N=7 # Se necesita colocar N+1 subinetervalos para tener 6
print(trapezoid(A,B, N-1))
```

#### 1.1.1 6 subintervalos

Resultado de la integración; 0.6762021273014618

#### 1.1.2 15 subintervalos

Resultado de la integración; 0.6818659017601713

## 1.2 Simpson

```
from numpy import *
import numpy as np

# Definición de la función a integrar
def f(x):
    return (exp((-x**2)/2)/np.sqrt(2*pi))  # Ejemplo de función

# Implementación del método de Simpson
def simpson(f, a, b, n):
    """
    Calcular la integral de la función f en el intervalo [a, b] usando el método de Simpson.

    :param f: Función a integrar
    :param a: Límite inferior de integración
    :param b: Límite superior de integración
    :param n: Número de subintervalos (debe ser par)
    :return: Aproximación de la integral de f desde a hasta b
    """
    # Verificar que n es par
    if n % 2 == 1:
        raise ValueError("El número de subintervalos 'n' debe ser par.")

    h = (b - a) / n  # Ancho de cada subintervalo
    x = np.linspace(a, b, n + 1)  # Puntos de evaluación
    fx = f(x)  # Evaluación de la función en los puntos

    # Aplicación de la fórmula de Simpson
    integral = fx[0] + fx[-1] + 4 * np.sum(fx[1:-1:2]) + 2 * np.sum(fx[2:-2:2])
    integral *= h / 3

    return integral

# Definición de los límites de integración y el número de subintervalos
a = -1.0  # Límite inferior
b = 1.0  # Límite superior
n = 6   # Número de subintervalos (debe ser par)

# Cálculo de la integral usando el método de Simpson
result = simpson(f, a, b, n)
print("Resultado de la integral con el método de Simpson =", result)
```

### 1.2.1 6 particiones

Resultado de la integral con el método de Simpson = 0.6827586139549787

### 1.2.2 15 particiones

Resultado de la integral con el método de Simpson = 0.6826908122781858

# 2 Gauss-Legendre

```
    from numpy import *
import numpy as np
from sys import version
import math
from scipy.integrate import quad
# Definición de parámetros iniciales
max_in = 6  # Número máximo de intervalos para la integración
vmin = -1.0  # Valor mínimo del rango de integración
vmax = 1.0  # Valor máximo del rango de integración

# Inicialización de arrays para almacenar los puntos de integración (x) y los pesos (w)
w = zeros((2001), float)  # Array de pesos de la cuadratura de Gauss
x = zeros((2001), float)  # Array de puntos de la cuadratura de Gauss

# Definición de la función a integrar: f(x) = e^(-x)
def f(x):
    return (exp((-x**2)/2)/np.sqrt(2*pi))

# Función para calcular los puntos y pesos de la cuadratura de Gauss
def gauss(npts, job, a, b, x, w):
    m = i = j = t = t1 = pp = p1 = p2 = p3 = 0.0
    eps = 3E-14  # Precisión deseada para el cálculo de los puntos y pesos
    m = int((npts + 1) / 2)  # Número de puntos necesarios

    # Bucle para calcular los puntos y pesos usando la fórmula de Gauss-Legendre
    for i in range(1, m + 1):
        # Estimación inicial del punto usando la aproximación coseno
        t = cos(math.pi * (float(i) - 0.25) / (float(npts) + 0.5))
        t1 = 1.0  # Inicialización para el bucle iterativo

        # Bucle iterativo para refinar la estimación de los puntos
        while (abs(t - t1) >= eps):
            p1 = 1.0  # Inicialización del polinomio de Legendre
            p2 = 0.0
            for j in range(1, npts + 1):
                p3 = p2  # Cambio de valores anteriores
                p2 = p1
                # Calculo del valor actual del polinomio de Legendre
                p1 = ((2.0 * float(j) - 1) * t * p2 - (float(j) - 1.0) * p3) / float(j)
```

```
            # Derivada del polinomio de Legendre en el punto t
            pp = npts * (t * p1 - p2) / (t * t - 1.0)
            t1 = t  # Almacena el valor previo
            t = t1 - p1 / pp  # Nuevo valor de t usando el método de Newton-Raphson

        # Asignación de los puntos de integración y sus simétricos
        x[i - 1] = -t
        x[npts - i] = t

        # Cálculo de los pesos asociados a los puntos
        w[i - 1] = 2.0 / ((1.0 - t * t) * pp * pp)
        w[npts - i] = w[i - 1]

    # Ajuste de los puntos y pesos según el tipo de intervalo (job)
    if (job == 0):  # Intervalo estándar (a, b)
        for i in range(0, npts):
            x[i] = x[i] * (b - a) / 2.0 + (b + a) / 2.0  # Escalamiento de los puntos
            w[i] = w[i] * (b - a) / 2.0  # Escalamiento de los pesos
    elif (job == 1):  # Intervalo (a, inf)
        for i in range(0, npts):
            xi = x[i]
            x[i] = a * b * (1.0 + xi) / (b + a - (b - a) * xi)
            w[i] = w[i] * 2.0 * a * b * b / ((b + a - (b - a) * xi) ** 2)
    elif (job == 2):  # Intervalo (-inf, inf)
        for i in range(0, npts):
            xi = x[i]
            x[i] = (b * xi + b + a + a) / (1.0 - xi)
            w[i] = w[i] * 2.0 * (a + b) / ((1.0 - xi) ** 2)

# Función para realizar la integración usando los puntos y pesos calculados
def gaussint(no, min, max):
    quadra = 0.0  # Inicialización del valor de la integral
    gauss(no, 0, min, max, x, w)  # Llama a la función gauss para obtener los puntos y peso

    # Calcula la integral sumando la función evaluada en los puntos por sus respectivos pes
    for n in range(0, no):
        quadra += f(x[n]) * w[n]
    return quadra  # Devuelve el valor calculado de la integral

# Realiza la integral
result = gaussint(max_in, vmin, vmax)  # Calcula la integral con i puntos
print("Resultado de la integral=", result)  # Mensaje final al usuario

## Calcula el valor de referencia usando la función de integración numérica de scipy
ref_value, _ = quad(f, vmin, vmax)
```

```
print("Valor de referencia de la integral con scipy =", ref_value)

# Calcula el error absoluto
error_abs = abs(result - ref_value)
print("Error absoluto =", error_abs)

# Calcula el error relativo
error_rel = error_abs / abs(ref_value)
print("Error relativo =", error_rel)
```

### 2.1 6 puntos

Resultado de la integral= 0.6826894870533804
Valor de referencia de la integral con scipy = 0.682689492137086
Error absoluto = 5.083705567621166e-09
Error relativo = 7.446585345421347e-09

#### 2.1.1 15 puntos

Resultado de la integral= 0.6826894921370831
Valor de referencia de la integral con scipy = 0.682689492137086
Error absoluto = 2.886579864025407e-15
Error relativo = 4.228247098090347e-15

## 3 Integrar numéricamente

$$\int_0^3 \frac{e^x \sin x}{(1+x^2)} dx \tag{2}$$

### 3.1 usando regla del trapecio

```
def func(x):
    return (e**(x)*sin(x))/(1+e**2)
def trapezoid(A, B, N):
    h=(B-A)/(N-1)
    sum = (func(A)+ func(B))/2
    for i in range(1, N-1):
        sum += func(A+i*h)
    return h* sum

# Se define los limites de integracion y el numero de subintervalos
A=-1
B=1
N=7 # Se necesita colocar N+1 subinetervalos para tener 6
print(trapezoid(A,B, N-1))
```

### 3.1.1 6 subintervalos

Resultado de la integración; 0.08524723492600471

### 3.1.2 15 subintervalos

Resultado de la integración; 0.07987447612674553

### 3.1.3 20 Subintervalos

Resultado de la integración; 0.07951605076501572

## 3.2 Simpson

```
     import numpy as np

# Definición de la función a integrar
def f(x):
    return (exp(x) * sin(x)) / (1 + x**2)  # Ejemplo de función

# Implementación del método de Simpson
def simpson(f, a, b, n):
    """
    Calcula la integral de la función f en el intervalo [a, b] usando el método de Simpson.

    :param f: Función a integrar
    :param a: Límite inferior de integración
    :param b: Límite superior de integración
    :param n: Número de subintervalos (debe ser par)
    :return: Aproximación de la integral de f desde a hasta b
    """
    # Verificar que n es par
    if n % 2 == 1:
        raise ValueError("El número de subintervalos 'n' debe ser par.")

    h = (b - a) / n  # Ancho de cada subintervalo
    x = np.linspace(a, b, n + 1)  # Puntos de evaluación
    fx = f(x)  # Evaluación de la función en los puntos

    # Aplicación de la fórmula de Simpson
    integral = fx[0] + fx[-1] + 4 * np.sum(fx[1:-1:2]) + 2 * np.sum(fx[2:-2:2])
    integral *= h / 3

    return integral

# Definición de los límites de integración y el número de subintervalos
a = 0.0  # Límite inferior
```

```
b = 3.0  # Límite superior
n = 6   # Número de subintervalos (debe ser par)

# Cálculo de la integral usando el método de Simpson
result = simpson(f, a, b, n)
print("Resultado de la integral con el método de Simpson =", result)
```

#### 3.2.1 6 particiones

Resultado de la integral con el método de Simpson = 2.8854018637270924

#### 3.2.2 15 particiones

Resultado de la integral con el método de Simpson = 2.8816659631412325

#### 3.2.3 20 particiones

Resultado de la integral con el método de Simpson = 2.881648560171607

### 3.3 Gauss-Legendre

```
    from numpy import *
from sys import version
import math
from scipy.integrate import quad
# Definición de parámetros iniciales
max_in = 6  # Número máximo de intervalos para la integración
vmin = 0.0  # Valor mínimo del rango de integración
vmax = 3.0  # Valor máximo del rango de integración

# Inicialización de arrays para almacenar los puntos de integración (x) y los pesos (w)
w = zeros((2001), float)  # Array de pesos de la cuadratura de Gauss
x = zeros((2001), float)  # Array de puntos de la cuadratura de Gauss

# Definición de la función a integrar: f(x) = e^(-x)
def f(x):
    return (exp(x)*sin(x))/(1+x**2)

# Función para calcular los puntos y pesos de la cuadratura de Gauss
def gauss(npts, job, a, b, x, w):
    m = i = j = t = t1 = pp = p1 = p2 = p3 = 0.0
    eps = 3E-14  # Precisión deseada para el cálculo de los puntos y pesos
    m = int((npts + 1) / 2)  # Número de puntos necesarios

    # Bucle para calcular los puntos y pesos usando la fórmula de Gauss-Legendre
```

```
    for i in range(1, m + 1):
        # Estimación inicial del punto usando la aproximación coseno
        t = cos(math.pi * (float(i) - 0.25) / (float(npts) + 0.5))
        t1 = 1.0  # Inicialización para el bucle iterativo

        # Bucle iterativo para refinar la estimación de los puntos
        while (abs(t - t1) >= eps):
            p1 = 1.0  # Inicialización del polinomio de Legendre
            p2 = 0.0
            for j in range(1, npts + 1):
                p3 = p2  # Cambio de valores anteriores
                p2 = p1
                # Calculo del valor actual del polinomio de Legendre
                p1 = ((2.0 * float(j) - 1) * t * p2 - (float(j) - 1.0) * p3) / float(j)

            # Derivada del polinomio de Legendre en el punto t
            pp = npts * (t * p1 - p2) / (t * t - 1.0)
            t1 = t  # Almacena el valor previo
            t = t1 - p1 / pp  # Nuevo valor de t usando el método de Newton-Raphson

        # Asignación de los puntos de integración y sus simétricos
        x[i - 1] = -t
        x[npts - i] = t

        # Cálculo de los pesos asociados a los puntos
        w[i - 1] = 2.0 / ((1.0 - t * t) * pp * pp)
        w[npts - i] = w[i - 1]

    # Ajuste de los puntos y pesos según el tipo de intervalo (job)
    if (job == 0):  # Intervalo estándar (a, b)
        for i in range(0, npts):
            x[i] = x[i] * (b - a) / 2.0 + (b + a) / 2.0  # Escalamiento de los puntos
            w[i] = w[i] * (b - a) / 2.0  # Escalamiento de los pesos
    elif (job == 1):  # Intervalo (a, inf)
        for i in range(0, npts):
            xi = x[i]
            x[i] = a * b * (1.0 + xi) / (b + a - (b - a) * xi)
            w[i] = w[i] * 2.0 * a * b * b / ((b + a - (b - a) * xi) ** 2)
    elif (job == 2):  # Intervalo (-inf, inf)
        for i in range(0, npts):
            xi = x[i]
            x[i] = (b * xi + b + a + a) / (1.0 - xi)
            w[i] = w[i] * 2.0 * (a + b) / ((1.0 - xi) ** 2)


# Función para realizar la integración usando los puntos y pesos calculados
```

```
def gaussint(no, min, max):
    quadra = 0.0  # Inicialización del valor de la integral
    gauss(no, 0, min, max, x, w)  # Llama a la función gauss para obtener los puntos y peso

    # Calcula la integral sumando la función evaluada en los puntos por sus respectivos pes
    for n in range(0, no):
        quadra += f(x[n]) * w[n]
    return quadra  # Devuelve el valor calculado de la integral

# Realiza la integral
result = gaussint(max_in, vmin, vmax)  # Calcula la integral con i puntos
print("Resultado de la integral=", result)  # Mensaje final al usuario

## Calcula el valor de referencia usando la función de integración numérica de scipy
ref_value, _ = quad(f, vmin, vmax)
print("Valor de referencia de la integral con scipy =", ref_value)

# Calcula el error absoluto
error_abs = abs(result - ref_value)
print("Error absoluto =", error_abs)

# Calcula el error relativo
error_rel = error_abs / abs(ref_value)
print("Error relativo =", error_rel)
```

### 3.3.1 6 puntos

Resultado de la integral= 2.8816429377925967
Valor de referencia de la integral con scipy = 2.881637273055187
Error absoluto = 5.664737409816212e-06
Error relativo = 1.965805156250741e-06

### 3.3.2 15 puntos

Resultado de la integral= 2.881637273055171
Valor de referencia de la integral con scipy = 2.881637273055187
Error absoluto = 1.5987211554602254e-14
Error relativo = 5.547961120607035e-15

# 4 Integrar 2 ahora con límites en (-2, 0)

$$\int_{-2}^{0} \frac{e^x \sin x}{(1+x^2)} dx \tag{3}$$

### 4.1 Gauss Legendre

Resultado de la integral= 0.4772498718592582
Valor de referencia de la integral con scipy = 0.47724986805182085
Error absoluto = 3.807437365388466e-09
Error relativo = 7.977869917347041e-09

## 5 Problema 4

Integrar numéricamente

$$f(x) = e^1 \tag{4}$$

$$g(x) = \frac{1 - e^x}{x} \tag{5}$$

en el intervalo (0,5) Elijan el método y número de particiones.

### 5.1 Método de Gauss Legendre

#### 5.1.1

$$\int_5^5 e^{x/x} dx \tag{6}$$

Resultado de la integral= 8.154845485377058
Valor de referencia de la integral con scipy = 8.154845485377136
Error absoluto = 7.815970093361102e-14
Error relativo = 9.584449033861293e-15

#### 5.1.2

$$\int_0^5 \frac{1 - e^x}{x} dx \tag{7}$$

Resultado de la integral= -37.99862157746428
Valor de referencia de la integral con scipy = -37.998621778467545
Error absoluto = 2.0100326736383067e-07
Error relativo = 5.289751521402074e-09