Facultad de Ciencias Físico Matemáticas (2023), **Febrero**, 1–6

REPORTE DE LABORATORIO COMPUTACIONAL

Aplicación del método de Romberg en la ecuación del cohete.

Jesús Eduardo Loera Casas*

Facultad de Ciencias Físico Matemáticas, San Nicolás de los Garzas, Nuevo León, México
*Corresponding author. Email: colab.git@gmail.com

Abstract

En este reporte implementamos el método de Romberg y el método de extrapolación de Richardson en los lenguajes de Fortran y Python. El método funciona para resolver integrales definidas de funciones de una variable real. Una vez implementado el método, este se aplicó para resolver la ecuación del cohete, donde se mostró que el método de extrapolación de Richardson es eficiente para iterar los resultados numéricos de obtenidos con el método de Romberg y mejorar la precisión de las integrales calculadas.

Keywords: Integración numérica, Método de Romberg, Método de Extrapolación de Richardson, Fortran, Python

Desde la invención del cálculo con Newton y Leibniz, en prácticamente todas las ramas de la física se suele recurrir a el cálculo de integrales para definir magnitudes físicas. Por ejemplo, cuando queremos verificar la normalización de una función de onda

$$\langle \psi | \psi \rangle = \int_{-\infty}^{\infty} \psi^* \psi \, dx$$

El cálculo de dichas integrales se puede volver complicado, por ello solemos recurrir a técnicas de integración numérica para resolver el problema de manera computacional.

1. El Método de Romberg

El método de Romberg (Scherer 2010) es un método de integración numérica que permite calcular el valor aproximado de las integrales de la forma

$$I = \int_a^b f(x) dx$$

donde f(x) es una función escalar de una variable real X.

El algoritmo de Romberg

El método de Romberg consiste en discretizar el intervalo $[a, b]$ en una malla de puntos equidistantes de manera que la distancia que separa a dos puntos contiguos está dada por

$$h_n = \frac{b-a}{2^{n-1}}$$

en donde $n \geq 1$ y $m \geq 1$.

Este es un método iterativo y recursivo, donde $R_{n,m}$ representa la aproximación a la integral numérica con un error del orden $O(h_n^{2m+1})$. Podemos encontrar el valor de $R_{n,m}$ de la siguiente manera:

$$R_{1,1} = \frac{1}{2}(b-a)(f(a)+f(b))$$

$$R_{n,1} = \frac{h_n}{2}\left[f(a)+f(b)+2\sum_{i=1}^{2^{n-1}} f(a+ih_n)\right]$$

$$R_{i,j} = \frac{4^{j-1}R_{i,j-1} - R_{i-1,j-1}}{4^{j-1}-1}$$

El corazón del método de Romberg es el *método de Extrapolación de Richardson*, un método que toma una secuencia convergente para construir otra secuencia que converge más rápido. Este es ampliamente utilizado en un sin fin de métodos numéricos interativos, pero su más icónica aplicación es justamente en la construcción del algoritmo de Romberg.

1.0.1 Resolver la ecuación del Cohete

Problema. The upward velocity of a rocket can be computed by the following formula:

$$v = uln\left(\frac{m_o}{m_o - qt}\right) - gt$$

where v = upward velocity, u = velocity at which fuel is expelled relative to the rocket, m_o = initial mass of the rocket at time $t = 0$, q = fuel consumption rate, and g = downward acceleration of gravity (assumed constant = $9.8 m/s^2$). If $u = 1800 m/s$, $m_o = 160000 kg$ and $q = 2500 kg/s$, use six-segment trapezoidal and Simpson's 1/3 rule, six-point Gauss quadrature, and $O(h^8)$ Romberg mehtods to determine how high the rocket will fly in 30 s.

- Solución con Python

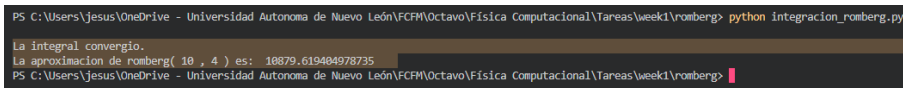

Figure 1. Se ejecutó el script de Python que resuelve el problema físico.

- Solución con Fortran

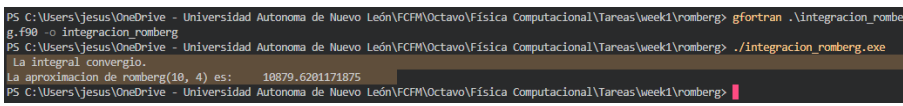

Figure 2. Se ejecutó el script de Python que resuelve el problema físico.

Vea una comparación de las respuestas obtenidas en cada script en la tabla 1.

Table 1. Se resolvió el problema de la altura alcanzada por el cohete con el script en Python y Fortran.

	Altura alcanzada en 30 segundos
Python	10879.61940497 metros
Fortran	10879.62011718 metros

2. Conclusión

Podemos observar que en ambos scripts donde se implementa el método de Romberg (en Python y Fortran), al calcular el valor de la integral numérica se converge a un resultado de manera rápida antes de alcanzar a calcular el valor de $R_{10,4}$ e incluso ambos valores calculados por los distintos programas son muy similares. De manera que queda en evidencia la eficiencia y la rapidez de este método numérico para resolver integrales de manera óptima.

References

Scherer, Philipp OJ. 2010. *Computational physics*. Springer.

Appendix 1. Programa en Python

```
# Programa elaborado por Jesus Eduardo Loera Casas
# Elaborado el dia 26/02/23

import numpy as np

"""
Este programa calcula la integral de una funcion continua de una
variable f(x) en el intervalo [a,b]
"""

def roomberg_00(a, b):
    return 0.5*(b-a)*(function(a)+function(b))
```

```
def romberg_n0(n, a, b):
    if (n==0):
        return roomberg_00(a, b)
    else:
        hn = (b-a)/(2**n)
        sum = 0
        for k in range (1, 2**(n-1) + 1):
            sum = sum + function(a + (2*k-1)*hn)
        Rn0 = 0.5*(romberg_n0(n-1, a, b)) + hn*sum
    return Rn0

def romberg_nm(n, m, a, b):
    # Definimos una tolarancia al error con cada iteracion
    tol = 0.00001
    # Creamos una matriz donde almacenaremos las iteraciones de romberg
    matrix = np.zeros([n,m])
    matrix[0,0] = roomberg_00(a,b)

    for i in range(1, n):
        matrix[i,0] = romberg_n0(i, a, b)
        # Evaluamos el criterio de convergencia
        if (abs(matrix[i,0]-matrix[i-1,0]) <= tol ):
            print("La integral convergio")
            return matrix[i,0]

    for j in range(1, m):
        for i in range(j, n):
            aux = (4**(j))*matrix[i, j-1] - matrix[i-1, j-1]
            romberg_ij = (1.0/((4.0**(j))-1))*(aux)
            matrix[i,j] = romberg_ij
            # Evaluamos el criterio de convergencia
            if (abs(matrix[i,j]-matrix[i-1,j]) <= tol ):
                print("La integral convergio.")
                return matrix[i,j]

    # Si no converge nos quedamos con el valor mas preciso que tenemos.
    return matrix[n,m]

"""
Funcion a integrar
"""
def function(x):
    g = 9.8; u = 1800; mo = 160000; q = 2500
    return u*np.log((mo)/(mo-q*x))-g*x

"""
Intervalo de integracion
"""
a = 0 ; b = 30
n = 10 ; m = 4

"""
Prueba del metodo
"""
integral_nm = romberg_nm(n, m, a, b)
print("La aproximacion de romberg(",n,",",m,") es: ", integral_nm)
```

Appendix 2. Programa en Fortran

```
! Programa elaborado por Jesus Eduardo Loera Casas
! Fecha 01/02/23

    REAL FUNCTION fx(x)
        IMPLICIT NONE
        REAL :: x
```

```
    ! Parametros
    REAL :: mo, g, q, u
    g = 9.8; u = 1800; mo = 160000; q = 2500
    ! ecuacion del cohete
    fx = u*log((mo)/(mo-q*x))-g*x
    RETURN
END

REAL FUNCTION romberg_00(a, b)
    IMPLICIT NONE
    REAL :: a, b, fx
    romberg_00 = 0.5*(b-a)*(fx(a)+fx(b))
    RETURN
END

RECURSIVE FUNCTION romberg_n0(n, a, b) result(integral)
    IMPLICIT NONE
    INTEGER :: n, k
    REAL :: a, b, fx, romberg_00, hn, sum, integral
    IF (n==0) THEN
        integral = romberg_00(a, b)
    ELSE
        hn = (b-a)/(2**n)
        sum = 0
        DO k = 1, 2**(n-1)
            sum = sum + fx(a + (2*k-1)*hn)
        END DO
        integral = 0.5*(romberg_n0(n-1, a, b)) + hn*sum
    END IF
    RETURN
END

REAL FUNCTION romberg_nm(n, m, a, b)
    IMPLICIT NONE
    INTEGER :: n, m, i, j
    REAL :: a, b, romberg_ij, aux, romberg_00, romberg_n0, tol
    REAL, DIMENSION ((n+1),(m+1)) :: matrix

    ! Definimos una tolarancia al error con cada iteracion
    tol = 0.00001

    matrix(1,1) = romberg_00(a,b)

    DO i = 2, n + 1
        matrix(i,1) = romberg_n0(i-1, a, b)
        ! Evaluamos el criterio de convergencia
        IF (abs(matrix(i,1)-matrix(i-1,1)) .le. tol ) THEN
            write(*,*) "La integral convergio con una"
            write(*,*) "tolerancia de: ", tol
            romberg_nm = matrix(i,1)
            RETURN
        END IF
    END DO

    DO j = 2, m+1
        DO i = j, n+1
            aux = (4**(j-1))*matrix(i, j-1) - matrix(i-1, j-1)
            romberg_ij = (1.0/((4.0**(j-1))-1))*(aux)
            matrix(i,j) = romberg_ij
            ! Evaluamos el criterio de convergencia
            IF (abs(matrix(i,j)-matrix(i-1,j)) .le. tol ) THEN
                write(*,*) "La integral convergio."
                romberg_nm = matrix(i,j)
                RETURN
            END IF
        END DO
    END DO
```

```
        write(*,*)"La integral no convergio con la tolerancia dada."
        romberg_nm = matrix(n+1, m+1)
        RETURN
    END

    PROGRAM main

        IMPLICIT NONE
        INTEGER :: n, m
        REAL :: a, b
        REAL :: romberg_nm, int_nm

        n = 10 ; m = 4
        a = 0  ; b = 30

        int_nm = romberg_nm(n, m, a, b)

        WRITE(*,*) "La aproximacion de romber es: ", int_nm

    END PROGRAM main
```