**Instituto Tecnológico Autónomo de México**

Algoritmos Numéricos por Computadora

# *"Integración Numérica"*

**Alumnos:**

Sofía Hernández Mendoza

Carlos Lozada Rojas

Miguel Ángel Cifuentes Jiménez

**Profesor:**

Doctor Ángel Fernando Kuri Morales

Miércoles, 14 de oct. de 20

## 1. Objetivos

Que el estudiante comprenda que es la integración numérica, en específico la cuadratura de Gauss y que hay distintas formas para resolverla.

## 2. Marco Teórico

La integración numérica es un instrumento de suma importancia que proporciona fórmulas y técnicas para analizar y calcular aproximaciones de integrales definidas. Gracias a ella se pueden calcular de forma muy acertada, valores de integrales definidas que no pueden calcularse analíticamente y, sobre todo, se puede efectuar ese cálculo en una computadora.

Constituye una amplia jerarquía de algoritmos para calcular el valor numérico de una integral definida y, por extensión, el término se usa a veces para describir algoritmos numéricos para resolver ecuaciones diferenciales. El término cuadratura numérica es más parecido al significado de integración numérica, especialmente si se aplica a integrales de una dimensión a pesar de que para el caso de dos o más dimensiones también se utiliza.

El problema básico considerado por la integración numérica es calcular una solución aproximada a la integral definida:

$$\int_a^b f(x)\,dx$$

Este problema también puede ser enunciado como un problema de valor inicial para una ecuación diferencial ordinaria, como sigue:

$$y'(x) = f(x), \quad y(a) = 0$$

Encontrar y(b) es equivalente a calcular la integral. Los métodos desarrollados para ecuaciones diferenciales ordinarias, como el método de Runge-Kutta, pueden ser aplicados al problema reformulado.

Existen integrales que no son tan sencillas de resolver de manera analítica o que definitivamente no tienen solución de forma analítica. Por esta razón, tenemos la posibilidad de resolver este tipo de problemas mediante procesos numéricos. La diferencia entre estas dos soluciones es la siguiente:

Solución analítica: nos da un resultado exacto.

Solución numérica: no da una aproximación del resultado

*Métodos para integrales unidimensionales*

Los métodos de integración numérica son una combinación de evaluaciones para obtener una aproximación integral. Una parte importante del análisis de cualquier método de integración numérica es analizar el comportamiento del error de aproximación como una función del número de evaluaciones del integrando. Un método que produce un pequeño error para un pequeño número de evaluaciones es normalmente considerado superior. Reduciendo el número de evaluaciones de la integral se reduce el número de operaciones aritméticas involucradas y eso reduce el error de redondeo total.

*Métodos basados en funciones de interpolación*

Hay un extenso número de métodos que se basan en aproximar la función a integrar f(x) por otro función g(x) de la cual se conoce la integral exacta. La función que sustituye la original se encuentra de forma que en un número específico de puntos tenga el mismo valor que la original. Como los puntos extremos forman parte siempre de este conjunto de puntos, la nueva función se llama una interpolación de la función original. Cuando los puntos extremos no se utilizan para encontrar la función que sustituye a la original entonces se dice extrapolación. Típicamente estas funciones son polinomios.

*Fórmulas de Newton – Cotes*

La interpolación con polinomios evaluada en puntos igualmente separados en [a,b] da las fórmulas de Newton-Cotes, de las que la regla del rectángulo, la del trapecio y la de Simpson son ejemplos. Si se escogen los nodos hasta k = n + 1 será la fórmula de Newton-Cotes cerrada y si se escogen k = n − 1 será la fórmula de Newton-Cotes abierta.

Estas fórmulas se basan en la idea de integrar una función polinomial en vez de f (x):

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

donde $P_n(x) = a_0 + a_1x + \cdots . + a_nx^n$ es un polinomio de aproximación de grado n para ciertos valores que se escogen apropiadamente (se suele conocer también como polinomio de interpolación, ya que la condición es que tome los mismos valores que la función original en los puntos elegidos).

*Regla del rectángulo*

El método más simple de este tipo es hacer a la función interpoladora ser una función constante (un polinomio de orden cero) que pasa a través del punto (a,f(a)). Este método se llama la regla del rectángulo:

$$\int_a^b f(x)dx \sim (b-a)f(a)$$

*Regla del punto medio*

Si en el método anterior la función pasa a través del punto $(\frac{a+b}{2}, f(\frac{a+b}{2}))$ este método se llama la regla del punto medio:

$$\int_a^b f(x)dx \sim (b-a)f(\frac{a+b}{2})$$

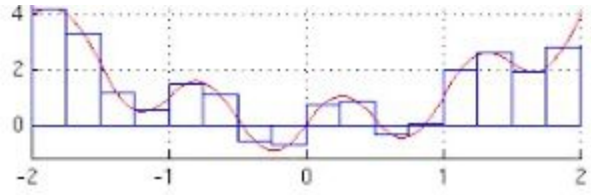


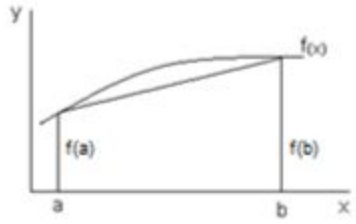

Ilustración de la regla del punto medio.

*Regla del trapecio*

Corresponde al caso donde n=1, es decir:

$$I = \int_a^b f(x)\ dx \approx \int_a^b P_1(x)dx$$

Donde P1(x) es un polinomio de de grado 1. En el gráfico trazamos la recta que une los puntos: (a, f(a)) y (b, f(b)) obteniendo un trapecio cuya superficie será, aproximadamente, el valor de la integral I.




Así tendremos:

$$I = \int_a^b f(x)\ dx \approx \int_a^b P_1(x)dx = \int_a^b (a_0 + a_1 x)\ dx = (b-a)\ \frac{f(a)+f(b)}{2}$$

conocida como Regla del Trapecio.

*Regla de Simpson*

La función interpoladora puede ser un polinomio de grado 2 que pasa a través de los puntos $(a, f(a))$, $(\frac{a+b}{2}, f(\frac{a+b}{2}))$ y $(b, f(b))$.

Este método se llama regla de Simpson:

$$(a, f(a)), \ (\frac{a+b}{2}, f(\frac{a+b}{2})) \text{ y } (b, f(b)).$$

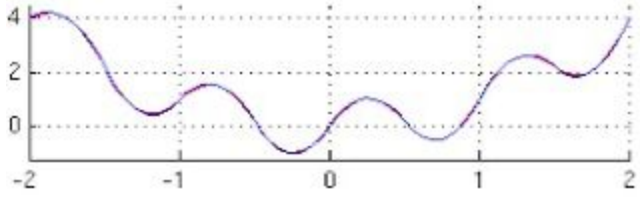



Ilustración de la regla de Simpson.

*Reglas compuestas*

Para cualquier regla interpoladora, se puede hacer una aproximación más precisa dividiendo el intervalo [a,b] en algún número n de subintervalos, hallando una aproximación para cada subintervalo, y finalmente sumando todos los resultados. Las reglas que surgen de hacer esto se llaman reglas compuestas, y se caracterizan por perder un orden de precisión global frente a las correspondientes simples, si bien globalmente dan valores más precisos de la integral, a costa eso sí de incrementar significativamente el coste operativo del método. Por ejemplo, la regla del trapecio compuesta puede expresarse como:

$$\int_a^b f(x)dx \sim \frac{b-a}{n}\left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f\left(a + k\frac{b-a}{n}\right)\right)$$

donde los subintervalos tienen la forma $[kh, (k+1)h]$ con $h = \frac{b-a}{n}$ y $k = 0, 1, 2, \ldots, n-1$.

*Cuadratura de Gauss*

Si se permite variar los intervalos entre los puntos de interpolación, se encuentra otro grupo de fórmulas de integración, llamadas fórmulas de cuadratura de Gauss. Una regla de cuadratura de Gauss es típicamente más precisa que una regla de Newton-Cotes que requiera el mismo número de evaluaciones del integrando, si el integrando es suave (es decir, si se puede derivar muchas veces).

*Fórmula de Rodrigues*

La fórmula de Rodrigues proporciona una representación diferencial de los polinomios de Legendre.

Partimos de las siguientes propiedades de la derivada de un monomio:

$$\begin{aligned} \left(\frac{d}{dx}\right)^n x^m &= \frac{m!}{(m-n)!}\, x^{m-n} \quad \text{para } n \leq m, \\ \left(\frac{d}{dx}\right)^n x^m &= 0 \quad \text{para } n > m. \end{aligned} \tag{2.31}$$

En particular se tiene que

$$\left(\frac{d}{dx}\right)^l x^{2l-2m} = \frac{(2l-2m)!}{(l-2m)!} x^{l-2m} \Rightarrow x^{l-2m} = \frac{(l-2m)!}{(2l-2m)!} \left(\frac{d}{dx}\right)^l x^{2l-2m}.$$

Usamos esta última relación en la expresión (2.24) de $P_l(x)$:

$$\begin{aligned} P_l(x) &= \sum_{m=0}^{[l/2]} (-1)^m \frac{1}{2^l m!(l-m)!} \left(\frac{d}{dx}\right)^l x^{2l-2m} \\ &= \frac{1}{2^l} \left(\frac{d}{dx}\right)^l \sum_{m=0}^{[l/2]} \frac{1}{m!(l-m)!} (-1)^m x^{2(l-m)}. \end{aligned} \tag{2.32}$$

Esta última expresión es muy similar a la fórmula del binomio de Newton. Sería igual si el sumatorio se extendiera hasta $l$ y estuviera multiplicado por $l!$ Pero, ¿es lícito extender el sumatorio hasta $l$? La respuesta es sí ya que

$$\sum_{m=0}^{l} \frac{(-1)^m x^{2(l-m)}}{m!(l-m)!} = \sum_{m=0}^{[l/2]} \frac{(-1)^m x^{2(l-m)}}{m!(l-m)!} + \sum_{m=[l/2]+1}^{l} \frac{(-1)^m x^{2(l-m)}}{m!(l-m)!} \tag{2.33}$$

y resulta que el segundo sumatorio, tras ser derivado $l$ veces, es nulo. Justificaremos esta afirmación escribiendo

$$\sum_{m=[l/2]+1}^{l} \frac{(-1)^m x^{2(l-m)}}{m!(l-m)!} = x^{2l-2[l/2]-2} + \text{monomios de grado menor}$$

Como

$$2[l/2] = \begin{cases} l & \text{si } l \text{ es par,} \\ l-1 & \text{si } l \text{ es impar,} \end{cases}$$

se tiene que

$$2l - 2[l/2] - 2 = \begin{cases} l-2 & \text{si } l \text{ es par,} \\ l-1 & \text{si } l \text{ es impar,} \end{cases}$$

y por tanto el monomio de mayor grado que añadimos en (2.33) es $x^{l-2}$ si $l$ es par, o $x^{l-1}$ si $l$ es impar. Pero, según (2.31), su derivada $l$-ésima es nula en cualquiera de estos dos casos. Esto significa que, efectivamente, los términos que añadimos, tras ser derivados $l$ veces, dan una contribución nula. Por tanto podemos escribir (2.32) en forma del binomio de Newton:

$$\begin{aligned} P_l(x) &= \frac{1}{2^l l!} \left(\frac{d}{dx}\right)^l \sum_{m=0}^{l} \frac{l!}{m!(l-m)!} (-1)^m x^{2(l-m)} \\ &= \frac{1}{2^l l!} \left(\frac{d}{dx}\right)^l (x^2-1)^l. \end{aligned} \tag{2.34}$$

Los polinomios de Legendre son las soluciones de la ecuación de Sturm-Liouville

$$\boxed{(1-x^2)\,y'' - 2x\,y' + l(l+1)\,y = 0}, \qquad -1 \le x \le 1, \tag{2.20}$$

que verifican la condición de ser regulares en $x = \pm 1$. En otros términos, son las soluciones de problema de Sturm-Liouville *singular*,

$$\mathcal{L}P_l(x) = -\lambda_l P_l(x), \tag{2.21}$$

donde $p(x) = 1 - x^2, q(x) = 0, r(x) = 1$ y $\lambda_l \equiv l(l+1)$, con las condiciones de contorno

$$P_l(\pm 1) = \text{finito}. \tag{2.22}$$

Cuando se resuelve la ecuación de Legendre mediante serie de potencias (véase la sección 2.3.1 siguiente) se encuentra que las condiciones de finitud de la solución en $x = \pm 1$ sólo pueden satisfacerse si $l = 0, 1, 2, \ldots$ Es decir, los autovalores del problema de Sturm-Liouville singular dado por (2.20) y (2.22) son

$$\lambda_l \equiv l(l+1), \quad l = 0, 1, 2, \ldots \tag{2.23}$$

Para estos valores de $l$, las soluciones (autofunciones) de la ecuación de Legendre que satisfacen las condiciones de contorno (2.22) existen y son los polinomios de Legendre, cuya representación en serie de potencias es

$$P_l(x) = \sum_{m=0}^{[l/2]} (-1)^m \frac{(2l-2m)!}{2^l m!\,(l-m)!\,(l-2m)!}\, x^{l-2m}, \tag{2.24}$$

donde por $[l/2]$ denotamos la parte entera de $l/2$. La constante de normalización se ha elegido de modo que $P_l(1) = 1$.

# 3. Desarrollo

# Codigo:

```
import sys
import math
from sympy import *

x = Symbol('x')
y = Symbol('y')

# Verifica que los valores ingresados sean tipo float
def check_format(x):
    while True:
        try:
            x = float(input("\""+x+"\")\t"))
            return x
```

```
        except:
            print("Must been a number")
            print("Try again")


# FORMULA DE RODRIGUES
# Fórmula para los polinomios de Legendre
#
# n es el grado del polinomio de Legendre
def rodrigues(n):
    x = Symbol("x")
    y = (x**2 - 1)**n
    polinomio = diff(y, x, n) / (2**n * math.factorial(n))
    return polinomio

def cuadra_Gauss(a, b, n):
    #Calcula raíces del polinomio de Legendre
    var = rodrigues(n)
    #--print(f"Rodrigues: {var}")
    x_roots = solve(var, x)

    # Calcula derivadas del polinomio de Legendre
    x_diff= diff(rodrigues(n))
    #--print(f"Valor de diferencial: {x_diff}")

    # Guarda los valores
    Suma = []
    Aux = [[] , []]

    for i in range(n):
        Suma.append(2/((1-x_roots[i]**2)*(x_diff.evalf(subs={x:x_roots[i]}))**2))
        Aux[0].append(y.evalf(subs={x:(b-a)*(x_roots[i]/2)+(a+b)/2}))
        Aux[1].append(Suma[i]*Aux[0][i])

    print(str(Suma))

    cuad = ((b-a)/2)*sum(Aux[1])
    exac = (integrate(y,x).evalf(subs={x:b}))-(integrate(y,x).evalf(subs={x:a}))
    error = ((exac-cuad))
    ER = abs((exac - cuad)/exac) * 100
```

```
    print ("\nResultado de cuadratura: ", cuad)
    print ("Resultado exacto : ", exac)
    print ("Porcentaje de error absoluto: ", error)
    print ("Error relativo: ", ER)

    return error


#-------------------------------------------------------------------------#
# CÓDIGO PARA HACER LA INTERPOLACIÓN DE UN SISTEMA DE ECUACIONES

def lassol(M, XY, CF):
    for i in range(0,M):
      rowmax = abs(XY[i][0])
      for j in range(1,M+1):
         rowmax = max(rowmax, abs(XY[i][j]))
      #endFor
      if (rowmax == 0):
       return False
      #endIf
      scale=1 / rowmax
      for j in range(0,M+1):
         XY[i][j] = XY[i][j]*scale
      #endFor
    #endFor
    for k in range(0, M):
       big = 0
       for i in range(k, M):
          temp = abs(XY[i][k])
          if (big < temp):
             big = temp
             ipiv = i
          #endIf
       #endFor
       if (big == 0):
          return False
       #endIf

#
```

```
#    Exchange column with largest element
#
       if (ipiv != k):
          for i in range(k, M+1):
             temp = XY[k][i]
             XY[k][i] = XY[ipiv][i]
             XY[ipiv][i] = temp
               #endFor
         #endIf



#
#    Eliminate all in column except first
#
       for i in range(k+1, M):
          quot = XY[i][k]/XY[k][k]
          for j in range(k+1,M+1):
             XY[i][j] = XY[i][j] - quot * XY[k][j]
               #endfor
       #endFor
    #endFor

    if (XY[M - 1][M - 1] == 0):
       return False
    #endIf
    CF[M - 1] = XY[M - 1][M] / XY[M - 1][M - 1]
    for i in range(M-2, -1, -1):
       sum = 0
       for j in range(i+1, M):
          sum = sum + XY[i][j] * CF[j]
       #endFor
       CF[i] = (XY[i][M] - sum) / XY[i][i]
    #endFor
    return True
#endLassol

def PassMat(M, rows, cols):
    print("La matriz en memoria: ")
    for i in range(0, rows):
       for j in range(0, cols):
```

```
            print("XY["+ str(i) +"]["+ str(j) +"]\t"+ str(M[i][j]))
        #endFor
    #endFor
    return
#endFunction

def NumCols(Linea, Linelen):
    cols = 1
    for i in range(0, LineLen):
        if (Linea[i] == "\t"):
            cols = cols + 1
        #endIf
    #endFor
    return cols
#endFunction

def getNum(Linea, numVar):
    countVar = -1
    first = 0
    for i in range(0, numVar+1):
        x = ""
        for j in range(first, len(Linea)):
            if (Linea[j] == "\t" or Linea[j] == "\n"):
                first = j + 1
                countVar = countVar + 1
                break
            else:
                x = x + Linea[j]
            #endif
        #endFor
        if (countVar == numVar):
            return float(x)
        #endIf
    #endFor
#endFunction

def polynomial(lista):
    while True:
        degree = len(lista)
        if(degree >= 1):
```

```
            break
        #endif
        print("Error, it may be, at least, a first degree")
    #endwhile
    Fx = [0]*(degree)

    for i in range(degree):
        Fx[i] = lista[i]
    ##EndFor
    print(f"{Fx}")
    return Fx
##EndFunction

## Evalua el polinomio
def evaluate_pol(Fx, x):
    res = 0
    degree = len(Fx)
    for i in range(degree ):
        # print(f"COEFICIENTE: {Fx[i]}")
        #print(f"VALOR i: {i}")
        #print(f"GRADO: {degree}")
        res += Fx[i]*x**(i)
    #print(f"\nEL RESULTADO PRUEBA ES: {res}")
    return res

#-------------------------------------------------------------------------#

#           INICIA PROGRAMA

# Symbol se utiliza para definir variables
# sin otorgarles algún valor en específico
# Se vuelven para usar Sympy
y = Symbol('y')
x = Symbol('x')
f = Symbol("f")

print("Seleccione como quiere ingresar la función ")
print("1) Ingresar la función manualmente ")
print("2) A partir de un documento externo (Sistema de ecuaciones)")
```

```
try:
   resp = int(input("Seleccione 1 o 2: "))
except:
   sys.exit("*** FIN DEL PROGRAMA ***")

if resp == 1:

   # Sympify convierte una expresión a una variable que se puede utilizar
   # con sympy
   y = sympify(input("Ingrese manualmente la función (escriba en términos de x  y sin y=
    ) "))
   print(type(y))
   print(f"Y: {y}")

   # Intervalos de integración
   print("Da el valor de los intervalos de integración: ")
   try:
      a = check_format("a")
      b = check_format("b")
   except:
       sys.exit("*** VERIFIQUE QUE LOS VALORES INGRESADOS HAYAN SIDO
    NÚMEROS ***")

   # Grado polinomio Legendre
   try:
      n = int(input("Grado polinomio de Legendre: \n"))#check_format("Grado: ")
   except:
      sys.exit("*** ERROR ***")


   cuadra_Gauss(a, b, n)



if resp == 2:
   # Intervalos de integración
   print("Da el valor de los intervalos de integración: ")
   try:
      a = float(input("El valor de a es: "))
      b = float(input("El valor de b es: "))
```

```
except:
   sys.exit("**** Los valores que dio no son números ****")

# Grado del polinomio de Legendre
try:
   n = int(input("Grado polinomio de Legendre: \n"))
except:
   sys.exit("*** ERROR ***")

# Código utilizado en el programa Interp.1.py
Arch = input("Dame el nombre del archivo a leer: \t")
Interp = Arch

try:
   FN = open(Arch, "r")
   print("Se abrió el archivo \""+ Arch +"\"" )
except:
   print("No se encontró el archivo \""+ Arch +"\"")
   sys.exit("**** Fin de programa ****")


lista = list(range(0))
lista = FN.readlines()
Filas = len(lista)
print("Filas: " + str(Filas))

FN = open(Arch, "r")
Lineas = list(range(Filas))
Lineas[0] = FN.readline()
LineLen = len(Lineas[0])
Columnas = NumCols(Lineas[0], LineLen)
print("Columnas: " + str(Columnas))

if (Filas != Columnas - 1):
   print("¡¡¡<Número de filas> debe ser = <Número de columnas-1>!!!")
   print("**** \tFin de Programa ****\n\n\n")
   sys.exit()

FN = open(Arch,"r")
XY = list(range(Filas))
```

```
for i in range(0, Filas):
   XY[i]=list(range(Columnas))

for i in range(0, Filas):
   Lineas[i] = FN.readline()
   for j in range(0, Columnas):
      L = Lineas[i]
      Num = getNum(L, j)     #j-th variablesin i-th line
      XY[i][j] = Num

FN.close()

print("\tMatriz de datos:")
for i in range(0, Filas):
   fila_i = ""
   for j in range(0, Columnas):
      XYij = XY[i][j]
      fila_i = fila_i + ("%10.4f" % (XY[i][j]))

   print(fila_i)


"""
      LA MATRIZ "XY" SE MODIFICA EN LASSOL
      POR ELLO ES INDISPENSABLE ALMACENAR "XY" EN OTRO ARREGLO
      EN ESTE CASO EL ARREGLO SE DENOMINA "AB"
      PARA TENER UNA COPIA DE XY ES IMPOSIBLE HACER
      SIMPLEMENTE AB=XY
      EL OBJETO "AB" ES IDÉNTICO AL OBJETO "XY"
"""

AB = list(range(Filas))

for i in range(0, Filas):
   AB[i] = list(range(Columnas))

for i in range(0, Filas):
   for j in range(0, Columnas):
      AB[i][j] = XY[i][j]
```

```
Vars = Filas
CF = list(range(Vars))
var = list()    # Almacena los valores de los coeficientes

if (not lassol(Vars, XY, CF)):
   print("****\n****\n\t\tUNSTABLE SYSTEM\n****\n****")
else:
   print("\n\tVALORES DE LA SOLUCIÓN\n")
   for i in range(0, Vars):
      var.append(CF[i])
      print("\tC[ %2.0f ] = %10.6f " % (i,CF[i]))


for i in range(0,Filas):
   F_Vars=0
   for j in range(0,Columnas-1):
      F_Vars=F_Vars+AB[i][j]*CF[j]
   #endFor
   print(" %2.0f ) %10.6f\t %10.6f " % (i, F_Vars, AB[i][Columnas-1]))

#
#   Obtenemos los valores de los coeficientes de la función interpolada
#
      print("\n\n**** VALORES DE LOS COEFICIENTES DE LA FUNCIÓN
  INTERPOLADA ****")
for i in range (0, Columnas-1):
   print(f"  {i}) {var[i]}")


#
# Escribimos el polinomio con los coeficientes
# obtenidos de la interpolación
#
PolIn = " "
count = len(var)

for i in range(len(var)):
   if i < len(var)-1:
```

```
            PolIn = PolIn +(str(var[i]*x**(count-1))) + "+"
        else:
            PolIn = PolIn +(str(var[i]*x**(count-1)))
        count = count -1

    print(f"{PolIn}")

    # Convertimos la variable "y" en una variable
    # tipo sympy para poder realizar la cuadratura
    y = sympify(PolIn)

    cuadra_Gauss(a, b, n)
```

---

```
import sys
import math

def check_format(x):
    while True:
        try:
            x=float(input("\""+x+"\")\t"))
            return x
        except:
                sys.exit("\n\tDEBE SER UN NÚMERO, INTENTE DE NUEVO") ## Exit
      program
    ##EndWhile

def transform(a,b,pol):
    global t1,t2,t3,t4,t5,dt
    dt=(b-a)/2
    if pol == 1:
        t1=(b+a)/2

    if pol == 2:
        t1=((b+a)+(b-a)*math.sqrt(1/3))/2
        t2=((b+a)+(b-a)*(-math.sqrt(1/3)))/2

    if pol == 3:
```

```
        t1=((b+a)+(b-a)*math.sqrt(3/5))/2
        t2=(b+a)/2
        t3=((b+a)+(b-a)*(-math.sqrt(3/5)))/2

    if pol == 4:
        t1=((b+a)+(b-a)*-0.861136312)/2
        t2=((b+a)+(b-a)*-0.339981044)/2
        t3=((b+a)+(b-a)*0.339981044)/2
        t4=((b+a)+(b-a)*0.861136312)/2


    if pol == 5:
        t1=((b+a)+(b-a)*-0.906179846)/2
        t2=((b+a)+(b-a)*-0.538469310)/2
        t3=(b+a)/2
        t4=((b+a)+(b-a)*0.538469310)/2
        t5=((b+a)+(b-a)*0.906179846)/2


    return


def resolve(pol):
    if pol == 1:
        q1=2
        res=q1*(C2*t1+C1)*dt

    if pol == 2:
        q1=1
        res=q1*((C3*t1**2+C2*t1+C1)*dt)+q1*((C3*t2**2+C2*t2+C1)*dt)

    if pol == 3:
        q1=5/9
        q2=8/9
        q3=5/9

      res=q1*((C4*t1**3+C3*t1**2+C2*t1+C1)*dt)+q2*((C4*t2**3+C3*t2**2+C2*t2+C
      1)*dt)+q3*((C4*t3**3+C3*t3**2+C2*t3+C1)*dt)

    if pol == 4:
```

```
        q1=0.3478548
        q2=0.6521452
        q3=0.6521452
        q4=0.3478548


      res=q1*((C5*t1**4+C4*t1**3+C3*t1**2+C2*t1+C1)*dt)+q2*((C5*t2**4+C4*t2**
      3+C3*t2**2+C2*t2+C1)*dt)+q3*((C5*t3**4+C4*t3**3+C3*t3**2+C2*t3+C1)*dt)
      +q4*((C5*t4**4+C4*t4**3+C3*t4**2+C2*t4+C1)*dt)

    if pol == 5:
        q1=0.2369269
        q2=0.4786287
        q3=0.5688889
        q4=0.4786287
        q5=0.2369269


      res=q1*((C6*t1**5+C5*t1**4+C4*t1**3+C3*t1**2+C2*t1+C1)*dt)+q2*((C6*t2**
      5+C5*t2**4+C4*t2**3+C3*t2**2+C2*t2+C1)*dt)+q3*((C6*t3**5+C5*t3**4+C4*t
      3**3+C3*t3**2+C2*t3+C1)*dt)+q4*((C6*t4**5+C5*t4**4+C4*t4**3+C3*t4**2+
      C2*t4+C1)*dt)+q5*((C6*t5**5+C5*t5**4+C4*t5**3+C3*t5**2+C2*t5+C1)*dt)
    return res

global C1,C2,C3,C4,C5,C6

print("\n\tPROPORCIONA EL NÚMERO DE COEFICIENTES DE LA ECUACIÓN
     DE GRADO:")
pol=check_format("Grado")
if (pol>6):
    sys.exit("\n\tESTA ECUACIÓN SUPERA EL GRADO 6") ## Exit program
#   endif
if pol == 1:
    print("\n\tPROPORCIONA LOS COEFICIENTES:")
    C1=check_format("c1")
    C2=check_format("c2")
       print("\n\tDA EL VALOR DE LOS INTERVALOS DE INTEGRACIÓN DE a
      HASTA b:")
    A=check_format("a")
    B=check_format("b")
```

```
if pol == 2:
   print("\n\tPROPORCIONA LOS COEFICIENTES:")
   C1=check_format("c1")
   C2=check_format("c2")
   C3=check_format("c3")
      print("\n\tDA EL VALOR DE LOS INTERVALOS DE INTEGRACIÓN DE a
    HASTA b:")
   A=check_format("a")
   B=check_format("b")

if pol == 3:
   print("\n\tPROPORCIONA LOS COEFICIENTES:")
   C1=check_format("c1")
   C2=check_format("c2")
   C3=check_format("c3")
   C4=check_format("c4")
      print("\n\tDA EL VALOR DE LOS INTERVALOS DE INTEGRACIÓN DE a
    HASTA b:")
   A=check_format("a")
   B=check_format("b")

if pol == 4:
   print("\n\tPROPORCIONA LOS COEFICIENTES:")
   C1=check_format("c1")
   C2=check_format("c2")
   C3=check_format("c3")
   C4=check_format("c4")
   C5=check_format("c5")
      print("\n\tDA EL VALOR DE LOS INTERVALOS DE INTEGRACIÓN DE a
    HASTA b:")
   A=check_format("a")
   B=check_format("b")

if pol == 5:
   print("\n\tPROPORCIONA LOS COEFICIENTES:")
   C1=check_format("c1")
   C2=check_format("c2")
   C3=check_format("c3")
   C4=check_format("c4")
```

```
    C5=check_format("c5")
    C6=check_format("c6")
       print("\n\tDA EL VALOR DE LOS INTERVALOS DE INTEGRACIÓN DE a
     HASTA b:")
    A=check_format("a")
    B=check_format("b")

transform(A,B,pol)
x=resolve(pol)
print ("\nEL ÁREA BAJO LA CURVA DE LA INTEGRAL ES: ", x)
```

## 4. Conclusiones

Con esta investigación pudimos comprender cómo son pocas las ecuaciones diferenciales que tienen una solución analítica sencilla, la mayor parte de las veces es necesario realizar aproximaciones, estudiar el comportamiento del sistema bajo ciertas condiciones y la importancia de los algoritmos numéricos para resolver este tipo de problemas matemáticos. Asimismo, se analizó distintas formas de realizar una cuadratura de Gauss, lo que nos permitió resolver el problema principal desde distintos métodos.

## 5. Referencias


*Wikipedia, (2020). "Integración numérica" [En línea]. Consultado el 6 de octubre del 2020. Disponible en: https://es.wikipedia.org/wiki/Integraci%C3%B3n_num%C3%A9ri ca*

*Scipy, "Numerical Methods using Python" [En línea]. Consultado el 1 de octubre del 2020. Disponible en: https://www.southampton.ac.uk/~fangohr/teaching/python/book /html/16-scipy.html*

*Kevin Briones del Carpio. (2019). Integración Numérica. Octubre 10, 2020, de Instituto Tecnológico de Tuxtla Gutierrez Sitio web: https://sites.google.com/site/sistrevolution/home/unidad-4/integracion-numerica*

*UNNE. (2018). Integración Numérica. Octubre 10, 2020, de UNNE Sitio web: http://www.ing.unne.edu.ar/assets/pdf/academica/departamentos/computacion/comp/IN.pdf*

*EDAN. (2009). Introducción a la integración numérica. Octubre 10, 2020, de EDAN Sitio web: http://departamento.us.es/edan/php/asig/LICFIS/LFIPC/Tema7FISPC0809.pdf*