EXAMEN 2: EJERCICIO 4, 7 Y 10

Autores:

1. David Montaño Castro
2. Sergio Israel Durán García

Paqueterías necesarias

```
In [1]:
import numpy as np
import sympy as sp
from sympy import Symbol,integrate,tan, log,ln, pi, sqrt, sin, cos,csc, exp, Abs, Min, Max#<----- Aquí
 puedes agregar la función que desees.
x = sp.Symbol("x")
```

Ejercicio 4

Elabora el código correspondiente para la construcción de los siguentes interpoladores polinómicos

Interpolador de Lagrange

```
In [2]:
def Inter_Lagrange(xi,fxi,est):
    """
    Función que calcula el interpolador de Lagrange en un punto estimado.

    Parámetros:

    xi : lista de los xi con el que se va a construir el interpolador.

    fxi : lista de las imágnees de los xi con los que se va a construir el interpolador.

    est : el punto dentro dentro del intervalo cuya imagen quiere ser calculada.
    """

    numeradores = []
    denominadores = []

    for i in range(0,len(xi)):  # For para calcular los numeradores de cada Li
        num = 1
        for j in xi:
            if j == xi[i]: # Salta el valor del arreglo cuyo Li está siendo calculado
                continue
            else:
                num = num * (est-j)

        numeradores.append(num)

    for i in range(0,len(xi)):  # For para calcular los denominadores de cada Li
        num = 1
        for j in xi:
            if j == xi[i]:  # Salta el valor del arreglo cuyo Li está siendo calculado
                continue
            else:
                num = num * (xi[i]-j)

        denominadores.append(1/num)

    numeradores = np.array(numeradores)
    denominadores = np.array(denominadores)
    fs = np.array(fxi)

    return print("La imagen de x = ",est,"es f(x) = ", np.sum(fs*numeradores*denominadores))
```

Ejemplo visto en clase

```
In [3]:
Inter_Lagrange([2,2.5,4],[.5,.4,.25],3)
```

```
La imagen de x =  3 es f(x) =  0.325
```

Ejemplo en video

```
In [4]:
Inter_Lagrange([0,1,2],[1,3,0],.5)
```

```
La imagen de x =  0.5 es f(x) =  2.625
```

Interpolador de Newton (adaptación a Python)

```
In [5]:
def Inter_Newton(xi,fxi,r):
    """
    Función que calcula el Interpolador de Newton

    Parámetros:

    xi : lista de los valores de las xi

    fxi : lista de las imágenes de los xi

    r : valor que quiere estimarse
    """
    r1 = len(xi)
    r2 = len(fxi)

    if r1 != r2:
        return "El método no se puede aplicar"
    else:
        mat = np.zeros((r1,r2))
        for i in range(0,r1):  # LLena toda la primera columna con los valores de las fxi
            mat[i,0] = fxi[i]
        for i in range(1,r1):  # Llena el resto de la matriz
            for j in range(1,i+1):
                mat[i,j] = (mat[i,j-1] - mat[i-1,j-1])/(xi[i] - xi[i-j])

    a = mat.shape
    a1 = a[0]
    a2 = a[1]

    vec_r = np.zeros(a1)
    for i in range(1,a1):
        prod = 1
        for j in range(0,i):
            prod = prod * (r - xi[j])
        vec_r[i-1] = prod

    inter = mat[0,0]
    for i in range(1,a1):
        inter = inter + (mat[i,i] * vec_r[i-1])

    return print("La imagen de x = ",r,"es f(x) = ", inter)
```

Ejemplo visto en clase

```
In [6]:
Inter_Newton([2,2.5,4],[.5,.4,.25],3)
```

```
La imagen de x =  3 es f(x) =  0.325
```

Ejemplo en video

```
In [7]:
Inter_Newton([1,0,-3],[2,4,-2],-1)
```

```
La imagen de x =  -1 es f(x) =  4.0
```

```
In [8]:
Inter_Newton([1,0,-3],[2,4,-2],-2)
```

```
La imagen de x =  -2 es f(x) =  2.0
```

```
In [9]:
Inter_Newton([1,0,-3],[2,4,-2],-3)
```

```
La imagen de x =  -3 es f(x) =  -2.0
```

Ejercicio 7

Elabora los códigos de derivación correspondientes a los siguientes métodos.

Primer Orden

Centrada a tres puntos

```
In [10]:
def Centrada_3(funcion,x0,x1,x2):
    """
    Función que calcula la derivada de primer orden de x1 por medio de 3 puntos con la técnica centrada

    Parámetros:

    funcion : función a la que se le quiere calcular la derivada

    x0 : Punto anterior

    x1 : Punto en el que se evaluará la derivada

    x2 : Punto posterior
    """

    x = sp.Symbol("x")

    estimacion = (funcion.subs(x,x2) - funcion.subs(x,x0))/(x2 - x0) # Calcular estimación

    verdadero = sp.diff(funcion,x).subs(x,x1) # Calcular el valor verdadero

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejemplo visto en clase

```
In [11]:
Centrada_3(exp(-x)/x,.9,1,1.1)
```

```
El valor de la estimación es:  -0.7456699514464014
 El verdadero valor es: -0.7357588823428847
```

Progresiva a tres puntos

```
In [12]:
def Progresiva_3(funcion,x0,x1,x2):
    """
    Función que calcula la derivada de primer orden de x0 por medio de 3 puntos con la técnica progresi
va

    Parámetros:

    funcion : función a la que se le quiere calcular la derivada

    x0 : Punto en el que se evaluará la derivada

    x1 : Primer punto posterior

    x2 : Segundo punto posterior
    """

    x = sp.Symbol("x")

    estimacion = (4*funcion.subs(x,x1) - funcion.subs(x,x2) - 3*funcion.subs(x,x0))/(x2 - x0) # Calcula
r estimación

    verdadero = sp.diff(funcion,x).subs(x,x0) # Calcular el valor verdadero

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejemplo visto en clase

```
In [13]:
Progresiva_3(exp(-x)/x,1,1.1,1.2)
```

```
El valor de la estimación es:  -0.7209659787558802
 El verdadero valor es: -0.7357588823428847
```

Regresiva a cinco puntos

```
In [14]:
def Regresiva_5(funcion,x0,x1,x2,x3,x4):
    """
    Función que calcula la derivada de primer orden de x4 por medio de 5 puntos con la técnica regresiv
a.

    Parámetros:

    funcion : función a la que se le quiere calcular la derivada

    x0 : Primer punto anterior

    x1 : Segundo punto anterior

    x2 : Tercer punto anterior

    x3 : Cuarto punto posterior

    x4 : Punto en el que se evaluará la derivada
    """

    x = sp.Symbol("x")

    estimacion = (-27*funcion.subs(x,x0) + 64*funcion.subs(x,x1) + 36*funcion.subs(x,x2) - 288*funcion.
subs(x,x3) + 215*funcion.subs(x,x4))/(132 * (x1 - x0)) # Calcular estimación

    verdadero = sp.diff(funcion,x).subs(x,x4) # Calcular el valor verdadero

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejemplo propuesto

```
In [15]:
Regresiva_5(exp(-x)/x,.6,.7,.8,.9,1)
```

```
El valor de la estimación es:  -0.7638520140921287
 El verdadero valor es: -0.7357588823428847
```

Centrada a cinco puntos

```
In [16]:
def Centrada_5(funcion,x0,x1,x2,x3,x4):
    """
    Función que calcula la derivada de primer orden de x2 por medio de 5 puntos con la técnica centrada

    Parámetros:

    funcion : función a la que se le quiere calcular la derivada

    x0 : Primer punto anterior

    x1 : Segundo punto anterior

    x2 : Punto en el que se evaluará la derivada

    x3 : Primer punto posterior

    x4 : Segundo punto posterior
    """

    x = sp.Symbol("x")

    estimacion = (funcion.subs(x,x0) - 8*funcion.subs(x,x1) + 8*funcion.subs(x,x3) - funcion.subs(x,x4
))/(12 * (x1 - x0)) # Calcular estimación

    verdadero = sp.diff(funcion,x).subs(x,x2) # Calcular el valor verdadero

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejemplo visto en clase

```
In [17]:
Centrada_5(exp(-x)/x,.8,.9,1,1.1,1.2)
```

```
El valor de la estimación es:  -0.7353382448010146
 El verdadero valor es: -0.7357588823428847
```

Segundo Orden

Centrada a tres puntos

```
In [18]:
def Centrada_3_2(funcion,x0,x1,x2):
    """
    Función que calcula la derivada de segundo orden de x1 por medio de 3 puntos con la técnica centrad
a

    Parámetros:

    funcion : función a la que se le quiere calcular la derivada

    x0 : Punto anterior

    x1 : Punto en el que se evaluará la derivada

    x2 : Punto posterior
    """

    x = sp.Symbol("x")

    estimacion = (funcion.subs(x,x2) + funcion.subs(x,x0) - 2*funcion.subs(x,x1))/((x1 - x0)**2) # Calc
ular estimación

    verdadero = sp.diff(sp.diff(funcion,x),x).subs(x,x1) # Calcular el valor verdadero

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejercicio visto en clase

```
In [19]:
Centrada_3_2(x**3+3*x**5,.9,1,1.1)
```

```
El valor de la estimación es:  66.30000000000024
 El verdadero valor es: 66.0
```

Progresiva a tres puntos

```
In [20]:
def Progresiva_3_2(funcion,x0,x1,x2):
    """
    Función que calcula la derivada de segundo orden de x0 por medio de 3 puntos con la técnica Progres
iva

    Parámetros:

    funcion : función a la que se le quiere calcular la derivada

    x0 : Punto en el que se evaluará la derivada

    x1 : Primer punto posterior

    x2 : Segundo punto posterior
    """

    x = sp.Symbol("x")

    estimacion = (funcion.subs(x,x2) + funcion.subs(x,x0) - 2*funcion.subs(x,x1))/((x1 - x0)**2) # Calc
ular estimación

    verdadero = sp.diff(sp.diff(funcion,x),x).subs(x,x0) # Calcular el valor verdadero

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejercicio propuesto

```
In [21]:
Progresiva_3_2(x**3+3*x**5,1,1.1,1.2)
```

```
El valor de la estimación es:  86.7899999999992
 El verdadero valor es: 66.0
```

Regresiva a cinco puntos

```
In [22]:
def Regresiva_5_2(funcion,x0,x1,x2,x3,x4):
    """
    Función que calcula la derivada de segundo orden de x4 por medio de 5 puntos con la técnica Regresi
va

    Parámetros:

    funcion : función a la que se le quiere calcular la derivada

    x0 : Primer punto anterior

    x1 : Segundo punto anterior

    x2 : Tercer punto anterior

    x3 : Cuarto punto anterior

    x4 : Punto en el que se evaluará la derivada
    """

    x = sp.Symbol("x")

    estimacion = (-384*funcion.subs(x,x1) +162*funcion.subs(x,x0) + 672*funcion.subs(x,x3) - 366*funcio
n.subs(x,x4) -84* funcion.subs(x,x2))/(-264 * (x1 - x0)**2) # Calcular estimación

    verdadero = sp.diff(sp.diff(funcion,x),x).subs(x,x4) # Calcular el valor verdadero

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejercicio propuesto

```
In [23]:
Regresiva_5_2(x**3+3*x**5,.6,.7,.8,.9,1)
```

```
El valor de la estimación es:  61.29272727272725
 El verdadero valor es: 66.0
```

Ejercicio propuesto

```
In [24]:
Regresiva_5_2(ln(x),.6,.7,.8,.9,1)
```

```
El valor de la estimación es:  -0.814855761635273
 El verdadero valor es: -1.0
```

Ejercicio propuesto

```
In [25]:
Regresiva_5_2((3*x)**ln(x),.6,.7,.8,.9,1)
```

```
El valor de la estimación es:  1.7415155412661285
 El verdadero valor es: 2.108336672144472
```

Centrada a cinco puntos

```
In [26]:
def Centrada_5_2(funcion,x0,x1,x2,x3,x4):
    """
    Función que calcula la derivada de segundo orden de x2 por medio de 5 puntos con la técnica centrad
a

    Parámetros:

    funcion : función a la que se le quiere calcular la derivada

    x0 : Primer punto anterior

    x1 : Segundo punto anterior

    x2 : Punto en el que se evaluará la derivada

    x3 : Primer punto posterior

    x4 : Segundo punto posterior
    """

    x = sp.Symbol("x")

    estimacion = (-funcion.subs(x,x0) + 8*funcion.subs(x,x1) + 8*funcion.subs(x,x3) - funcion.subs(x,x4
) - 14*funcion.subs(x,x2))/(4 * (x1 - x0)**2) # Calcular estimación

    verdadero = sp.diff(sp.diff(funcion,x),x).subs(x,x2) # Calcular el valor verdadero

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejercicio visto en clase

```
In [27]:
Centrada_5_2(x**3+3*x**5,.8,.9,1,1.1,1.2)
```

```
El valor de la estimación es:  65.40000000000038
 El verdadero valor es: 66.0
```

Ejercicio 10

Elabora los códigos correpondientes a los siguientes métodos de integración numérica

Trapecio compuesto

```
In [28]:
def Trapecio_Compuesto(funcion,inf,sup,num_nodos):
    """
    Trapecio compuesto para aproximar valores de integrales definidas

    Parámetros:

    funcion : función a integrar

    inf : límite inferior de la integral

    sup : límite superior de la integral

    num_nodos : número de nodos que se desean calcular entre ambos límites. DEBE SER VALOR PAR.
    """

    x = sp.Symbol("x")

    xi = np.linspace(inf,sup,num_nodos) # Genera nodos
    fxi = []

    for i in xi: # Calcula las imagenes de los nodos
        fxi.append(float(funcion.subs(x,i)))

    fxo = fxi[0] # Guarda el primer y último elemento de las imagenes
    fxn = fxi[-1]
    h = xi[1] - xi[0] # Calcular el valor de h (separación enter nodos)

    del fxi[0] # Elimina el primer y último elemento de las imagenes
    del fxi[-1]

    estimacion = (h/2)* (fxo + 2*np.sum(np.array(fxi)) + fxn) # Estimación

    verdadero = integrate(funcion,(x,inf,sup)) # Integral verdadera

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejemplo visto en clase

```
In [29]:
Trapecio_Compuesto(-1/x**2,1,2,4)
```

```
El valor de la estimación es:  -0.5158333333333333
 El verdadero valor es: -0.5
```

Ejemplo ayudante

```
In [30]:
Trapecio_Compuesto(exp(-x),1,2,4)
```

```
El valor de la estimación es:  0.2346933677191053
 El verdadero valor es: 0.23254415793482963
```

Ejemplo de la tarea

```
In [31]:
Trapecio_Compuesto(x**2 * ln(x),1,3,6)
```

```
El valor de la estimación es:  7.113224267846189
 El verdadero valor es: 6.998621709124098
```

Simpson 1/3 compuesto

```
In [32]:
def Simpson_tercio(funcion,inf,sup,num_nodos):
    """
    Simpson 1/3 para aproximar valores de integrales definidas

    Parámetros:

    funcion : función a integrar

    inf : límite inferior de la integral

    sup : límite superior de la integral

    num_nodos : número de nodos que se desean calcular entre ambos límites. DEBE SER VALOR IMPAR.

    """

    x = sp.Symbol("x")

    xi = np.linspace(inf,sup,num_nodos) # Genera nodos
    fxi = []

    for i in xi: # Calcula las imagenes de los nodos
        fxi.append(float(funcion.subs(x,i)))

    fxo = fxi[0] # Guarda el primer y último elemento de las imagenes
    fxn = fxi[-1]
    h = xi[1] - xi[0] # Calcular el valor de h (separación enter nodos)

    del fxi[0] # Elimina el primer y último elemento de las imagenes
    del fxi[-1]

    # Listas de comprimida
    estimacion = (h/3) * (fxo + 4 * np.sum(np.array([x for x in fxi if fxi.index(x)%2 == 0])) + 2 * (n
p.sum(np.array([x for x in fxi if fxi.index(x)%2 != 0]))) + fxn)

    verdadero = integrate(funcion,(x,inf,sup))

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejemplo visto en clase

```
In [33]:
Simpson_tercio(x**2 * ln(x),2,4,5)
```

```
El valor de la estimación es:  21.50349557488515
 El verdadero valor es: 21.50366500017559
```

Ejercicio de la tarea

```
In [34]:
Simpson_tercio(x**2 * exp(-2*x),2,4,5)
```

```
El valor de la estimación es:  0.05606306467844655
 El verdadero valor es: 0.05608733445238534
```

Simpson 3/8 compuesto

```
In [35]:
def Simpson_Octavo(funcion,inf,sup,num_nodos):
    """
    Simpson 3/8 para aproximar valores de integrales definidas

    Parámetros:

    funcion : función a integrar

    inf : límite inferior de la integral

    sup : límite superior de la integral

    num_nodos : número de nodos que se desean calcular entre ambos límites. DEBE SER VALOR PAR.

    """

    x = sp.Symbol("x")

    xi = np.linspace(inf,sup,num_nodos) # Genera nodos
    fxi = []

    for i in xi: # Calcula las imagenes de los nodos
        fxi.append(float(funcion.subs(x,i)))

    fxo = fxi[0] # Guarda el primer y último elemento de las imagenes
    fxn = fxi[-1]
    h = xi[1] - xi[0] # Calcular el valor de h (separación enter nodos)

    del fxi[0] # Elimina el primer y último elemento de las imagenes
    del fxi[-1]

    estimacion = (3*h/8) * (fxo + 2 * np.sum(np.array([x for x in fxi if (fxi.index(x)-2)%3 == 0])) + 3
 * (np.sum(np.array([x for x in fxi if (fxi.index(x)-2)%3 != 0]))) + fxn)

    verdadero = integrate(funcion,(x,inf,sup))

    return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:", floa
t(verdadero))
```

Ejercicio de la tarea

```
In [36]:
Simpson_Octavo(x**2 * ln(x),1,3,4)
```

```
El valor de la estimación es:  6.995897295596805
 El verdadero valor es: 6.998621709124098
```

Cuadratura Gaussiana

```
In [37]:
def Cuadratura_Gaussiana(funcion,inf,sup,num_nodos = 4):
    """
    Cuadratura Gaussiana para aproximar el valor de una integral.

    Parámetros:

    funcion : función a la cual se le va a calcular la integral

    inf : límite inferior de la integral

    sup : límite superior de la integral

    num_nodos : número de nodos con los que se va a calcular la integral. Solo disponible para 2,3 y 4
 nodos. Opcional en 4.
    """
    x = sp.Symbol("x")

    if num_nodos == 2: # Dependiendo del número de nodos, se seleccionan los parámetros correspondiente
s
        ws = np.array([1,1])
        xs = np.array([-sqrt(1/3),sqrt(1/3)])
    elif num_nodos == 3:
        ws = np.array([.55555556,.88888889,.55555556])
        xs = np.array([-0.77459667,0,.77459667])
    elif num_nodos == 4:
        ws = np.array([.3478548,.6521452,.6521452,.3478548])
        xs = np.array([.8611363116,.3399810436,-0.3399810436,-0.8611363116])
    else:
        return "Solo hay disponibles 2,3 y 4 nodos"

    evaluaciones = []

    if inf == -1 and sup == 1:# En caso de que los límites ya concuerden, no hagas todo el proceso
        for i in xs:
            evaluaciones.append(float(funcion.subs(x,i)))
        evaluaciones = np.array(evaluaciones)

        estimacion = np.dot(evaluaciones,ws) # Producto punto entre ws y evaluaciones

        verdadero = integrate(funcion,(x,inf,sup))

        return print("El valor de la estimación es: ",float(estimacion),"\n","El verdadero valor es:",
float(verdadero))

    else:
        X = (x*(sup-inf)+ (inf+sup))/2 # transformacion t
        derivada_X = sp.diff(X,x) # derivada de t

        real_fuc = funcion.subs(x,X) # Funcion evaluada con el valor de t

        for i in xs:
            evaluaciones.append(float(real_fuc.subs(x,i)))
        evaluaciones = np.array(evaluaciones)

        estimacion = np.dot(evaluaciones,ws) # Producto punto entre ws y evaluaciones

        verdadero = integrate(funcion,(x,inf,sup))

        return print("El valor de la estimación es: ",float(derivada_X)*float(estimacion),"\n","El verd
adero valor es:", float(verdadero))
```

Ejercicio visto en clase

```
In [38]:
Cuadratura_Gaussiana(exp(-x**2),0,3,3)
```

```
El valor de la estimación es:  0.8845439314938828
 El verdadero valor es: 0.8862073482595212
```

Ejercicio de la tarea (4 nodos)

```
In [39]:
Cuadratura_Gaussiana(x**2 * exp(-2*x),2,4,4)
```

```
El valor de la estimación es:  0.05608748401373847
 El verdadero valor es: 0.05608733445238534
```

Ejercicio propuesto (-1 y 1)

```
In [40]:
Cuadratura_Gaussiana(x**2+x/2,-1,1,4)
```

```
El valor de la estimación es:  0.6666666101779625
 El verdadero valor es: 0.6666666666666666
```

Ejercicio propuesto (-1 y 1)

```
In [41]:
Cuadratura_Gaussiana(ln(x+3)- x/(x-2),-1,1,4)
```

```
El valor de la estimación es:  2.3560240477404806
 El verdadero valor es: 2.356107660695891
```