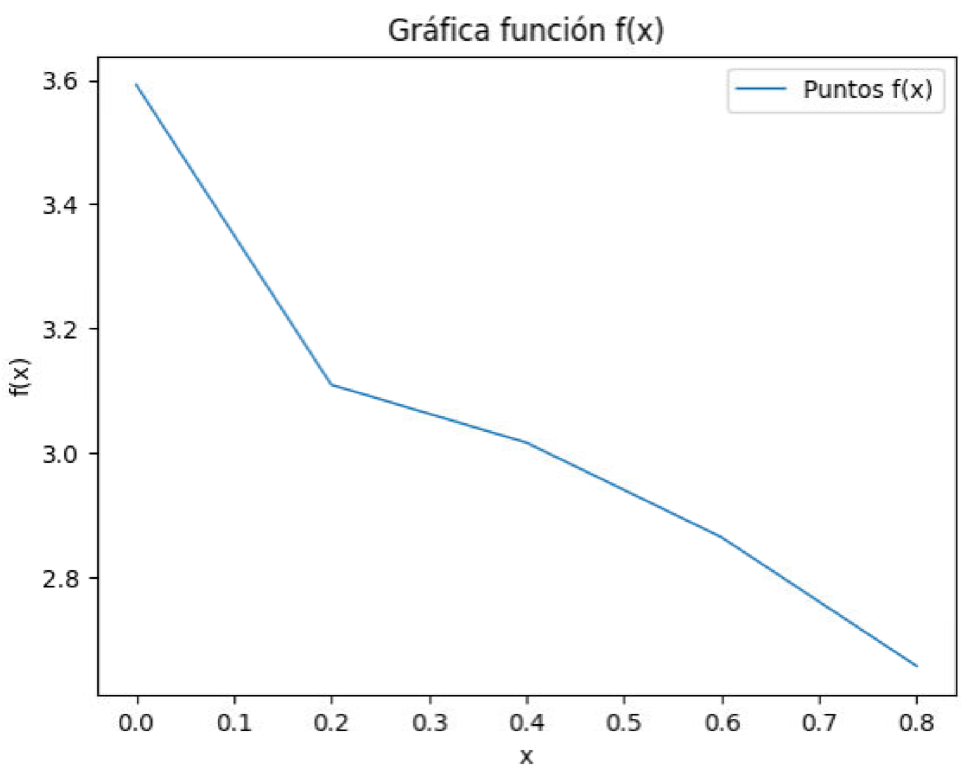
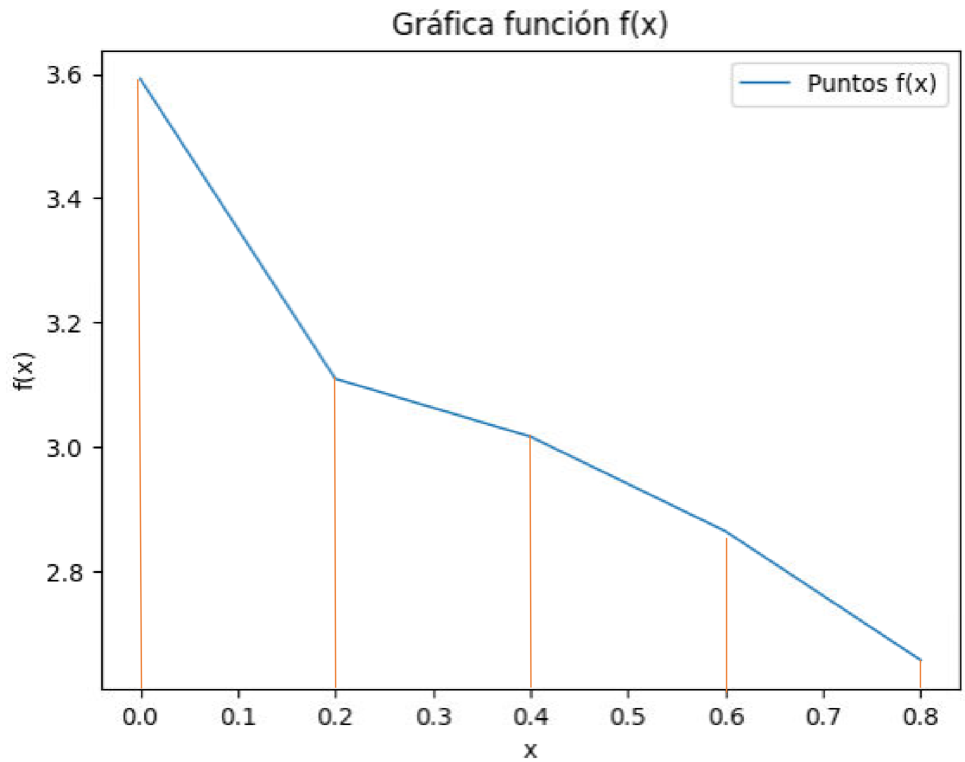
Ejercicio 7.5 Aproximar $\int_0^{0.8} f(x)\,dx$ utilizando los tres métodos siguientes.

```python
import numpy as np
import matplotlib.pyplot as plt

# input, datos polinomio
xi = np.array([0, 0.2, 0.4, 0.6, 0.8])
fi = np.array([3.592, 3.110, 3.017, 2.865, 2.658])


# Gráfica
plt.plot(xi,fi,'-', linewidth=1, label='Puntos f(x)')

plt.legend()
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gráfica función f(x)')
plt.show()
```

x	$f(x)$
0	3.592
0.2	3.110
0.4	3.017
0.6	2.865
0.8	2.658

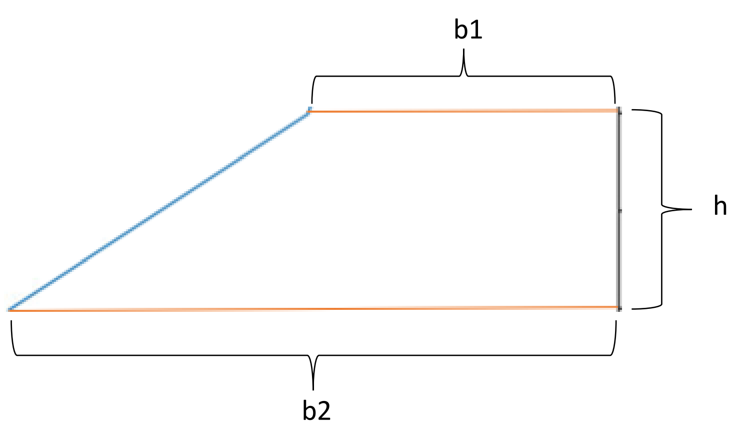


1. Regla del trapecio



El área de un trapecio está dada por $h\left(\frac{b_1 + b_2}{2}\right)$

Y en donde el área total debajo de la curva es igual a la suma de todas las áreas de los trapecios



T1 = 0.2 * ((3.592 + 3.110) / 2) = 0.6702

T2 = 0.2 * ((3.110 + 3.017) / 2) = 0.6127

T3 = 0.2 * ((3.017 + 2.865) / 2) = 0.5882

T4 = 0.2 * ((2.865 + 2.658) / 2) = 0.5523

Área total = 2.4234 u^2 -> aproximación

2. Regla de Simpson

$$\int_a^b f(x)\,dx = \left(\frac{b-a}{6}\right)\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

Donde a es igual a 0 y b es igual a 0.8 y

f ((a+b) / 2) = f ((0 + 0.8) / 2) = f (0.4) = 3.017

= ((0.8 - 0) / 6) [3.592 + 4 (3.017) + 2.658]

= (0.8 / 6) (18.318)

= 2.4424 u^2 -> aproximación

3. Regla Romberg

Esta regla al ser iterativa, se tiene la siguiente definición

$R_{k,j}$ k = 1, … , n

J = 1, … , k

$h_k = (b-a) / 2^{k-1}$

Primero se debe encontrar la semilla que es igual a $R_{1,1}$

$R_{1,1}$ = h1 / 2 [f(a) + f(b)]

= (b-a) /2 [f(a) + f(b)]

El siguiente algoritmo fue implementado para la solución del método.

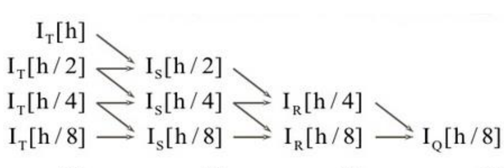
Romberg utiliza una tabla de iteraciones hasta que llega a su última iteración con la aproximación utilizada que en este caso la aproximación es de -> 2.44600 u^2

```python
from __future__ import print_function
from numpy import add, isscalar, asarray, arange
from scipy.integrate.quadrature import tupleset

def RombergMethod(y, dx, show=False):

    axis=-1
    y = asarray(y)
    nd = len(y.shape)
    Nsamps = y.shape[axis]
    Ninterv = Nsamps-1
    n = 1
    k = 0

    while n < Ninterv:
        n <<= 1
        k += 1

    R = {}
    all = (slice(None),) * nd
    slice0 = tupleset(all, axis, 0)
    slicem1 = tupleset(all, axis, -1)
    h = Ninterv*asarray(dx)*1.0
    R[(1,1)] = (y[slice0] + y[slicem1])/2.0*h
    slice_R = all
    start = stop = step = Ninterv
```

```python
    for i in range(2,k+1):
        start >>= 1
        slice_R = tupleset(slice_R, axis, slice(start,stop,step))
        step >>= 1
        R[(i,1)] = 0.5*(R[(i-1,1)] + h*add.reduce(y[slice_R],axis))
        for j in range(2,i+1):
            R[(i,j)] = R[(i,j-1)] + \
                       (R[(i,j-1)]-R[(i-1,j-1)]) / ((1 << (2*(j-1)))-1)
        h = h / 2.0

    if show:
        precis = 5
        width = 8
        formstr = "%" + str(width) + '.' + str(precis)+'f'

        print('\nMétodo de Romberg')
        for i in range(1,k+1):
            for j in range(1,i+1):
                print(formstr % R[(i,j)], end=' ')
            print()
        print('')

    return R[(k,k)]
```

```python
def main():
    data_sample = [3.592, 3.110, 3.017, 2.865, 2.685]
    res = RombergMethod(data_sample, 0.2, show=True)

    print('Resultado: {}u²'.format(res))

if __name__ == "__main__":
    main()
```

En el arreglo data_sample se guardan los valores f(x) de la tabla suministrada por el ejercicio.



- Expresión general: $I_{kj} = \dfrac{4^{j-1} I_{k,j-1} - I_{k-1,j-1}}{4^{j-1} - 1}$
- Error de orden h^{2j}
- Exacta para polinomios de grado 2j-1

```
Método de Romberg
 2.51080
 2.46220  2.44600

Resultado: 2.446u²
```