MODELADO Y SIMULACIÓN
Reglas de Newton Cotes

CLASE 4

Ing. Omar Cáceres

Integración Numérica (Reglas de Newton-Cotes

Las reglas de Newton-Cotes son una familia de métodos de integración numérica utilizados para aproximar integrales definidas. Se basan en la interpolación de la función a integrar mediante polinomios y en la evaluación de la integral de estos polinomios.

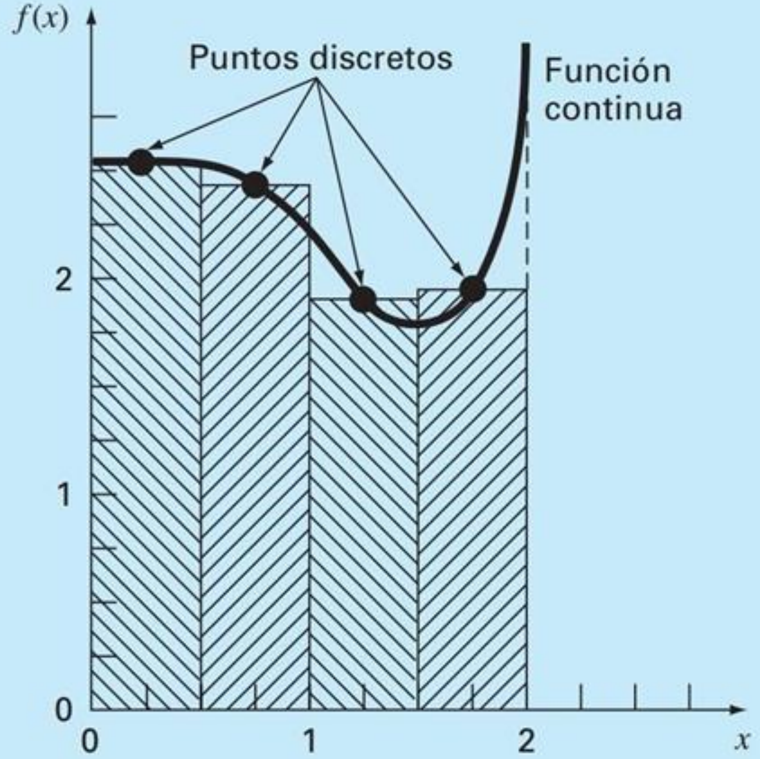
a)

$$\int_0^2 \frac{2 + \cos\,(1 + x^{3/2})}{\sqrt{1 + 0.5\ \text{sen}\ x}}\ e^{\,0.5x}\ dx$$

b)

x	$f(x)$
0.25	2.599
0.75	2.414
1.25	1.945
1.75	1.993

c)



Regla del rectángulo medio compuesta

$$\int_a^b f_{(x)}dx \approx h\sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right)$$

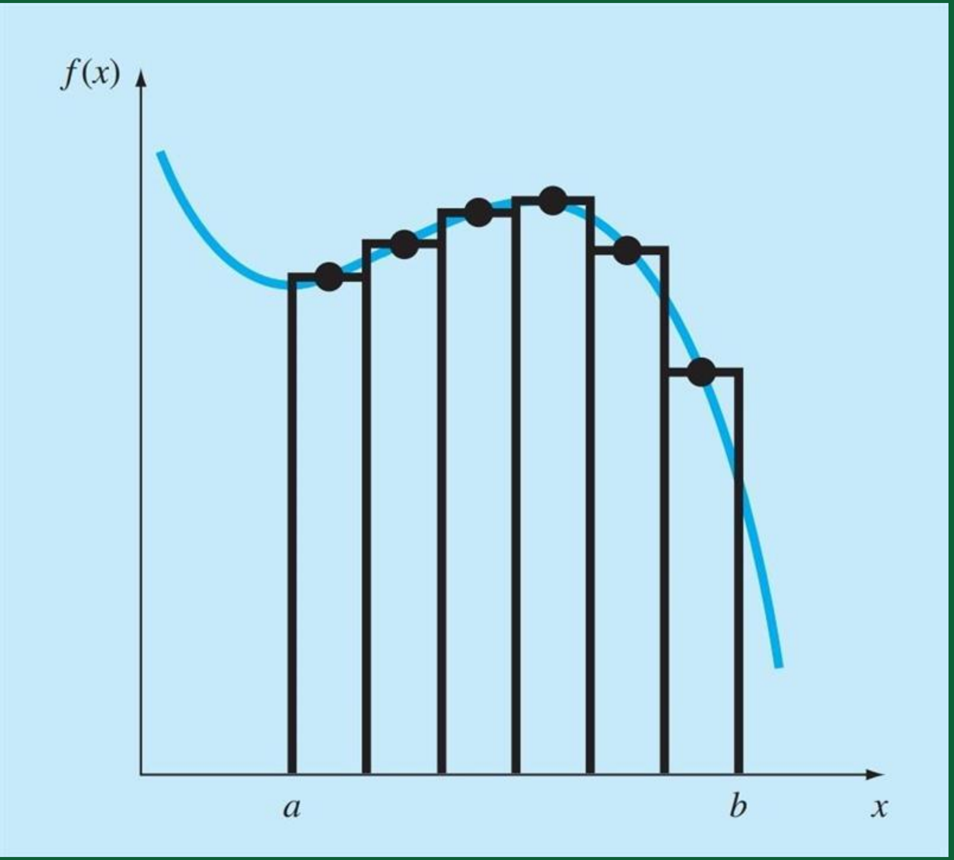
$$\int_a^b f_{(x)}dx \approx h\sum_{i=0}^{n-1} f(\bar{x}_i)$$

Donde

$\bar{x}_{i=}\, punto\ medio$

i	x_i	$\bar{x}_i$	$f(\bar{x}_i)$

$$h = \frac{(b-a)}{n}$$



MODELADO Y SIMULACIÓN
Reglas de Newton Cotes

CLASE 4

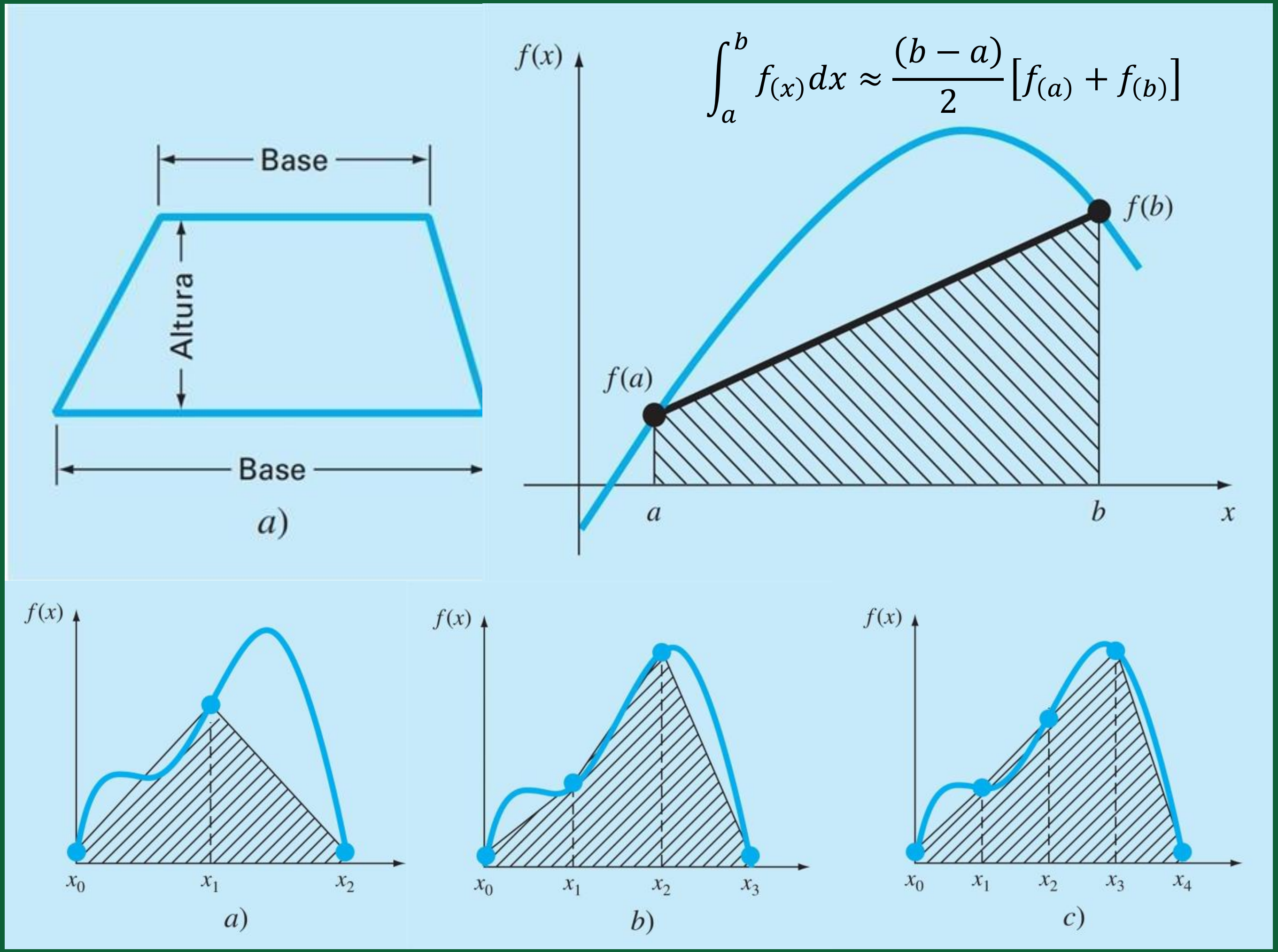
Ing. Omar Cáceres

Existen varias reglas de Newton-Cotes, dependiendo del número de puntos utilizados en la interpolación.

Regla del trapecio (Newton-Cotes de orden 1): Utiliza dos puntos para aproximar la integral como la suma de áreas de un trapezoide.

Forma compuesta

Esta consiste en dividir en varios subintervalos a un paso constante



Forma compuesta

Se divide el área de subintervalos para mejorar la precisión, acá se resume la formula:

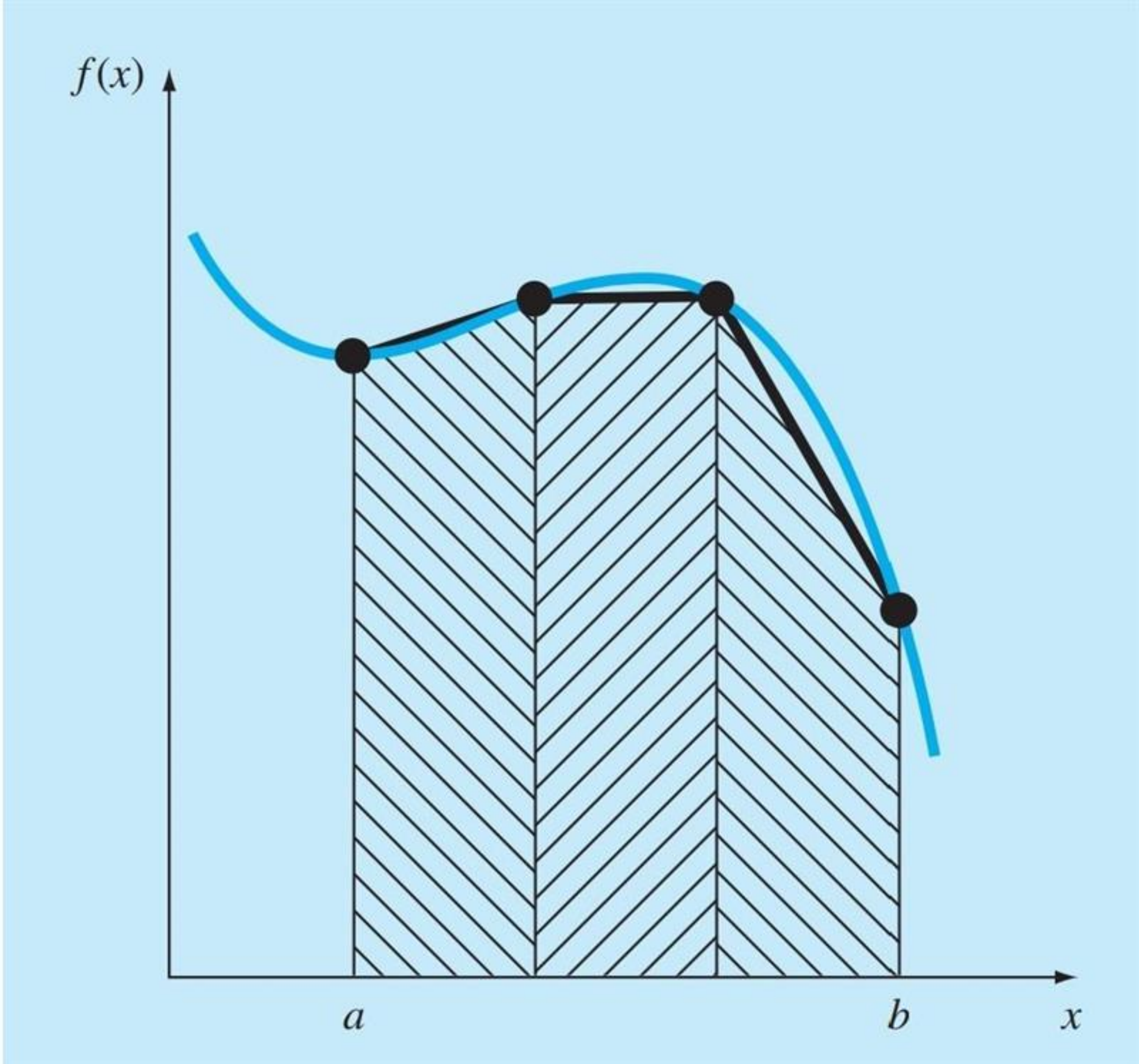
$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{n-1} f(a+ih) + f(b)\right)$$

Donde h es el paso, y n el número de subintervalos:

$$h = \frac{(b-a)}{n}$$

Error de truncamiento

$$E_T = -\frac{(b-a)^3}{12n^2} f''_{(\xi)}$$



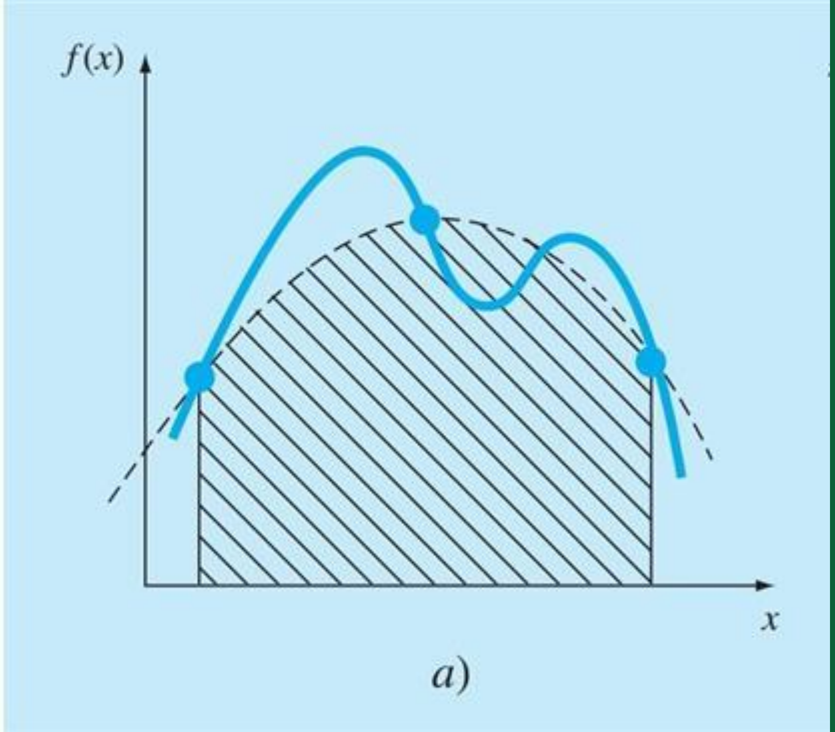
Simpson

Regla de Simpson 1/3, (Newton-Cotes)

a) Usa tres puntos y ajusta un polinomio cuadrático para mejorar la precisión. Para esta regla (n) solo puede ser par

$$\int_a^b f_{(x)}dx \approx \frac{h}{3}\left(f_{(a)} + 4f\left(\frac{a+b}{2}\right) + f_{(b)}\right)$$

$$h = \frac{(b-a)}{2}$$



Error de truncamiento

$$E = -\frac{(b-a)^5}{2880} f^{(4)}(\xi)$$

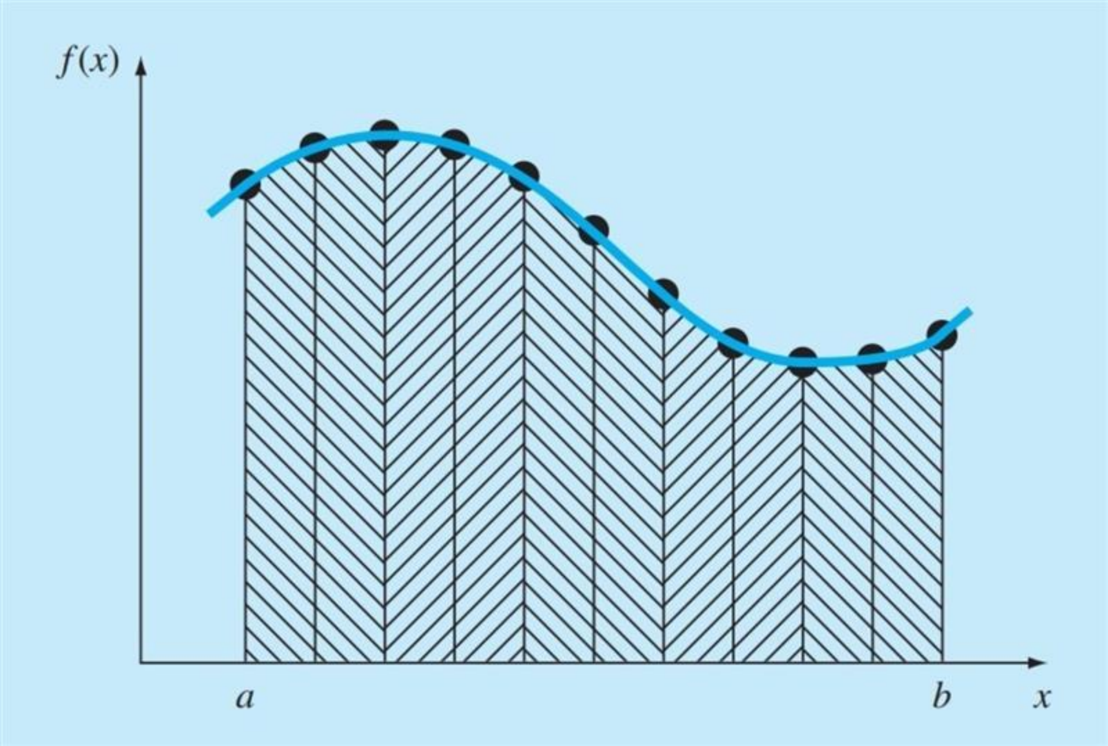
$donde\ \xi \epsilon [a, b]$

$$E = -\frac{h^5}{90} f^{(4)}(\xi)$$

Simpson

b) Reglas de Simpson 1/3 compuesta

$$\int_a^b f_{(x)}dx \approx \frac{h}{3}\left(f_{(a)} + 4\sum_{impares} f_{(a+ih)} + 2\sum_{pares} f_{(a+ih)} + f_{(b)}\right)$$



Error de truncamiento

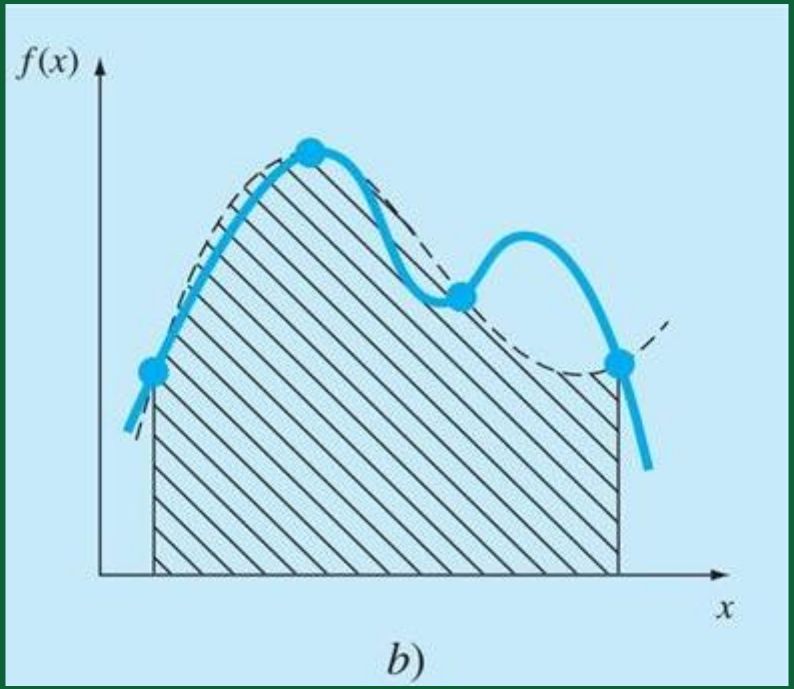
$$E_{comp} = -\frac{(b-a)^5}{180n^4} f^{(4)}(\xi)$$

$$E_{comp} = -\frac{(b-a)}{180} h^4 f^{(4)}(\xi)$$

$$\begin{cases} h = \dfrac{(\mathrm{b}-\mathrm{a})}{n} \\ \text{n es par} \end{cases}$$

Simpson

b) Reglas de Simpson 3/8 y otras de orden superior: Utilizan más puntos y polinomios de mayor grado para obtener aproximaciones más precisas. Es una extensión de la regla 1/3 pero esta usa 4 puntos. n=3



$$\boxed{\int_a^b f(x)dx \approx \frac{3h}{8}\left(f_{(a)} + 3f(x_1) + 3f(x_2) + f_{(b)}\right)}$$

Error de truncamiento

$$h = \frac{(b-a)}{3}$$

$$\boxed{E_T = -\frac{3}{80}h^5 f^{(4)}(\xi)}$$

Simpson 3/8 compuesta, n debe ser múltiplo de 3

$$\int_a^b f(x)dx \approx \frac{3h}{8}\left(f_{(a)} + 3\sum_{impares} f(x_i) + 3\sum_{pares} f\left(x_j\right) + 2\sum_{mult.3} f(x_i) + f_{(b)}\right)$$

$$h = \frac{(b-a)}{3n}$$

Error de truncamiento

$$E = -\frac{(b-a)^5}{6480} f^{(4)}(\xi)$$

```
import numpy as np

# Definimos la función a integrar
def funcion(x):
    return np.sin(x)  # Puedes cambiarla por otra función
# Límites de integración
a = 0  # Límite inferior
b = np.pi  # Límite superior

# Número de subintervalos
n = 10  # Ajusta según la precisión deseada

# Paso
h = (b - a) / n

# Puntos medios
x_medio = np.linspace(a + h/2, b - h/2, n)

# Aplicamos la regla del rectángulo medio compuesta
integral = h * np.sum(funcion(x_medio))
print(f"Integral aproximada con la regla del rectángulo medio compuesta:
{integral:.4f}")
```

```
import numpy as np

# Definimos la función a integrar
def funcion(x):
    return np.sin(x)  # Puedes cambiarla por cualquier otra función

# Límites de integración
a = 0  # Límite inferior
b = np.pi/2  # Límite superior

# Evaluamos la función en los extremos
fa = funcion(a)
fb = funcion(b)

# Aplicamos la regla del trapecio simple
integral = (b - a) / 2 * (fa + fb)

print(f"Integral aproximada con la regla del trapecio simple: {integral:.4f}")
```

```
import numpy as np

# Definimos la función a integrar
def funcion(x):
    return np.sin(x)  # Puedes cambiarla por otra función

# Límites de integración
a = 0  # Límite inferior
b = np.pi  # Límite superior
# Número de subintervalos
n = 10  # Ajusta según la precisión deseada

# Paso
h = (b - a) / n

# Puntos de evaluación
x = np.linspace(a, b, n + 1)
y = funcion(x)

# Aplicamos la regla del trapecio compuesta
integral = (h / 2) * (y[0] + 2 * np.sum(y[1:n]) + y[-1])
print(f"Integral aproximada con la regla del trapecio compuesta: {integral:.4f}")
```

```python
import numpy as np

# Definimos la función a integrar
def funcion(x):
    return np.sin(x)  # Puedes cambiarla por cualquier otra función

# Límites de integración
a = 0  # Límite inferior
b = np.pi  # Límite superior

# Punto medio
m = (a + b) / 2

# Evaluamos la función en los tres puntos
fa = funcion(a)
fm = funcion(m)
fb = funcion(b)

# Aplicamos la regla de Simpson simple
integral = (b - a) / 6 * (fa + 4 * fm + fb)

print(f"Integral aproximada con Simpson simple: {integral:.4f}")
```

```
import numpy as np

# Definimos la función a integrar
def funcion(x):
    return np.sin(x)  # Puedes cambiarla por cualquier otra función

# Límites de integración
a = 0  # Límite inferior
b = np.pi  # Límite superior
# Número de subintervalos (debe ser par)
n = 10  # Ajusta según la precisión deseada

# Paso
h = (b - a) / n
# Puntos
x = np.linspace(a, b, n + 1)
y = funcion(x)

# Aplicamos la regla de Simpson compuesta
S = y[0] + y[-1] + 4 * np.sum(y[1:n:2]) + 2 * np.sum(y[2:n-1:2])
integral = (h / 3) * S
print(f"Integral aproximada con Simpson compuesta: {integral:.4f}")
```

```
import numpy as np
# Definimos la función a integrar
def funcion(x):
return np.sin(x)  # Puedes cambiarla por otra función

# Límites de integración
a = 0  # Límite inferior
b = np.pi  # Límite superior
# Paso
h = (b - a) / 3
# Puntos de evaluación
x1 = a + h
x2 = a + 2*h

# Evaluamos la función en los puntos
fa = funcion(a)
fx1 = funcion(x1)
fx2 = funcion(x2)
fb = funcion(b)
# Aplicamos la regla de Simpson 3/8 simple
integral = (3 * h / 8) * (fa + 3 * fx1 + 3 * fx2 + fb)
print(f"Integral aproximada con Simpson 3/8 simple: {integral:.4f}")
```

Regla de Simpson 3/8 Compuesta

```
import numpy as np

# Definimos la función a integrar
def funcion(x):
    return np.sin(x)  # Puedes cambiarla por otra función
# Límites de integración
a = 0  # Límite inferior
b = np.pi  # Límite superior

# Número de subintervalos (debe ser múltiplo de 3)
n = 9  # Ajusta según la precisión deseada
# Paso
h = (b - a) / n
# Puntos de evaluación
x = np.linspace(a, b, n + 1)
y = funcion(x)

# Aplicamos la regla de Simpson 3/8 compuesta
S = y[0] + y[-1] + 3 * np.sum(y[1:n:3]) + 3 * np.sum(y[2:n:3]) + 2 *
np.sum(y[3:n-1:3])
integral = (3 * h / 8) * S
print(f"Integral aproximada con la regla de Simpson 3/8: {integral:.4f}")
```