Integrales

Importación de las librerías necesarias

In [7]:
```
import numpy as np # biblioteca de cálculo numérico y análisis de datos
import pylab as plt # biblioteca para la generación de gráficos a partir de
listas o arrays
from sympy import * # Librería de Cálculo
from sympy.plotting import plot as symplot # Librería para los gráficos
from sympy.abc import x, y, h # Carga de un simbólico "x"
from sympy.plotting.pygletplot import PygletPlot as Plot # Librería para lo
s gráficos
import math
```

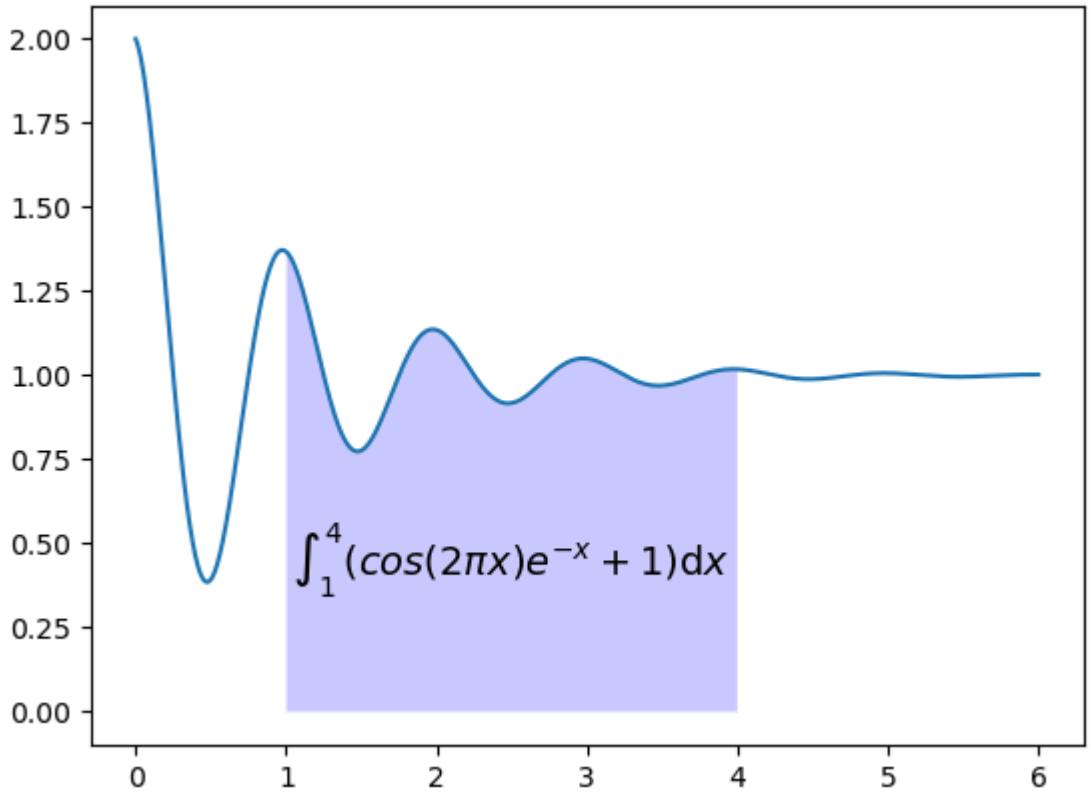
In [8]:
```
# Para imprimir todas las líneas
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
# Solo la última
#InteractiveShell.ast_node_interactivity = "last_expr"
```

Definición de área bajo una curva

Dada la gráfica de una función, se puede definir la integral definida como el área bajo la función entre dos valores a y b.

Así por ejemplo para la función $cos(2\pi x)e^{-x} + 1$ entre los valores 1 y 4 se puede representar como

In [2]:
```
from matplotlib import pyplot as plt

a=0
b=6
n=1000

# Creamos la función simbólica
f = cos(2*pi*x)*exp(-x)+1
# Definimos los subintervalos entre a y b como n valores espaciados uniform
emente
xvalues = np.linspace(a,b,n)
# Obtenemos los valores correspondientes para la y
yvalues = lambdify(x, f)(xvalues)

plt.plot(xvalues,yvalues)     # Dibujamos la curva a partir de sus coordena
das x, y
# Rellenamos el área de integración en color azul
plt.fill_between(xvalues,y1=yvalues,y2=0,where=(xvalues>=1)&(xvalues<=4), f
acecolor='blue',alpha=0.2)
# Añadimos un texto explicativo en la posición (2.5, 0.4)
plt.text(2.5,0.4,r"$\int_{1}^4(cos(2πx)e^{-x}+1)\mathrm{d}x$",horizontalali
gnment='center',fontsize=14)
plt.show()
```



Una forma de calcular una aproximación del área bajo una curva sería mediante una suma finita. De esta forma, se define la Suma de Riemann como una aproximación del valor de una integral mediante una suma finita.

La suma se calcula dividiendo la región en formas (rectángulos, trapezoides, cuadrados, polinomios cuadráticos o cúbicos, etc.) que si las juntamos todas se forma un zona similar a la región que se está midiendo. Después, calculando el área para cada una de estas formas y sumándolas todas, se puede encontrar una aproximación numérica para una integral definida.

Hay que tener en cuenta que la región rellenada por las formas pequeñas generalmente no es exactamente la misma forma que la región que se está midiendo, con lo que la suma de Riemann será una aproximación diferente del área que se está midiendo. El error que se comete, se puede reducir utilizando formas cada vez más pequeñas. Así, a medida que las formas se hacen cada vez más pequeñas, la suma se acerca a la integral buscada.

Una de las formas más simples es el rectángulo. Así pues, vamos a aproximar el área bajo la curva mediante una suma de áreas de rectángulos. Para ello, primeramente, vamos a realizar una partición del intervalo $[a, b]$ en n particiones de longitud Δx.

Para calcular la altura de los rectángulos vamos a elegir dos valores el valor de la función en el punto inicial izquierdo y el valor en el punto final derecho de cada elemento de la partición. Estas sumas de las áreas de esos rectángulos se denominarán suma izquierda en un caso, y suma derecha en el otro.

Así pues, tomando cada intervalo de igual longitud $\Delta x = \frac{(b-a)}{n}$, con n número de intervalos definidos mediante la partición $\{a = x_1, x_2, \ldots, x_n, x_{n+1} = b\}$, y tomando como alturas de los rectángulos los valores de $f(x_i)$ y $f(x_{i+1})$ respectivamente, se obtienen las áreas de los rectángulos.

Ahora solo queda sumar las áreas de los rectángulos para obtener una aproximación, así la Suma Izquierda será

$$L_n = \sum_{i=1}^{n} \Delta x f(x_i) = \Delta x \sum_{i=1}^{n} f(x_i) = \frac{b-a}{n} \sum_{i=1}^{n} f(x_i),$$

y la Suma Derecha

$$R_n = \sum_{i=1}^{n} \Delta x f(x_{i+1}) = \Delta x \sum_{i=1}^{n} f(x_{i+1}) = \frac{b-a}{n} \sum_{i=1}^{n} f(x_{i+1})$$

Esto se puede implementar en python de la forma siguiente

In [10]:
```
# Función que calcula la suma izquierda donde
# function: función a integrar de forma simbólica
# a: límite inferior de integración
# b: límite superior de integración
# n: número de intervalos entre a y b

def L_n(function, a, b, n):
    xvalues = np.linspace(a, b, n) # Devuelve n valores espaciados uniforme
mente en el intervalo [a, b]

    # Obtenemos los valores correspondientes a f(xi)
    yvalues = lambdify(x, function)(xvalues[:-1])

    Lsum = 0
    for i in yvalues:
        Lsum += i

    Ax = (b-a)/n
    Lsum = Ax * Lsum

    return Lsum

# Función que calcula la suma derecha
def R_n(function, a, b, n):
    xvalues = np.linspace(a, b, n) # Devuelve n valores espaciados uniforme
mente en el intervalo [a, b]

    # Obtenemos los valores correspondientes a f(xi+1)
    yvalues = lambdify(x, function)(xvalues[1:])

    Rsum = 0
    for i in yvalues:
        Rsum += i

    Ax = (b-a)/n
    Rsum = Ax * Rsum

    return Rsum

# La siguiente función llama a las dos anteriores y devuelve una tupla con
sus valores
def Riemann_sum(function, a, b, n):
     Ln = L_n(function, a, b, n)
     Rn = R_n(function, a, b, n)
     return (Ln, Rn)

## ejemplo
n = 10 #100
a = 0
b = np.pi

function = sin(x)**2 * x**2 * 10**-x
Ln, Rn = Riemann_sum(function, a, b, n)

print(f"La Suma Izquierda de: {function} de {a:.2f} a {b:.2f} es {Ln:.6f}")
print(f"La Suma Derecha   de: {function} de {a:.2f} a {b:.2f} es {Rn:.6f}")
```

```
La Suma Izquierda de: x**2*sin(x)**2/10**x de 0.00 a 3.14 es 0.089965
La Suma Derecha   de: x**2*sin(x)**2/10**x de 0.00 a 3.14 es 0.089965
```

También podemos usar la regla trapezoidal que consiste en aproximar la altura de la función a una constante en cada subintervalo $[a, b]$ como si fuera un polinomio lineal. En este caso, la región bajo el polinomio lineal es un trapezoide.

En cada sub-intervalo x_i y x_{i+1}, el trapezoide tiene de área

$$\frac{f(x_i) + f(x_{i+1})}{2}\Delta x$$

Y sumando estas áreas en todos los sub-intervalos obtenemos:

$$\int_a^b f(x)dx = \sum_{i=1}^{n} \frac{f(x_i) + f(x_{i+1})}{2}\Delta x$$

Esto se puede implementar fácilemente en python mediante el siguiente código

In [11]:
```
def suma_trapezoidal(function, a, b, n):
    Ax = (b - a)/n

    xvalues = np.linspace(a, b, n) # Devuelve n valores espaciados uniforme
mente en el intervalo [a, b]

    # Obtenemos los valores de f(xi+1)
    yvalues = lambdify(x, function)(xvalues)

    sum = 0
    for i in range(len(yvalues)-1):
      sum += (yvalues[i]+yvalues[i+1])/2

    sum = Ax * sum

    return sum

n= 100
a = 0
b = np.pi

f = sin(x)**2 * x**2 * 10**-x

STn = suma_trapezoidal(f, a, b, n)
print(f"La Suma Trapezoidal de: {f} de {a} a {b:.2f} es {STn:.6f}")
```

```
La Suma Trapezoidal de: x**2*sin(x)**2/10**x de 0 a 3.14 es 0.098944
```

Cálculo de integrales en Python

Para calcular integrales definidas, la librería *scipy* proporciona un comando de cálculo inmediato *quad*. Este comando devuelve una tupla cuyo primer elemento es el valor de la integral y tiene la sintaxis siguiente *quad(función, a, b)*

In [12]:
```
# scipy es una biblioteca de herramientas y algoritmos matemáticos, entre ellos quad para calcular integrales
from scipy.integrate import quad

# Necesitamos definir una función de usuario para poder utilizar la función quad
# Como mínimo, incluirá el argumento x
def integrand(x):
    return x**2+x+1

a=0
b=3
J = quad(integrand,a,b)
J # Tupla valor del integral y error absoluto
```

Out[12]: (16.5, 1.8318679906315083e-13)

Con Python, se pueden resolver integrales indefinidas de forma simbólica con la ayuda de la librería *SymPy*. Para ello, vamos a utilizar el objeto *integrate* cuya sintaxis es la siguiente: *integrate(funcion, variable)*.

In [13]:
```
from sympy import integrate

fx = x**3-6*x # Creando la función simbólica

dx1=integrate(fx,x)
display(dx1)

fx2 = x**2-3*x + 2
dx2=integrate(fx2)
display(dx2)
```

$$\frac{x^4}{4} - 3x^2$$

$$\frac{x^3}{3} - \frac{3x^2}{2} + 2x$$

La función *Integral* da el mismo resultado

In [24]:
```
from sympy import Integral

fx = x**3-6*x # Creando la función simbólica
dx1 = Integral(fx,x).doit()
display(dx1)

fx2 = x**2-3*x+2
dx2=Integral(fx2).doit()
display(dx2)

dx3=Integral(x**2-3*x+2).doit()
display(dx3)
```

$$\frac{x^4}{4} - 3x^2$$

$$\frac{x^3}{3} - \frac{3x^2}{2} + 2x$$

$$\frac{x^3}{3} - \frac{3x^2}{2} + 2x$$

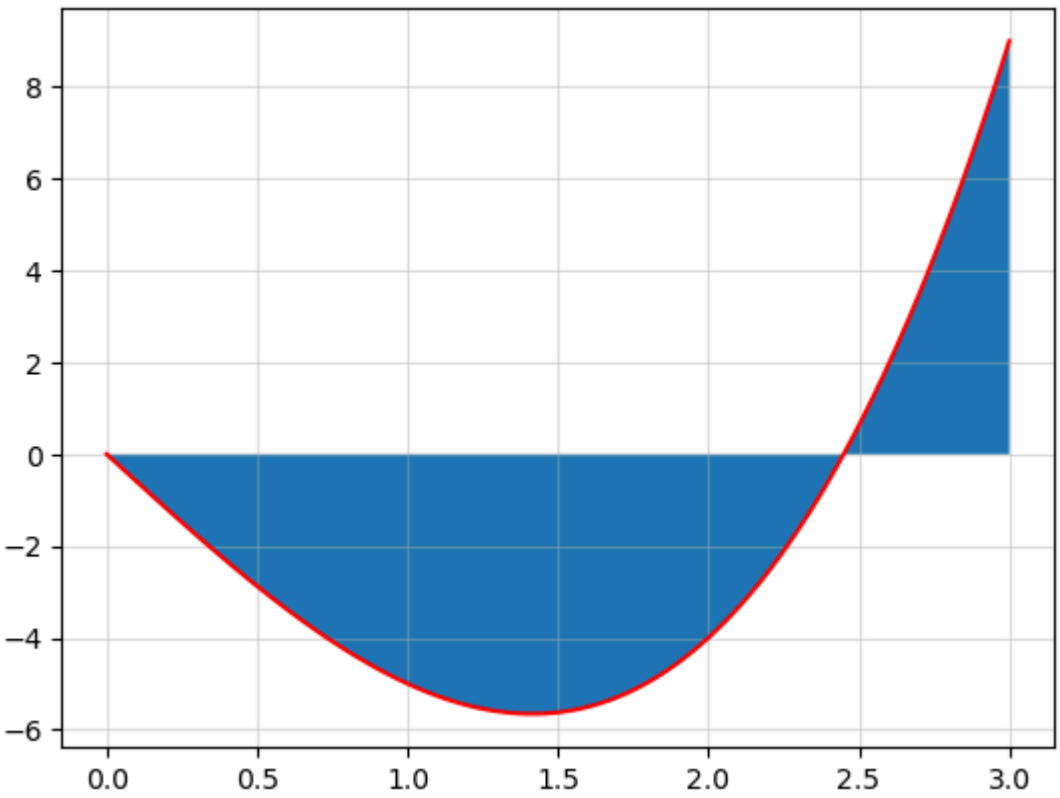
En *SymPy*, también se pueden calcular integrales definidas mediante las funciones *integrate*, considerando el hecho que deben añadirse los límites de integración:

In [15]:
```
from sympy import integrate

fx = x**3-6*x # Creando la función simbólica

Id2 = integrate(fx,(x,0,3))
display(Id2)

xx = np.linspace(0,3,100)
yy = lambdify(x,fx)(xx)

plt.grid(alpha=0.4)
plt.plot(xx,yy,'r')
plt.fill_between(xx,yy)
```

$$-\frac{27}{4}$$

Out[15]: [<matplotlib.lines.Line2D at 0x1c4398f8750>]

Out[15]: <matplotlib.collections.PolyCollection at 0x1c43b445fd0>



Cuando la expresión a integrar incluye más de una variable simbólica, es necesario especificar de manera explícita la variable respecto a la cual se integra, de lo contrario Python nos mostrará un error.

In [16]:
```
from sympy.abc import a,b,c
from sympy import integrate

f = a*x**2+b*x+c
integrate(f,x)
```

Out[16]: $$\frac{ax^3}{3} + \frac{bx^2}{2} + cx$$

En SymPy, también se puede calcular una integral definida mediante *integrate*, considerando el hecho que deben añadirse los límites de integración, mediante la sintaxis:

In [17]:
```
from sympy import cos,pi
from sympy import integrate, init_printing
from sympy.abc import x

f = x**2 - 3*x + 2
ID = integrate(f)
ID

# o integrales definidas
Res1 = integrate(cos(x), (x,0,pi/2.0))
print("la integral definida de cos(x) entre 0 y pi/2 es {}".format(Res1))

Res2 = integrate(x, (x, 0, 5))
print("la integral definida de x entre 0 y 5 es {}".format(N(Res2)))
```

Out[17]:
$$\frac{x^3}{3} - \frac{3x^2}{2} + 2x$$

```
la integral definida de cos(x) entre 0 y pi/2 es 1
la integral definida de x entre 0 y 5 es 12.5000000000000
```

Integrales múltiples

Aunque se van a resolver integrales dobles, la sintaxis y metodología se puede extrapolar de forma fácil a cualquier integral múltiple.

Para resolver

$$\int_a^b \int_c^d 1 \, dydx,$$

hay que recordar que este tipo de integrales dobles se pueden resolver de forma iterada

$$I_1 = \int_c^d 1 \, dy \Rightarrow I = \int_a^b I_1 \, dx.$$

En Python se puede realizar exactamente lo mismo

In [31]:
```
from sympy.abc import x,y,z,a,b,c,d
from sympy import simplify

I1 = integrate(1, (y,c,d))
simplify(integrate(I1, (x,a,b)))
```

Out[31]:
$$(a - b)(c - d)$$

También existen funciones para el cálculo de integrales dobles, triples o múltiples. La mecánica para este tipo de integración se ha agrupado en las funciones *dblquad, tplquad* y *nquad*. Hay que remarcar, que los límites de todas las integrales internas deben definirse como funciones.

La sintaxis general de la función *dblquad* es *dblquad(func, a, b, gfun, hfun)*. Donde, *func* es el nombre de la función a integrar, *a* y *b* son los límites inferior y superior de la variable *x*, respectivamente, mientras que *gfun* y *hfun* son los nombres de las funciones que definen los límites inferior y superior de la variable *y*.

Como ejemplo, veamos el cálculo de la integral doble

$$\int_0^{1/2} \int_0^{\sqrt{1-4y^2}} 16xy \, dxdy$$

In [32]:
```
from scipy.integrate import dblquad

# definición de las funciones
def integrand(x, y):
  return 16*x*y

def xlimit(y):
  return sqrt(1-4*y**2)

i = dblquad(integrand, 0, 0.5, 0, xlimit)
print(i)
```

```
(0.5, 1.7092350012594845e-14)
```

Ejercicios

Ejercicio 1

Dadas las funciones:

- $f(x) = x^2 - 2x + 3$ en $[-2, 3]$
- $f(x) = sen(2x)$ en $[-1, 5]$
- $f(x) = -x^2 + 8x + 5$ en $[-2, 3]$

calcula:

1. El valor de la suma izquierda Ln para $n = 6$
2. El valor de la suma derecha Rn para $n = 6$
3. El valor de la suma trapezoidal STn para $n = 6$
4. Valor exacto de la integral (utilizar la función *quad*)
5. El error relativo (Valor real menos valor aproximado y dividido por el valor real) para las tres aproximaciones.

In [18]:
```
from scipy.integrate import quad

f1_ej1 = x**2-2*x+3
f2_ej1 = sin(2*x)
f3_ej1 = -x**2 +8*x +5

def f1_ej1_quad(x):
    return x**2-2*x+3

def f2_ej1_quad(x):
    return sin(2*x)

def f3_ej1_quad(x):
    return -x**2 +8*x +5

print("Resultado de 1) y 2)")
print(f"Suma izquierda: {round(L_n(f1_ej1,-2,3,6),4)}, suma derecha: {round
(R_n(f1_ej1,-2,3,6),4)}")
print(f"Suma izquierda: {round(L_n(f2_ej1,-1,5,6),4)}, suma derecha: {round
(R_n(f2_ej1,-1,5,6),4)}")
print(f"Suma izquierda: {round(L_n(f3_ej1,-2,3,6),4)}, suma derecha: {round
(R_n(f3_ej1,-2,3,6),4)}")
print("Resultado de 3)")
print(f"Suma trapezoidal: {round(suma_trapezoidal(f1_ej1,-2,3,6),4)}")
print(f"Suma trapezoidal: {round(suma_trapezoidal(f2_ej1,-1,5,6),4)}")
print(f"Suma trapezoidal: {round(suma_trapezoidal(f3_ej1,-2,3,6),4)}")
print("Resultado de 4)")
print(f"Integral con quad: {round(quad(f1_ej1_quad, -2, 3)[0],4)}")
print(f"Integral con quad: {round(quad(f2_ej1_quad, -1, 5)[0],4)}")
print(f"Integral con quad: {round(quad(f3_ej1_quad, -2, 3)[0],4)}")
print("Resultado de 5)")
print("f1")
print(f"Error relativo con izq.: {round((quad(f1_ej1_quad, -2, 3)[0] - L_n
(f1_ej1,-2,3,6))/quad(f1_ej1_quad, -2, 3)[0],4)}")
print(f"Error relativo con der.: {round((quad(f1_ej1_quad, -2, 3)[0] - R_n
(f1_ej1,-2,3,6))/quad(f1_ej1_quad, -2, 3)[0],4)}")
print(f"Error relativo con tra.: {round((quad(f1_ej1_quad, -2, 3)[0] - suma
_trapezoidal(f1_ej1,-2,3,6))/quad(f1_ej1_quad, -2, 3)[0],4)}")
print("f2")
print(f"Error relativo con izq.: {round((quad(f2_ej1_quad, -1, 5)[0] - L_n
(f2_ej1,-1,5,6))/quad(f2_ej1_quad, -1, 5)[0],4)}")
print(f"Error relativo con der.: {round((quad(f2_ej1_quad, -1, 5)[0] - R_n
(f2_ej1,-1,5,6))/quad(f2_ej1_quad, -1, 5)[0],4)}")
print(f"Error relativo con tra.: {round((quad(f2_ej1_quad, -1, 5)[0] - suma
_trapezoidal(f2_ej1,-1,5,6))/quad(f2_ej1_quad, -1, 5)[0],4)}")
print("f3")
print(f"Error relativo con izq.: {round((quad(f3_ej1_quad, -2, 3)[0] - L_n
(f3_ej1,-2,3,6))/quad(f3_ej1_quad, -2, 3)[0],4)}")
print(f"Error relativo con der.: {round((quad(f3_ej1_quad, -2, 3)[0] - R_n
(f3_ej1,-2,3,6))/quad(f3_ej1_quad, -2, 3)[0],4)}")
print(f"Error relativo con tra.: {round((quad(f3_ej1_quad, -2, 3)[0] - suma
_trapezoidal(f3_ej1,-2,3,6))/quad(f3_ej1_quad, -2, 3)[0],4)}")
```

```
Resultado de 1) y 2)
Suma izquierda: 20.8333, suma derecha: 16.6667
Suma izquierda: -0.1004, suma derecha: 0.2649
Suma izquierda: 12.5, suma derecha: 41.6667
Resultado de 3)
Suma trapezoidal: 18.75
Suma trapezoidal: 0.0822
Suma trapezoidal: 27.0833
Resultado de 4)
Integral con quad: 21.6667
Integral con quad: 0.2115
Integral con quad: 33.3333
Resultado de 5)
f1
Error relativo con izq.: 0.0385
Error relativo con der.: 0.2308
Error relativo con tra.: 0.1346
f2
Error relativo con izq.: 1.4749
Error relativo con der.: -0.2525
Error relativo con tra.: 0.6112
f3
Error relativo con izq.: 0.625
Error relativo con der.: -0.25
Error relativo con tra.: 0.1875
```

In [ ]:

```
Resultado de 1) y 2)
Suma izquierda: 20.8333, suma derecha: 16.6667
Suma izquierda: -0.1004, suma derecha: 0.2649
Suma izquierda: 12.5000, suma derecha: 41.6667
Resultado de 3)
Suma trapezoidal: 18.7500
Suma trapezoidal: 0.0822
Suma trapezoidal: 27.0833
Resultado de 4)
Integral con quad: 21.6667
Integral con quad: 0.2115
Integral con quad: 33.3333
Resultado de 5)
f1
Error relativo con izq.: 0.0385
Error relativo con der.: 0.2308
Error relativo con tra.: 0.1346
f2
Error relativo con izq.: 1.4749
Error relativo con der.: -0.2525
Error relativo con tra.: 0.6112
f3
Error relativo con izq.: 0.6250
Error relativo con der.: -0.2500
Error relativo con tra.: 0.1875
```

Ejercicio 2 (Regla de Simpson)

La regla de Simpson consiste en aproximar la integral de una función f con polinomios cuadráticos. Así, dada una partición p del intervalo $[a, b]$ para algún número par n. En cada subintervalo, el área debajo de f se aproxima con el área debajo de un polinomio cuadrático:

$$[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})]\frac{\Delta x}{3}$$

Sumando estas áreas para todos los subintervalos obtenemos la aproximación de la integral:

$$\int_a^b f(x)dx = \sum_{i=2}^{n-1}[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})]\frac{\Delta x}{3} = [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x$$

Implementa en python la Regla de Simpson y pruebala para las funciones del ejercicio 1.

In [19]:
```
def simpson(f,a,b,n):
  Ax = (b-a)/n
  xx = np.linspace(a,b,n)
  yy = lambdify(x,f)(xx)

  xx = xx[~np.isnan(yy)] # Elimina todos los nan (ej. sqrt(-1))
  yy = yy[~np.isnan(yy)] # Elimina todos los nan (ej. sqrt(-1))

  # sum(yy[0:-1:2]) -> suma todos los numeros desde 0 hasta el (final-1) co
n saltos de 2 en 2 (saltos pares 0,2,4,6,8,...)
  # sum(4*yy[1::2]) -> suma el resultado de la multiplicación de todos los
numeros desde 0 hasta el final con saltos de 2 en 2 (saltos impares 1,3,5,
7,...)
  # sum(yy[2::2]) -> suma todos los numeros desde 2 hasta el final con salt
os de 2 en 2 (saltos pares 2,4,6,8,...)
  simp = Ax/3 * (sum(yy[0:-1:2])+sum(4*yy[1::2])+sum(yy[2::2]))
  return simp

# mismo que el anterior
def simpson2(f,a,b,n):
  Ax = (b-a)/n
  xx = np.linspace(a,b,n)
  yy = lambdify(x,f)(xx)

  xx = xx[~np.isnan(yy)] # Elimina todos los nan (ej. sqrt(-1))
  yy = yy[~np.isnan(yy)] # Elimina todos los nan (ej. sqrt(-1))

  simp = 0
  for i in range(0,len(xx)-2,2):
    simp = simp + yy[i]+4*yy[i+1]+yy[i+2]
  return Ax/3 * simp

print("Metodo 1")
print(simpson(f1_ej1,-2,3,100))
print(simpson(f2_ej1,-1,5,100))
print(simpson(f3_ej1,-2,3,100))

print("Metodo 2")
print(simpson2(f1_ej1,-2,3,100))
print(simpson2(f2_ej1,-1,5,100))
print(simpson2(f3_ej1,-2,3,100))
```

```
Metodo 1
21.651683501683507
0.18656882638835257
33.66750841750842
Metodo 2
21.155007992381723
0.23886197018946806
32.00256776519402
```

In [ ]:

```
Metodo 1
21.651683501683507
0.18656882638835265
33.66750841750842
Metodo 2
21.155007992381723
0.23886197018946814
32.00256776519402
```

Ejercicio 3

Calcula:

1. $\int e^{4x} dx$
2. $\int x^5 \log x dx$
3. $\int \cos(\sin x) dx$
4. $\int_{-1}^{1} x^2 \, dx$
5. $\int_{-\pi}^{\pi} \cos(x) \, dx$
6. $\int_{4}^{\infty} \frac{1}{x^2-5x+4} \, dx$

In [36]:
```
f1_ej3 = exp(4*x)
f2_ej3 = x**5*log(x)
f3_ej3 = cos(sin(x))
f4_ej3 = x**2
f5_ej3 = cos(x)
f6_ej3 = 1/(x**2-5*x+4)

print("1)")
display(integrate(f1_ej3))
print("2)")
display(integrate(f2_ej3))
print("3)")
display(integrate(f3_ej3))
print("4)")
display(integrate(f4_ej3,(x,-1,1)))
print("5)")
display(integrate(f5_ej3,(x,-pi,pi)))
print("6)")
display(integrate(f6_ej3,(x,4,oo)))
```

1)

$$\frac{e^{4x}}{4}$$

2)

$$\frac{x^6 \log(x)}{6} - \frac{x^6}{36}$$

3)

$$\int \cos(\sin(x))\, dx$$

4)

$$\frac{2}{3}$$

5)

$$0$$

6)

$$\infty$$

In [ ]:

1)

$$\frac{e^{4x}}{4}$$

2)

$$\frac{x^6 \log(x)}{6 \log(10)} - \frac{x^6}{36 \log(10)}$$

3)

$$\int \cos(\sin(x))\, dx$$

4)

$$\frac{2}{3}$$

5)

$$0$$

6)

$$\infty$$

Ejercicio 4

Sean las funciones $f(x) = x$ y $g(x) = (x+1)^2$, calcula:

1. $\int_0^2 (f+g)dx$
2. $\int_0^2 f\,dx + \int_0^2 g\,dx$

Compara los resultados.

In [39]:
```
f_ej4 = x
g_ej4 = (x + 1)**2

integrate(f_ej4+g_ej4,(x,0,2))
integrate(f_ej4,(x,0,2)) + integrate(g_ej4,(x,0,2))
```

Out[39]:
$$\frac{32}{3}$$

Out[39]:
$$\frac{32}{3}$$

In [ ]:

$$\frac{32}{3}$$

$$\frac{32}{3}$$

Ejercicio 5

Sea la función $f(x) = \sin x$ y $k = 5$, calcula:

1. $\int_{-\pi/2}^{\pi} k\, f\, dx$
2. $k \int_{-\pi/2}^{\pi} f\, dx$

Compara los resultados.

In [46]:
```
from sympy.abc import k

f_ej5 = sin(x)

integrate(k*f_ej5,(x,-pi/2,pi)).subs(k,5)
(k*integrate(f_ej5,(x,-pi/2,pi))).subs(k,5)
```

Out[46]: 5

Out[46]: 5

In [ ]:

5

5

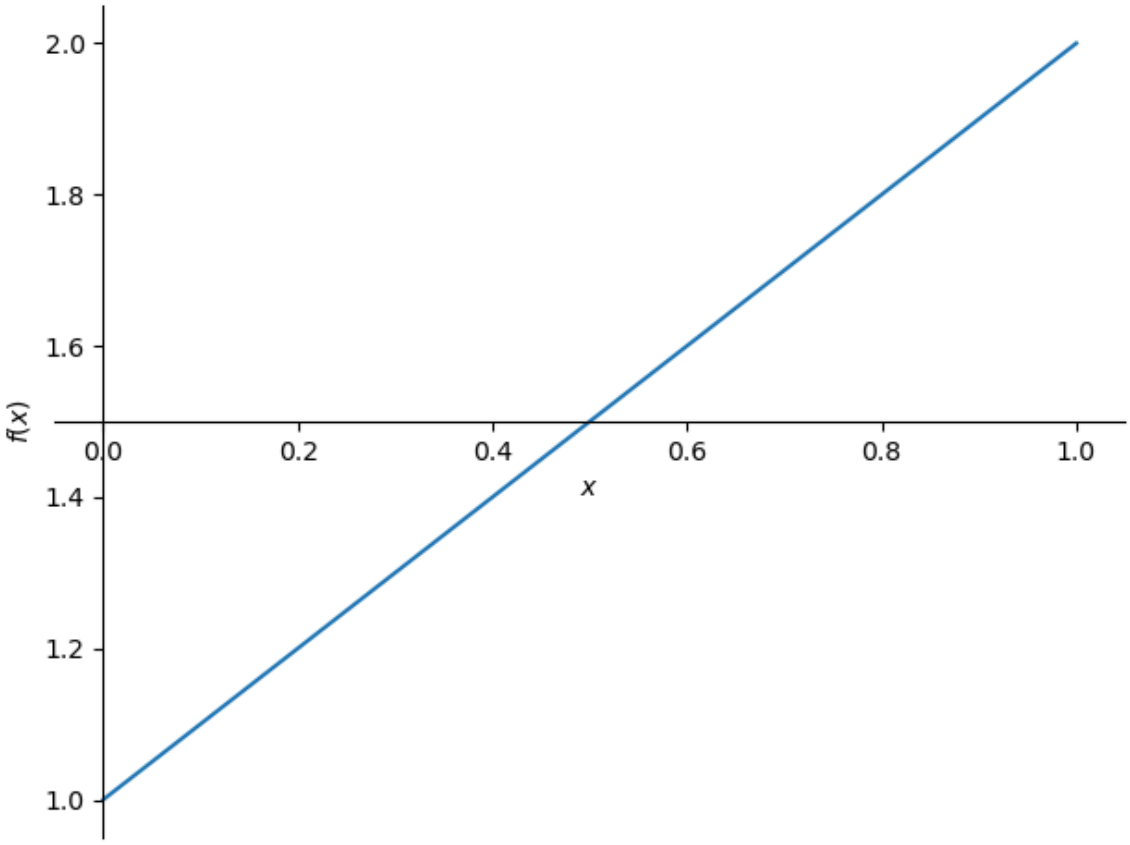
Ejercicio 6

Calcula el área de la región limitada por las siguientes gráficas utilizando el método de Simpson con $\Delta x = (b-a)/n = 0.01$.

1. $y = x + 1$, con $0 \leq x \leq 1$
2. $y = x^2 + 1$, con $1 \leq x \leq 2$

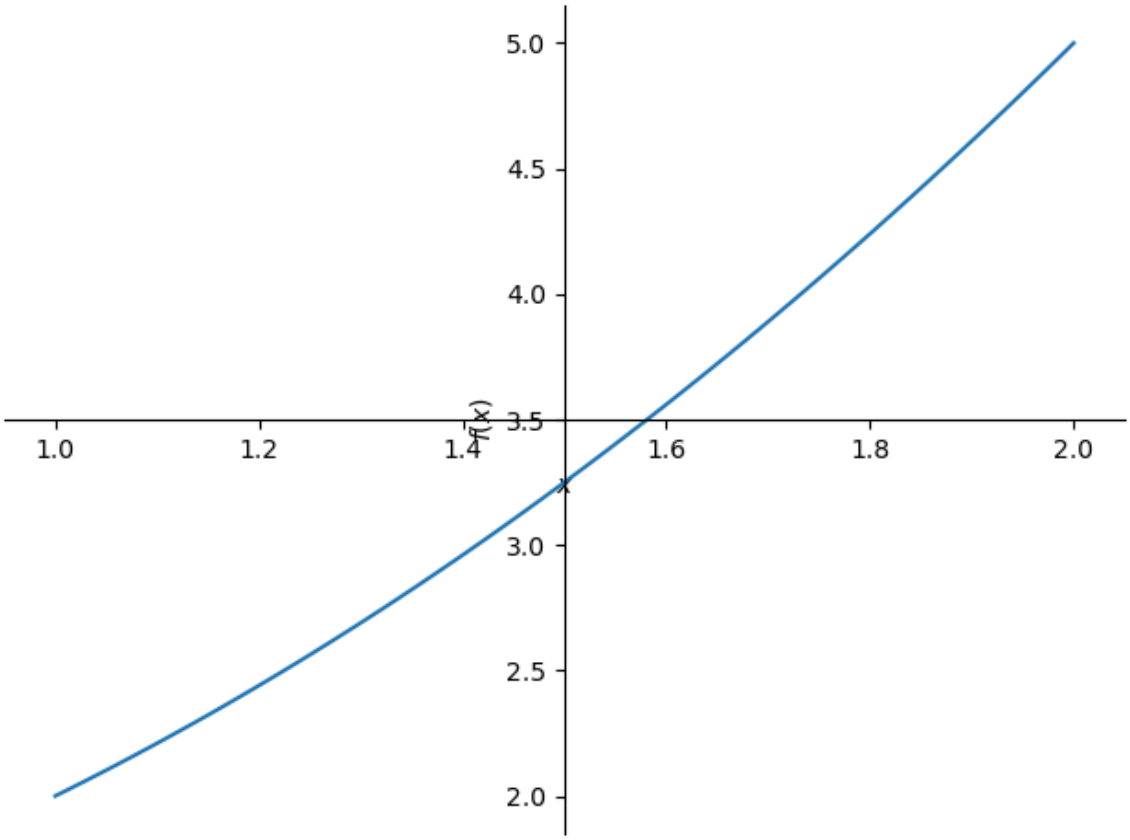
In [47]:
```
# 1
n = int((1-0)/0.01)
e61 = simpson(x+1,0,1,n)
print(f"Simpson: {e61}")
print(f"Exacta: {integrate((x+1),(x,0,1))}")

symplot(x+1,(x,0,1))

# 2
n = int((2-1)/0.01)
e62 = simpson(x**2+1,1,2,n)
print(f"Simpson: {e62}")
print(f"Exacta: {integrate((x**2+1),(x,1,2))}")

symplot(x**2+1,(x,1,2))
```

```
Simpson: 1.4983501683501688
Exacta: 3/2
```



Out[47]: <sympy.plotting.plot.Plot at 0x1c43c042d50>

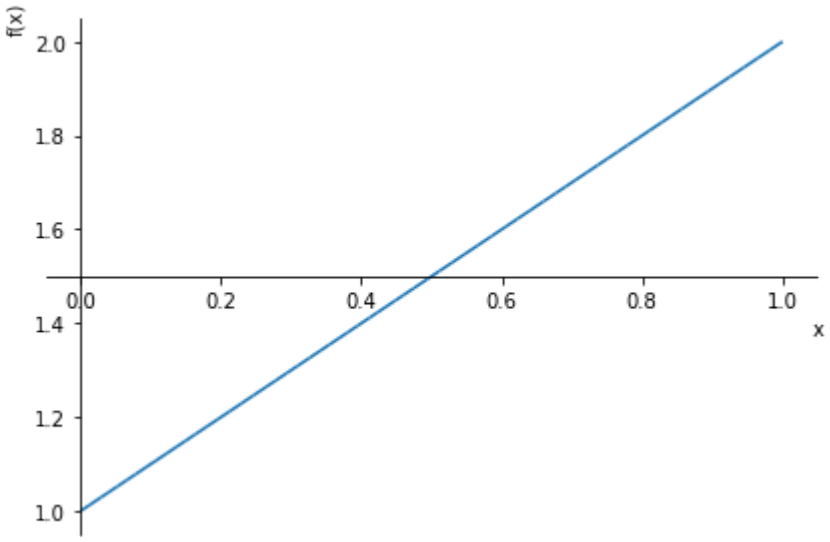
```
Simpson: 3.333400673400673
Exacta: 10/3
```



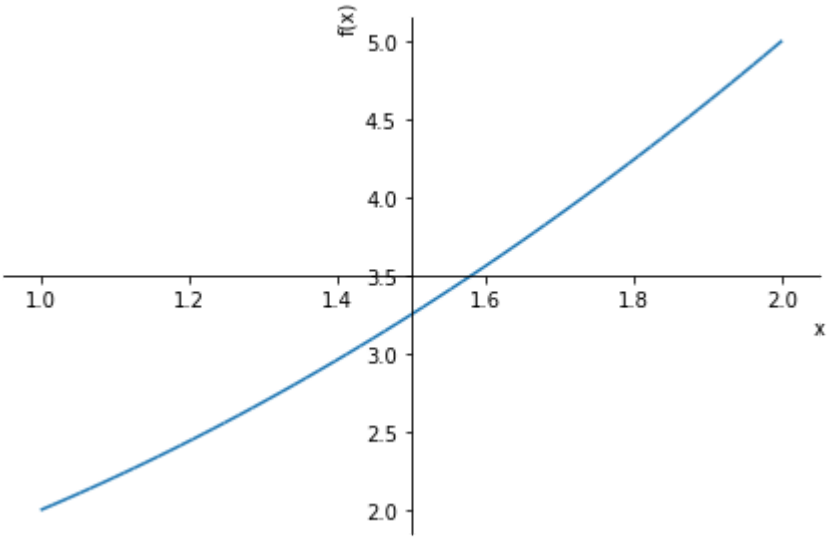
Out[47]: <sympy.plotting.plot.Plot at 0x1c43b828e10>

In [ ]:

```
Simpson: 1.4983501683501688
Exacta: 3/2
```




```
Simpson: 3.333400673400673
Exacta: 10/3
```



Out[ ]: <sympy.plotting.plot.Plot at 0x7f1c8aa70dd0>

Ejercicio 7

Calcula el volumen del sólido de revolución generado al girar alrededor del eje x las siguientes gráficas. Como en el ejercicio anterior utiliza el método de Simpson con $\Delta x = (b-a)/n = 0.01$ para obtener la integral definida correspondiente.

1. $y = \sqrt{x-1}$, la recta $x = 3$ y el eje de ordenadas.
2. $y = 1 - x^2$, y las rectas $x = -1$ y $x = 1$.

In [48]:
```
# 1
a = 0
b = 3
n = int((b-a)/0.01)
f71 = sqrt(x-1)
i71 = pi*integrate(f71**2,(x,a,b))
s71 = math.pi*simpson(f71,a,b,n)
print(f"Integral por revolución: {N(i71):.6f}, Simpson: {s71:.6f}")

# 2
a = -1
b = 1
n = int((b-a)/0.01)
f72 = 1 -x**2
i72 = pi*integrate(f72**2,(x,a,b))
s72 = math.pi*simpson(f72,a,b,n)
print(f"Integral por revolución: {N(i72):.6f}, Simpson: {s72:.6f}")
```

```
Integral por revolución: 4.712389, Simpson: 5.933310
Integral por revolución: 3.351032, Simpson: 4.167741

<lambdifygenerated-31>:2: RuntimeWarning: invalid value encountered in sqrt
  return sqrt(x - 1)
```

In [ ]:

```
Integral por revolución: 4.712389, Simpson: 5.933310
Integral por revolución: 3.351032, Simpson: 4.167741

<string>:2: RuntimeWarning: invalid value encountered in sqrt
```

Ejercicio 8

Calcula:

1. La integral doble $\iint_{\mathbb{R}^2} \frac{x^2}{2y} dxdy$, con $1 \leq x \leq 2$ y $1 \leq y \leq 4$
2. La integral triple $\iiint_{\mathbb{Q}} x^2 \sin(z) dxdydz$, con $0 \leq x \leq \sqrt{5}$, $0 \leq y \leq 2\pi$ y $0 \leq z \leq \arctan 2$
3. La integral doble $\iint_{\mathbb{R}^2} xydydx$, con $0 \leq x \leq 1$ y $1 - x \leq y \leq 1 - x^2$
4. La integral doble $\iint_{\mathbb{R}^2} (x + y^3) dydx$, con $1 \leq x \leq 4$ y $x \leq y \leq x^2$
5. La integral doble $\iint_{\mathbb{R}^2} dydx$, con $-1 \leq x \leq 3$ y $0 \leq y \leq x + 2$
6. Calcular el volumen bajo la superficie $f(x, y) = 6 - x - y$ en el rectángulo $[0, 3] \times [0, 2]$
7. Calcular el volumen bajo la superficie $f(x, y) = 6 - x - y$ en el área del plano XY delimitada por $0 \leq x \leq 1$ y $2x \leq y \leq -x^2 + 3$

In [68]:
```
from sympy import symbols, integrate, sqrt, pi, atan

# Definir las variables
x, y, z = symbols('x y z')

# Primera integral
integral1 = integrate(x**2/2*y, (x, 1, 2), (y, 1, 4))
print("1)")
display(integral1)

# Segunda integral
integral2 = integrate(integrate(integrate(x**2*sin(z), (x, 0, sqrt(5))),
(y, 0, 2*pi)), (z, 0, atan(2)))
print("2)")
display(integral2)

# Tercera integral
integral3 = integrate(integrate(x*y, (y, 1-x, 1-x**2)), (x, 0, 1))
print("3)")
display(integral3)

# Cuarta integral
integral4 = integrate(integrate(x+y**3, (y, x, x**2)), (x, 1, 4))
print("4)")
display(integral4)

# Quinta integral
integral5 = integrate(integrate(1, (y, 0, x+2)), (x, -1, 3))
print("5)")
display(integral5)

# Sexta integral
integral6 = integrate(integrate(6 - x - y, (x, 0, 3)), (y, 0, 2))
print("6)")
integral6

# Septima integral
integral7 = integrate(integrate(6 - x - y, (y, 2*x, -x**2 + 3)), (x, 0, 1))
print("7)")
integral7
```

1)

$$\frac{35}{4}$$

2)

$$-\frac{10\pi}{3}+\frac{10\sqrt{5}\pi}{3}$$

3)

$$\frac{1}{24}$$

4)

$$\frac{145467}{20}$$

5)

$$12$$

6)

Out[68]: $$21$$

7)

Out[68]: $$\frac{389}{60}$$

In [ ]:

1)

$$\frac{35}{4}$$

2)

$$-\frac{10\pi}{3} + \frac{10\sqrt{5}\pi}{3}$$

3)

$$\frac{1}{24}$$

4)

$$\frac{145467}{20}$$

5)

$$12$$

6)

$$21$$

7)

$$\frac{389}{60}$$

Ejercicio 9

Dada la función $f(x, y) = x^2 + 4y^2$ se pide:

1. Representar la gráfica de la función en el rectángulo $[0, 2] \times [0, 1]$
2. Calcular el volumen del sólido limitado por el rectángulo $[0, 2] \times [0, 1]$ mediante una integral doble.

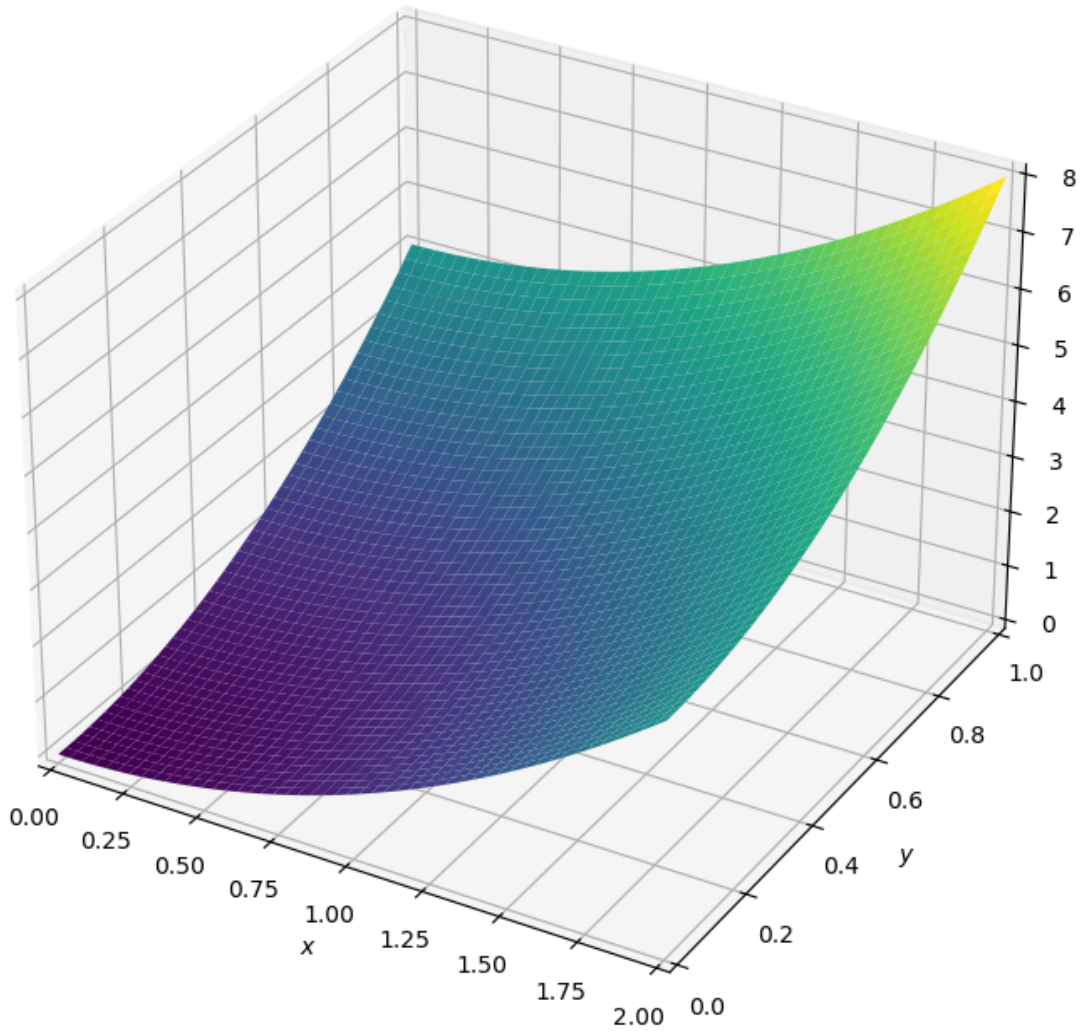
In [ ]:
```
f_ej9 = x**2 + 4*y**2
```

In [72]:
```
from sympy import symbols, integrate

# Definir las variables
x, y = symbols('x y')

# Definir la función
f = x**2 + 4*y**2

# Definir los límites de integración
x_limits = (x, 0, 2)
y_limits = (y, 0, 1)

# Calcular la integral doble
integral_volumen = integrate(integrate(f, y_limits), x_limits)
print("Volumen del sólido limitado por el rectángulo [0, 2] x [0, 1]:")
integral_volumen
```

```
Volumen del sólido limitado por el rectángulo [0, 2] x [0, 1]:
```

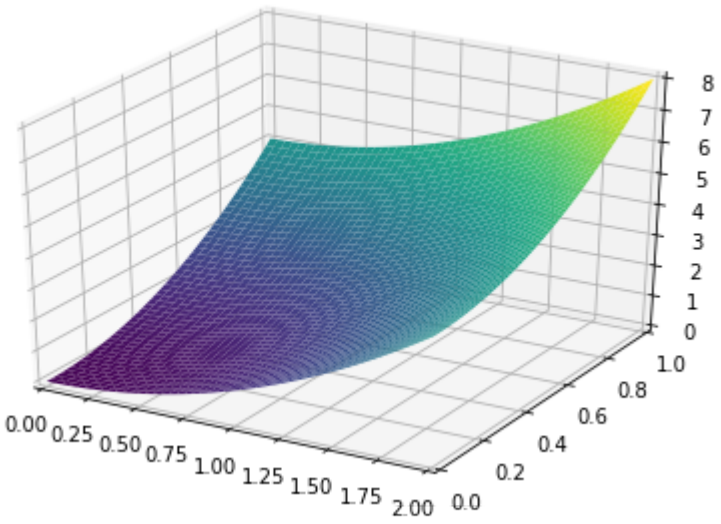
Out[72]: $\frac{16}{3}$

In [73]:
```
from sympy import symbols
from sympy.plotting import plot3d

# Definir las variables
x, y = symbols('x y')

# Definir la función
func = x**2 + 4*y**2

# Graficar la función en el rango especificado
plot3d(func, (x, 0, 2), (y, 0, 1), size=(10, 7))
```



Out[73]: <sympy.plotting.plot.Plot at 0x1c43c6c22d0>

In [ ]:



$$\frac{16}{3}$$