Example 1

Let us consider the following equation, with unknown z and parameter φ:

$$z^2 - \frac{2}{\cos\varphi} z + \frac{5}{\cos^2\varphi} - 4 = 0$$

with $-\frac{\pi}{2} < \varphi < \frac{\pi}{2}$. Solve this equation for z.

In [ ]:
```python
# first way
import sympy as sp

# define the symbols
z, phi = sp.symbols("z varphi")

# define the domain
constrain = sp.And(-sp.pi / 2 < phi, phi < sp.pi / 2)

# define the function
eq = z**2 - 2 / sp.cos(phi) * z + 5 / (sp.cos(phi) ** 2) - 4

# solve the equation
solutions = sp.solveset(eq, z)

# refine the solutions
solutions.refine(constrain).simplify()
```

In [ ]:
```python
# second way
import sympy as sp

# define the symbols
z, phi = sp.symbols("z varphi")

eq = z**2 - 2 / sp.cos(phi) * z + 5 / (sp.cos(phi) ** 2) - 4

solutions = sp.nonlinsolve([eq, -sp.pi / 2 < phi, phi < sp.pi / 2], z)

solutions = list(solutions)
# solutions
# solutions[0][0].simplify()
solutions[1][0].simplify()
```

In [ ]:
```python
# third way
import sympy as sp

# define the symbols
z, phi = sp.symbols("z varphi")

constrain = sp.And(-sp.pi / 2 < phi, phi < sp.pi / 2)

if constrain:
    eq = z**2 - 2 / sp.cos(phi) * z + 5 / (sp.cos(phi) ** 2) - 4
    solutions = sp.solveset(eq, z)
    solutions = solutions.simplify()
    print(solutions)
```

Example 2

Let us consider the seqence of numbers $u_n = \dfrac{n^{100}}{100^n}$.

- Compute the first 10 terms of the sequence.
- What is the limit of the sequence when n goes to infinity?
- From which value of n does $u_n \in (0, 10^{-8})$ hold?

In [ ]:
```
import sympy as sp
from IPython.display import display

# define the symbols
n = sp.symbols("n", integer=True, positive=True)

u = n**100 / 100**n
s = sp.SeqFormula(u)
display(s)

# compute the limit
limit = sp.limit(u, n, sp.oo)
print(f"Limit: {limit}")

# s[:10]
num_terms = 110
for i in range(num_terms):
    next_term = s.coeff(i)
    if next_term == 0:
        continue
    if next_term < 1e-8:
        break
print(f"n th term: {i}")
print(f"Value: {s.coeff(i).evalf():.3e}")
```

Example 3

Check the following function is harmonic:

$$f(x, y) = \frac{1}{2}\ln\left(x^2 + y^2\right)$$

for all $(x, y) \neq (0, 0)$.

- Hint: A function f is said *harmonic* when its Laplacian $\Delta f = \partial_x f + \partial_y f$ is zero.

In [ ]:
```python
import sympy as sp

# define the symbols
x, y = sp.symbols("x y")

# define the equation
f = 1 / 2 * sp.ln(x**2 + y**2)

grad_f = sp.diff(f, x, x) + sp.diff(f, y, y)

print(f"laplacian of f = {grad_f.simplify()}")
```

In [ ]:
```python
# second method
# compute the hessian
hessian = sp.hessian(f, (x, y))

sp.trace(hessian).simplify()
print(f"laplacian of f = {sp.trace(hessian).simplify()}")
```

Example 4

- Is the following matrix A diagonalizable?

$$A = \begin{pmatrix} 2 & 4 & 3 \\ -4 & -6 & -3 \\ 3 & 3 & 1 \end{pmatrix}$$

- Write a Python function that prints if a given matrix is diagonalizable or not and returns True or False, respectively.
- Write another function to compute the diagonal form of the matrix if it is diagonalizable and compute the Jordan form otherwise.

In [ ]:
```python
import sympy as sp

A = sp.Matrix([[2, 4, 3], [-4, -6, -3], [3, 3, 1]])
D = sp.diag(1, 2, 3)

print(A.is_diagonalizable())


def check_diagonal(matrix):
    status = matrix.is_diagonalizable()
    if status:
        print("The matrix is diagonalizable")
        return True
    else:
        print("The matrix is not diagonalizable")
        return False


def diagonalize(matrix):
    if check_diagonal(matrix):
        print("Calculating the diagonal form")
        P, D = matrix.diagonalize()
        print(f"P = {P}")
        print(f"D = {D}")
```

```
        return P, D
    else:
        print("Calculating the Jordan form")
        T, J = matrix.jordan_form()
        print(f"T = {T}")
        print(f"J = {J}")
        return T, J


diagonalize(A)
```

Example 5

Let us compute, for $x \in \mathbb{R}$, the integral

$$\int_0^{+\infty} \frac{x \cos u}{u^2 + x^2} du$$

In [ ]:
```
import sympy as sp

# define the symbols
u = sp.symbols("u", positive=True)
# if x>0
x = sp.symbols("x", positive=True)
int_p = sp.integrate(x * sp.cos(u) / (u**2 + x**2), (u, 0, sp.oo)).simpli
# if x<0
x = sp.symbols("x", negative=True)
int_n = sp.integrate(x * sp.cos(u) / (u**2 + x**2), (u, 0, sp.oo)).simpli

sp.Piecewise((int_p, x > 0), (int_n, x < 0))
```

Example 6

Consider the following IVP:

$$\frac{dy}{dt} = -0.2y, \quad y(0) = 50$$

- Compute the exact solution of the IVP.
- Compute the approximate solution of the IVP using the scipy function `solve_ivp`.
- Plot the exact and approximate solutions, and error between them.

In [ ]:
```
import sympy as sp
from IPython.display import display
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt


#### Exact solution ####


# define the symbols
t = sp.symbols("t", positive=True)
```

```python
y = sp.Function("y")(t)


# define the rhs function
def f(t, y):
    return -0.2 * y


# define the equation
eq = y.diff(t) - f(t, y)

solution = sp.dsolve(eq, y, ics={y.subs(t, 0): 50})
exact_solution = solution.rhs

display(exact_solution)

#### Numerical solution ####

# define the initial condition
y0 = [50]

# define the time interval
t_inter = [0, 10]

# solve the ODE
numerical_solution = solve_ivp(
    f, t_inter, y0, t_eval=np.linspace(t_inter[0], t_inter[1], 100)
)
numerical_solution


#### Plot the solutions ####


exact_solution = sp.lambdify(t, exact_solution, "numpy")
t_values = np.linspace(t_inter[0], t_inter[1], 100)
y_values = exact_solution(t_values)

fig = plt.figure()
plt.plot(numerical_solution.t, numerical_solution.y[0], "o", label="Numer
plt.plot(t_values, y_values, label="Exact solution")
plt.xlabel("t")
plt.ylabel("y")
plt.legend(loc="best")
plt.title("Exact vs Numerical solution")
plt.show()
plt.close(fig)

abs_error = np.abs(numerical_solution.y[0] - exact_solution(numerical_sol
relative_error = abs_error / np.abs(exact_solution(numerical_solution.t))
fig = plt.figure()
# plt.plot(numerical_solution.t, abs_error, label="Absolute error")
plt.plot(numerical_solution.t, relative_error, label="Relative error")
plt.xlabel("t")
plt.ylabel("Errors")
# plt.yscale("log")
plt.legend(loc="best")
plt.title("Errors")
plt.show()
plt.close(fig)
```

Example 7

Solve the differential equation $y'' - 3y' - 4y = sin(x)$ using the Laplace transform with initial conditions $y(0) = 1$ and $y'(0) = -1$.

In [ ]:
```
import sympy as sp
import matplotlib.pyplot as plt
import numpy as np

x = sp.symbols("x")
s = sp.symbols("s")
y = sp.Function("y")
Y = sp.Function("Y")

lhs = y(x).diff(x, x) - 3 * y(x).diff(x) - 4 * y(x)
rhs = sp.sin(x)

### Laplace transform ###

f = sp.laplace_transform(lhs, x, s, noconds=True)

## the following two lines are for a newer version of sympy
# g = sp.laplace_correspondence(f, {y: Y})
# laplace_lhs = sp.laplace_initial_conds(g, x, {y: [1, -1]})


## for an older version of sympy
# replace the laplace transform of y and y' with Y(s) and sY(s) - y(0) re
laplace_lhs = f.subs(
    {
        sp.laplace_transform(y(x), x, s, noconds=True): Y(s),
        sp.laplace_transform(y(x).diff(x), x, s, noconds=True): s * Y(s)
    }
)

print("The Laplace transform of left hand side is:")
display(laplace_lhs)

# subsitute the initial conditions
laplace_lhs = laplace_lhs.subs({y(0): 1, y(x).diff(x).subs(x, 0): -1})

print("The Laplace transform of left hand side with initial conditions is
display(laplace_lhs)

laplace_rhs = sp.laplace_transform(rhs, x, s, noconds=True)

laplace_solution = sp.solve(laplace_lhs - laplace_rhs, Y(s))

solution = sp.inverse_laplace_transform(laplace_solution[0], s, x).simpli

print("The solution of the ODE by Laplace transform is:")
display(sp.Eq(y(x), solution))

exaxt_lambda = sp.lambdify(x, solution, "numpy")


### Exact solution ###
```

```
exact_solution = sp.dsolve(lhs - rhs, y(x), ics={y(0): 1, y(x).diff(x).su
print("The exact solution of the ODE is:")
display(exact_solution)

### Plot the solution ###

solution = sp.lambdify(x, solution, "numpy")
x_values = np.linspace(-1.5, 1.5, 100)
y_values = solution(x_values)

fig = plt.figure()
plt.plot(x_values, y_values, label="Laplace Solution")
plt.plot(x_values, exaxt_lambda(x_values), "--", label="Exact Solution")

plt.xlabel("x")
plt.ylabel("y")
plt.title("Solution of the ODE")
plt.show()
plt.close(fig)
```

Example 8

Consider a stone tossed into the air from ground level with an initial velocity of 15 m/sec. Its height in meters at time t seconds is given by

$$h(t) = 15t - 4.9t^2.$$

Compute the average velocity of the stone over the given time interval $[1, 1.005]$.

In [ ]:
```
import sympy as sp

# define the symbols
t = sp.symbols("t", positive=True)
h = 15 * t - 4.9 * t**2

# the average velocity in an interval [a, b] is given by
t0 = 1
t1 = 1.005
v_avg = (h.subs(t, t1) - h.subs(t, t0)) / (t1 - t0)

print(f"The average velocity is {v_avg.round(1)} m/s")
```

Example 9

According to Newton's law of universal gravitation, the force F between two bodies of constant mass m_1 and m_2 is given by the formula

$$F = \frac{Gm_1m_2}{d^2},$$

where G is the gravitational constant and d is the distance between the bodies.

- Suppose that $G, m1$, and $m2$ are constants. Find the rate of change of force F with respect to distance d.
- Find the rate of change of force F with gravitational constant

$G = 6.67 \times 10^{-11} Nm^2/kg^2$, on two bodies 10 meters apart, each with a mass of 1000 kilograms.

In [ ]:
```
import sympy as sp
from IPython.display import display

# define the symbols
m1, m2 = sp.symbols("m_1:3")
G = sp.symbols("G")
d = sp.symbols("d")

# define the force
F = G * m1 * m2 / d**2

# compute the derivative
dF = F.diff(d)

print("The rate of change of the force is:")
display(dF)

result = dF.subs({G: 6.67e-11, d: 10, m1: 1000, m2: 1000})
print("The rate of change of the force at given values is:")
display(result)
```

Example 10

The price p (in dollars) and the demand x for a certain digital clock radio is given by the price–demand function

$$p = 10 - 0.001x.$$

- Find the revenue function $R(x)$.
- Find the marginal revenue function $MR(x)$.
- Find the price that maximizes the revenue.

In [ ]:
```
import sympy as sp
from IPython.display import display

# define the symbols
x = sp.symbols("x")

# define the price-demand function
p = 10 - 0.001 * x

# define the revenue function
R = p * x

# compute the derivative
dR = R.diff(x)

# compute the maximum revenue
x_max = sp.solve(dR, x)[0]
R_max = R.subs(x, x_max)

print("The revenue function is:")
display(R)
```

```
print("The marginal revenue function is:")
display(dR)
print(f"The maximum revenue is at\nx = {x_max:.0f} and R({x_max:.0f}) = {
```