Solvers

The solvers module in SymPy implements methods for solving equations. Here is a list of the most commonly used methods:

Algebraic equations

There are two high-level functions to solve equations, `solve()` and `solveset()` .

In [1]:
```
import sympy as sp
from sympy.abc import x, y


sp.solve(x**2 - y, x)
```

Out[1]: `[-sqrt(y), sqrt(y)]`

In [2]:
```
sp.solveset(x**2 - y, x)
```

Out[2]: $\left\{-\sqrt{y}, \sqrt{y}\right\}$

In [4]:
```
sp.solve(x**2 - y, x, dict=True)
```

Out[4]: `[{x: -sqrt(y)}, {x: sqrt(y)}]`

In [5]:
```
eqn = sp.Eq(x**2, y)
eqn
```

Out[5]: $x^2 = y$

In [8]:
```
solutions = sp.solve(eqn, x, dict=True)
solutions
# solutions[0][x]
```

Out[8]: `[{x: -sqrt(y)}, {x: sqrt(y)}]`

In [14]:
```
solutions_set = sp.solveset(eqn, x)
for sol in solutions_set:
    print(sol)

solution_list = list(solutions_set)
solution_list[0]
```

```
sqrt(y)
-sqrt(y)
```

Out[14]: $\sqrt{y}$

In [18]:
```
# x = sp.Symbol("x")
x = sp.Symbol("x", real=True)
sp.solve(x**4 - 256, x, dict=True)
```

Out[18]: [{x: -4}, {x: 4}]

In [20]:
```python
x = sp.Symbol("x", real=True)
expr = (x - 4) * (x - 3) * (x - 2) * (x - 1)
solution = sp.solve(expr, x)
print(solution)
solution_outside_2_3 = [sol for sol in solution if sol < 2 or sol > 3]
print(solution_outside_2_3)
```

```
[1, 2, 3, 4]
[1, 4]
```

In [21]:
```python
from sympy.abc import x

sp.solveset(x**4 - 256, x, domain=sp.S.Reals)
```

Out[21]: $\{-4, 4\}$

In [24]:
```python
sp.solve(x**2 + 1, x)
```

Out[24]: [-I, I]

Why Solveset?

- `solveset` has an alternative consistent input and output interface: `solveset` returns a set object and a set object takes care of all types of output. For cases where it does not "know" all the solutions a `ConditionSet` with a partial solution is returned. For input it only takes the equation, the variables to solve for and the optional argument `domain` over which the equation is to be solved.
- `solveset` can return infinitely many solutions. For example solving for $\sin(x) = 0$ returns $\{2n\pi | n \in \mathbb{Z}\} \cup \{2n\pi + \pi | n \in \mathbb{Z}\}$, whereas `solve` only returns $[0, \pi]$.
- There is a clear code level and interface level separation between solvers for equations in the complex domain and the real domain. For example solving $e^x = 1$ when x is to be solved in the complex domain, returns the set of all solutions, that is $\{2ni\pi | n \in \mathbb{Z}\}$, whereas if is to be solved in the real domain then only $\{0\}$ is returned.

In [25]:
```python
solution = sp.solveset(sp.sin(x), x)
solution
```

Out[25]: $\{2n\pi \mid n \in \mathbb{Z}\} \cup \{2n\pi + \pi \mid n \in \mathbb{Z}\}$

In [4]:
```python
sp.solveset(x**2 + 1, x)  # Complex solution set is default
# sp.solveset(x**2 + 1, x, domain=sp.S.Reals)
```

Out[4]: $\emptyset$

In [15]:
```python
# system of equations
sp.solve([x - 3, y**2 - 1])  # solve can handle system of equations

# sp.solveset can handle only uni-variate equations
```

```python
# sp.solveset([x - 3, y**2 - 1]) # will raise error


x, y, z = sp.symbols("x y z")

Eqns = [3 * x + 2 * y - z - 1, 2 * x - 2 * y + 4 * z + 2, -x + y / 2 - z]

sp.linsolve(Eqns, x, y, z)
```

Out[15]: $\{(1, \ -2, \ -2)\}$

In [ ]:
```python
# Non-linear system of equations
sp.nonlinsolve([x - 3, y**2 - 1], [x, y])
```

In [23]:
```python
# If any equation does not depend on the symbol(s) given,
# it will be eliminated from the equation set
sp.solve(x - y, x, dict=True)
# sp.solveset(x - y, x)
# sp.solve([x - y, y - 3], x, dict=True)
```

Out[23]: [{x: y}]

In [32]:
```python
# In case the system is underdetermined, the function will return a param

A = sp.Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
b = sp.Matrix([3, 6, 9])

sp.linsolve((A, b), x, y, z)
# sp.linsolve((A, b)) # if the parametric symbols are not given, it will
```

Out[32]: $\{(z - 1, \ 2 - 2z, \ z)\}$

In [42]:
```python
import numpy as np

a = np.array([[1, 2], [3, 5]])
b = np.array([1, 2])
x = np.linalg.solve(a, b)
x
```

Out[42]: array([-1., 1.])

In [44]:
```python
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
b = np.array([3, 6, 9])
x = np.linalg.lstsq(A, b, rcond=None)[0]
x
```

Out[44]: array([-0.16666667, 0.33333333, 0.83333333])

In [33]:
```python
# linsolve can accept augmented matrix as well
aug = sp.Matrix([[2, 1, 3, 1], [2, 6, 8, 3], [6, 8, 18, 5]])

sp.linsolve(aug, x, y, z)
```

Out[33]: $\left\{\left(\frac{3}{10}, \ \frac{2}{5}, \ 0\right)\right\}$

In [34]:
```python
a, b, c, d, e, f = sp.symbols("a, b, c, d, e, f")
```

```
eqns = [a * x + b * y - c, d * x + e * y - f]

sp.linsolve(eqns, x, y)
```

Out[34]: $$\left\{\left(\frac{-bf+ce}{ae-bd},\ \frac{af-cd}{ae-bd}\right)\right\}$$

In [37]:
```
# use sympy to reduce inequalities

sp.solve(x**2 - 4 > 0, x)
# sp.solveset(x**2 - 4 > 0, x)
```

Out[37]: $$(-\infty < x \wedge x < -2) \vee (2 < x \wedge x < \infty)$$

In [17]:
```
sp.Abs(x - 5) - 3
sp.solve(sp.Abs(x - 5) - 3 > 0, x)
```

Out[17]: $$(-\infty < x \wedge x < 2) \vee (8 < x \wedge x < \infty)$$

In [29]:
```
sp.solve_univariate_inequality(sp.sin(x) > 0, x, relational=False)
```

Out[29]: $$(0, \pi)$$

In [32]:
```
domain = domain = sp.Interval(0, sp.oo)

sp.solve_univariate_inequality(x**2 >= 4, x)
# sp.solve_univariate_inequality(x**2 >= 4, x, relational=False)
# sp.solve_univariate_inequality(x**2 >= 4, x, domain=domain)
```

Out[32]: $$(2 \leq x \wedge x < \infty) \vee (x \leq -2 \wedge -\infty < x)$$

Differential equations

In [1]:
```
# undefined functions

x = sp.Symbol("x")
f = sp.Function("f")
g = sp.Function("g")(x)
# f(x)

# f(0)
# g(0) # will raise error
g.subs(x, 0)
```

Out[1]: $$g(0)$$

In [8]:
```
f = sp.symbols("f", cls=sp.Function)(x)
f
```

Out[8]: $$f(x)$$

In [9]:
```
g.diff()
# f(x).diff()
# f(x).diff(x)
# sp.Derivative(f(x), x)
```

Out[9]: $\frac{d}{dx}g(x)$

In [22]:
```python
# assumptions can be passed to the function

f = sp.Function("f", real=True)
f(x).is_real
```

Out[22]: True

In [24]:
```python
x.diff(x)

# if y is variable
# y.diff(x)

# if y is a function of x
# y = sp.Function("y")(x)
# y.diff(x)
```

Out[24]: 1

In [26]:
```python
f(x).diff(x).subs(x, 0)
```

Out[26]: $\left.\frac{d}{dx}f(x)\right|_{x=0}$

In [27]:
```python
# solve differential equations

f = sp.Function("f")
sp.dsolve(f(x).diff(x) - f(x), f(x))
```

Out[27]: $f(x) = C_1 e^x$

In [25]:
```python
f = sp.Function("f")(x)
f_x = f.diff(x)
f_xx = f.diff(x, x)

sp.dsolve(f_xx + f, f)
```

Out[25]: $f(x) = C_1 \sin(x) + C_2 \cos(x)$

In [4]:
```python
# system of differential equations

t = sp.symbols("t")

x, y = sp.symbols("x, y", cls=sp.Function)

eq = (
    sp.Eq(sp.Derivative(x(t), t), 12 * t * x(t) + 8 * y(t)),
    sp.Eq(sp.Derivative(y(t), t), 21 * x(t) + 7 * t * y(t)),
)

sp.dsolve(eq)
# sp.pprint(sp.dsolve(eq))
# sp.dsolve(eq, [x(t), y(t)])
```

Out[4]:
```
[Eq(x(t), C1*x0(t) + C2*x0(t)*Integral(8*exp(Integral(7*t, t))*exp(Integ
ral(12*t, t))/x0(t)**2, t)),
 Eq(y(t), C1*y0(t) + C2*(y0(t)*Integral(8*exp(Integral(7*t, t))*exp(Inte
gral(12*t, t))/x0(t)**2, t) + exp(Integral(7*t, t))*exp(Integral(12*t,
t))/x0(t)))]
```

In [33]:
```
# initial conditions

f, g = sp.symbols("f g", cls=sp.Function)

x = sp.symbols("x")

eqs = [sp.Eq(f(x).diff(x), g(x)), sp.Eq(g(x).diff(x), f(x))]

sp.dsolve(eqs, [f(x), g(x)])
# sp.dsolve(eqs, [f(x), g(x)], ics={f(0): 1, g(2): 3})
```

Out[33]:
```
[Eq(f(x), -C1*exp(-x) + C2*exp(x)), Eq(g(x), C1*exp(-x) + C2*exp(x))]
```

In [34]:
```
# derivative conditions
eqn = sp.Eq(f(x).diff(x), f(x))

sp.dsolve(eqn, f(x), ics={f(x).diff(x).subs(x, 1): 2})
```

Out[34]:

$$f(x) = \frac{2e^x}{e}$$

In [36]:
```
y = sp.Function("y")

result = sp.dsolve(sp.Derivative(y(x), x, x) + 9 * y(x), y(x))
result
# result.rhs
# y_res = result.rhs
# # y_res.subs(x, 0)
# C1, C2 = sp.symbols("C1, C2")
# y_res.subs({C1: 9, C2: sp.pi})
```

Out[36]:

$$y(x) = C_1 \sin(3x) + C_2 \cos(3x)$$

In [ ]:
```
# not all equations can be solved by dsolve

y = sp.Function("y")

x, C = sp.symbols("x C1:2")

# NotImplementedError will be raised
sp.dsolve(sp.Derivative(y(x), x, 3) - (y(x) ** 2), y(x)).rhs
```

Example: Seperable ODE

Let the following Cauchy problem be given:

$$\begin{cases} \dfrac{df(t)}{dt} = -2tf(t) \\ f(0) = 1 \end{cases}$$

whose exact solution is $f(t) = e^{-t^2}$.

In [38]:
```
t = sp.symbols("t")
f = sp.Function("f")

eq = sp.Eq(f(t).diff(t), -2 * t * f(t))

print("ODE class:", sp.classify_ode(eq)[0])

result = sp.dsolve(eq, f(t), ics={f(0): 1})
f = result.rhs
f
```

```
ODE class: separable
```

Out[38]: e^{-t^2}

Example: Linear ODE

Let the following Cauchy problem be given:

$$\begin{cases} \frac{d}{dt} f(t) = \frac{t f(t)}{1+t^2} + 1 \\ f(0) = 0 \end{cases}$$

whose exact solution is $f(t) = \sqrt{1+t^2}\ln(t+\sqrt{1+t^2})$

In [43]:
```
t = sp.symbols("t")
f = sp.Function("f")

f_exact = sp.sqrt(1 + t**2) * sp.ln(t + sp.sqrt(1 + t**2))

eq = sp.Eq(f(t).diff(t), (t / (1 + t**2)) * f(t) + 1)
eq
print("ODE class:", sp.classify_ode(eq)[0])
f_result = sp.dsolve(eq, ics={f(0): 0}).rhs
# f_result = sp.dsolve(eq, hint="1st_linear", ics={f(0): 0}).rhs
f_result
# f_result.simplify()
```

```
ODE class: factorable
```

Out[43]: $\frac{t^2 \operatorname{asinh}(t) + \operatorname{asinh}(t)}{\sqrt{t^2+1}}$

In [44]:
```
sp.checkodesol(eq, f_result)
```

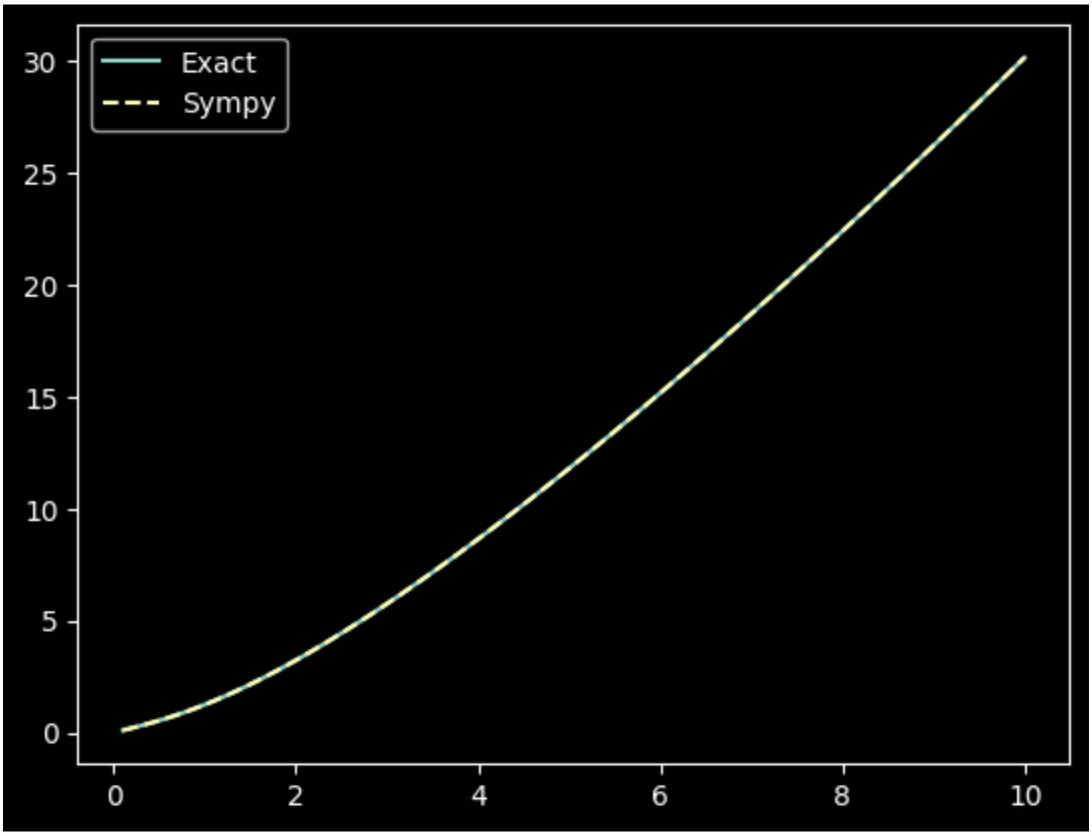
Out[44]:
```
(True, 0)
```

In [45]:
```
import matplotlib.pyplot as plt
import numpy as np

f_exact_lambdified = sp.lambdify(t, f_exact, modules="numpy")
f_result_lambdified = sp.lambdify(t, f_result, modules="numpy")

t_vals = np.linspace(0.1, 10, 100)
```

```python
fig = plt.figure()
plt.plot(t_vals, f_exact_lambdified(t_vals), label="Exact")
plt.plot(t_vals, f_result_lambdified(t_vals), "--", label="Sympy")
plt.legend()
plt.show()
plt.close(fig)
```



Partial Differential Equations

In [1]:

```python
import sympy as sp
from sympy.abc import x, y

f = sp.Function("f")(x, y)  # f is a function of x and y

# fx will be the partial derivative of f with respect to x
fx = sp.Derivative(f, x)

# fy will be the partial derivative of f with respect to y
fy = sp.Derivative(f, y)
```

In [9]:

```python
f = sp.Function("f")  # varibales are not passed

u = f(x, y)  # now u is a function of x and y

ux = u.diff(x)

uy = u.diff(y)

eq = sp.Eq(1 + (2 * (ux / u)) + (3 * (uy / u)), 0)

sp.pdsolve(eq)
# sp.pdsolve(eq, f(x, y))
```

In [10]:
```
result = sp.pdsolve(eq).rhs
sp.checkpdesol(eq, result)
```

Out[10]: (True, 0)

Example: Linear partial differential equation with constant coefficients

The general form of this partial differential equation is

$$a\frac{\partial f(x,y)}{\partial x} + b\frac{\partial f(x,y)}{\partial y} + cf(x,y) = G(x,y)$$

where a, b, c are constants and $G(x,y)$ is a function of x and y.

The general solution of the PDE is:

$$f(x,y) = \left.\left[F(\eta) + \frac{1}{a^2+b^2}\int^{ax+by} G\left(\frac{a\xi+b\eta}{a^2+b^2}, \frac{-a\eta+b\xi}{a^2+b^2}\right) e^{\frac{c\xi}{a^2+b^2}}\, d\xi\right] e^{-\frac{c\xi}{a^2+b^2}}\right|_{\substack{\eta=-a\eta\\ \xi=ax}}$$

In [12]:
```
x, y, a, b, c = sp.symbols("x y a b c")
f, G = sp.symbols("f G", cls=sp.Function)

u = f(x, y)
ux = u.diff(x)
uy = u.diff(y)

genform = a * ux + b * uy + c * u - G(x, y)
genform
```

Out[12]:

$$a\frac{\partial}{\partial x}f(x,y) + b\frac{\partial}{\partial y}f(x,y) + cf(x,y) - G(x,y)$$

In [16]:
```
# sp.pdsolve(genform)
sp.pdsolve(genform, hint="1st_linear_constant_coeff_Integral")
```

Out[16]:

$$f(x,y) = \left.\left(F(\eta) + \frac{\int^{ax+by} G\left(\frac{a\xi+b\eta}{a^2+b^2}, \frac{-a\eta+b\xi}{a^2+b^2}\right) e^{\frac{c\xi}{a^2+b^2}}\, d\xi}{a^2+b^2}\right) e^{-\frac{c\xi}{a^2+b^2}}\right|_{\substack{\eta=-ay+bx\\ \xi=ax+by}}$$