Równania różniczkowe zwyczajne

Cel zajęć:

1. Poznanie metod rozwiązywania równań różniczkowych zwyczajnych z wykorzystaniem SymPy
2. Poznanie metod rozwiązywania równań różniczkowych zwyczajnych z warunkami poczatkowymi
3. Podstawowe metody analizy i wizualizacj rozwiązań
4. Nabycie umiejętności samodzielnego zaplanowania oraz realizacji zadań zawierających równania różniczkowe

W czasie zajęć będziemy korzystali z nastepujących funkcji i operatorów:

- `.rhs` - zwraca prawą stronę równania
- `.lhs` - zwraca lewą stronę równania
- `sp.dsolve()` - rozwiązuje równanie różniczkowe
- `sp.Function()` - definiuje funkcję

Importujemy niezbędne moduły

```
import sys
import mpmath
sys.modules['sympy.mpmath'] = mpmath
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
import sympy as sp
sp.init_printing() #aby ładnie się drukowało
```

Chcemy rozwiązać równanie różniczkowe:

y'(t) = - λ y(t)

```
#Definiujemy symbole, funkcję
x, t, l, C1 = sp.symbols('x t lambda C1')
y = sp.Function('y')(t)
y
```

$$y(t)$$

```
#Zapisujemy równanie różniczkowe (strona lewa, strona prawa)
dydt = y.diff(t)
expr = sp.Eq(dydt, -l*y)
expr
```

$$\frac{d}{dt}y(t) = -\lambda y(t)$$

```
#Rozwiązujemy równanie
y_t_sol = sp.dsolve(expr)
y_t_sol
```

$$y(t) = C_1 e^{-\lambda t}$$

```
C1, l = sp.symbols('C1 lambda')
C1
```

$$C_1$$

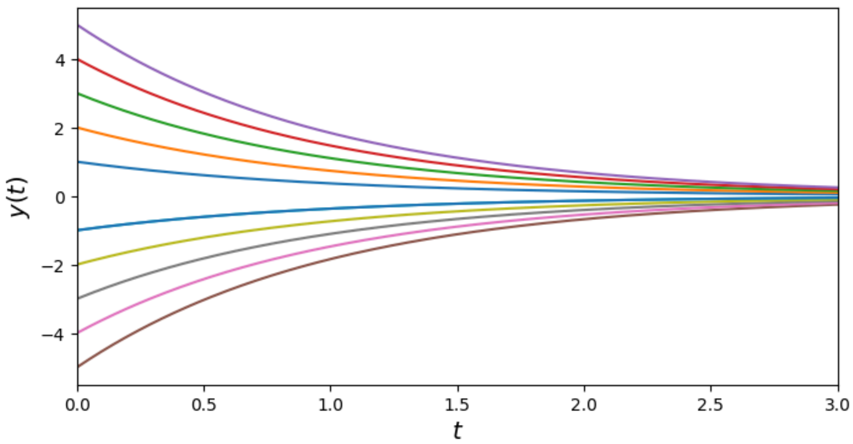
```
# y_t_sol_c = C1* sp.exp(-l*t)
```

```
y_t_sol_c1 = C1* sp.exp(-t)
```

```
fig, ax = plt.subplots(figsize=(8, 4))
tt = np.linspace(0, 3, 250)
for CC in [1,2,3,4,5]:
  y_t = sp.lambdify(t, y_t_sol_c1.subs({C1:CC}), 'numpy')
  ax.plot(tt, y_t(tt))
  # ax.plot(tt, y_t(tt), label=r"$\lambda = %.1f$" % ll)

for CC in range (-5,0):
  y_t = sp.lambdify(t, y_t_sol_c1.subs({C1:CC}), 'numpy')
  ax.plot(tt, y_t(tt))
  # ax.plot(tt, y_t(tt), label=r"$\lambda = %.1f$" % ll)

print(C1,CC)
ax.set_xlabel(r"$t$", fontsize=14)
ax.set_ylabel(r"$y(t)$", fontsize=14)
ax.plot(tt, y_t(tt).real)
ax.set_xlim(0, 3)
# ax.legend()
```

```
C1 -1
```

$(0.0,\ 3.0)$



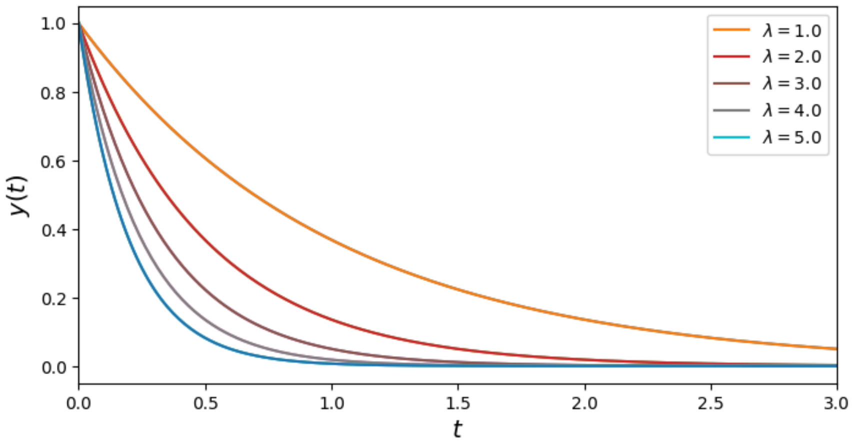

```
# y_t_sol.rhs.subs({l:1})
```

```
y_t_sol_w = sp.exp(-l*t)
y_t_sol_w
```

$e^{-\lambda t}$

```
fig, ax = plt.subplots(figsize=(8, 4))
tt = np.linspace(0, 3, 250)
for ll in [1., 2., 3., 4., 5.0]:
  y_t = sp.lambdify(t, y_t_sol_w.subs({l:ll}), 'numpy')
  ax.plot(tt, y_t(tt))
  ax.plot(tt, y_t(tt), label=r"$\lambda = %.1f$" % ll)


ax.set_xlabel(r"$t$", fontsize=14)
ax.set_ylabel(r"$y(t)$", fontsize=14)
ax.plot(tt, y_t(tt).real)
ax.set_xlim(0, 3)
ax.legend()
```

```
<matplotlib.legend.Legend at 0x792994346590>
```



A teraz rozwiążmy równanie z podanymi warunkami:

x f''(x) + f'(x) = x**3;

f(1) = 0; f'(2) = 1.

```
#Wprowadzamy funkcję f(x)
f = sp.Function('f')(x)
f
```

$$f(x)$$

```
#Wprowadzamy pochodne f(x)
f.diff()
```

$$\frac{d}{dx}f(x)$$

```
f.diff(x,x)
```

$$\frac{d^2}{dx^2}f(x)$$

```
#Zapisujemy równanie różniczkowe:
diff_eq = sp.Eq(x*f.diff(x,x) + f.diff(x), x**3)
diff_eq
```

$$x\frac{d^2}{dx^2}f(x) + \frac{d}{dx}f(x) = x^3$$

```
#Zapiszmy stronę prawą i lewą równania
diff_eq.rhs
```

$$x^3$$

```
diff_eq.lhs
```

$$x\frac{d^2}{dx^2}f(x) + \frac{d}{dx}f(x)$$

```
#Rozwiązujemy równanie
sol = sp.dsolve(diff_eq, f)
sol
```

$$f(x) = C_1 + C_2\log{(x)} + \frac{x^4}{16}$$

```
#Sprawdźmy typ rozwiązania:
type(sol)
```

```
sympy.core.relational.Equality
```

```
#wprowadźmy zmienną
exp = sol.rhs
exp
```

$$C_1 + C_2 \log(x) + \frac{x^4}{16}$$

```
#Sprawdźmy, jakie symbole(zmienne) występują w powyższej funkcji
exp.free_symbols
```

$$\{C_1, C_2, x\}$$

```
print(exp.free_symbols)
```

```
{C2, C1, x}
```

```
#Przypiszmy nazwy dla symboli C1 i C2:
_, C1, C2= tuple(exp.free_symbols)
C1
```

$$C_1$$

```
#Podstawmy wartości C1 = 3 oraz C2 = 5:
exp.subs({C1:3, C2:5})
```

$$C_2 \log(5) + \frac{673}{16}$$

```
#Podstawmy wartości C1 = 1 oraz C2 = 2:
exp.subs({C1:1, C2:2})
```

$$C_2 \log(2) + 2$$

Korzystamy z podanych w treści zadania warunków:

```
#Utwórzmy słownik z podanymi warunkami
ics = {f.subs(x, 1): 0, f.diff().subs(x, 2): 1}
ics
```

$$\left\{ f(1) : 0,\ \left.\frac{d}{dx} f(x)\right|_{x=2} : 1 \right\}$$

```
#Rozwiążmy problem początkowy
ivp = sp.dsolve(diff_eq, ics=ics).rhs
ivp
```

$$\frac{x^4}{16} - 2\log(x) - \frac{1}{16}$$

```
#Sprawdźmy, czy warunki zostały spełnione
ivp.subs(x, 1)
```

$$0$$

```
ivp.diff().subs(x, 2)
```

$$1$$

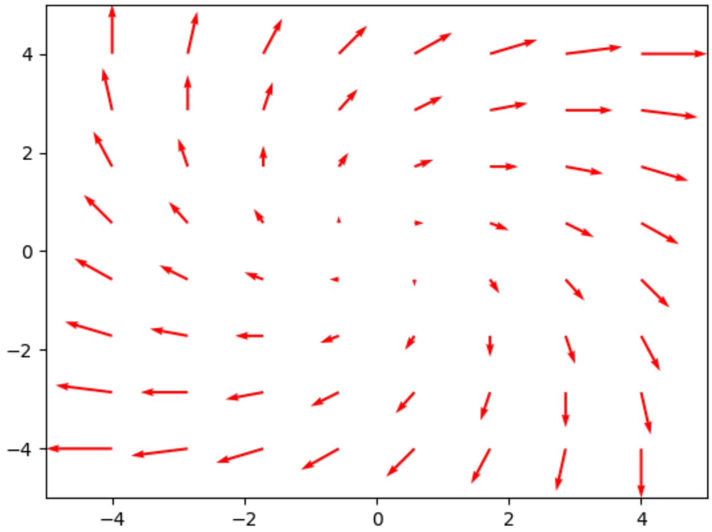
```
#Sprawdźmy też, czy nasze rozwiązanie spełnia równanie (zamieniając x na ivp)
(x*ivp.diff(x,x) + ivp.diff(x)).simplify()
```

$$x^3$$

Powyżej rozwiązaliśmy równanie z warunkami początkowymi. Warto zebrać wszystkie te kroki w funkcji wielokrotnego użytku.

Poniżej inspiracja do rysowania pól wektorowych

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

# plt.style.use('mpl-gallery-nogrid')

# make data
x = np.linspace(-4, 4, 8)
y = np.linspace(-4, 4, 8)
X, Y = np.meshgrid(x, y)
U = X + Y
V = Y - X

# plot
fig, ax = plt.subplots()

ax.quiver(X, Y, U, V, color="r", angles='xy',
          scale_units='xy', scale=8, width=.004)

ax.set(xlim=(-5, 5), ylim=(-5, 5))

plt.show()
```



Wprowadźmy funkcję do rysowania pola kierunków dla równania różniczkowego:

```
def plot_direction_field(x, y_x, f_xy, x_lim=(-5, 5), y_lim=(-5, 5), ax=None):
  f_np = sp.lambdify((x, y_x), f_xy, 'numpy')
  x_vec = np.linspace(x_lim[0], x_lim[1], 20)
  y_vec = np.linspace(y_lim[0], y_lim[1], 20)
  # print('*****')

  if ax is None:
    # _, ax = plt.subplots(figsize=(4, 4))
    ax = plt.subplots(figsize=(4, 4))

  dx = x_vec[1] - x_vec[0]
  dy = y_vec[1] - y_vec[0]
  # print(x_vec[1],x_vec[0],dx,dy)

  for m, xx in enumerate(x_vec):
    for n, yy in enumerate(y_vec):
      Dy = f_np(xx, yy) * dx
      Dx = 0.8 * dx**2 / np.sqrt(dx**2 + Dy**2)
      Dy = 0.8 * Dy*dy / np.sqrt(dx**2 + Dy**2)
      ax.plot([xx - Dx/2, xx + Dx/2],
              [yy - Dy/2, yy + Dy/2], 'b', lw=0.5)
  ax.axis('tight')
  ax.set_title(r"$%s$" %
                   (sp.latex(sp.Eq(y(x).diff(x), f_xy))),fontsize=16)
  return ax
```

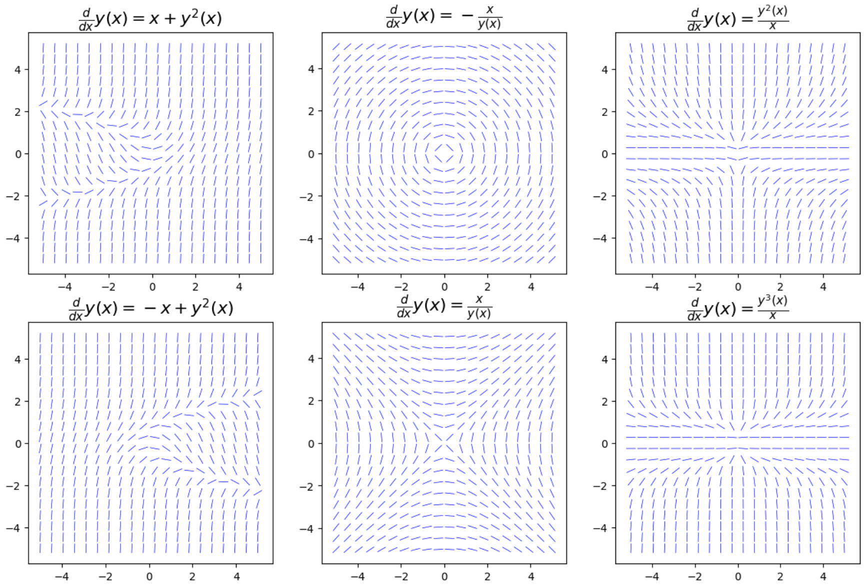
Zastosujmy wprowadzoną funkcję 'plot_direction_field' do wizualizacji pól kierunkowych dla podanych niżej przykładów równań różniczkowych:

```
x = sp.symbols("x")
```

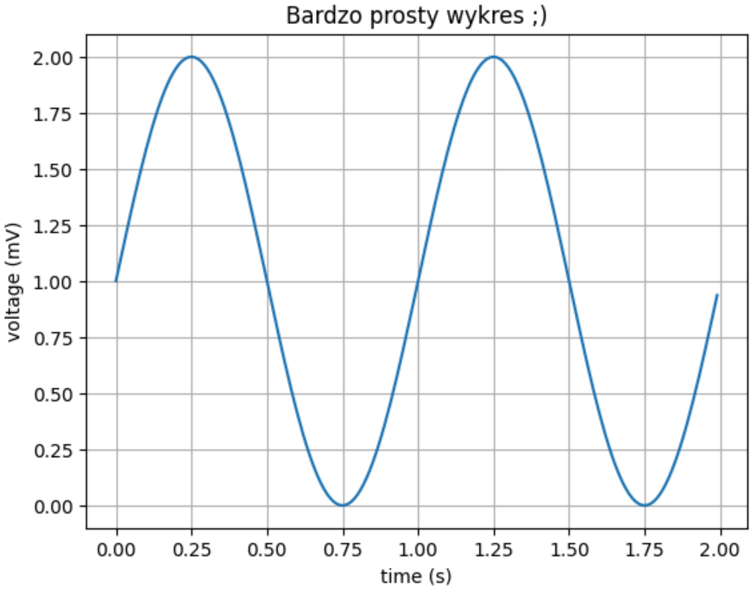
```
y = sp.Function("y")
```

```
fig, axes = plt.subplots(2, 3, figsize=(14, 9)) #definiujemy przestrzeń oraz układ rysunków
plot_direction_field(x, y(x), y(x)**2 + x, ax=axes[0,0])
plot_direction_field(x, y(x), -x / y(x), ax=axes[0,1])
plot_direction_field(x, y(x), y(x)**2 / x, ax=axes[0,2])
plot_direction_field(x, y(x), y(x)**2 - x, ax=axes[1,0])
plot_direction_field(x, y(x), x / y(x), ax=axes[1,1])
plot_direction_field(x, y(x), y(x)**3 / x, ax=axes[1,2])
```

```
<Axes: title={'center': '$\\frac{d}{d x} y{\\left(x \\right)} = \\frac{y^{3}
{\\left(x \\right)}}{x}$'}>
```




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

# Dane do wykresu
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

fig, ax = plt.subplots()
ax.plot(t, s)

ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='Bardzo prosty wykres ;)')
ax.grid()

fig.savefig("test.png")
plt.show()
```




```
#krótkie wprowadzenie równań
t = sp.symbols('t')
x = sp.Function('x')

diffeq = sp.Eq(x(t).diff(t, t) - x(t), sp.cos(t))
res = sp.dsolve(diffeq, ics={x(0): 0, sp.diff(x(t), t).subs(t,0): 0})
```

```
res
```

$$x(t) = \frac{e^{t}}{4} - \frac{\cos(t)}{2} + \frac{e^{-t}}{4}$$

Zadania do pracy samodzielnej:

1. y' + 2xy = 0
2. y' = y sin x
3. 2 sqrt(x) y' = sqrt(1 - y^2)
4. y' = (64xy)^(1/3)
5. (1 - x^2) y' = 2y
6. y' = x y^3
7. y^3 y' = (y^4 + 1) cos x
8. y' = y exp(x), y(0) = 2e
9. y' = 3 x^2 (y^2 +1), y(0) = 1
10. 2y y' = x/sqrt(x^2-16), y(5) = 2
11. tan(x) y' = y, y(pi/2) = pi/2
12. y' + y = 2, y(0) = 0
13. y' - 2y = 3 exp(2x), y(0) = 0

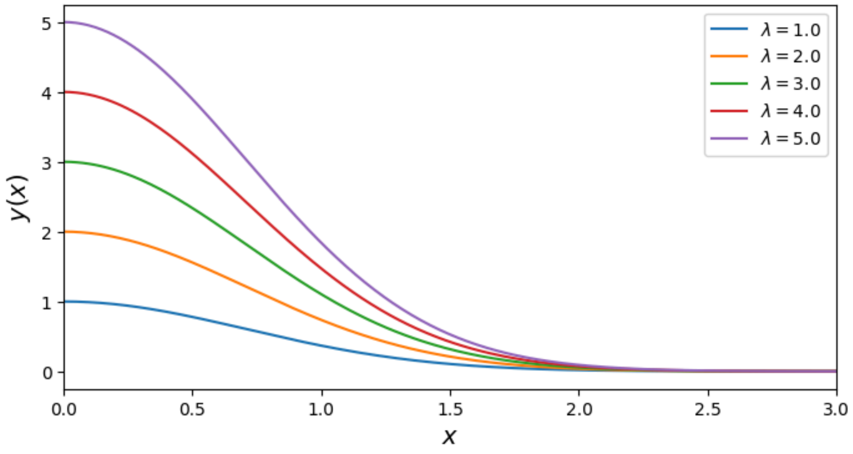
Niecierpliwie czekam na wasze sprawozdania :)

y' + 2xy = 0

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx + 2*y*x,0)
y_x_sol = sp.dsolve(expr)
y_x_sol
```

$$y(x) = C_1 e^{-x^2}$$

```
sol = sp.dsolve(sp.Derivative(y, x) + 2 * x * y, y)
xx = np.linspace(0, 3, 250)
fig, ax = plt.subplots(figsize=(8, 4))
for ll in [1., 2., 3., 4., 5.0]:
    y_lambda = sol.subs('C1', ll)
    y_lambda_func = sp.lambdify(x, y_lambda.rhs, 'numpy')
    ax.plot(xx, y_lambda_func(xx), label=r"$\lambda = %.1f$" % ll)
ax.set_xlabel(r"$x$", fontsize=14)
ax.set_ylabel(r"$y(x)$", fontsize=14)
ax.set_xlim(0, 3)
ax.legend()
plt.show()
```

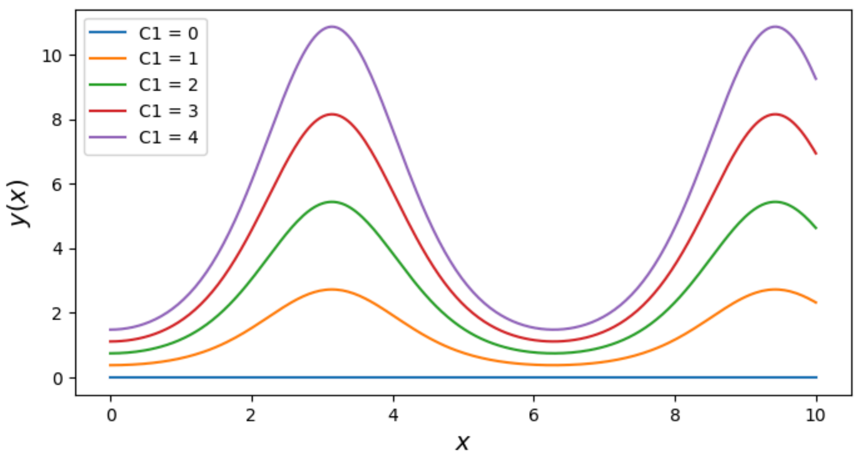


y' = y sin x

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx, sp.sin(x))
y_t_sol = sp.dsolve(expr)
y_t_sol
```

$$y(x) = C_1 - \cos(x)$$

```
diff_eq = sp.Eq(sp.Derivative(y, x), y * sp.sin(x))
sol = sp.dsolve(diff_eq, y)
xx = np.linspace(0, 10, 250)
fig, ax = plt.subplots(figsize=(8, 4))
for C_value in [0, 1, 2, 3, 4]:
    y_values = [sol.subs('C1', C_value).rhs.subs(x, val).evalf() for val in xx]
    ax.plot(xx, y_values, label=f"C1 = {C_value}")
ax.set_xlabel(r"$x$", fontsize=14)
ax.set_ylabel(r"$y(x)$", fontsize=14)
ax.legend()
plt.show()
```

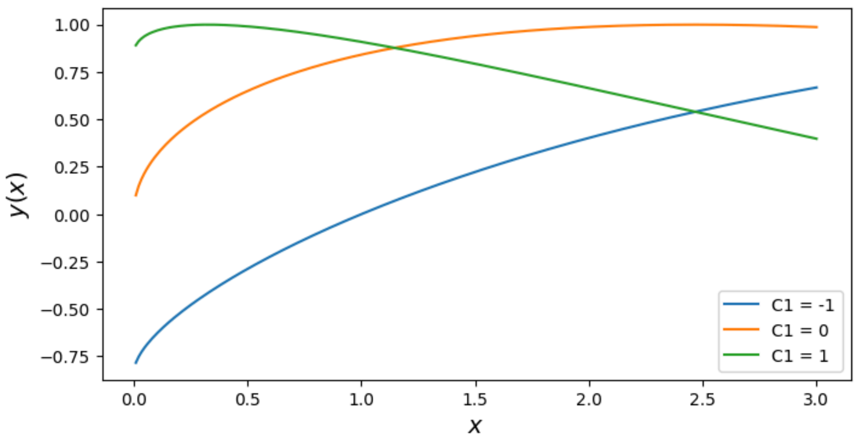


2 sqrt(x) y' = sqrt(1 - y^2)

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(2*(x**(1/2))*dydx, (1-y**2)**(1/2))
y_t_sol = sp.dsolve(expr)
y_t_sol
```

$$\begin{cases} -1.0i \operatorname{acosh}\left(1.0y(x)\right) & \text{for}\ 1.0\left|y^2(x)\right| > 1 \\ 1.0 \operatorname{asin}\left(1.0y(x)\right) & \text{otherwise} \end{cases} = C_1 + 1.0\sqrt{x}$$

```
diff_eq = sp.Eq(2 * sp.sqrt(x) * sp.diff(y, x), sp.sqrt(1 - y**2))
sol = sp.dsolve(diff_eq, y)
xx = np.linspace(0.01, 3, 250)
fig, ax = plt.subplots(figsize=(8, 4))
for C_value in [-1, 0, 1]:
    y_values = [sol.subs('C1', C_value).rhs.subs(x, val).evalf() for val in xx]
    ax.plot(xx, y_values, label=f"C1 = {C_value}")
ax.set_xlabel(r"$x$", fontsize=14)
ax.set_ylabel(r"$y(x)$", fontsize=14)
ax.legend()
plt.show()
```

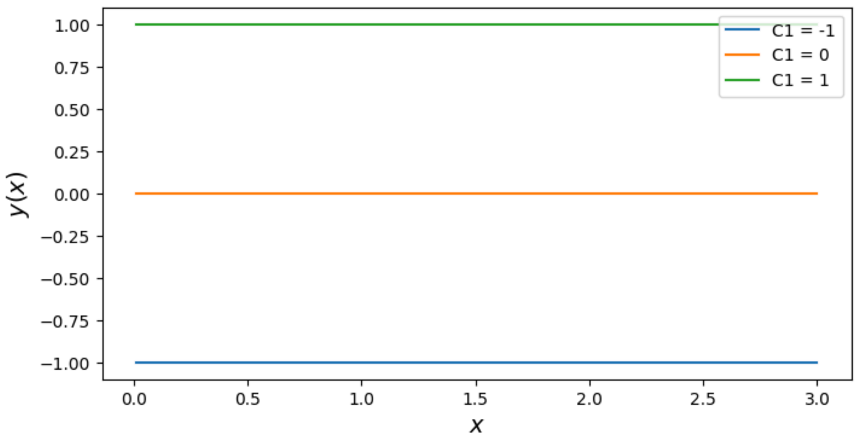


y' = (64xy)^(1/3)

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx, (64*x*y)**(1/3))
y_t_sol = sp.dsolve(expr)
y_t_sol
```

$$-\frac{3.0\left(xy(x)\right)^{\frac{4}{3}}}{y^{1.33333333333333}(x)} + 1.5y^{0.666666666666667}(x) = C_1$$

```
diff_eq = sp.Eq(sp.Derivative(y, x), (64*x*y)**(1/3))
sol = sp.dsolve(diff_eq, y)
fig, ax = plt.subplots(figsize=(8, 4))
for C_value in [-1, 0, 1]:
    y_values = [sol.subs('C1', C_value).rhs.subs(x, val).evalf() for val in xx]
    ax.plot(xx, y_values, label=f"C1 = {C_value}")
ax.set_xlabel(r"$x$", fontsize=14)
ax.set_ylabel(r"$y(x)$", fontsize=14)
ax.legend()
plt.show()
```

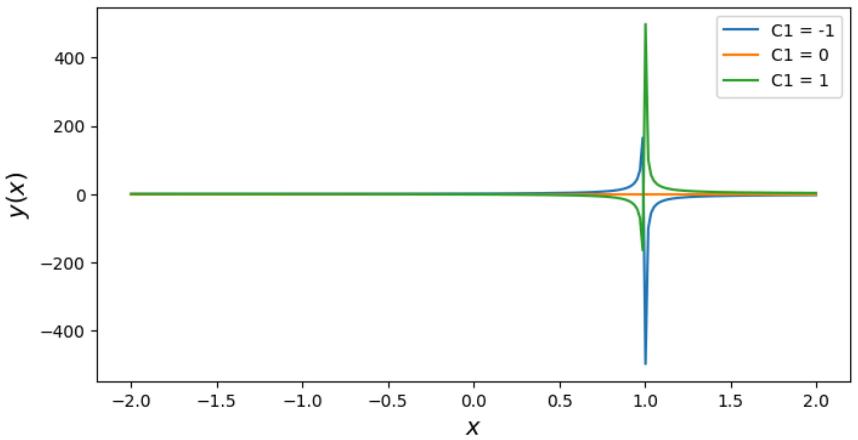


(1 - x^2) y' = 2y

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx*(1-x**2), 2*y)
y_t_sol = sp.dsolve(expr)
y_t_sol
```

$$y(x) = \frac{C_1 (x+1)}{x-1}$$

```
diff_eq = sp.Eq((1 - x**2) * sp.Derivative(y, x), 2 * y)
sol = sp.dsolve(diff_eq, y)
xx = np.linspace(-2, 2, 250)
fig, ax = plt.subplots(figsize=(8, 4))
for C_value in [-1, 0, 1]:
    y_values = [sol.subs('C1', C_value).rhs.subs(x, val).evalf() for val in xx]
    ax.plot(xx, y_values, label=f"C1 = {C_value}")
ax.set_xlabel(r"$x$", fontsize=14)
ax.set_ylabel(r"$y(x)$", fontsize=14)
ax.legend()
plt.show()
```

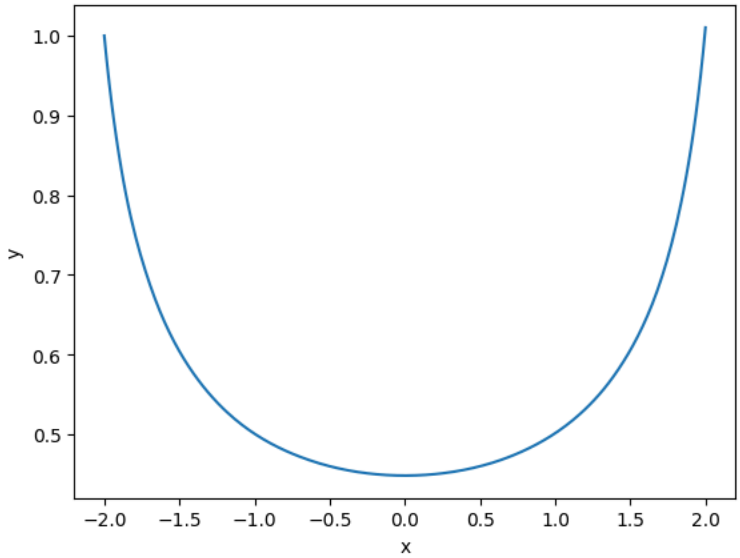


y' = x y^3

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx, x*y**3)
y_t_sol = sp.dsolve(expr)
y_t_sol
```

$$\left[y(x) = -\sqrt{-\frac{1}{C_1 + x^2}},\ y(x) = \sqrt{-\frac{1}{C_1 + x^2}} \right]$$

```
from scipy.integrate import solve_ivp
def diff_eq(x, y):
    return x * y**3
x0 = -2
y0 = [1]
x_range = (-2, 2)
sol = solve_ivp(diff_eq, x_range, y0, t_eval=np.linspace(*x_range, 250))
plt.plot(sol.t, sol.y[0])
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



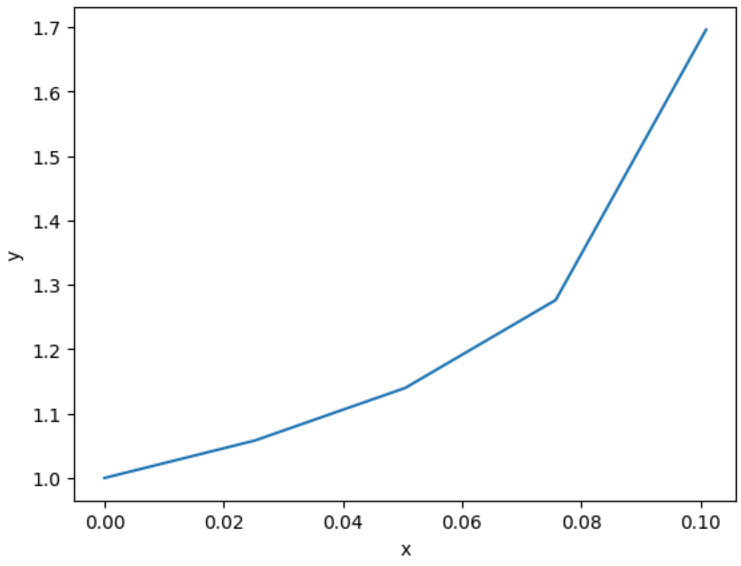
y^3 y' = (y^4 + 1) cos x

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx*y**3, sp.cos(x)*(y**4+1))
y_t_sol = sp.dsolve(expr)
y_t_sol
```

$\left[y(x) = -(-1)^{\frac{3}{4}} \sqrt[4]{C_1 e^{4\sin(x)} + 1},\ y(x) = (-1)^{\frac{3}{4}} \sqrt[4]{C_1 e^{4\sin(x)} + 1},\ y(x) = -\sqrt[4]{C_1 e^{4\sin(x)} -\right.$

```
def diff_eq(x, y):
    return y**3 * (y**4 + 1) * np.cos(x)
y0 = [1]
x_range = (0, 2*np.pi)
sol = solve_ivp(diff_eq, x_range, y0, t_eval=np.linspace(*x_range, 250))
plt.plot(sol.t, sol.y[0])
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
<ipython-input-54-d2cb4149a2fa>:2: RuntimeWarning: overflow encountered in multiply
  return y**3 * (y**4 + 1) * np.cos(x)
<ipython-input-54-d2cb4149a2fa>:2: RuntimeWarning: overflow encountered in power
  return y**3 * (y**4 + 1) * np.cos(x)
```

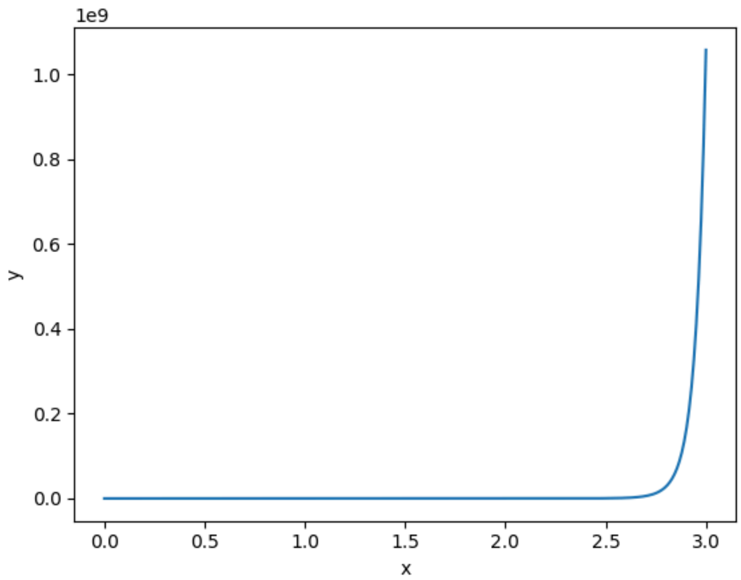


y' = y exp(x), y(0) = 2e

```
import math
e = math.e
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx, y*sp.exp(x))
y_t_sol = sp.dsolve(expr)
ics = {y.subs(x, 0): 2*e}
ivp = sp.dsolve(expr, ics=ics).rhs
ivp
```

$$2.0e^{e^x}$$

```
diff_eq = sp.Eq(sp.Derivative(y, x), y * sp.exp(x))
sol = sp.dsolve(diff_eq, y, ics={y.subs(x, 0): 2 * sp.exp(1)})
y_func = sp.lambdify(x, sol.rhs, 'numpy')
x_vals = np.linspace(0, 3, 250)
y_vals = y_func(x_vals)
plt.plot(x_vals, y_vals)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

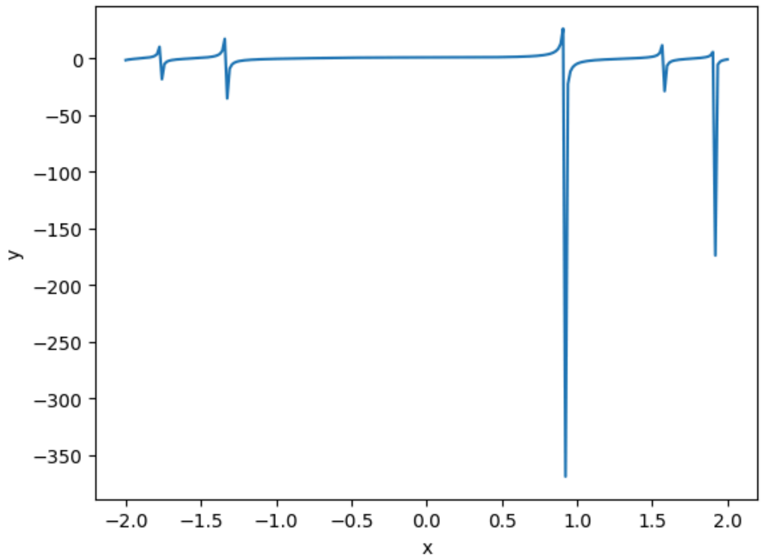


y' = 3 x^2 (y^2 +1), y(0) = 1

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx, 3*x**2*(y**2+1))
y_t_sol = sp.dsolve(expr)
ics = {y.subs(x, 0): 1}
ivp = sp.dsolve(expr, ics=ics).rhs
ivp
```

$$\tan\left(x^3 + \frac{\pi}{4}\right)$$

```
diff_eq = sp.Eq(sp.Derivative(y, x), 3 * x**2 * (y**2 + 1))
sol = sp.dsolve(diff_eq, y, ics={y.subs(x, 0): 1})
y_func = sp.lambdify(x, sol.rhs, 'numpy')
x_vals = np.linspace(-2, 2, 250)
y_vals = y_func(x_vals)
plt.plot(x_vals, y_vals)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

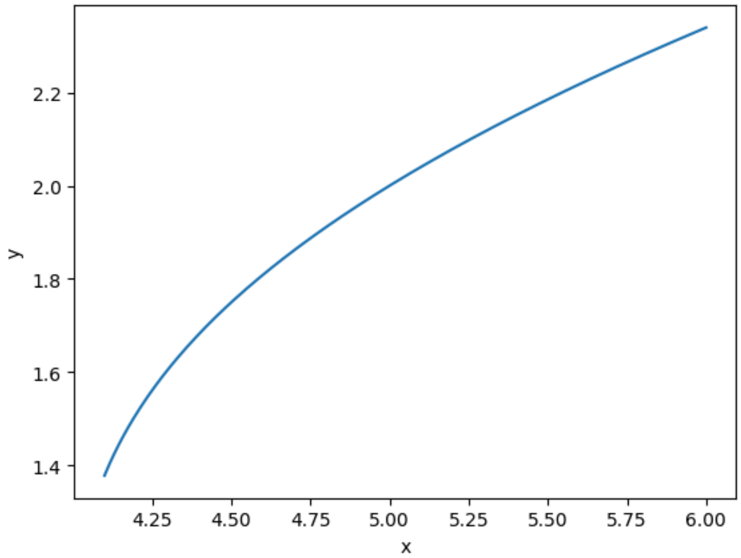


2y y' = x/sqrt(x^2-16), y(5) = 2

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(2*y*dydx, x/(x**2-16)**(1/2))
y_t_sol = sp.dsolve(expr)
ics = {y.subs(x, 5): 2}
ivp = sp.dsolve(expr, ics=ics).rhs
ivp
```

$$2.0\sqrt{1.0\sqrt{0.0625x^2 - 1.0} + 0.25}$$

```
diff_eq = sp.Eq(2 * y * sp.Derivative(y, x), x / sp.sqrt(x**2 - 16))
sol = sp.dsolve(diff_eq, y, ics={y.subs(x, 5): 2})
y_func = sp.lambdify(x, sol.rhs, 'numpy')
x_vals = np.linspace(4.1, 6, 250)
y_vals = y_func(x_vals)
plt.plot(x_vals, y_vals)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

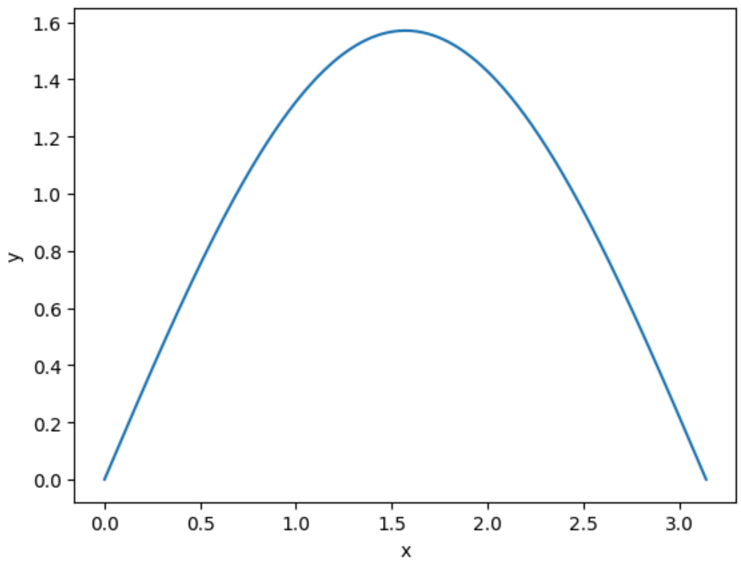


tan(x) y' = y, y(pi/2) = pi/2

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx*sp.tan(x), y)
y_t_sol = sp.dsolve(expr)
ics = {y.subs(x, sp.pi/2): sp.pi/2}
ivp = sp.dsolve(expr, ics=ics).rhs
ivp
```

$$\frac{\pi \sin(x)}{2}$$

```
diff_eq = sp.Eq(sp.tan(x) * sp.Derivative(y, x), y)
sol = sp.dsolve(diff_eq, y, ics={y.subs(x, sp.pi/2): sp.pi/2})
y_func = sp.lambdify(x, sol.rhs, 'numpy')
x_vals = np.linspace(0, np.pi, 250)
y_vals = y_func(x_vals)
plt.plot(x_vals, y_vals)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



y' + y = 2, y(0) = 0

```
x, C1 = sp.symbols('x C1')
y = sp.Function('y')(x)
dydx = y.diff(x)
expr = sp.Eq(dydx + y, 2)
y_t_sol = sp.dsolve(expr)
ics = {y.subs(x, 0): 0}

diff_eq = sp.Eq(sp.Derivative(y, x) + y, 2)
sol = sp.dsolve(diff_eq, y, ics={y.subs(x, 0): 0})
y_func = sp.lambdify(x, sol.rhs, 'numpy')
x_vals = np.linspace(-1, 2, 250)
y_vals = y_func(x_vals)
plt.plot(x_vals, y_vals)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```