```
f.series(x0=0, n=4).removeO()
```

$$\hookrightarrow \quad -\frac{x^3}{6} + x$$

Removing the higher-order terms is frequently useful when we would like to use the nth-order **Taylor polynomial**, a truncated Taylor series, as an approximation of a function.

4.7 Solving Ordinary Differential Equations

Engineering analysis regularly includes the solution of differential equations. **Differential equations** are those equations that contain derivatives. An **ordinary differential equation (ODE)** is a differential equation that contains only ordinary, as opposed to partial, derivatives. A **linear ODE**—one for which constant multiples and sums of solutions are also solutions—is an important type that represent **linear, time-varying (LTV) systems**. For this class of ODEs, it has been proven that for a set of initial conditions, a unique solution exists (Kreyszig 2010; p. 108).

A **constant-coefficient, linear ODE** can represent **linear, time-invariant (LTI) systems**. An LTV or LTI system model can be represented as a scalar nth-order ODE, or as a system of n 1st-order ODEs. As a scalar nth-order linear ODE, with independent time variable t, output function $y(t)$, forcing function $f(t)$, and constant coefficients a_i, has the form

$$y^{(n)}(t) + a_{n-1}y^{(n-1)}(t) + \cdots a_1 y'(t) + a_0 y(t) = f(t). \tag{4.23}$$

The forcing function $f(t)$ can be written as a linear combination of derivatives of the input function $u(t)$ with $m + 1 \leq n + 1$ constant coefficients b_j, as follows:

$$f(t) = b_m u^{(m)}(t) + b_{m-1}u^{(m-1)}(t) + \cdots + b_1 u'(t) + b_0 u(t).$$

Alternatively, the same LTI system model can be represented by a system of n 1st-order ODEs, which can be written in vector form as

$$\boldsymbol{x}'(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \tag{4.24a}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t), \tag{4.24b}$$

where $\boldsymbol{x}(t)$ is called the state vector, $\boldsymbol{u}(t)$ is called the input vector, and $\boldsymbol{y}(t)$ is called the output vector (they are actually vector-valued functions of time), and $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are matrices containing constants derived from system parameters (e.g., a mass, a spring constant, a capacitance, etc.). Equation (4.24) is called an LTI **state-space model**, and it is used to model a great many engineering systems.

Solving ODEs and systems of ODEs is a major topic of mathematical engineering analysis. It is typically the primary topic of one required course and a secondary

topic of several others. Understanding when these solutions exist, whether they are unique, and how they can be found adds much to the understanding of engineering systems. However, it is also true that CASs such as SymPy offer the engineer excellent tools for making quick and adaptable work of this task.

Consider the ODE

$$3y'(t) + y(t) = f(t),$$

where the forcing function $f(t)$ is defined piecewise as

$$f(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 0. \end{cases}$$

The SymPy `dsolve()` function can find the **general solution** (i.e., a family of solutions for any initial conditions) with the following code:

```
t = sp.symbols("t", nonnegative=True)
y = sp.Function("y", real=True)
f = 1   # Or sp.Piecewise(), but t≥0 already restricts f(t)
ode = sp.Eq(3*y(t).diff(t) + y(t), f)   # Define the ODE
sol = sp.dsolve(ode, y(t)); sol   # Solve
```

$\hookrightarrow$ $y(t) = C_1 e^{-\frac{t}{3}} + 1$

The solution is returned as an `sp.Eq()` equation object. Note the unknown constant C_1 in the solution. To find the **specific solution** (i.e., the general solution with the initial condition applied to determine C_1) for a given initial condition $y(0) = 5$,

```
sol = sp.dsolve(ode, y(t), ics={y(0): 5}); sol
```

$\hookrightarrow$ $y(t) = 1 + 4e^{-\frac{t}{3}}$

Now consider the ODE

$$y''(t) + 5y'(t) + 9y(t) = 0.$$

The SymPy `dsolve()` function can find the general solution with the following code:

```
ode = sp.Eq(y(t).diff(t, 2) + 5*y(t).diff(t) + 9*y(t), 0)
sol = sp.dsolve(ode, y(t)); sol
```

$\hookrightarrow$ $y(t) = \left(C_1 \sin\left(\frac{\sqrt{11}t}{2} \right) + C_2 \cos\left(\frac{\sqrt{11}t}{2} \right) \right) e^{-\frac{5t}{2}}$

This is a decaying sinusoid. Applying two initial conditions, $y(0) = 4$ and $y'(0) = 0$, we obtain the following:

```
sol = sp.dsolve(
    ode, y(t),
    ics={y(0): 4, y(t).diff(t).subs(t, 0): 0}
); sol
```

$$y(t) = \left(\frac{20\sqrt{11}\sin\left(\frac{\sqrt{11}t}{2}\right)}{11} + 4\cos\left(\frac{\sqrt{11}t}{2}\right) \right) e^{-\frac{5t}{2}}$$

We see here that to apply the initial condition $y'(0) = 0$, the derivative must be applied before substituting $t \to 0$.

Solving sets (i.e., systems) of first-order differential equations is similar. Consider the set of differential equations

$$y_1'(t) = y_2(t) - y_1(t) \text{ and } y_2'(t) = y_1(t) - y_2(t).$$

To find the solution for initial conditions $y_1(0) = 1$ and $y_2(0) = -1$, we can use the following technique:

```python
t = sp.symbols("t", nonnegative=True)
y1, y2 = sp.symbols("y1, y2", cls=sp.Function, real=True)
odes = [y1(t).diff(t) + y1(t) - y2(t), y2(t).diff(t) + y2(t) - y1(t)]
ics = {y1(0): 1, y2(0): -1}
sol = sp.dsolve(odes, [y1(t), y2(t)], ics=ics)
print(sol)
```

```
[Eq(y1(t), exp(-2*t)), Eq(y2(t), -exp(-2*t))]
```

In engineering, it is common to express a set of differential equations as a state-space model, as in equation (4.24). The following example demonstrates how to solve these with SymPy.

Example 4.3

Consider the electromechanical schematic of a direct current (DC) motor shown in figure 4.6. A voltage source $V_S(t)$ provides power, the armature winding loses some energy to heat through a resistance R and stores some energy in a magnetic field due to its inductance L, which arises from its coiled structure. An electromechanical interaction through the magnetic field, shown as M, has torque constant K_M and induces a torque on the motor shaft, which is supported by bearings that lose some energy to heat via a damping coefficient B. The rotor's mass has rotational moment of inertia J, which stores kinetic energy. We denote the voltage across an element with v, the current through an element with i, the angular velocity across an element with Ω, and the torque through an element with T.
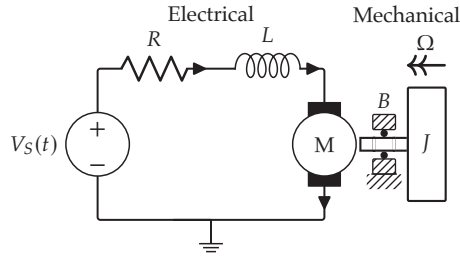
Figure 4.6. An electromechanical schematic of a DC motor.

A state-space model state equation in the form of equation (4.24a) can be derived for this system, with the result as follows:

$$\underbrace{\frac{d}{dt}\begin{bmatrix} \Omega_J \\ i_L \end{bmatrix}}_{x'(t)} = \underbrace{\begin{bmatrix} -B/J & K_M/J \\ -K_M/L & -R/L \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \Omega_J \\ i_L \end{bmatrix}}_{x(t)} + \underbrace{\begin{bmatrix} 0 \\ 1/L \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} V_S \end{bmatrix}}_{u(t)} \ .$$

We choose $y = \begin{bmatrix} \Omega_J \end{bmatrix}$ as the output vector, which yields output equation (i.e., equation (4.24b))

$$\underbrace{\begin{bmatrix} \Omega_J \end{bmatrix}}_{y(t)} = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} \Omega_J \\ i_L \end{bmatrix}}_{x(t)} + \underbrace{\begin{bmatrix} 0 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} V_S \end{bmatrix}}_{u(t)} \ .$$

Together, these equations are a state-space model for the system.

Solve the state equation for $x(t)$ and the output equation for $y(t)$ for the following case:

- The input voltage $V_S(t) = 1$ V for $t \geq 0$
- The initial condition is $x(0) = \mathbf{0}$

We begin by defining the parameters and functions of time as SymPy symbolic variables and unspecified functions as follows:

```
R, L, K_M, B, J = sp.symbols("R, L, K_M, B, J", positive=True)
W_J, i_L, V_S = sp.symbols(
    "W_J, i_L, V_S", cls=sp.Function, real=True
)  # Ω_J, i_L, V_S
t = sp.symbols("t", real=True)
```

Now we can form the symbolic matrices and vectors:

```
A_ = sp.Matrix([[-B/J, K_M/J], [-K_M/L, -R/L]])   # A
B_ = sp.Matrix([[0], [1/L]])   # B
C_ = sp.Matrix([[1, 0]])   # C
D_ = sp.Matrix([[0]])   # D
x = sp.Matrix([[W_J(t)], [i_L(t)]])   # x
u = sp.Matrix([[V_S(t)]])   # u
y = sp.Matrix([[W_J(t)]])   # y
```

The input and initial conditions can be encoded as follows:

```
u_subs = {V_S(t): 1}
ics = {W_J(0): 0, i_L(0): 0}
```

The set of first-order ODEs comprising the state equation can be defined as follows:

```
odes = x.diff(t) - A_*x - B_*u
print(odes)
```

$$
\begin{bmatrix}
\frac{BW_J(t)}{J} + \frac{d}{dt}W_J(t) - \frac{K_M i_L(t)}{J} \\
\frac{K_M W_J(t)}{L} + \frac{d}{dt}i_L(t) + \frac{R i_L(t)}{L} - \frac{V_S(t)}{L}
\end{bmatrix}
$$

```
x_sol = sp.dsolve(list(odes.subs(u_subs)), list(x), ics=ics)
```

The symbolic solutions for $x(t)$ are lengthy expressions. Instead of printing them, we will graph them for the following set of parameters:

```
params = {
    R: 1,   # (Ohms)
    L: 0.1e-6,   # (H)
    K_M: 7,   # (N·m/A)
    B: 0.1e-6,   # (N·m/(rad/s))
    J: 2e-6,   # (kg·m²)
}
```

Create a numerically evaluable version of each function as follows:

```
W_J_ = sp.lambdify(t, x_sol[0].rhs.subs(params), modules="numpy")
i_L_ = sp.lambdify(t, x_sol[1].rhs.subs(params), modules="numpy")
```

Graph each solution as follows:

```
t_ = np.linspace(0, 0.000002, 201)
fig, axs = plt.subplots(2, sharex=True)
axs[0].plot(t_, W_J_(t_))
axs[1].plot(t_, i_L_(t_))
axs[1].set_xlabel("Time (s)")
axs[0].set_ylabel("$\\Omega_J(t)$ (rad/s)")
axs[1].set_ylabel("$i_L(t)$ (A)")
plt.show()
```
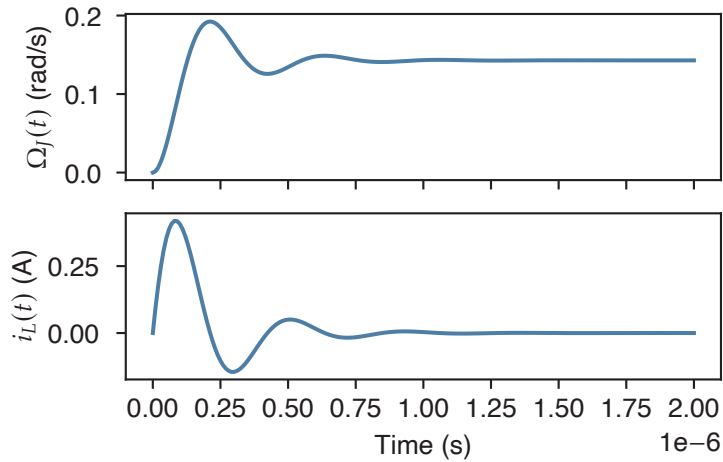
Figure 4.7. The state response to a unit step voltage input.

The output equation is trivial in this case, yielding only the state variable $\Omega_J(t)$, for which we have already solved. Therefore, we have completed the analysis.

4.8   Problems

Problem 4.1 5Z   Let $s \in \mathbb{C}$. Use SymPy to perform a partial fraction expansion on the following expression:

$$\frac{(s+2)(s+10)}{s^4 + 8s^3 + 117s^2 + 610s + 500}.$$

Problem 4.2 IE   Let $x, a_1, a_2, a_3, a_4 \in \mathbb{R}$. Use SymPy to combine the cosine and sine terms that share arguments into single sinusoids with phase shifts in the following expression:

$$a_1 \sin(x) + a_2 \cos(x) + a_3 \sin(2x) + a_4 \cos(2x)$$

Problem 4.3 K2   Let $s \in \mathbb{C}$ and $a_i, b_j \in \mathbb{R}$. Consider the following expression:

$$\frac{a_3 s^3 + a_2 s^2}{a_1 s + a_0} \cdot \frac{b_2 s^2}{b_1 s + b_0}.$$

Use SymPy to write the expression as a fraction with the numerator and denominator as polynomials in s with the terms collected such that each power of s appears only once in each polynomial.

Problem 4.4 QR   Let $t \in \mathbb{R}$. Use SymPy to simplify the following expression:

$$\frac{\cos(t) + \sin(t)}{\cos(t) - \sin(t)} + (\cos(t) + \sin(t))^2.$$

First try using the `simplify()` function. Then try using the `trigsimp()` function.

Problem 4.5 W1   Let $x, y, z \in \mathbb{R}$ be variables and $a, b, c \in \mathbb{R}$ be parameters. Suppose we have the following solutions for x, y, and z in terms of a, b, and c:

$$x = a^2 + b^2 \tag{4.25}$$
$$y = a^2 - b^2 \tag{4.26}$$
$$z = a^2 b^2. \tag{4.27}$$

Furthermore, suppose the numerical values of the parameters are

$$a = 1, \quad b = 2, \quad c = 3.$$

Use SymPy to substitute the solutions for x, y, and z into the following expression:

$$\frac{x+y}{x-y} + (x+y)^2 + (x-y)^2 + z^2.$$