# 4 Symbolic Analysis

## Problem Solutions

KB

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

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

**Solution 4.1** 5Z First, load SymPy and define the expression:

```
import sympy as sp
s = sp.symbols("s", complex=True)
expr = (s + 2)*(s + 10)/(s**4 + 8*s**3 + 117*s**2 + 610*s + 500)
```

The SymPy `apart()` method can be directly applied to find the partial fraction expansion:

```
expr.apart()
```

$$-\frac{2\cdot(7s-136)}{253\,(s^2+2s+100)}+\frac{3}{92\,(s+5)}+\frac{1}{44\,(s+1)}$$

**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:

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

**Solution 4.2** IE First, load SymPy as follows:

```
import sympy as sp
```

Unfortunately, the SymPy `trigsimp()` method doesn't combine the cosine and sine terms that share an argument. However, we can define a functions `trig_two_to_one()` for performing the two-to-one conversion. The identities we will use are as follows:

$$A \sin u + B \cos u = \sqrt{A^2 + B^2} \sin(u + \arctan(B/A))$$
$$= \sqrt{A^2 + B^2} \cos(u - \arctan(A/B)).$$

We begin by defining two functions, each of which returns a dictionary of replacement rules for sin(u) in the identities above.

```
def _trig_two_to_one_sin_rule(A, B, u):
  """Replacement rule for sin(u) in A sin(u) + B cos(u) for
  converting to a single sin term with a phase

  The identity applied is:
    A sin u + B cos u = sqrt(A^2 + B^2) sin(u + atan(B/A))
  Returns: A dictionary for replacing sin(u)
  """
  identity = A*sp.sin(u) + B*sp.cos(u) + \
    - sp.sqrt(A**2 + B**2)*sp.sin(u + sp.atan(B/A))
  sol = sp.solve(identity, [sp.sin(u)], dict=True)
  return sol[0]

def _trig_two_to_one_cos_rule(A, B, u):
  """Replacement rule for sin(u) in A sin(u) + B cos(u) for
  converting to a single cos term with a phase

  The identity applied is:
    A sin u + B cos u = sqrt(A^2 + B^2) cos(u - atan(A/B))
  Returns: A dictionary for replacing sin(u)
  """
  identity = A*sp.sin(u) + B*sp.cos(u) + \
    - sp.sqrt(A**2 + B**2)*sp.cos(u - sp.atan(A/B))
  sol = sp.solve(identity, [sp.sin(u)], dict=True)
  return sol[0]
```

Now we can write a function to perform the trigonometric simplification, as follows:

```python
def trig_two_to_one(expr: sp.Expr, to: str = "sin"):
  """Rewrites sin and cos terms that share arguments to single sin
  or cos terms

  Applies the following identity:
    A sin u + B cos u = sqrt(A^2 + B^2) sin(u + atan(B/A))
  or
    A sin u + B cos u = sqrt(A^2 + B^2) cos(u - atan(A/B))
  depending on the ``to`` argument.

  Args:
    expr: The symbolic expression containing sin(u) and cos(u)
    to: Rewrite with "sin" (default) or "cos"
  """
  expr = expr.simplify()
  # Identify sin terms
  w1 = sp.Wild("w1", exclude=[1])
  w2 = sp.Wild("w2")
  sin_terms = expr.find(w1*sp.sin(w2))
  sin_arg_amps = {}  # To be: {sin argument: sine amplitude}
  for term in sin_terms:
    arg_amp_rules = term.match(w1*sp.sin(w2))
    sin_arg_amps[arg_amp_rules[w2]] = arg_amp_rules[w1]
  # Identify cos terms
  cos_terms = expr.find(w1*sp.cos(w2))
  cos_arg_amps = {}  # To be: {sin argument: sine amplitude}
  for term in cos_terms:
    arg_amp_rules = term.match(w1*sp.cos(w2))
    cos_arg_amps[arg_amp_rules[w2]] = arg_amp_rules[w1]
  # Replace with wildcard rule
  for sin_arg, sin_amp in sin_arg_amps.items():
    if to == "sin":
      if sin_arg in cos_arg_amps.keys():
        cos_amp = cos_arg_amps[sin_arg]
        sin_rule = _trig_two_to_one_sin_rule(
          sin_amp, cos_amp, sin_arg
        )
        expr = expr.subs(sin_rule)
    elif to == "cos":
      if sin_arg in cos_arg_amps.keys():
        cos_amp = cos_arg_amps[sin_arg]
        cos_rule = _trig_two_to_one_cos_rule(
          sin_amp, cos_amp, sin_arg
        )
        expr = expr.subs(cos_rule)
  return expr.simplify()
```

The expression of interest in the problem is defined here:

```
x, a1, a2, a3, a4 = sp.symbols("x, a1, a2, a3, a4", real=True)
expr = a1*sp.sin(x) + a2*sp.cos(x) + a3*sp.sin(2*x) + a4*sp.cos(2*x)
```

Apply the function to the expression, rewriting in terms of sine (first) and cosine (second):

```
trig_two_to_one(expr, to="sin")
trig_two_to_one(expr, to="cos")
```

$$\sqrt{a_1^2+a_2^2}\sin\left(x+\operatorname{atan}\left(\frac{a_2}{a_1}\right)\right)+\sqrt{a_3^2+a_4^2}\sin\left(2x+\operatorname{atan}\left(\frac{a_4}{a_3}\right)\right)$$

$$\sqrt{a_1^2+a_2^2}\cos\left(x-\operatorname{atan}\left(\frac{a_1}{a_2}\right)\right)+\sqrt{a_3^2+a_4^2}\cos\left(2x-\operatorname{atan}\left(\frac{a_3}{a_4}\right)\right)$$

**Problem 4.3** KR Consider the following equation, where $x \in \mathbb{C}$ and $a, b, c \in \mathbb{R}_+$,

$$ax^2 + bx + \frac{c}{x} + b^2 = 0.$$

Use SymPy to solve for $x$.

**Solution 4.3** KR First, load SymPy as follows:

```
import sympy as sp
```

Now define the symbolic variables and the equation:

```
x = sp.symbols("x", complex=True)
a, b, c = sp.symbols("a, b, c", positive=True)
eq = a*x**2 + b*x + c/x + b**2  # == 0
```

Let's try `sp.solve()`, as follows:

```
sol = sp.solve(eq, x, dict=True)
print(sol)
```

```
[{x: -(-3*b**2/a + b**2/a**2)/(3*(sqrt(-4*(-3*b**2/a + b**2/a**2)**3
↪  + (27*c/a - 9*b**3/a**2 + 2*b**3/a**3)**2)/2 + 27*c/(2*a) -
↪  9*b**3/(2*a**2) + b**3/a**3)**(1/3)) - (sqrt(-4*(-3*b**2/a +
↪  b**2/a**2)**3 + (27*c/a - 9*b**3/a**2 + 2*b**3/a**3)**2)/2 +
↪  27*c/(2*a) - 9*b**3/(2*a**2) + b**3/a**3)**(1/3)/3 - b/(3*a)}, {x:
↪  -(-3*b**2/a + b**2/a**2)/(3*(-1/2 -
↪  sqrt(3)*I/2)*(sqrt(-4*(-3*b**2/a + b**2/a**2)**3 + (27*c/a -
↪  9*b**3/a**2 + 2*b**3/a**3)**2)/2 + 27*c/(2*a) - 9*b**3/(2*a**2) +
↪  b**3/a**3)**(1/3)) - (-1/2 - sqrt(3)*I/2)*(sqrt(-4*(-3*b**2/a +
↪  b**2/a**2)**3 + (27*c/a - 9*b**3/a**2 + 2*b**3/a**3)**2)/2 +
↪  27*c/(2*a) - 9*b**3/(2*a**2) + b**3/a**3)**(1/3)/3 - b/(3*a)}, {x:
↪  -(-3*b**2/a + b**2/a**2)/(3*(-1/2 +
↪  sqrt(3)*I/2)*(sqrt(-4*(-3*b**2/a + b**2/a**2)**3 + (27*c/a -
↪  9*b**3/a**2 + 2*b**3/a**3)**2)/2 + 27*c/(2*a) - 9*b**3/(2*a**2) +
↪  b**3/a**3)**(1/3)) - (-1/2 + sqrt(3)*I/2)*(sqrt(-4*(-3*b**2/a +
↪  b**2/a**2)**3 + (27*c/a - 9*b**3/a**2 + 2*b**3/a**3)**2)/2 +
↪  27*c/(2*a) - 9*b**3/(2*a**2) + b**3/a**3)**(1/3)/3 - b/(3*a)}]
```

This is a cubic solution, so it is unwieldy.

**Problem 4.4** G9 Let $w, x, y, z \in \mathbb{R}$. Consider the following system of equations:

$$\begin{aligned} 8w - 6x + 5y + 4z &= -20 \\ 2y - 2z &= 10 \\ 2w - x + 4y + z &= 0 \\ w + 4x - 2y + 8z &= 4. \end{aligned}$$

Use SymPy to solve the system for $w$, $x$, $y$, and $z$.

**Solution 4.4** G9 Load SymPy as follows:

```
import sympy as sp
```

Now define the symbolic variables:

```
w, x, y, z = sp.symbols("w, x, y, z", complex=True)
```

Define the set of equations:

```
S = [
  8*w - 6*x + 5*y + 4*z + 20,  # == 0
  2*y - 2*z - 10,  # == 0
  2*w - x + 4*y + z,  # == 0
  w + 4*x - 2*y + 8*z - 4, # == 0
]
```

Let's try `sp.solve()`, as follows:

```
sol = sp.solve(S, [w, x, y, z], dict=True)
print(sol)
```

```
[{w: 564/85, x: 773/85, y: 14/85, z: -411/85}]
```

**Problem 4.5** 49 You are designing the truss structure shown in figure 4.6, which is to support the hanging of an external load $\boldsymbol{f}_C = -f_C\hat{\boldsymbol{j}}$, where $f_C > 0$. Your organization plans to offer customers the following options:

- Any width (i.e., $2w$)
- A selection of maximum load magnitudes $L = f_C/\alpha \in \Gamma$, where $\Gamma = \{1\text{ kN}, 2\text{ kN}, 4\text{ kN}, 8\text{ kN}, 16\text{ kN}\}$, and where $\alpha$ is the factor of safety

As the designer, you are to develop a design curve for the dimension $h$ versus half-width $w$ for each maximum load $L \in \Gamma$, under the following design constraints:

- Minimize the dimension $h$
- The tension in all members is no more than a given $T$
- The compression in all members is no more than a given $C$
- The magnitude of the support force at pin A is no more than a given $P_A$
- The magnitude of the support force at pin D is no more than a given $P_D$

Use a static analysis and the method of joints to develop a solution for the force in each member $F_{AB}$, $F_{AC}$, etc., and the reaction forces using the sign convention that tension is positive and compression is negative. Create a Python function that returns $h$ as a function of $w$ and $L$ for a given set of design parameters $\{T, C, P_A, P_C, \alpha\}$. Use the function to create a design curve $h$ versus $2w$ for each $L \in \Gamma$ and a factor of safety of $\alpha = 5$.

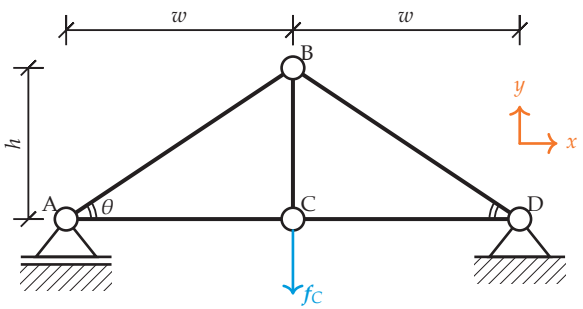


Figure 4.6. A truss with pinned joints, supported by a hinge and a floating support, with an applied load $\boldsymbol{f}_C$.