```python
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 $f_C = -f_C\hat{j}$, 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 α 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 for a given set of design parameters $\{T, C, P_A, P_D, \alpha, L\}$. Use the function to create a design curve h versus $2w$ for each $L \in \Gamma$, maximum tension $T = 81$ kN, maximum compression $C = 81$ kN, maximum support A load $P_A = 50$ kN, maximum support D load $P_D = 50$ kN, and a factor of safety of $\alpha = 5$.

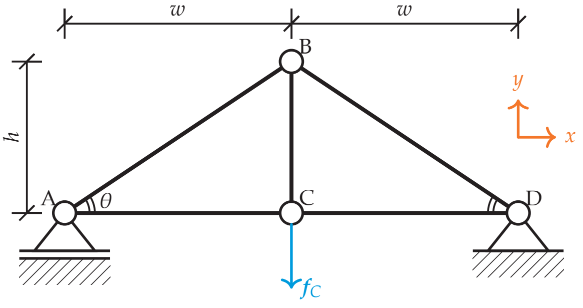

Figure 4.6. A truss with pinned joints, supported by a hinge and a floating support, with an applied load f_C.

Solution 4.5 49

```
#### Import Packages {-}
```

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

Static Analysis Using the method of joints, we proceed through the joints, summing forces in the x- and y-directions. We will assume all members are in tension, and their sign will be positive if this is the case and negative, otherwise. Beginning with joint A, which includes one reaction force R_A from the support,

$$\Sigma F_x = 0; \qquad F_{AC} + F_{AB}\cos\theta = 0 \tag{4.1}$$

$$\Sigma F_y = 0; \qquad R_A + F_{AB}\sin\theta = 0. \tag{4.2}$$

The angle θ is known in terms of the dimensions w and h as

$$\theta = \arctan\frac{h}{w}.$$

These equations can be encode symbolically as follows:

```
RA, FAB, FAC, theta= sp.symbols(
  "RA, FAB, FAC, theta", real=True
)
h, w = sp.symbols("h, w", positive=True)
eqAx = FAC + FAB*sp.cos(theta)
eqAy = RA + FAB*sp.sin(theta)
theta_wh = sp.atan(h/w)
```

Proceeding to joint B,

$$\Sigma F_x = 0; \qquad -F_{AB}\cos\theta + F_{BD}\cos\theta = 0 \tag{4.3}$$

$$\Sigma F_y = 0; \qquad -F_{AB}\sin\theta - F_{BD}\sin\theta - F_{BC} = 0. \tag{4.4}$$

Encoding these equations,

```
FBD, FBC = sp.symbols("FBD, FBC", real=True)
eqBx = -FAB*sp.cos(theta) + FBD*sp.cos(theta)
eqBy = -FAB*sp.sin(theta) - FBD*sp.sin(theta) - FBC
```

For joint C, there is an externally applied force f_C, so the analysis proceeds as follows:

$$\Sigma F_x = 0; \qquad -F_{AC} + F_{CD} = 0 \tag{4.5}$$

$$\Sigma F_y = 0; \qquad F_{BC} - f_C = 0. \tag{4.6}$$

Encoding these equations,

```
FCD = sp.symbols("FCD", real=True)
fC = sp.symbols("fC", positive=True)
eqCx = -FAC + FCD
eqCy = FBC - fC
```

For joint D, the pinned reaction forces R_{Dx} and R_{Dy} are present, so the analysis proceeds as follows:

$$\Sigma F_x = 0; \qquad -F_{CD} - F_{BD}\cos\theta + R_{Dx} = 0 \tag{4.7}$$

$$\Sigma F_y = 0; \qquad F_{BD}\sin\theta + R_{Dy} = 0. \tag{4.8}$$

Encoding these equations,

```
RDx, RDy = sp.symbols("RDx, RDy", real=True)
eqDx = -FCD - FBD*sp.cos(theta) + RDx
eqDy = FBD*sp.sin(theta) + RDy
```

In total, we have 8 force equations and 8 unknown forces (5 member forces and 3 reaction forces). Let's construct the system and solve it for the unkown forces, as follows:

```
S_forces = [
  eqAx, eqAy, eqBx, eqBy, eqCx, eqCy, eqDx, eqDy
]  # 8 force equations
member_forces = [FAB, FAC, FBC, FBD, FCD]  # 5 member forces
reaction_forces = [RA, RDx, RDy]  # 3 reaction forces
forces_unknown = member_forces + reaction_forces  # 8 unkown forces
sol_forces = sp.solve(S_forces, forces_unknown, dict=True); sol_forces
```

```
[{FAB: -fC/(2*sin(theta)),
  FAC: fC*cos(theta)/(2*sin(theta)),
  FBC: fC,
  FBD: -fC/(2*sin(theta)),
  FCD: fC*cos(theta)/(2*sin(theta)),
  RA: fC/2,
  RDx: 0,
  RDy: fC/2}]
```

This solution is in terms of f_C and θ. Because w and h are our design parameters, let's substitute `eqtheta` such that our solution is rewritten in terms of f_F, w, and h. Create a list of solutions as follows:

```python
forces_wh = {}  # Initialize
for force in forces_unknown:
  force_wh = force.subs(
    sol_forces[0]
  ).subs(
    theta, theta_wh
  ).simplify()
  forces_wh[force] = force_wh
  print(f"{force} = {force_wh}")
```

```
FAB = -fC*sqrt(h**2 + w**2)/(2*h)
FAC = fC*w/(2*h)
FBC = fC
FBD = -fC*sqrt(h**2 + w**2)/(2*h)
FCD = fC*w/(2*h)
RA = fC/2
RDx = 0
RDy = fC/2
```

Define the constraints as follows:

```python
C, T, PA, PD = sp.symbols("C, T, PA, PD", positive=True)
member_constraints = {"Tension": T, "Compression": C}
reaction_constraints = {
  PA: (RA, 0),
  PD: (RDx, RDy)
}  # Max magnitude: (x force, y force)
```

Define a function that encodes the constraints as a list of expressions that must be nonnegative.

```python
def get_force_constraints(
  forces: dict,
  member_forces: list,
  member_constraints: dict,
  reaction_constraints: dict,
):
  """Returns a list of expressions that must be nonnegative

  Args:
    - forces: Force solutions {force symbol: solution}
    - member_forces: List of member symbols [FAB, FAC ...]
    - member_constraints:
      {"Tension": max tension, "Compression": max compression}
    - reaction_constraints:
      {max force symbol: (max x force, max y force)}
  """
  force_constraints = []
  # Append member constraints
  for f_name, f_value in forces.items():
    if f_name in member_forces:
      if f_value > 0:  # Tension
        force_constraints.append(
          member_constraints["Tension"] - f_value
        )
      elif f_value < 0:  # Compression
        force_constraints.append(
          member_constraints["Compression"] + f_value
        )
  # Append reaction constraints
  for constraint, pair in reaction_constraints.items():
    force_constraints.append(
      constraint - sp.sqrt(pair[0]**2 + pair[1]**2).subs(forces_wh)
    )
  return force_constraints
```

Now apply the function:

```python
constraints = get_force_constraints(
  forces_wh, member_forces, member_constraints, reaction_constraints
)
print(constraints)
```

```
[C - fC*sqrt(h**2 + w**2)/(2*h), T - fC*w/(2*h), T - fC, C -
↪  fC*sqrt(h**2 + w**2)/(2*h), T - fC*w/(2*h), PA - fC/2, PD - fC/2]
```

At this point, we can solve for h in each expression that includes h. Because these constraints are to be nonnegative, solving with equality gives the minimum $h_{\min}$.

Expressions that don't include h must still be checked. We can use this opportunity to sort the constraint expressions into those that involve h and those that do not.

```
h_mins = []
constraints_to_check = []
for constraint in constraints:
  if h in constraint.free_symbols:
    # Solve for h
    h_min_sol = sp.solve(constraint, h)
    for h_min in h_min_sol:
      h_mins.append(h_min)
  else:
    constraints_to_check.append(constraint)
```

Inspect the `h_mins` and `constraints_to_check`:

```
print(f"Min h solutions: {h_mins}")
print(f"Constraints to be checked: {constraints_to_check}")
```

```
Min h solutions: [fC*w*sqrt(1/(2*C - fC))/sqrt(2*C + fC), fC*w/(2*T),
↪  fC*w*sqrt(1/(2*C - fC))/sqrt(2*C + fC), fC*w/(2*T)]
Constraints to be checked: [T - fC, PA - fC/2, PD - fC/2]
```

Finally, define a function to design the truss:

```
def truss_designer(
  h_mins: list,
  constraints_to_check: list,
  design_params: dict,
):
  """Returns an expression for h(w) using the design parameters provided
  """
  L = sp.symbols("L", positive=True)
  alpha = sp.symbols("alpha", positive=True)
  for constraint in constraints_to_check:
    constraint_ = constraint.subs(fC, L*alpha).subs(design_params)
    if constraint_ < 0:
      raise Exception(
        f"Design failed for constraint: {constraint} < 0"
      )
  h_mins_ = []
  for h_min in h_mins:
    h_mins_.append(h_min.subs(fC, L*alpha).subs(design_params))
  return max(h_mins_)  # Maximum h_min
```

Define the given design parameters in a dictionary:

```python
alpha = sp.symbols("alpha", positive=True)
design_parameters = {
  T: 81e3, C: 81e3, PA: 50e3, PD: 50e3, alpha: 5,
}  # Forces in N
Gamma = [1e3, 2e3, 4e3, 8e3, 16e3]
```

Loop through the L loads, running `truss_designer()`:

```python
h_mins_w = []
L = sp.symbols("L", positive=True)
for L_ in Gamma:
  design_parameters[L] = L_
  h_min = truss_designer(
    h_mins, constraints_to_check, design_parameters
  )
  h_mins_w.append(h_min)
  print(h_min)
```

```
0.0308789086390917*w
0.0618463369994117*w
0.124408521678431*w
0.254802914503365*w
0.56790458868584*w
```

Convert these symbolic expressions into numerically evaluable function, as follows:

```python
h_min_funs = []
for h_min in h_mins_w:
  h_min_funs.append(
    sp.lambdify([w], h_min, modules=np)
  )
```

Plot the design curves as follows:

```python
w_ = np.linspace(1, 20, 101)
fig, ax, = plt.subplots()
for ih, h_min_fun in enumerate(h_min_funs):
  ax.semilogy(2*w_, h_min_fun(w_), label=f"L = {Gamma[ih]}")
ax.set_xlabel(f"width $2w$ (m)")
ax.set_ylabel(f"height $h$ (m)")
ax.grid()
ax.legend()
plt.show()
```

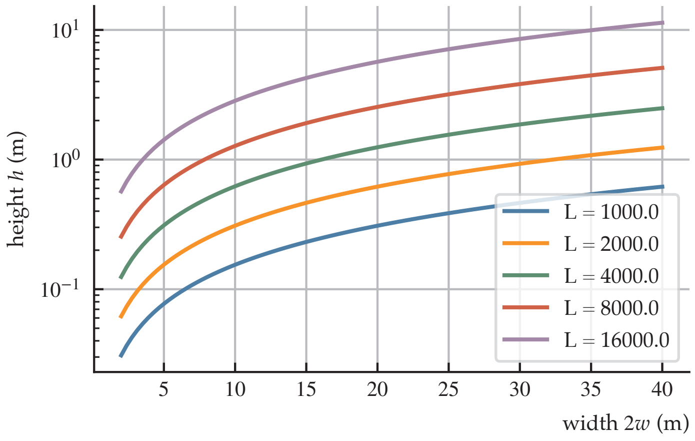

Figure S4.2. Truss design curve.