Thermal Design of 2 Rooms and Feed-back Indoor Temperature Control

June 15, 2022

Authors:

- Vyshnav THANIYIL CHANGATTU
- Anton PILHAGE
- Henry LY

Objective: - Physical analysis of model rooms with 7 concrete walls including insulation, 3 doors, 4 windows, and ventillations. - Model a controller for indoor temperature.

1 Model

Here, we define the parameters required of the model and develop mathemtical model of the design.

1.1 Elements of the thermal network

The thermal resistances for conduction are of the form:

$$R_{cd} = \frac{w}{\lambda S}$$

where:

- w is the width of the material, m;
- λ - thermal conductvity, W/m K;
- S - surface area of the wall, m²

The thermals resistance for convection are of the form:

$$R_{cv} = \frac{1}{hS}$$

where: - h is the convection coefficient, W/m² K; - S - surface area of the wall, m².

The thermal capacities of the wall are of the form:

$$C_{wall} = \rho_{wall} c_{p,wall} w_{wall} S_{wall}$$

The thermal capacity of the air is:

$$C_{air} = \rho_{air} c_{air} V_{air}$$

The total shortwave incident irradiation on the wall i, G_i, may be estimated as a function of the direct solar irradiation incident on the surface of the walls, G_i^o:

$$S_i G_i = S_i G_i^o + \sum_{j=1}^{n} F_{j,i} S_j \rho_j G_j$$

where: - S_i is the area of the surface of the wall i [m²]; - G_i - total irradiation received directly and by mutiple reflections on surface i [W/m²]; - G_i^o - irradiance received directly from the sun on surface i [W/m²]; - $F_{j,i}$ - view factor between surface j and surface i, $0 F_{j,i} 1$; - ρ_j - reflectivity of surface j, $0 \rho_j 1$.

By taking into account the *reciprocity relation*: $S_i F_{i,j} = S_j F_{j,i}$, the previous equation becomes:

$$\begin{bmatrix} 1-\rho_1 F_{1,1} & -\rho_2 F_{1,2} & ... & -\rho_n F_{1,n} \\ -\rho_1 F_{2,1} & 1-\rho_2 F_{2,2} & ... & -\rho_n F_{2,n} \\ ... & ... & ... & ... \\ -\rho_1 F_{n,1} & -\rho_2 F_{n,1} & ... & 1-\rho_n F_{n,n} \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ ... \\ G_n \end{bmatrix} = \begin{bmatrix} G_1^o \\ G_2^o \\ ... \\ G_n^o \end{bmatrix}$$

or

$$[I - \rho F] G = G^o$$

where:

- \$I = $\begin{bmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & 1 \end{bmatrix}$, \$ is the identity matrix;
- $\rho = \begin{bmatrix} \rho_1 & 0 & ... & 0 \\ 0 & \rho_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & \rho_n \end{bmatrix}$ - matrix of reflexivities;
- $F = \begin{bmatrix} F_{1,1} & F_{1,2} & ... & F_{1,n} \\ F_{2,1} & F_{2,2} & ... & F_{2,n} \\ ... & ... & ... & ... \\ F_{n,1} & F_{n,2} & ... & F_{n,n} \end{bmatrix}$ - matrix of view factors;
- $G = \begin{bmatrix} G_1 \\ G_2 \\ ... \\ G_n \end{bmatrix}$ - vector of unknown total irradiances;

- $G^o = \begin{bmatrix} G_1^o \\ G_2^o \\ ... \\ G_n^o \end{bmatrix}$ - vector of direct solar irradiances.

The unknown total irradiances are then

$$G = [I - \rho F]^{-1} G^o$$

The radiative short wave (i.e. solar) heat flow rate on each surface is:

$$\Phi = SG$$

where: - $\Phi = \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ ... \\ \Phi_n \end{bmatrix}$ - vector of total heat flow rates due to solar radiation [W];

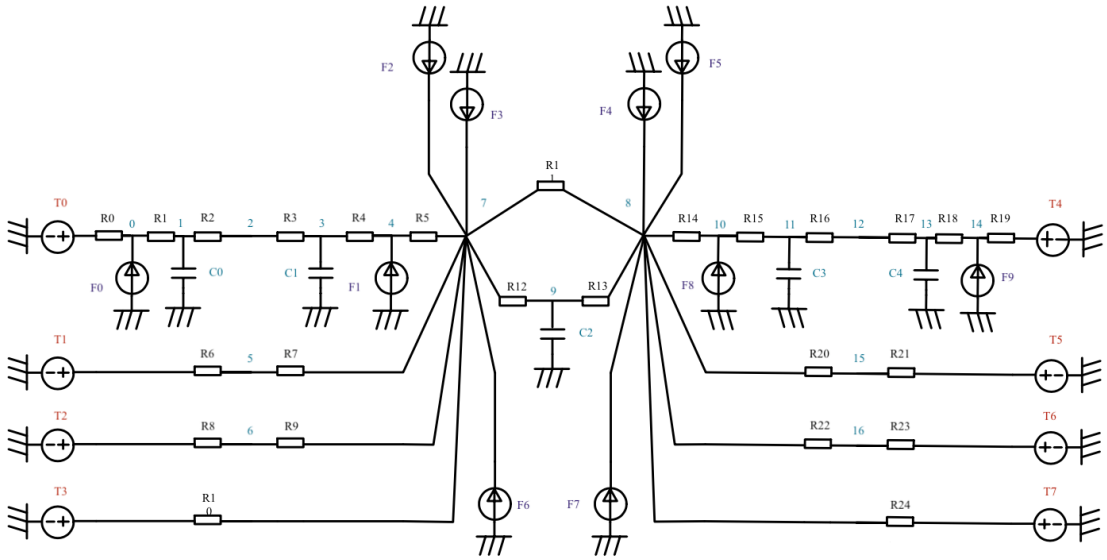
- $S = \begin{bmatrix} S_1 & 0 & ... & 0 \\ 0 & S_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & S_n \end{bmatrix}$ - matrix of surface areas of walls i [m²].

1.2 Importing Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1.3 Physical Analysis and Mathematical Modelling

We consider 2 rooms with 2 exterior doors and one interior door connecting two rooms, as well as 2 windows per each room, HVAC system and ventillation chambers.
Considering HVAC system controlled by a P - controller.



> Ther-

mal circuit.

[2]:
```
# P-controler gain
Kp = 1e4
```

[3]:
```
l_room = 3                # m length of the cubic room
Sg = l_room**2            # m² surface of the glass wall
Sc = Si = 5 * Sg     # m² surface of concrete & insulation of the 5 walls
```

[4]:
```
Va = l_room**3            # m³ volume of air
ACH = 1              # air changes per hour
Va_dot = ACH * Va / 3600     # m³/s air infiltration
```

Thermophysical propwerties of air is given as:

[5]:
```
air = {'Density': 1.2,                      # kg/m³
       'Specific heat': 1000}               # J/kg.K
```

Thermophysical properties and surface area of materials given below:

[6]:
```
wall = {'Conductivity': [1.4, 0.027, 1.4],  # W/m.K
        'Density': [2300, 55, 2500],        # kg/m³
        'Specific heat': [880, 1210, 750],  # J/kg.K
        'Width': [0.2, 0.08, 0.004],
        'Surface': [5 * l_room**2, 5 * l_room**2, l_room**2],  # m²
        'Slices': [4, 2, 1]}                # number of discretizations
wall = pd.DataFrame(wall, index=['Concrete', 'Insulation', 'Glass'])
```

Radiative Properties of the surface:

[7]:
```
_wLW = 0.9      # long wave wall emmisivity (concrete)
_wSW = 0.2      # absortivity white surface
_gLW = 0.9      # long wave glass emmisivity (glass pyrex)
_gSW = 0.83     # short wave glass transmitance (glass)
_gSW = 0.1      # short wave glass absortivity

 = 5.67e-8      # W/m².K Stefan-Bolzmann constant
```

The view factor between the wall and the window (glass):

[8]:
```
Fwg = 1 / 5      # view factor wall - glass
```

The mean temperature of the surfaces (used for the linearization of the radiative heat exchange) is:

[9]:
```
Tm = 20 + 273    # mean temp for radiative exchange

# convection coefficients, W/m² K
h = pd.DataFrame([{'in': 4., 'out': 10}])
```

1.3.1 Thermal Conductance

[10]:
```
G_cd = wall['Conductivity'] / wall['Width'] * wall['Surface']

# Convection
Gw = h * wall['Surface'][0]     # wall
Gg = h * wall['Surface'][2]     # glass

# Long-wave radiation exchnage
GLW1 = _wLW / (1 - _wLW) * wall['Surface']['Insulation'] * 4 *  * Tm**3
GLW2 = Fwg * wall['Surface']['Insulation'] * 4 *  * Tm**3
GLW3 = _gLW / (1 - _gLW) * wall['Surface']['Glass'] * 4 *  * Tm**3
# long-wave exg. wall-glass
GLW = 1 / (1 / GLW1 + 1 / GLW2 + 1 / GLW3)

# ventilation & advection
Gv = Va_dot * air['Density'] * air['Specific heat']

# glass: convection outdoor & conduction
Ggs = float(1 / (1 / Gg['out'] + 1 / (2 * G_cd['Glass'])))
```

1.3.2 Thermal Capacities

[11]:
```
C = wall['Density'] * wall['Specific heat'] * wall['Surface'] * wall['Width']
C['Air'] = air['Density'] * air['Specific heat'] * Va
```

1.3.3 Incidence Matrix

[12]:
```
A = np.zeros([25, 17])
A[0, 0] = 1
A[1, 0], A[1, 1] = -1, 1
A[2, 1], A[2, 2] = -1, 1
A[3, 2], A[3, 3] = -1, 1
A[4, 3], A[4, 4] = -1, 1
A[5, 4], A[5, 7] = -1, 1
A[6, 5] = 1
A[7, 5], A[7, 7] = -1, 1
A[8, 6] = 1
A[9, 6], A[9, 7] = -1, 1
A[10, 7] = 1
A[11, 7], A[11, 8] = -1, 1
A[12, 7], A[12, 9] = -1, 1
A[13, 8], A[13, 9] = 1, -1
A[14, 8], A[14, 10] = -1, 1
A[15, 10], A[15, 11] = -1, 1
A[16, 11], A[16, 12] = -1, 1
A[17, 12], A[17, 13] = -1, 1
```

```
A[18, 13], A[18, 14] = -1, 1
A[19, 14] = -1
A[20, 8], A[20, 15] = -1, 1
A[21, 15] = -1
A[22, 8], A[22, 16] = -1, 1
A[23, 16] = -1
A[24, 8] = -1

np.set_printoptions(suppress=False)
print(f'A = \n{A}')
```

```
A =
[[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [-1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. -1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. -1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. -1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -1.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. -1.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. -1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. -1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. -1.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1. -1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

[13]:
```
G = np.diag([Gw.iloc[0]['out'], G_cd['Concrete'], G_cd['Concrete'],␣
↪G_cd['Insulation'], G_cd['Insulation'],
             Gg.iloc[0]['in'], Gw.iloc[0]['out'], G_cd['Glass'], Gw.
↪iloc[0]['out'], Ggs,
              Gv, Kp, G_cd['Concrete'], G_cd['Concrete'], Gg.iloc[0]['in'],
               G_cd['Insulation'], G_cd['Insulation'], G_cd['Concrete'],␣
↪G_cd['Concrete'],
              Gw.iloc[0]['out'], G_cd['Glass'], Gw.iloc[0]['out'], Ggs, Gw.
↪iloc[0]['out'], Gv
```

```
            ])

np.set_printoptions(precision=3, threshold=16, suppress=True)
with np.printoptions(threshold=np.inf):
    print(f'G: \n{G}')
```

```
G:
[[ 450.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.  315.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.    0.  315.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.    0.    0.   15.187   0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.    0.    0.    0.   15.187   0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.    0.    0.    0.    0.   36.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.    0.    0.    0.    0.    0.  450.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.    0.    0.    0.    0.    0.    0.
  3150.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.    0.    0.    0.    0.    0.    0.
     0.  450.    0.    0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.    0.    0.    0.    0.    0.    0.
     0.    0.   88.732   0.    0.    0.    0.
     0.    0.    0.    0.    0.    0.    0.
     0.    0.    0.    0.   ]
 [   0.    0.    0.    0.    0.    0.    0.
```

```
      0.        0.        0.        9.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.    10000.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.      315.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.      315.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
     36.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.       15.187     0.        0.        0.        0.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.       15.187     0.        0.        0.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.      315.        0.        0.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.      315.        0.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.      450.        0.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.     3150.
      0.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
      0.        0.        0.        0.        0.        0.        0.
    450.        0.        0.        0.   ]
[     0.        0.        0.        0.        0.        0.        0.
```

```
       0.        0.        0.        0.        0.        0.        0.
       0.        0.        0.        0.        0.        0.        0.
       0.       88.732     0.        0.    ]
 [     0.        0.        0.        0.        0.        0.        0.
       0.        0.        0.        0.        0.        0.        0.
       0.        0.        0.        0.        0.        0.        0.
       0.        0.      450.        0.    ]
 [     0.        0.        0.        0.        0.        0.        0.
       0.        0.        0.        0.        0.        0.        0.
       0.        0.        0.        0.        0.        0.        0.
       0.        0.        0.        9.    ]]
```

1.3.4 Conductance Matrix

G = np.diag([Gw.iloc[0]['out'], Gw.iloc[0]['out'], Gw.iloc[0]['out'], Gw.iloc[0]['out'], Gw.iloc[0]['out'], Gw.iloc[0]['out'], Gw.iloc[0]['out'], 2 * G_cd['Concrete'], 2 * G_cd['Concrete'], 2 * G_cd['Concrete'], 2 * G_cd['Concrete'], 2 * G_cd['Concrete'], 2 * G_cd['Concrete'], 2 * G_cd['Concrete'], 2 * G_cd['Concrete'], 2 * G_cd['Insulation'], 2 * G_cd['Insulation'], 2 * G_cd['Insulation'], GLW, Gw.iloc[0]['in'], Gg.iloc[0]['in'], Ggs, 2 * G_cd['Glass'], Gv, Kp]) # 2*G means 2 sides of the surface

np.set_printoptions(precision=3, threshold=16, suppress=True) with np.printoptions(threshold=np.inf): print(f'G: G')

1.3.5 Capacity Matrix

[14]:
```
C = np.diag([0, C['Concrete'], 0, C['Concrete'], 0, 0, 0, 0,
             0, C['Concrete'], 0, C['Concrete'], 0, C['Concrete'], 0, 0, 0])
with np.printoptions(threshold=np.inf):
    print(C)
```

```
[[       0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.]
 [       0. 18216000.        0.        0.        0.        0.        0.
         0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.]
 [       0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.]
 [       0.        0.        0. 18216000.        0.        0.        0.
         0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.]
 [       0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.]
 [       0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.        0.        0.        0.        0.
         0.        0.        0.]
```

```
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0. 18216000.        0.        0.        0.        0.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0. 18216000.        0.        0.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0.        0.        0. 18216000.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.]
[       0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.        0.        0.        0.        0.
        0.        0.        0.]]
```

1.3.6 Input Vector

If the flow in a source is from low potential to high potential (i.e. from - to +), the source is positive . Let's give some arbitrary non-zero values to the temperature sources:

[15]:
```
b = np.zeros(25)
b[[0, 6, 8, 18, 19, 20, 21, 22]] = 10 + np.array([0, 60, 80, 180, 190, 200, 210,␣
↪220])
```

[16]:
```
f = np.zeros(17) # 17 Nodes
f[[0, 4, 7, 8, 10, 14]] = 1000 + np.array([0, 4000, 7000, 8000, 10000, 14000])
```

[17]:
```
y = np.ones(17)
```

[18]:
```
u = np.hstack([b[np.nonzero(b)], f[np.nonzero(f)]])
u
```

[18]:
```
array([   10.,    70.,    90.,   190.,   200.,   210.,   220.,   230.,
        1000.,  5000.,  8000.,  9000., 11000., 15000.])
```

1.4 State-Space Representation

– Explanation –

[20]:
```
[As, Bs, Cs, Ds] = tc2ss(A, G, b, C, f, y)
```

Let's compare the steady state results obtained with the differential-algebraic equations of the thermal circuit:

$$y_{tc} = (A^T G A)^{-1}(A^T G b + f)$$

and the state-space:

$$y_{ss} = (-C_s A_s^{-1} B_s + D_s)u$$

[21]:
```
yss = (-Cs @ np.linalg.inv(As) @ Bs + Ds) @ u
ytc = np.linalg.inv(A.T @ G @ A) @ (A.T @ G @ b + f)

print(np.array_str(yss, precision=3, suppress_small=True))
print(np.array_str(ytc, precision=3, suppress_small=True))
print(f'Max error in steady-state between thermal circuit and state-space:\
 {max(abs(yss - ytc)):.2e}')
```

```
[  12.271   12.339   12.408 ... -159.674   39.491   15.904]
[  12.271   12.339   12.408 ... -159.674   39.491   15.904]
Max error in steady-state between thermal circuit and state-space: 3.69e-13
```

1.5 Dynamic Simulation

[22]:
```
b = np.zeros(25)
f = np.zeros(17)

f[[0, 4, 5, 7, 8, 12]] = 1
b[[0, 6, 7, 16, 17, 18]] = 1
```

Let's consider the output of the circuit temperature to be the indoor(Room 2) temperature(node 7)

[23]:
```
y = np.zeros(17)
y[[5]] = 1
```

The state-space representation is obtained from the differential-algebraic equations of the thermal circuit:

[24]:
```
[As, Bs, Cs, Ds] = tc2ss(A, G, b, C, f, y)
```

1.5.1 Time Step

The maximum time step for numerical stability of Euler explicit itegration in time is given by the minimum eigenvalue λ of the state matrix A_s:

$$\Delta t \leq \min(-2/\lambda_i) = \min T_i/2$$

where T_i is the time constants, $T_i = - 1_{\overline{\lambda_i}}$

[25]:
```
# Assume:
dtmax = min(-2. / np.linalg.eig(As)[0])
print(f'Maximum time step: {dtmax:.2f} s')
```

```
Maximum time step: 95166.46 s
```

[26]:
```
dt = 5000
```

1.5.2 Step response

Let's obtain the dynamic response of the system to a step input. For the duration of simulation:

[27]:
```
duration = 3600 * 24 * 4        # [s]
n = int(np.floor(duration / dt))
t = np.arange(0, n * dt, dt)    # time
```

[28]:
```
n
```

[28]:
```
69
```

For the input vector **u**, the outdoor temperatures will be `To = 1`, the indoor set-point temperature will `Tsp =0`, and the heat flow sources will be zero.

[29]:
```
# Vectors of state and input (in time)
n_tC = As.shape[0]              # no of state variables (temps with capacity)
# u = [To To To Tsp Phio Phii Qaux Phia]
u = np.zeros([12, n])
u[0:8, :] = np.ones([8, n])
```

[30]:
```
temp_exp = np.zeros([n_tC, t.shape[0]])
temp_imp = np.zeros([n_tC, t.shape[0]])
```

By integrating the state-space model

$$\begin{cases} \dot{\theta} = A_s\theta + B_s u \\ y = C_s\theta + D_s u \end{cases}$$

by using Euler forward (or explicit)

$$\theta_{k+1} = (I + \Delta t A)\theta_k + \Delta t B u_k$$

and Euler backward (or implicit) integration

$$\theta_{k+1} = (I - \Delta t A)^{-1}(\theta_k + \Delta t B u_k)$$

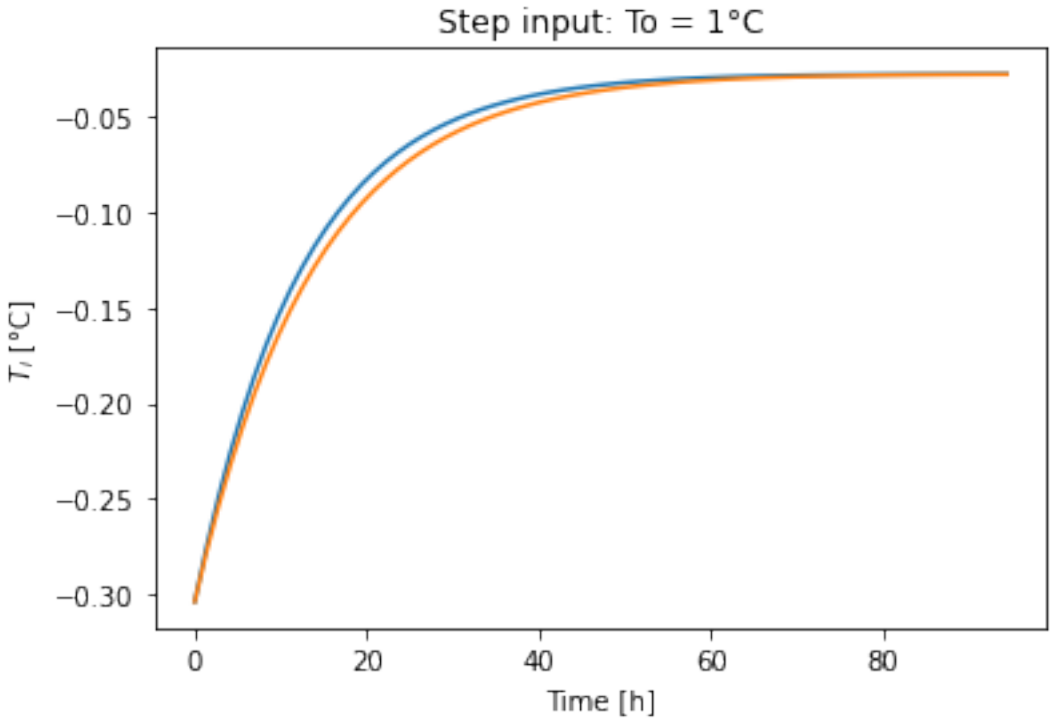
[31]:
```
I = np.eye(n_tC)
for k in range(n - 1):
    temp_exp[:, k + 1] = (I + dt * As) @ temp_exp[:, k] + dt * Bs @ u[:, k]
    temp_imp[:, k + 1] = np.linalg.inv(I - dt * As) @ (temp_imp[:, k] + dt * Bs␣
 ↪@ u[:, k])
```

[32]:
```
u.shape
```

[32]:
```
(12, 69)
```

[33]:
```
y_exp = Cs @ temp_exp + Ds @  u
y_imp = Cs @ temp_imp + Ds @  u

fig, ax = plt.subplots()
ax.plot(t / 3600, y_exp.T, t / 3600, y_imp.T)
ax.set(xlabel='Time [h]',
       ylabel='$T_i$ [°C]',
       title='Step input: To = 1°C')
plt.show()
```



[34]:
```
b = np.zeros(25)
b[[0, 6, 7, 16, 17, 18]] = 1
f = np.zeros(17)

ytc = np.linalg.inv(A.T @ G @ A) @ (A.T @ G @ b + f)
print('Steady-state indoor temperature obtained with:')
print(f'- DAE model: {ytc[6]:.4f} °C')
print(f'- response to step input:{float(y_exp[:, -2]):.4f} °C')
```

```
Steady-state indoor temperature obtained with:
- DAE model: 0.1355 °C
- response to step input:-0.0276 °C
```

1.5.3 Simulation with weather data

Read the hourly weather data file from start date to end date. The data are for a year (the choice of 2000 for the year is arbitrary).

[35]:
```
filename = 'FRA_Lyon.074810_IWEC.epw'
start_date = '2000-01-03 12:00:00'
end_date = '2000-02-05 18:00:00'

# Read weather data from Energyplus .epw file
[data, meta] = read_epw(filename, coerce_year=None)
weather = data[["temp_air", "dir_n_rad", "dif_h_rad"]]
del data
weather.index = weather.index.map(lambda t: t.replace(year=2000))
weather = weather[(weather.index >= start_date) & (
    weather.index < end_date)]
```

From weather data, calculate the hourly solar radiation on a tilted surface, `t1` [W]:

[36]:
```
surface_orientation = {'slope': 90,
                       'azimuth': 0,
                       'latitude': 45}
albedo = 0.2
rad_surf1 = sol_rad_tilt_surf(weather, surface_orientation, albedo)
rad_surf1['t1'] = rad_surf1.sum(axis=1)
```

The weather data is at the time-step of 1h. It needs to be at time step of `dt`.

[37]:
```
data = pd.concat([weather['temp_air'], rad_surf1['t1']], axis=1)
print(data)
data1 = data.resample(str(dt) + 'S').pad()
data1=data1[1:]
data1=data1.append(data.iloc[-1,:])
print(data1)
data = data.rename(columns={'temp_air': 'To'})
```

```
                           temp_air         t1
2000-01-03 12:00:00+01:00       6.8   98.949619
2000-01-03 13:00:00+01:00       8.0  171.364399
2000-01-03 14:00:00+01:00      10.0  164.029026
2000-01-03 15:00:00+01:00       9.0   97.877824
2000-01-03 16:00:00+01:00       8.0   13.300000
...                             ...         ...
2000-02-05 13:00:00+01:00      11.0  671.116452
2000-02-05 14:00:00+01:00      12.0  485.272692
2000-02-05 15:00:00+01:00      12.0  293.359833
2000-02-05 16:00:00+01:00      11.0  106.332549
2000-02-05 17:00:00+01:00      10.0    8.400000

[798 rows x 2 columns]
                           temp_air         t1
2000-01-03 12:30:00+01:00       6.8   98.949619
2000-01-03 13:53:20+01:00       8.0  171.364399
2000-01-03 15:16:40+01:00       9.0   97.877824
2000-01-03 16:40:00+01:00       8.0   13.300000
2000-01-03 18:03:20+01:00       4.9    0.000000
...                             ...         ...
2000-02-05 12:10:00+01:00      11.0  709.796200
2000-02-05 13:33:20+01:00      11.0  671.116452
2000-02-05 14:56:40+01:00      12.0  485.272692
2000-02-05 16:20:00+01:00      11.0  106.332549
2000-02-05 17:00:00+01:00      10.0    8.400000

[575 rows x 2 columns]
```

Let's consider the indoor temperature setpoint `Ti` and auxiliary heat flow `Qa` constant for the whole simulation period.

[38]:
```
data['Ti'] = 20 * np.ones(data.shape[0])
data['Qa'] = 0 * np.ones(data.shape[0])
```

The simulation time vector, at time step `dt`is:

[39]:
```
t = dt * np.arange(data.shape[0])
```

The input vector is formed by the temperature sources $\left[T_o, T_o, T_o, T_{i,sp}\right]$ and the heat flow sources $\left[\Phi_o, \Phi_i, \dot{Q}_a, \Phi_a\right]$

[40]:
```
u = pd.concat([data['To'], data['To'], data['To'], data['To'], data['To'],␣
 ↪data['To'], data['To'],
               data['Ti'],
               _wSW * wall['Surface']['Concrete'] * data['t1'],
               _gSW * _wSW * wall['Surface']['Glass'] * data['t1'],
               data['Qa'],
               _gSW * wall['Surface']['Glass'] * data['t1']], axis=1)
```

[41]:
```
print(u)
```

```
                            To    To    To    To    To    To    To    Ti  \
2000-01-03 12:00:00+01:00   6.8   6.8   6.8   6.8   6.8   6.8   6.8  20.0
2000-01-03 13:00:00+01:00   8.0   8.0   8.0   8.0   8.0   8.0   8.0  20.0
2000-01-03 14:00:00+01:00  10.0  10.0  10.0  10.0  10.0  10.0  10.0  20.0
2000-01-03 15:00:00+01:00   9.0   9.0   9.0   9.0   9.0   9.0   9.0  20.0
2000-01-03 16:00:00+01:00   8.0   8.0   8.0   8.0   8.0   8.0   8.0  20.0
...                         ...   ...   ...   ...   ...   ...   ...   ...
2000-02-05 13:00:00+01:00  11.0  11.0  11.0  11.0  11.0  11.0  11.0  20.0
2000-02-05 14:00:00+01:00  12.0  12.0  12.0  12.0  12.0  12.0  12.0  20.0
2000-02-05 15:00:00+01:00  12.0  12.0  12.0  12.0  12.0  12.0  12.0  20.0
2000-02-05 16:00:00+01:00  11.0  11.0  11.0  11.0  11.0  11.0  11.0  20.0
2000-02-05 17:00:00+01:00  10.0  10.0  10.0  10.0  10.0  10.0  10.0  20.0

                                  t1          t1   Qa         t1
2000-01-03 12:00:00+01:00   890.546572   147.830731  0.0   89.054657
2000-01-03 13:00:00+01:00  1542.279592   256.018412  0.0  154.227959
2000-01-03 14:00:00+01:00  1476.261234   245.059365  0.0  147.626123
2000-01-03 15:00:00+01:00   880.900415   146.229469  0.0   88.090041
2000-01-03 16:00:00+01:00   119.700000    19.870200  0.0   11.970000
...                                ...          ...  ...         ...
2000-02-05 13:00:00+01:00  6040.048072  1002.647980  0.0  604.004807
2000-02-05 14:00:00+01:00  4367.454230   724.997402  0.0  436.745423
2000-02-05 15:00:00+01:00  2640.238495   438.279590  0.0  264.023849
2000-02-05 16:00:00+01:00   956.992943   158.860829  0.0   95.699294
2000-02-05 17:00:00+01:00    75.600000    12.549600  0.0    7.560000

[798 rows x 12 columns]
```

The initial value of the state-vector can be different of zero:

[42]:
```
temp_exp = 20 * np.ones([As.shape[0], u.shape[0]])
```

[43]:
```
temp_exp.shape
```

[43]:
```
(5, 798)
```

Explicit Euler integration in time

$$\theta_{k+1} = (I + \Delta t A)\theta_k + \Delta t B u_k$$

[44]:
```
for k in range(u.shape[0] - 1):
    temp_exp[:, k + 1] = (I + dt * As) @ temp_exp[:, k]\
     + dt * Bs @ u.iloc[k, :]
```

yields the time variation of state variable θ, from which we obtain the variation of the output (i.e. indoor temperature):

$$y = C_s\theta + D_s u$$

and the variation of the heat flow of the HVAC system:

$$q_{HVAC} = K_p(T_{i,sp} - y)$$

where K_p is the gain of the P-controller and $T_{i,sp}$ is the HVAC-setpoint for the indoor temperature.

[45]:
```
y_exp = Cs @ temp_exp + Ds @ u.to_numpy().T
q_HVAC = Kp * (data['Ti'] - y_exp[0, :])

fig, axs = plt.subplots(2, 1)
# plot indoor and outdoor temperature
axs[0].plot(t / 3600, y_exp[0, :], label='$T_{indoor}$')
axs[0].plot(t / 3600, data['To'], label='$T_{outdoor}$')
axs[0].set(xlabel='Time [h]',
           ylabel='Temperatures [°C]',
           title='Simulation for weather')
axs[0].legend(loc='upper right')

# plot total solar radiation and HVAC heat flow
axs[1].plot(t / 3600,  q_HVAC, label='$q_{HVAC}$')
axs[1].plot(t / 3600, data['t1'], label='$_{total}$')
axs[1].set(xlabel='Time [h]',
           ylabel='Heat flows [W]')
axs[1].legend(loc='upper right')

fig.tight_layout()
```