Model conversion (linear diff equation to matrices) for PySIP

In [1]:
```
from sympy import Matrix, symbols, linear_eq_to_matrix, Eq, diff, Derivative
from sympy.core.sympify import sympify
```

In [2]:
```
import numpy as np
import itertools
```

Beware of special variables, greek letters... that need to be expressed as symbols

In [3]:
```
Ci, xi = symbols('Ci x_i')
locals={'Ci' : Ci, 'xi' : xi}
```

Model TwTi_RoRiRcAwAi

In [4]:
```
states = [
        ('TEMPERATURE', 'xw', 'wall temperature'),
        ('TEMPERATURE', 'xi', 'indoor temperature'),
    ]

params = [
        ('THERMAL_RESISTANCE', 'Ro', 'between the outdoor and the wall node'),
        ('THERMAL_RESISTANCE', 'Ri', 'between the wall node and the indoor'),
        ('THERMAL_RESISTANCE', 'Rc', 'between the indoor and the sensor'),
        ('THERMAL_CAPACITY', 'Cw', 'Wall'),
        ('THERMAL_CAPACITY', 'Ci', 'indoor'),
        ('SOLAR_APERTURE', 'Aw', 'of the wall (m2)'),
        ('SOLAR_APERTURE', 'Ai', 'of the windows (m2)'),
        ('STATE_DEVIATION', 'sigw_w', ''),
        ('STATE_DEVIATION', 'sigw_i', ''),
        ('MEASURE_DEVIATION', 'sigv', ''),
        ('INITIAL_MEAN', 'x0_w', ''),
        ('INITIAL_MEAN', 'x0_i', ''),
        ('INITIAL_DEVIATION', 'sigx0_w', ''),
        ('INITIAL_DEVIATION', 'sigx0_i', ''),
    ]

inputs = [
        ('TEMPERATURE', 'To', 'outdoor air temperature'),
        ('POWER', 'Qgh', 'global horizontal solar radiation'),
        ('POWER', 'Qh', 'HVAC system heat'),
    ]

outputs = [('TEMPERATURE', 'xi', 'indoor air temperature')]

#equation CTSM-R like
model_state_eqs = ['dxw ~ ((xi-xw)/(Ri*Cw) + (To-xw)/(Ro*Cw) + Aw*Qgh/Cw)*dt + sigw_w*dw_w',
                   'dxi ~ ((xw-xi)/(Ri*Ci) + (Qh + Ai*Qgh)/Ci)*dt + sigw_i*dw_i']
model_obs_eqs = ['Ti ~ xi']
```

Magic function

In [6]:
```
def state_space_matrices_from_equations(model_state_eqs,model_obs_eqs, inputs, outputs):
    eqs = [sympify(f"Eq({s.split('~')[1].split('*dt')[0]},0)",locals=locals)  for s in model_state_eqs]
    states_symbs = [sympify(f"{s.split('~')[0].split('/')[0].split('d')[-1]}",locals=locals)  for s in model_state_eqs]
    inputs_symbs = symbols([i[1] for i in inputs])
    obs = [sympify(f"Eq({s.split('~')[1]},0)",locals=locals)  for s in model_obs_eqs]
    A,b = linear_eq_to_matrix(eqs, states_symbs)
    B,b = linear_eq_to_matrix(eqs, inputs_symbs)
    C,b = linear_eq_to_matrix(obs, states_symbs)
    D,b = linear_eq_to_matrix(obs, inputs_symbs)

    dA, dB = {},{}

    for p in params:
        par = Matrix([symbols(p[1])])
        dA_rows, dB_rows = [],[]
        for r in range(A.shape[0]):
            _list_A = [i[0] for i in A.row(r).jacobian(par).tolist()]
            _list_B = [i[0] for i in B.row(r).jacobian(par).tolist()]
            dA_rows.append(_list_A)
            dB_rows.append(_list_B)
        dA[p[1]] = Matrix(dA_rows)
        dB[p[1]] = Matrix(dB_rows)


    return A,B,C,D, dA, dB
```

Matrices evaluation (with jacobians)

In [7]:
```
A, B, C, D, dA, dB = state_space_matrices_from_equations(model_state_eqs,model_obs_eqs, inputs, outputs)
```

In [8]:
```
A
```

Out[8]:
$$\begin{bmatrix} -\frac{1}{CwRo} - \frac{1}{CwRi} & \frac{1}{CwRi} \\ \frac{1}{CiRi} & -\frac{1}{CiRi} \end{bmatrix}$$

In [9]:
```
B
```

Out[9]:
$$\begin{bmatrix} \frac{1}{CwRo} & \frac{Aw}{Cw} & 0 \\ 0 & \frac{Ai}{Ci} & \frac{1}{Ci} \end{bmatrix}$$

In [10]:
```
C
```

Out[10]:
$$\begin{bmatrix} 0 & 1 \end{bmatrix}$$

In [11]:
```
D
```

Out[11]:
$$\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

In [12]:
```
dA['Ro']
```

Out[12]:
$$\begin{bmatrix} \frac{1}{CwRo^2} & 0 \\ 0 & 0 \end{bmatrix}$$

In [13]:
```
dA['Ri']
```

Out[13]:
$$\begin{bmatrix} \frac{1}{CwRi^2} & -\frac{1}{CwRi^2} \\ -\frac{1}{CiRi^2} & \frac{1}{CiRi^2} \end{bmatrix}$$

In [14]:
```
dA['Cw']
```

Out[14]:
$$\begin{bmatrix} \frac{1}{Cw^2Ro} + \frac{1}{Cw^2Ri} & -\frac{1}{Cw^2Ri} \\ 0 & 0 \end{bmatrix}$$

In [15]:
```
dA['Ci']
```

Out[15]:
$$\begin{bmatrix} 0 & 0 \\ -\frac{1}{Ci^2Ri} & \frac{1}{Ci^2Ri} \end{bmatrix}$$

In [16]:
```
dB['Ro']
```

Out[16]:
$$\begin{bmatrix} -\frac{1}{CwRo^2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In [17]:
```
dB['Cw']
```

Out[17]:
$$\begin{bmatrix} -\frac{1}{Cw^2Ro} & -\frac{Aw}{Cw^2} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In [18]:
```
dB['Ci']
```

Out[18]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -\frac{Ai}{Ci^2} & -\frac{1}{Ci^2} \end{bmatrix}$$

In [19]:
```
dB['Aw']
```

Out[19]:
$$\begin{bmatrix} 0 & \frac{1}{Cw} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In [20]:
```
dB['Ai']
```

Out[20]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{Ci} & 0 \end{bmatrix}$$

Numerical evaluation of matrices

In [21]:
```
Ro,Ri,Rc,Cw,Ci,Aw,Ai,sigw_w,sigw_i,sigv,x0_w,x0_i,sigx0_w,sigx0_i = range(1,15)
theta = [Ro,Ri,Rc,Cw,Ci,Aw,Ai,sigw_w,sigw_i,sigv,x0_w,x0_i,sigx0_w,sigx0_i]
```

In [22]:
```
theta
```

Out[22]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

In [23]:
```
_dict = {symb : p for symb, p in zip(symbols([i[1] for i in params]), theta)}
```

In [24]:
```
_dict
```

Out[24]:
```
{Ro: 1,
 Ri: 2,
 Rc: 3,
 Cw: 4,
 Ci: 5,
 Aw: 6,
 Ai: 7,
 sigw_w: 8,
 sigw_i: 9,
 sigv: 10,
 x0_w: 11,
 x0_i: 12,
 sigx0_w: 13,
 sigx0_i: 14}
```

In [25]:
```
A.evalf(subs=_dict).tolist()
```

Out[25]:
```
[[-0.375000000000000, 0.125000000000000],
 [0.100000000000000, -0.100000000000000]]
```

In [26]:
```
B.evalf(subs=_dict).tolist()
```

Out[26]:
```
[[0.250000000000000, 1.50000000000000, 0],
 [0, 1.40000000000000, 0.200000000000000]]
```

In [27]:
```
dA['Ri'].evalf(subs=_dict).tolist()
```

Out[27]:
```
[[0.0625000000000000, -0.0625000000000000],
 [-0.0500000000000000, 0.0500000000000000]]
```