Elastic properties :
$Y_A = 67\,GPa$
$Y_M = 26\,GPa$

Transformation Temperature
$M_f = 9°C$
$M_S = 18°C$
$A_S = 35°C$
$A_f = 49°C$

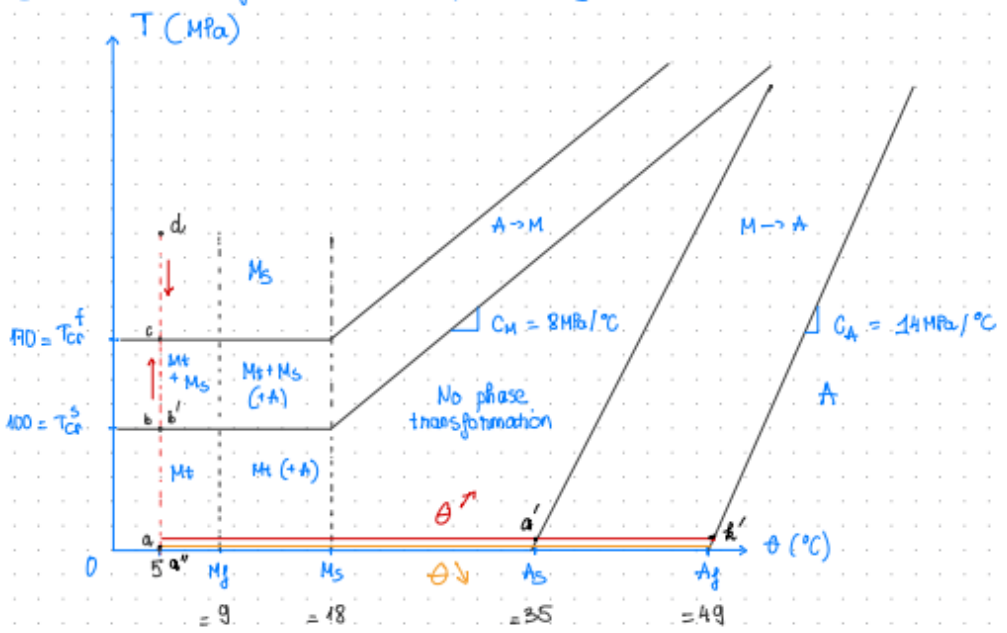
Transformation Constraints
$C_M = 8\,MPa/°C$
$C_A = 14\,MPa/°C$
$T_{cr}^s = 100\,MPa$
$T_{cr}^f = 170\,MPa$

Max Recovery Strain
$\xi_L = 0.07$

Material Heat
$\theta = 5°C$

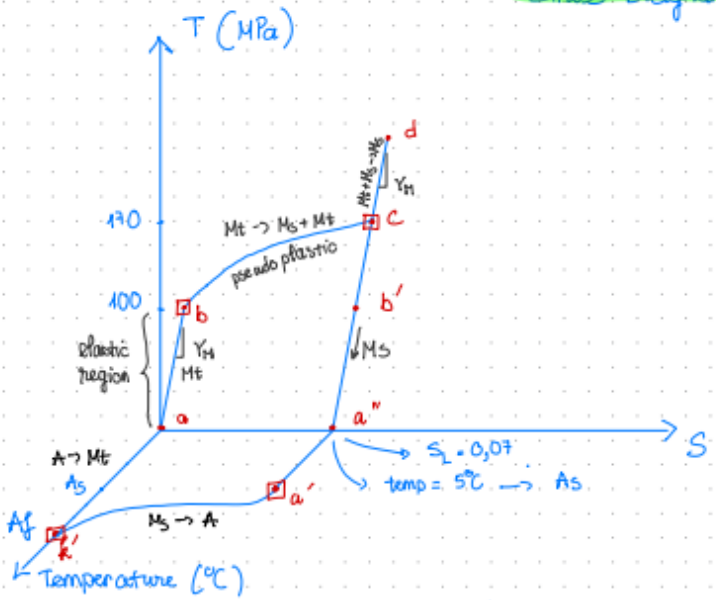
(1) What is the SMAs behavior at the given?
$\theta = 5°C < M_f = 9°C \Rightarrow$ The SMA behavior can be shape memory effect (SME)

(2) Draw stress cycle on phase diagram ? (Phase Diagram)



(3) Sketch graphically T-S curve with different specific regions ?
→ Determined transitional points : b, c

Stress Diagram



(4) Evaluate stress at the transitional points in the T-S curve ?

- $T_b = T_{cr}^s = 100\,MPa$
- $T_c = T_{cr}^f = 170\,MPa$
- $T_{a'} = 0$
- $T_{k'} = 0$

} Stress free condition ⇒ Reverse martensitic transformation.
→ $\xi_0 = 1 \Rightarrow \xi_0 = \frac{1}{2}\left(1 + \cos\left[a_M \cdot (\theta - A_S)\right]\right)$

Code File:

```
import math
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


class Shape_Memory_Effect:
    def __init__(self, YA, YM, Ms, Mf, As, Af, CM, CA, TCRS, TCRF, SL, temp, shiS0, shiT0):
        self.YA = YA
        self.YM = YM
        self.Ms = Ms
        self.Mf = Mf
        self.As = As
        self.Af = Af
        self.CM = CM
        self.CA = CA
        self.TCRS = TCRS
        self.TCRF = TCRF
        self.SL = SL
        self.temp = temp
        self.shiS0 = shiS0
        self.shiT0 = shiT0
        self.shi_lst = []

    def stress_calculated(self):
        global Ta, Tb, Tc
        Ta = 0
        Tb = self.TCRS
        Tc = self.TCRF

    def predicted_through_stress(self):
        global Tb_predicted, Tc_predicted, Td_predicted, Taf_predicted, Tkf_predicted, Ta_predicted
        Tb_predicted = np.linspace(Ta, Tb, 20)
        Tc_predicted = np.linspace(Tb, Tc, 20)
        for i in Tc_predicted:
            shiS = ((1 - self.shiS0) / 2) * math.cos((math.pi / (self.TCRS - self.TCRF)) * (i - self.TCRF)) + ((1 + self.shiS0) / 2)
            self.shi_lst.append(shiS)
```

```
        self.shi_lst = np.array(self.shi_lst)
        Td_predicted = np.linspace(Tc,Ta,num=20, endpoint=True)
        Taf_predicted = np.linspace(0,0, num=20, endpoint=True)
        Tkf_predicted = np.linspace(0,0, num=20, endpoint=True)
        Ta_predicted = np.linspace(0,0, num=20, endpoint=True)

    def predicted_through_strain(self):
        global Sb_predicted, Sc_predicted, Sd_predicted
        Sb_predicted = Tb_predicted/self.YM
        Sc_predicted = (Tc_predicted/self.YM) + self.SL*self.shi_lst
        Sd_predicted = (Td_predicted/self.YM) + self.SL


    def predicted_through_temp(self):
        global temp_b_predicted, temp_c_predicted, temp_d_predicted, temp_a_f_predicted, temp_k_f_predicted,
temp_a_predicted
        temp_b_predicted = np.linspace(self.temp,self.temp,num=20, endpoint=True)
        temp_c_predicted = np.linspace(self.temp,self.temp, num=20, endpoint=True)
        temp_d_predicted = np.linspace(self.temp,self.temp, num=20, endpoint=True)
        temp_a_f_predicted = np.linspace(self.temp, self.As, num=20, endpoint=True)
        temp_k_f_predicted = np.linspace(self.As, self.Af, num=20, endpoint=True)
        temp_a_predicted = np.linspace(self.Af, self.temp, num=20, endpoint=True)

    def predicted_through_strain_by_temp(self):
        global strain_a_f_predicted, strain_k_f_predicted, strain_a_predicted
        # At a'
        strain_a_f_predicted = np.linspace(self.SL,self.SL, num=20, endpoint=True)
        # At k'
        shi_M_ls = []
        for i in temp_k_f_predicted:
            shi_M = 1/2*(1 + math.cos((math.pi/(self.Af-self.As))*(i-self.As)))
            shi_M_ls.append(shi_M)
        shi_M_ls = np.array(shi_M_ls)
        strain_k_f_predicted = self.SL*shi_M_ls
        # At a
        strain_a_predicted = np.linspace(0,0, num=20, endpoint=True)

    def plot_T_S_Temp_curve(self):
        strain_combined = np.concatenate((Sb_predicted, Sc_predicted, Sd_predicted, strain_a_f_predicted, strain_k_f_predicted,
strain_a_predicted))
```

```
        stress_combined = np.concatenate((Tb_predicted, Tc_predicted, Td_predicted, Taf_predicted, Tkf_predicted, Ta_predicted))

        temp_combined = np.concatenate((temp_b_predicted, temp_c_predicted, temp_d_predicted, temp_a_f_predicted, temp_k_f_predicted, temp_a_predicted))

        # Create a 3D plot

        fig = plt.figure(figsize=(10, 8))

        ax = fig.add_subplot(111, projection='3d')


        # Plot the 3D curve

        ax.plot(strain_combined, stress_combined, temp_combined, label='Stress-Strain-Temp Curve', color='blue')


        # Set axis labels

        ax.set_xlabel('Strain')

        ax.set_ylabel('Temperature (°C)')

        ax.set_zlabel('Stress (MPa)')

        ax.set_title('Stress-Strain-Temperature Relationship')


        # Add legend and grid

        ax.legend()

        ax.grid(True)


        # Show plot

        plt.show()
def main():
    Mf = 9  # °C

    Ms = 18  # °C

    As = 35  # °C

    Af = 49  # °C

    CM = 8  # MPa/°C (slope for martensite)

    CA = 14  # MPa/°C (slope for austenite)

    YM = 26000

    YA = 67000

    TCRS = 100  # MPa (start transformation stress)

    TCRF = 170  # MPa (finish transformation stress)

    shiT0 = 1

    shiS0 = 0

    SL = 0.07

    temp = 5

    proc = Shape_Memory_Effect(YA, YM, Ms, Mf, As, Af, CM, CA, TCRS, TCRF, SL, temp, shiS0, shiT0)

    proc.stress_calculated()
```

```
    proc.predicted_through_stress()
    proc.predicted_through_strain()
    proc.predicted_through_temp()
    proc.predicted_through_strain_by_temp()
    proc.plot_T_S_Temp_curve()
if __name__ == "__main__":
    main()
```

Result: