simulation_pièce_MCP+classique.py

```python
import numpy as np
import matplotlib.pyplot as plt


# %%
# ------------------------------
# Paramètres physiques et géométriques
# ------------------------------
L = 0.1        # épaisseur du mur (m)
N = 50         # nombre de points spatiaux
dx = L / (N - 1)
p = 1         #fraction de MCP dans un noeud

#MCP
rho = 800      # masse volumique (kg/m^3)
cp = 2000      # capacité thermique (J/(kg.K))
k = 0.5        # conductivité thermique (W/(m.K))

#Laine de verre
rho_v = 25     # masse volumique (kg/m^3)
cp_v = 1030    # capacité thermique (J/(kg.K))
k_v = 0.046      # conductivité thermique (W/(m.K))

# Paramètres spécifiques au PCM
L_latent = 15e4   # chaleur latente (J/kg)
T_m = 20.0          # température de fusion (°C)
delta = 5.0         # intervalle autour de T_m pour la transition (°C)

# %%
# ------------------------------
# Paramètres de simulation
# ------------------------------
dt = 0.5              # pas de temps
total_time = 86400*5  # temps total
steps = int(total_time / dt)

# Paramètres de la pièce
h_conv = 10         # coefficient convectif intérieur (W/(m^2.K))
A = 24.0             # surface d'échange (m^2)
C_room = 1e4        # capacité thermique de la pièce (J/K)

# Fonction de température extérieure (sinusoïdale sur 24 h)
def T_ext(t):
    # Moyenne 20 °C, amplitude 10 °C
    return 20 + 10 * np.sin(2 * np.pi * t / 86400)

# %%

# ------------------------------
# Fonctions pour la méthode enthalpique (PCM)
# ------------------------------
def compute_H_pcm(T):
    """
    Calcule l'enthalpie H pour un profil de température T, en tenant compte du
PCM.
    """
    H = np.zeros_like(T)
    mask_solid = T < (T_m - delta)
    mask_liquid = T > (T_m + delta)
    mask_mushy = ~(mask_solid | mask_liquid)

    H[mask_solid] = (rho * cp * T[mask_solid]) * p + (rho_v * cp_v *
T[mask_solid]) * (1-p)
    H[mask_liquid] = (rho * cp * T[mask_liquid] + rho * L_latent) * p + (rho_v *
cp_v * T[mask_liquid]) * (1-p)
```

```
063|     H[mask_mushy] = (rho * cp * T[mask_mushy] \
064|                     + rho * L_latent * ((T[mask_mushy] - (T_m - delta)) / (2 *
delta)) ) * p \
065|                     +(rho_v * cp_v * T[mask_mushy]) * (1-p)
066|     return H
067|
068| def T_from_H_pcm(H):
069|     """
070|     Récupère la température T à partir de l'enthalpie H pour le matériau PCM.
071|     """
072|     T = np.zeros_like(H)
073|     H_low = (rho * cp * (T_m - delta)) * p + (rho_v * cp_v * (T_m - delta)) *
(1-p)
074|     H_high = (rho * cp * (T_m + delta) + rho * L_latent) * p + (rho_v * cp_v
*(T_m + delta)) * (1-p)
075|
076|     mask_solid = H < H_low
077|     mask_liquid = H > H_high
078|     mask_mushy = ~(mask_solid | mask_liquid)
079|
080|     # Zone solide
081|     T[mask_solid] = H[mask_solid] / ( rho * cp * p + rho_v * cp_v * (1 - p) )
082|     # Zone liquide
083|     T[mask_liquid] = (H[mask_liquid] - rho * L_latent * p) / ( (rho * cp * p) +
rho_v * cp_v * (1 - p) )
084|     # Zone mushy (transition)
085|     T[mask_mushy] = (H[mask_mushy] + (rho * L_latent / (2 * delta)) * (T_m -
delta) * p) \
086|                     / ( (rho * cp + (rho * L_latent / (2 * delta))) *p + rho_v *
cp_v * (1 - p) )
087|     return T
088|
089| # %%
090| # -------------------------------
091| # Initialisation des profils muraux et de la pièce
092| # -------------------------------
093| T_init_wall = 20.0   # température initiale du mur (°C)
094|
095| # Pour la simulation PCM
096| T_pcm = np.ones(N) * T_init_wall
097| H_pcm = compute_H_pcm(T_pcm)
098|
099| # Pour l'isolation classique (aucun PCM, mise à jour directe de T)
100| T_classic = np.ones(N) * T_init_wall
101|
102| # Températures initiales de la pièce (pour chaque cas)
103| T_room_pcm = 22.0
104| T_room_classic = 22.0
105|
106| # Pour l'enregistrement des résultats
107| time_arr = []
108| T_room_pcm_arr = []
109| T_room_classic_arr = []
110| T_interior_pcm_arr = []     # température à la face intérieure du mur (x = L) -
PCM
111| T_interior_classic_arr = [] # même pour isolation classique
112| T_ext_arr = []
113| # %%
114|
115| # -------------------------------
116| # Boucle temporelle de simulation
117| # -------------------------------
118| for step in range(steps):
119|     t = step * dt
120|     # Condition extérieure imposée à x = 0
121|     T_ext_val = T_ext(t)
122|
```

```
    # ----- Mise à jour pour la simulation PCM -----
    # Condition limite extérieure
    T_pcm[0] = T_ext_val
    H_pcm[0] = compute_H_pcm(np.array([T_pcm[0]]))[0]

    # Condition convective intérieure (x = L)
    # Approximation par DF: T_pcm[-1] = ( T_pcm[-2] + (h_conv * dx /
k)*T_room_pcm ) / (1 + (h_conv * dx / k))
    T_pcm[-1] = (T_pcm[-2] + (h_conv * dx / (k * p + k_v * (1-p))) * T_room_pcm)
/ (1 + (h_conv * dx / (k * p + k_v * (1-p))))
    H_pcm[-1] = compute_H_pcm(np.array([T_pcm[-1]]))[0]

    # Récupérer T à partir de H pour mettre à jour la conduction
    T_temp_pcm = T_from_H_pcm(H_pcm)
    H_new_pcm = H_pcm.copy()
    # Mise à jour des noeuds internes (schéma explicite)
    for i in range(1, N - 1):
        d2Tdx2 = (T_temp_pcm[i+1] - 2 * T_temp_pcm[i] + T_temp_pcm[i-1]) / dx**2
        # dH/dt = k * d2T/dx^2
        H_new_pcm[i] = H_pcm[i] + dt * (k * p + k_v * (1 - p)) * d2Tdx2
    H_pcm = H_new_pcm.copy()
    T_pcm = T_from_H_pcm(H_pcm)

    # ----- Mise à jour pour la simulation en isolation classique -----
    T_classic[0] = T_ext_val  # condition extérieure
    T_classic[-1] = (T_classic[-2] + (h_conv * dx / k) * T_room_classic) / (1 +
(h_conv * dx / k))
    T_new_classic = T_classic.copy()
    for i in range(1, N - 1):
        d2Tdx2_classic = (T_classic[i+1] - 2 * T_classic[i] + T_classic[i-1]) /
dx**2
        # Equation classique: rho*cp * dT/dt = k * d2T/dx^2
        T_new_classic[i] = T_classic[i] + dt * (k / (rho * cp)) * d2Tdx2_classic
    T_classic = T_new_classic.copy()

    # ----- Mise à jour de la température de la pièce -----
    # C_room * dT_room/dt = h_conv * A * (T_interior_wall - T_room)
    T_room_pcm += dt * (h_conv * A / C_room) * (T_pcm[-1] - T_room_pcm)
    T_room_classic += dt * (h_conv * A / C_room) * (T_classic[-1] -
T_room_classic)

    # Enregistrement des résultats tous les 100 pas
    if step % 100 == 0:
        time_arr.append(t / 3600.0)  # temps en heures
        T_room_pcm_arr.append(T_room_pcm)
        T_room_classic_arr.append(T_room_classic)
        T_interior_pcm_arr.append(T_pcm[-1])
        T_interior_classic_arr.append(T_classic[-1])
        T_ext_arr.append(T_ext_val)
# %%

# -------------------------------
# Visualisation des résultats
# -------------------------------
plt.figure(figsize=(10, 6))
plt.plot(time_arr, T_room_pcm_arr, label='Température pièce (MCP)')
plt.plot(time_arr, T_room_classic_arr, label='Température pièce (Isolation
classique)')
plt.plot(time_arr, T_ext_arr,       label='Température extérieure',
linestyle='--')
plt.xlabel('Temps (heures)')
plt.ylabel('Température (°C)')
plt.title('Évolution de la température de la pièce sur 5 jours')
plt.legend()
plt.grid(True)
plt.show()

```

```
plt.figure(figsize=(10, 6))
plt.plot(time_arr, T_interior_pcm_arr,    label='Temp. intérieure du mur (MCP)')
plt.plot(time_arr, T_interior_classic_arr,label='Temp. intérieure du mur
(Isolation classique)')
plt.plot(time_arr, T_ext_arr,             label='Température extérieure',
linestyle='--')
plt.xlabel('Temps (heures)')
plt.ylabel('Température (°C)')
plt.title('Évolution de la température à la face intérieure du mur')
plt.legend()
plt.grid(True)
plt.show()

# %%
```