

Régulation thermique des serres : de la rétroaction PID à la commande prédictive

Youssef AOUAD

2025–2026

Plan de la présentation

- 1 Introduction et problématique
- 2 Modèle physique simplifié (3 EDO)
- 3 Le problème du mismatch et le modèle enrichi (5 EDO)
- 4 Implémentation numérique
- 5 Métriques de comparaison
- 6 Régulation PID
- 7 Pourquoi passer au MPC
- 8 Commande prédictive (MPC)
- 9 Détermination des paramètres optimaux du MPC
- 10 Comparaison MPC vs PID
- 11 Perspectives
- 12 Conclusion

Qu'est-ce qu'une serre ?

- Enceinte close à couverture transparente
- Capte le rayonnement solaire
- Limite les pertes thermiques



Pourquoi réguler une serre ?

- Maintenir une température optimale pour la culture
- Protéger les plantes des aléas climatiques
- **Enjeu énergétique** : le chauffage représente une part majeure des coûts



serre visitée en Octobre 2025

Ancrage au thème : cycles et boucles

Deux lectures du thème

- **Cycles** : la serre subit des perturbations climatiques cycliques
- **Boucles** : sa régulation repose sur des boucles de rétroaction

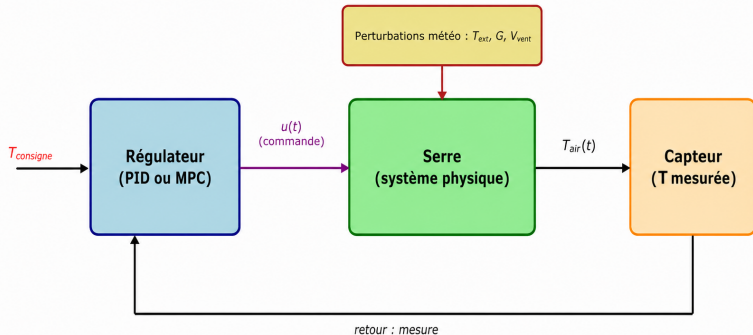
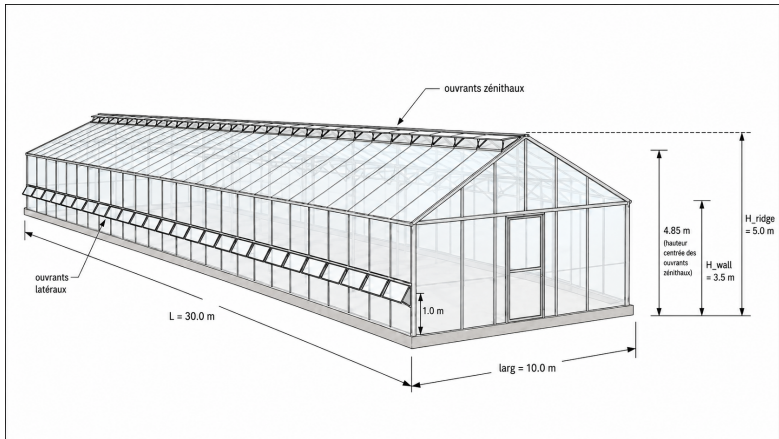


Schéma-bloc de la régulation de la serre

Dans une serre, dans quelle mesure une commande prédictive améliore-t-elle le respect des consignes thermiques et l'efficacité énergétique par rapport à une régulation PID ?

Dimensionnement de la serre



Serre de type à *pignon*, deux types d'ouvants (latéraux + zénithaux), couverture en polycarbonate alvéolaire 8 mm.

Grandeur	Symbole	Valeur
Longueur, largeur	L, l	30, 10 m
Hauteur gouttière, faîtage	H_w, H_r	3.5, 5.0 m (toit à pignon)
Surface au sol	A_{sol}	300 m ²
Volume	V	1275 m ³
Surface enveloppe	A_{env}	≈ 608 m ²
Capacité thermique air	C_{air}	1.54 MJ/K
Capacité thermique sol	C_{sol}	115 MJ/K
Capacité thermique vitrage	C_{vit}	7.0 MJ/K
Convection int / ext	$h_{\text{int}}, h_{\text{ext}}$	3, 20 W/m ² /K
Transmissivité vitrage	τ	0.7 (polycarbonate 8 mm)

Grandeur	Symbole	Valeur
Échange convectif sol-air	$h_{\text{sol-air}}$	5 W/m ² /K
Conductivité thermique du sol	λ_{sol}	1.0 W/m/K
Épaisseur de sol simplifiée	e_{sol}	0.30 m
Température du sol profond	T_{deep}	15°C
Constante de Stefan-Boltzmann	σ	5.67×10^{-8} W/m ² /K ⁴
Émissivité du vitrage	ε_{vit}	0.85
Émissivité effective vitrage-sol	$\varepsilon_{\text{eff,vs}}$	0.79
Absorptivité solaire du vitrage	α_{vit}	0.07
Fraction solaire interceptée par la canopée	η_{can}	≈ 0.75
Fraction canopée transmise à l'air	β	0.5
Écart ciel-extérieur simplifié	ΔT_{ciel}	20°C

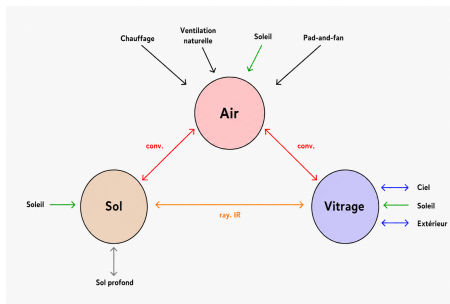
Construction du modèle : trois compartiments thermiques

Le modèle distingue trois variables d'état :

- T_{air} : l'air intérieur
- T_{sol} : le sol (couche unique)
- T_{vit} : le vitrage

Trois actionneurs :

- P_{chauf} : chauffage
- U_{ven} : ouvrants (ventilation naturelle)
- f_{pad} : pad-and-fan



- Air parfaitement mélangé (température uniforme)
- Air gaz parfait à la pression atmosphérique
- Canopée non dynamique : $T_{\text{can}} \approx T_{\text{air}}$
- Vitrage : échange radiatif avec le ciel, convectif avec l'air extérieur
- Sol profond à température constante T_{deep}
- h_{int} et h_{ext} constants
- Flux thermique de la ventilation dépendant linéairement de l'ouverture des ouvrants
- Pas d'infiltration d'air de l'extérieur

*Ce modèle simplifié servira de **modèle interne au MPC**.*

EDO 1 – Air intérieur

$$\begin{aligned} C_{\text{air}} \dot{T}_{\text{air}} = & (1 - \beta) \eta_{\text{can}} \tau_{\text{vit}} A_{\text{sol}} G + P_{\text{chauf}} + h_{\text{sol-air}} A_{\text{sol}} (T_{\text{sol}} - T_{\text{air}}) + h_{\text{int}} A_{\text{env}} (T_{\text{vit}} - T_{\text{air}}) \\ & - \rho_{\text{air}} c_p \frac{f_{\text{pad}}}{3600} V (T_{\text{air}} - T_{\text{pad}}) \\ & - \Phi_{\text{vent}} \end{aligned}$$

EDO 2 – Vitrage

$$\begin{aligned} C_{\text{vit}} \dot{T}_{\text{vit}} = & \alpha_{\text{vit}} A_{\text{env}} G + h_{\text{int}} A_{\text{env}} (T_{\text{air}} - T_{\text{vit}}) \\ & + \varepsilon_{\text{eff,vs}} \sigma A_{\text{sol}} (T_{\text{sol}}^4 - T_{\text{vit}}^4) - h_{\text{ext}} A_{\text{env}} (T_{\text{vit}} - T_{\text{ext}}) \\ & - \varepsilon_{\text{vit}} \sigma A_{\text{env}} (T_{\text{vit}}^4 - T_{\text{ciel}}^4) \end{aligned}$$

EDO 3 – Sol

$$\begin{aligned} C_{\text{sol}} \dot{T}_{\text{sol}} = & (1 - \eta_{\text{can}}) \tau_{\text{vit}} A_{\text{sol}} G + \varepsilon_{\text{eff,vs}} \sigma A_{\text{sol}} (T_{\text{vit}}^4 - T_{\text{sol}}^4) \\ & - h_{\text{sol-air}} A_{\text{sol}} (T_{\text{sol}} - T_{\text{air}}) \\ & - \frac{\lambda_{\text{sol}}}{e_{\text{sol}}} A_{\text{sol}} (T_{\text{sol}} - T_{\text{deep}}) \end{aligned}$$

Les perturbations imposées au MPC et au PI sont périodiques, avec $t_h = t/3600$ en heures.

Température extérieure

$$\theta_T = (t_h - 5) \bmod 24, \quad A_T = 8^\circ\text{C}$$

$$T_{\text{ext}}(t) = \begin{cases} T_{\text{moy}} - A_T \cos\left(\frac{\pi\theta_T}{9}\right), & 0 \leq \theta_T \leq 9, \\ T_{\text{moy}} + A_T \cos\left(\frac{\pi(\theta_T - 9)}{15}\right), & 9 < \theta_T < 24. \end{cases}$$

Rayonnement solaire

$$\theta_G = t_h \bmod 24$$

$$G(t) = \begin{cases} 0, & \theta_G \leq t_{\text{lever}} \text{ ou } \theta_G \geq t_{\text{coucher}}, \\ G_{\text{max}} \sin\left(\frac{\pi(\theta_G - t_{\text{lever}})}{t_{\text{coucher}} - t_{\text{lever}}}\right), & t_{\text{lever}} < \theta_G < t_{\text{coucher}}. \end{cases}$$

Vitesse du vent

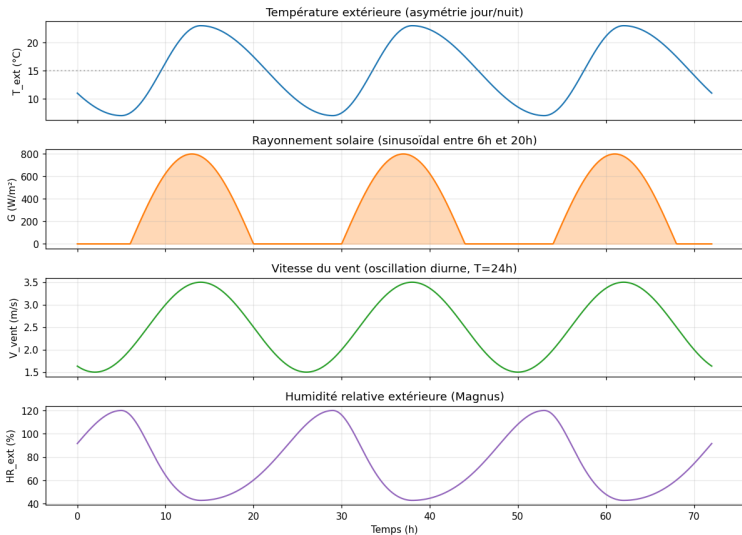
$$V_{\text{vent}}(t) = 2.5 + \sin\left(\frac{2\pi(t_h - 8)}{24}\right) \quad [\text{m/s}]$$

Humidité relative extérieure

$$HR_{\text{ext}}(t) = 100 \frac{VP_{\text{air,ext}}}{VP_{\text{sat}}(T_{\text{ext}}(t))}$$

$$VP_{\text{sat}}(T) = 610.94 \exp\left(\frac{17.625 T}{T + 243.04}\right)$$

Perturbations météo : évolution temporelle



Les commandes sont supposées constantes sur chaque pas d'échantillonnage T_s .

Chauffage

$$\Phi_{\text{chauf}} = P_{\text{chauf}}, \quad 0 \leq P_{\text{chauf}} \leq P_{\text{max}}$$

Pad-and-fan

$$\Phi_{\text{pad}} = \rho_{\text{air}} c_p \frac{f_{\text{pad}}}{3600} V (T_{\text{air}} - T_{\text{pad}})$$

Ventilation naturelle simplifiée

$$\Phi_{\text{vent}} = \rho_{\text{air}} c_p \frac{f_{\text{vent,max}}}{3600} U_{\text{ven}} V (T_{\text{air}} - T_{\text{ext}})$$

Constat

Si la serre simulée utilise le même modèle que le MPC, les résultats sont **trop optimistes** : on évalue le contrôleur sur ses propres prédictions.

- Une vraie serre a des phénomènes que le modèle 3 EDO ignore
- Le MPC doit fonctionner malgré ces écarts
- **Décision** : construire un modèle de **référence enrichi** (5 EDO) qui jouera le rôle de la « serre réelle »

⇒ Le modèle 3 EDO reste utilisé comme modèle interne du MPC ; le modèle à 5 EDO servira quant à lui d'environnement de simulation.

À partir des références *Reyes-Rosas et al. (2017)* et *Chen et al. (2017)* :

- **Sol à 3 couches** (0–5, 5–20, 20–50 cm) : conduction de Fourier

$$k_{ij} = \frac{2\lambda}{e_i + e_j}$$

- **Convection intérieure variable** (régime turbulent, McAdams 1954) :

$$h_{\text{int}} = 0,83 \cdot |T_{\text{air}} - T_{\text{vit}}|^{1/3}$$

- **Température de ciel** (Swinbank 1963) :

$$T_{\text{ciel}} = 0,0552 \cdot T_{\text{ext,K}}^{1,5} - 273,15$$

- **Ventilation naturelle non-linéaire** (Bernoulli, tirage + vent) :

$$D_{\text{vent}} = \sqrt{C_d^2 S_{\text{eq}}^2 \cdot 2gH_{\text{SR}} \frac{\Delta T}{T_K} + S_{\text{eq,w}}^2 C_w V_{\text{vent}}^2}$$

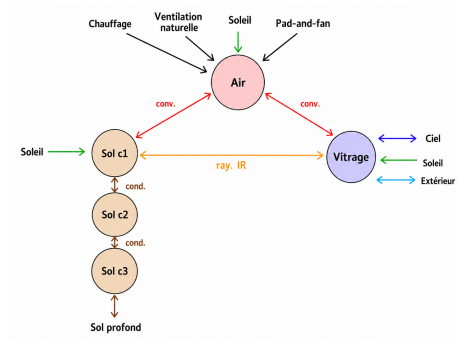
- **Convection extérieure variable** : $h_{\text{ext}} = 7,2 + 3,8 V_{\text{vent}}$ (Garzoli & Blackwell)
- **Infiltration permanente d'air de l'extérieur** : $V_{\text{loss}} = 0,033 \text{ m}^3/\text{s}$

5 variables d'état :

- T_{air}
- T_{s1}, T_{s2}, T_{s3} (3 couches sol)
- T_{vit}

Conduction inter-couches

entre $T_{s1} \leftrightarrow T_{s2} \leftrightarrow T_{s3} \leftrightarrow T_{\text{deep}}$



Air (identique au 3 EDO, mais avec h_{int} variable, D_{vent} , V_{loss}) :

$$C_{\text{air}} \dot{T}_{\text{air}} = \dots - \rho c_p D_{\text{vent}} (T_{\text{air}} - T_{\text{ext}}) - \rho c_p V_{\text{loss}} (T_{\text{air}} - T_{\text{ext}})$$

Couche 1 (surface 0–5 cm) :

$$C_{s1} \dot{T}_{s1} = \Phi_{\text{sol}}^{s1} + \varepsilon_{\text{eff}} \sigma A_{\text{sol}} (T_{\text{vit}}^4 - T_{s1}^4) - h_{\text{sa}} A_{\text{sol}} (T_{s1} - T_{\text{air}}) - k_{12} A_{\text{sol}} (T_{s1} - T_{s2})$$

Couche 2 (5–20 cm) :

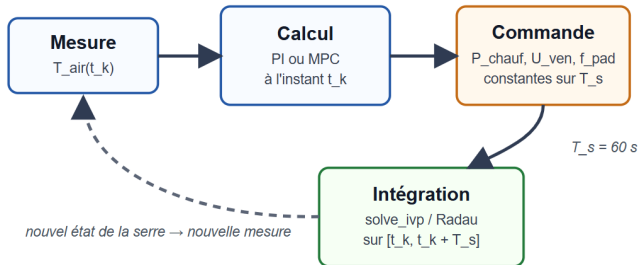
$$C_{s2} \dot{T}_{s2} = k_{12} A_{\text{sol}} (T_{s1} - T_{s2}) - k_{23} A_{\text{sol}} (T_{s2} - T_{s3})$$

Couche 3 (20–50 cm) :

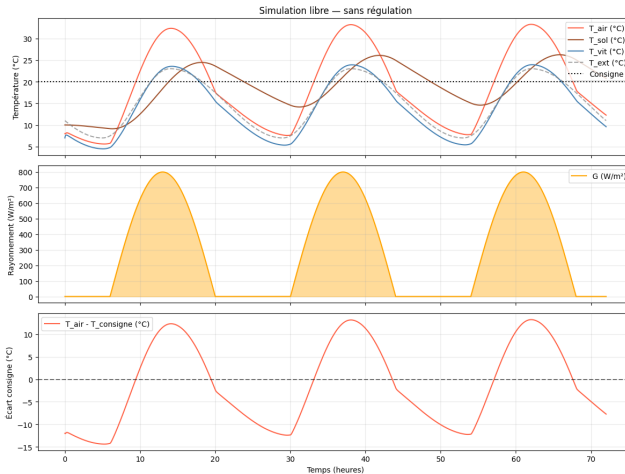
$$C_{s3} \dot{T}_{s3} = k_{23} A_{\text{sol}} (T_{s2} - T_{s3}) - k_{3d} A_{\text{sol}} (T_{s3} - T_{\text{deep}})$$

Vitrage : ajoute T_{ciel} Swinbank, h_{ext} variable.

- Intégration des EDO par `scipy.solve_ivp`, méthode Radau (adaptée aux systèmes raides)
- Résolution **pas par pas** : commande gelée sur chaque intervalle $T_s = 60$ s
- Reflète le fonctionnement réel d'un régulateur échantillonné



Comportement sans régulation (5 EDO libre)



La serre suit passivement le cycle jour–nuit : régulation nécessaire.

Pour comparer les régulateurs, on utilise deux indicateurs :

Erreur intégrée absolue (IAE)

$$IAE = \int_0^{t_f} |T_{\text{air}}(t) - T_{\text{consigne}}| dt \quad [\text{K} \cdot \text{s}]$$

Mesure la qualité du suivi de consigne (plus c'est petit, mieux c'est).

Énergie totale consommée

$$E_{\text{total}} = E_{\text{chauf}} + E_{\text{pad}} = \int P_{\text{chauf}} dt + \int \frac{f_{\text{pad}}}{f_{\text{max}}} P_{\text{fan,max}} dt \quad [\text{kWh}]$$

Inclut le chauffage *et* la puissance électrique des ventilateurs ($P_{\text{fan,max}} = 3 \text{ kW}$).

Compromis fondamental : on cherche à minimiser simultanément IAE et E_{total} .

Loi de commande

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt}$$

- **P** : réagit à l'erreur présente
- **I** : annule l'erreur statique
- **D** : anticipe la variation de l'erreur

$e(t) = T_{\text{consigne}} - T_{\text{air}}(t)$; régulateur **réactif** (pas d'anticipation des perturbations).

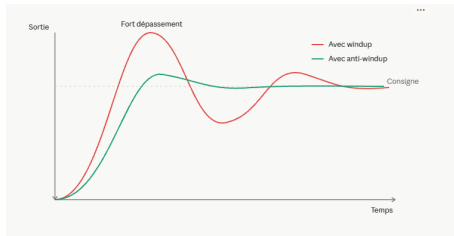
Anti-windup par *clamping*

Problème

Sur saturation des actionneurs, l'intégrale continue de croître → commande « bloquée » même quand l'erreur change de signe (*windup*).

Solution : clamping

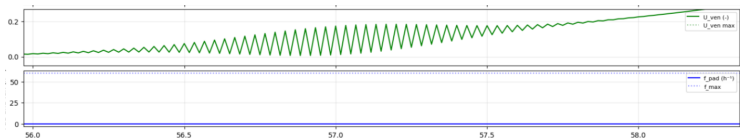
On gèle l'intégration de l'erreur dès que la commande sature et que l'erreur tend à aggraver la saturation.



Le problème du *chattering*

Constat en simulation

Les actionneurs (surtout les ouvrants U_{ven}) oscillent rapidement entre deux états successifs $\Rightarrow$ usure mécanique, comportement irréaliste.



Cause : le PI réévalue la commande tous les $T_s = 60$ s sans tenir compte de l'inertie mécanique réelle des ouvrants.

- **Limiteurs de vitesse** sur chaque actionneur :

$$\Delta P \leq 20 \text{ kW/min}, \quad \Delta U \leq 0.05/\text{min}, \quad \Delta f \leq 6 \text{ h}^{-1}/\text{min}$$

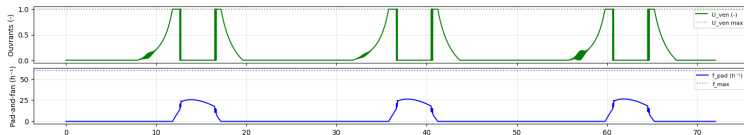
- **Sécurité split-range** : interdire chauffage et refroidissement simultanés
- **Inertie numérique sur U_{ven} uniquement** (modèle mécanique simplifié) :

$$U_{\text{filtered}} = U_{\text{old}} + \alpha_U (U_{\text{target}} - U_{\text{old}}), \quad \alpha_U = 0,8$$

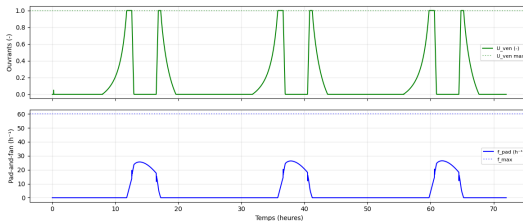
puis limiteur de vitesse classique

- **Condition physique** sur pad-and-fan : ne s'active que si $T_{\text{air}} > T_{\text{pad}}$

Comparaison avant/après intégration de l'anti-chattering



commandes qu'impose le PI sans anti-chattering



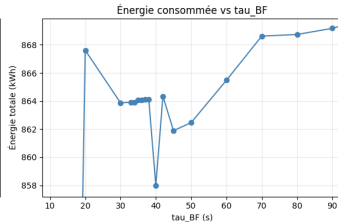
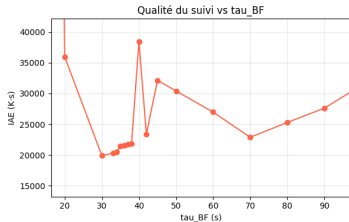
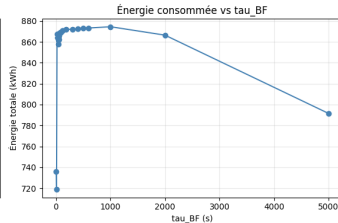
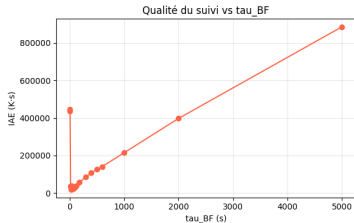
commandes qu'impose le PI avec anti-chattering

Approximation premier ordre de l'air

$$K_p = \frac{C_{\text{air}}}{\tau_{\text{BF}}}, \quad K_i = \frac{G_{\text{pertes}}}{\tau_{\text{BF}}}$$

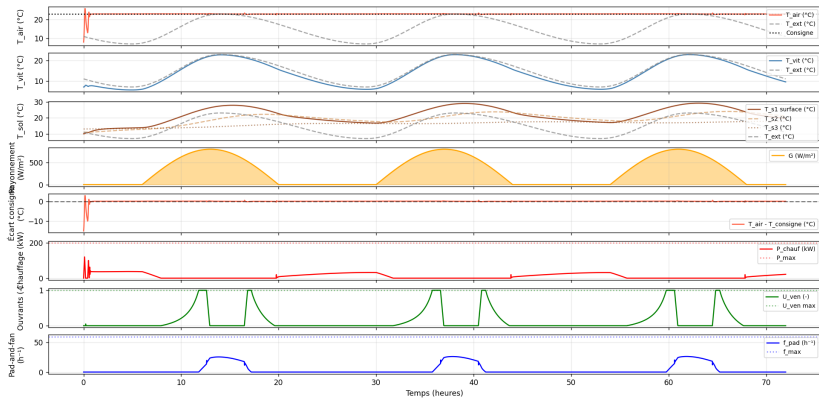
- τ_{BF} : constante de temps de boucle fermée souhaitée (seul paramètre libre)
- $G_{\text{pertes}} = h_{\text{sa}}A_{\text{sol}} + h_{\text{int}}A_{\text{env}}$: pertes thermiques linéaires

Réglage du PI : méthode physique 2

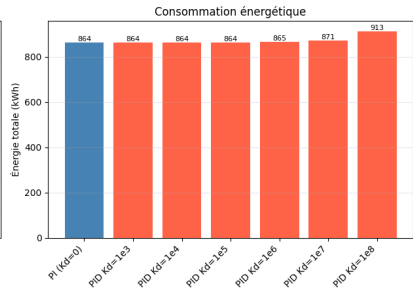
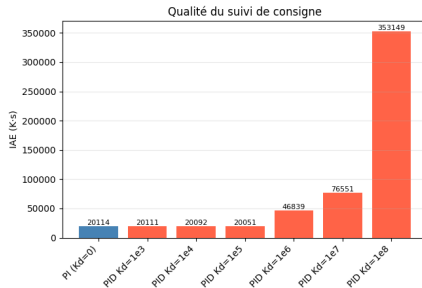


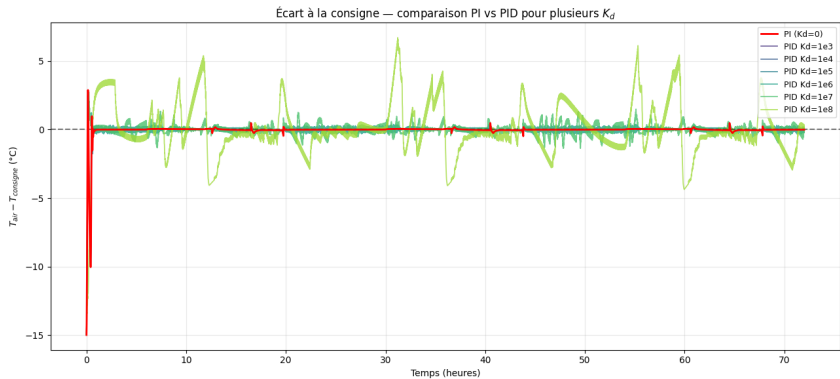
Choix de τ_{BF} : 32 s $\Rightarrow K_p = 48\,100$ W/K, $K_i = 104$ W/K/s.

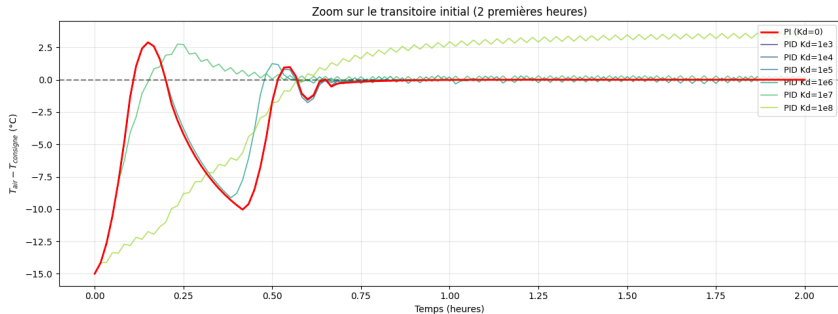
Régulation PI : résultat



Comparaison PI vs PID pour différentes valeurs de K_d







- K_d faible : aucun effet
- K_d élevé : instabilité
- Jamais d'amélioration

Conclusion : le terme D est inutile sur un système à forte inertie thermique. On retient le **PI**.

Un régulateur aveugle au futur

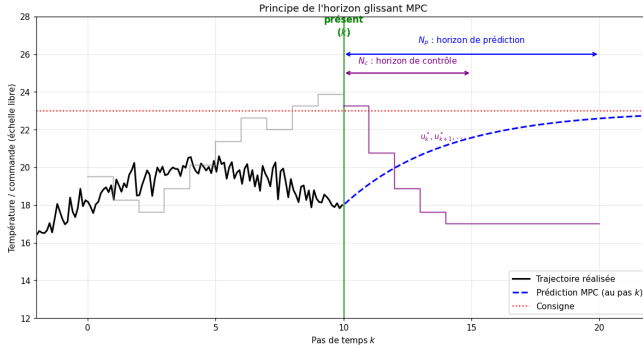
Le PID ne réagit qu'à l'erreur **présente et passée**.

- Il ne « sait » pas que le soleil va se lever
- Il subit les perturbations au lieu de les anticiper
- Pas de gestion explicite des contraintes (saturation gérée *a posteriori*)
- Pas de compromis explicite suivi / énergie
- Or les perturbations d'une serre sont **cycliques et prévisibles** (météo)

⇒ Le MPC exploite directement la cyclicité et les contraintes.

Le principe du MPC

- 1 **Prédire** l'évolution future à l'aide d'un modèle
- 2 **Optimiser** une séquence de commandes sur un horizon
- 3 **Appliquer** la première commande, puis recommencer (horizon glissant)



Fonction de coût à minimiser

$$J = Q \sum_{k=1}^{N_p} (T_{\text{air},k} - T_{\text{consigne}})^2 + R_c \sum_{k=1}^{N_c} u_{c,k}^2 + R_p \sum_{k=1}^{N_c} u_{p,k}^2 \\ + R_{\Delta c} \sum_{k=1}^{N_c} (\Delta u_{c,k})^2 + R_{\Delta v} \sum_{k=1}^{N_c} (\Delta u_{v,k})^2 + R_{\Delta p} \sum_{k=1}^{N_c} (\Delta u_{p,k})^2$$

- Q : pénalise l'écart à la consigne (suivi)
- R_c, R_p : pénalisent l'amplitude des commandes (énergie)
- $R_{\Delta c}, R_{\Delta v}, R_{\Delta p}$: pénalisent *séparément* la variation $\Delta u = u_k - u_{k-1}$ de chacun des trois actionneurs (**anti-chattering intégré**)
- Contraintes explicites : $u_{\min} \leq u_k \leq u_{\max}$

→ Pénaliser Δu dans J lisse les commandes *naturellement*.

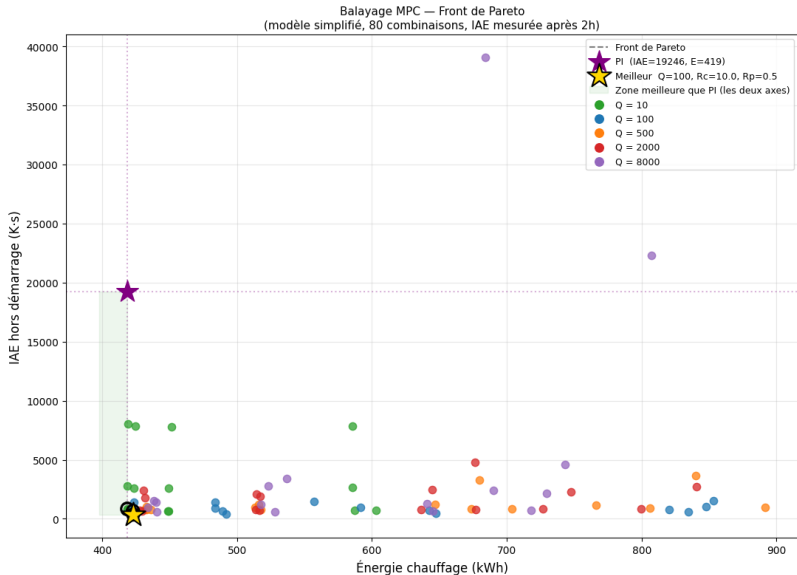
- Horizons : $N_p = 20$ min (prédiction), $N_c = 5$ min (contrôle)
- Pas d'échantillonnage : $T_s = 60$ s
- Intégrateur RK4 à pas fixe pour la prédiction (plus rapide que Radau)
- Optimiseur SLSQP (`scipy.optimize.minimize`)

Méthode :

- On utilise le modèle à 3 EDO comme modèle de la serre réelle (pas de mismatch)
- Grille de 80 combinaisons
- Pour chaque triplet : IAE et E_{total} sur 3 jours
- Sélection : minimum de la distance normalisée à l'optimum

Meilleur triplet retenu : $Q = 100$, $R_c = 10$, $R_p = 0.5$.

Front de Pareto

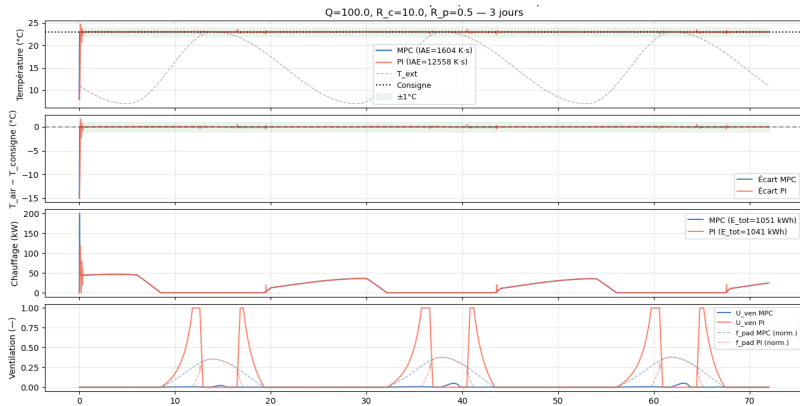


Top 5 triplets

#	Q	R_c	R_p	IAE	E (kWh)
1	100	10.0	0.5	360	423.0
2	10	10.0	0.5	808	418.6
3	10	2.0	0.5	654	422.8
4	100	10.0	2.0	652	422.9
5	10	2.0	2.0	667	423.0

- $R_{\Delta u, \text{chauf}} = 1$
- $R_{\Delta u, \text{ven}} = 50$
- $R_{\Delta u, \text{pad}} = 10$

MPC parfait vs PI (modèle 3 EDO, sans mismatch)



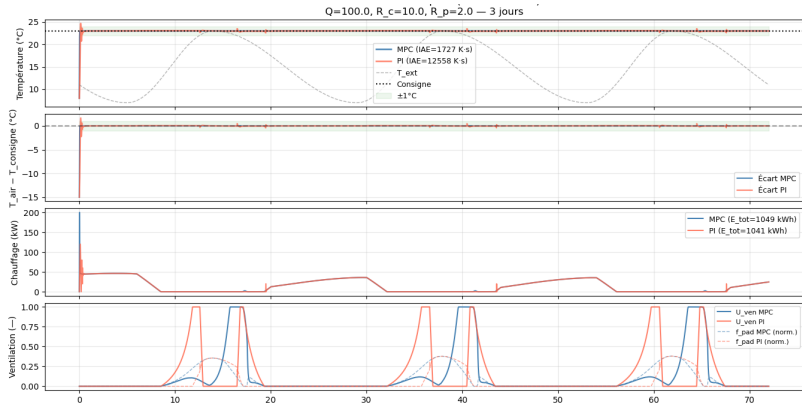
Performances PI vs MPC idéal

Commande	IAE (K.s)	E_{total} (kWh)
MPC idéal	3880	1070.4
PI	12558	1041.3

$\Delta \text{IAE} = 69.1\%$ en faveur du MPC

$\Delta E_{\text{total}} = -2.8\%$ en faveur du MPC

Changements de paramètres



Performances PI vs MPC réel

Commande	IAE (K.s)	E_{total} (kWh)
MPC réel	63160	899.4
PI	20114	863.9

$\Delta \text{IAE} = -214.0\%$ en faveur du MPC

$\Delta E_{\text{total}} = -4.1\%$ en faveur du MPC

Influence de la qualité des prévisions météo

Résumé comparatif

Cas	IAE (K·s)	E_chauf (kWh)	E_pad (kWh)	Temps (s)
Cas 1 — Prévisions parfaites	64 708	868,00	24,33	1 322,4
Cas 2 — Météo constante	46 201	891,38	14,85	1 161,9
Cas 3 — DES (Holt)	66 339	872,07	24,20	1 317,3

Dégradation par rapport au cas parfait (Cas 1)

Cas	Δ IAE	Δ E_chauf
Cas 2 — Météo constante	-28,6 %	+2,7 %
Cas 3 — DES (Holt)	+2,5 %	+0,5 %

Jusqu'ici : objectif = maintenir $T_{\text{air}} = T_{\text{consigne}}$ exactement.

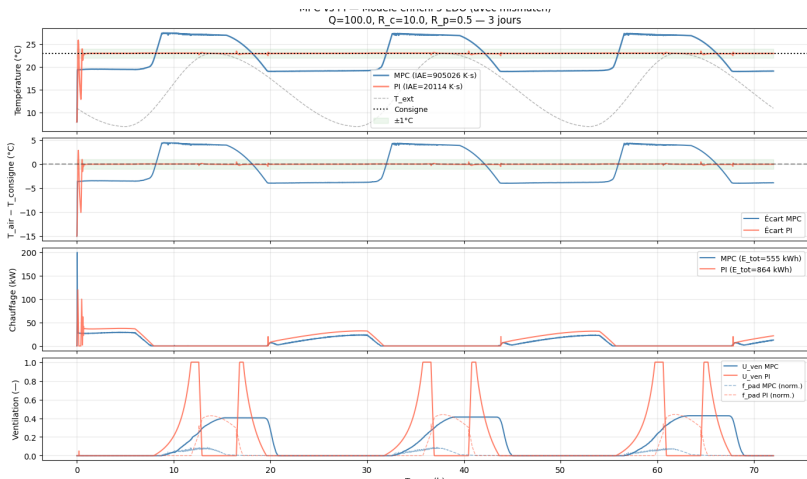
En agriculture, ce qui compte est en réalité une **plage acceptable** :

$$T_{\min} \leq T_{\text{air}} \leq T_{\max}$$

- Le MPC peut formuler ces bornes comme **contraintes molles**
- Il choisit alors la trajectoire qui minimise l'énergie tout en respectant la plage
- Référence : Van Beveren et al. (2015) montrent jusqu'à 47 % d'économie d'énergie
- Cette approche est **impossible** avec un PID, qui ne connaît qu'une consigne ponctuelle

⇒ Un atout structurel du MPC, pas seulement une amélioration quantitative.

Relâchement de la contrainte de suivi



Relâchement de la contrainte de suivi

Contrôleur	IAE (K·s)	E_chauf (kWh)	E_pad (kWh)	E_total (kWh)	Temps (s)
MPC	905 026	552,2	3,2	555,4	1 769
PI	20 114	848,5	15,4	863,9	—
Écart ($\Delta IAE / \Delta E_{total}$)	-4399,4 % MPC			+35,7 % MPC	

PI (avec anti-chattering)

- Simple, robuste
- Réglage analytique direct
- Suit la consigne en moyenne
- Aveugle au futur
- Pas de gestion des contraintes

MPC

- Anticipe les perturbations cycliques
- Gère explicitement les contraintes
- Compromis suivi/énergie réglable
- Plus coûteux à implémenter

Réponse à la problématique

Merci de votre attention

Questions ?

Annexe — Serre_reelle_libre.py (extraits) |

```
1 import numpy as np
2 from scipy.integrate import solve_ivp
3 import matplotlib.pyplot as plt
4
5 # --- Géométrie à pignon ---
6 L, larg = 30.0, 10.0
7 H_wall, H_ridge = 3.5, 5.0
8 dH_roof = H_ridge - H_wall
9 l_pente = np.sqrt((larg/2)**2 + dH_roof**2)
10 A_sol = L * larg
11 V = L*larg*H_wall + 0.5*L*larg*dH_roof
12 A_env = 2*L*l_pente + 2*L*H_wall + 2*(larg*H_wall + 0.5*larg*dH_roof)
13
14 # --- Capacités thermiques ---
15 rho_air, cp_air = 1.2, 1006.0
16 C_air = rho_air * V * cp_air
17 rho_sol, cp_sol = 1600.0, 800.0
18 e_s1, e_s2, e_s3 = 0.05, 0.15, 0.30
19 C_s1 = rho_sol * cp_sol * e_s1 * A_sol
20 C_s2 = rho_sol * cp_sol * e_s2 * A_sol
21 C_s3 = rho_sol * cp_sol * e_s3 * A_sol
22 rho_vit, cp_vit, e_vit = 1200.0, 1200.0, 0.008
23 C_vit = rho_vit * e_vit * cp_vit * A_env
24
25 # --- Coefficients d'echange variables (modele de reference) ---
26 def h_int_ref(T_air, T_vit):
27     dT = max(abs(T_air - T_vit), 1e-2)
28     return 0.83 * dT**(1.0/3.0)
29
30 def h_ext_ref(V_vent_now):
31     return 7.2 + 3.8 * V_vent_now
32
33 def T_ciel_swinbank(T_ext_C):
34     T_K = T_ext_C + 273.15
35     return 0.0552 * T_K**1.5 - 273.15
36
```

```

37 # --- Ventilation naturelle non-linéaire (Bernoulli) ---
38 def D_vent(U_ven, T_air, T_ext_now, V_vent_now):
39     if U_ven <= 1e-6: return 0.0
40     S_lat = S_lat_max * U_ven
41     S_roof = S_roof_max * U_ven
42     T_ext_K = T_ext_now + 273.15
43     dT = abs(T_air - T_ext_now)
44     S_eq_buoy_sq = (S_lat**2 * S_roof**2) / (S_lat**2 + S_roof**2)
45     term_buoy = C_d**2 * S_eq_buoy_sq * 2.0 * g_grav * H_SR * dT / T_ext_K
46     S_eq_wind = C_d * (S_lat + S_roof) / 2.0
47     term_wind = S_eq_wind**2 * C_w * V_vent_now**2
48     return np.sqrt(term_buoy + term_wind)
49
50 # --- Modele 5 EDO ---
51 def serre(t, Y):
52     T_air, T_s1, T_s2, T_s3, T_vit = Y
53     # ... (bilans d'énergie sur chaque compartiment) ...
54     return [dTair_dt, dTs1_dt, dTs2_dt, dTs3_dt, dTvit_dt]

```

Annexe — PI_final.py (extraits) |

```
1  # --- Bornes et gains PI ---
2  P_max = 200_000.0
3  f_max = 60.0
4  P_fan_max = 3000.0
5  KP = 48100 # W/K
6  KI = 104 # W/K/s
7  T_s = 60.0
8
9  # --- Anti-chattering ---
10 DP_MAX_STEP = 20_000.0 # W par pas
11 DU_MAX_STEP = 0.05 # ouverture par pas
12 DF_MAX_STEP = 6.0 # h^-1 par pas
13 ALPHA_U = 0.8 # inertie numerique ouvrants
14
15 def rate_limit(x_target, x_old, dx_max):
16     return x_old + np.clip(x_target - x_old, -dx_max, dx_max)
17
18 def smooth_rate_limit_U(U_target, U_old, dU_max, alpha):
19     U_filtered = U_old + alpha * (U_target - U_old)
20     return U_old + np.clip(U_filtered - U_old, -dU_max, dU_max)
21
22 # --- Regulateur PI avec anti-windup clamping ---
23 class PI_Controller:
24     def __init__(self, Kp, Ki, u_min, u_max, Ts):
25         self.Kp, self.Ki = Kp, Ki
26         self.u_min, self.u_max = u_min, u_max
27         self.Ts = Ts
28         self.integral = 0.0
29     def update(self, consigne, mesure):
30         e = consigne - mesure
31         u_brut = self.Kp * e + self.Ki * self.integral
32         u_sat = max(self.u_min, min(self.u_max, u_brut))
33         sat_h = (u_brut > self.u_max) and (e > 0)
34         sat_l = (u_brut < self.u_min) and (e < 0)
35         if not (sat_h or sat_l):
36             self.integral += e * self.Ts
```

Annexe — PI_final.py (extraits) ||

```
37         return u_sat
38
39 # --- Logique split-range refinee ---
40 def update_cmd(T_air_mesure, t_now):
41     u_sat = pi.update(T_consigne, T_air_mesure)
42     T_ext_now = T_ext(t_now)
43     if u_sat >= 0:
44         P_target, U_target, f_target = u_sat, 0.0, 0.0
45     else:
46         cooling_demand = -u_sat
47         if T_air_mesure > T_ext_now + 1.0:
48             P_vent_max = rho_air*cp_air*(f_vent_max/3600)*V*(T_air_mesure-T_ext_now)
49             if cooling_demand <= P_vent_max:
50                 P_target, U_target, f_target = 0.0, cooling_demand/P_vent_max, 0.0
51             else:
52                 P_target, U_target = 0.0, 1.0
53                 remaining = cooling_demand - P_vent_max
54                 T_pad_now = T_ext_now - eps_pad*(T_ext_now-T_wb(T_ext_now, HR_ext(t_now)))
55                 f_target = min(k_pad*remaining, f_max) if T_air_mesure > T_pad_now else 0.0
56         else:
57             P_target, U_target = 0.0, 0.0
58             f_target = min(k_pad*cooling_demand, f_max)
59 # Securite split-range : exclure chauff/refroid simultanes
60 P_old, U_old, f_old = current_command
61 if P_target > 1e-9 and (U_old > 1e-9 or f_old > 1e-9):
62     P_target = 0.0
63 if (U_target > 1e-9 or f_target > 1e-9) and P_old > 1e-9:
64     U_target = f_target = 0.0
65 # Appliquer les limiteurs de variation
66 current_command[0] = rate_limit(P_target, P_old, DP_MAX_STEP)
67 current_command[1] = smooth_rate_limit_U(U_target, U_old, DU_MAX_STEP, ALPHA_U)
68 current_command[2] = rate_limit(f_target, f_old, DF_MAX_STEP)
```

Annexe — compMPC_PID_modele_simple.py (extraits) |

```
1 # --- Parametres MPC ---
2 Q_mpc = 50.0      # poids suivi de consigne
3 R_chauf = 10.0     # cout chauffage normalise
4 R_pad = 10.0       # cout pad-and-fan normalise
5 R_dU_chauf = 5.0   # anti-chattering chauffage
6 R_dU_ven = 50.0    # anti-chattering ouvrants (plus penalise)
7 R_dU_pad = 10.0    # anti-chattering pad-and-fan
8
9 Np = 20            # horizon de prediction (min)
10 Nc = 5             # horizon de controle (min)
11 T_s = 60.0
12
13 # --- Modele interne MPC (3 EDO simplifie) ---
14 def f_simp(t, Y, cmd):
15     T_air, T_sol, T_vit = Y
16     P_c, U_v, f_p = cmd
17     # ... bilans d'energie air/sol/vitrage ...
18     return np.array([dT_a, dT_s, dT_v])
19
20 def rk4_step(t, Y, cmd):
21     k1 = f_simp(t, Y, cmd)
22     k2 = f_simp(t+T_s/2, Y+T_s/2*k1, cmd)
23     k3 = f_simp(t+T_s/2, Y+T_s/2*k2, cmd)
24     k4 = f_simp(t+T_s, Y+T_s*k3, cmd)
25     return Y + (T_s/6.0)*(k1 + 2*k2 + 2*k3 + k4)
26
27 # --- Prediction sur l'horizon Np ---
28 def predict_mpc(Y0, x_norm, t0):
29     Y = Y0.copy(); t = t0
30     T_pred = np.empty(Np)
31     for i in range(Np):
32         k = min(i, Nc-1)
33         cmd = np.array([x_norm[k*3]*P_max, x_norm[k*3+1], x_norm[k*3+2]*f_max])
34         Y = rk4_step(t, Y, cmd)
35         t += T_s
36         T_pred[i] = Y[0]
```

```

37     return T_pred
38
39 # --- Fonction de cout avec 3 termes anti-chattering separees ---
40 def cost_mpc(x_norm, Y0, t0, u_prev):
41     T_pred = predict_mpc(Y0, x_norm, t0)
42     J_track = Q_mpc * np.mean((T_pred - T_consigne)**2)
43     x_mat = x_norm.reshape(Nc, 3)
44     J_act = R_chauf*np.mean(x_mat[:,0]**2) + R_pad*np.mean(x_mat[:,2]**2)
45
46     # 3 termes Delta U separees
47     u_chauf_aug = np.r_[u_prev[0], x_mat[:, 0]]
48     J_dU_chauf = R_dU_chauf * np.mean(np.diff(u_chauf_aug)**2)
49     u_ven_aug = np.r_[u_prev[1], x_mat[:, 1]]
50     J_dU_ven = R_dU_ven * np.mean(np.diff(u_ven_aug)**2)
51     u_pad_aug = np.r_[u_prev[2], x_mat[:, 2]]
52     J_dU_pad = R_dU_pad * np.mean(np.diff(u_pad_aug)**2)
53
54     return J_track + J_act + J_dU_chauf + J_dU_ven + J_dU_pad
55
56 # --- Boucle MPC : horizon glissant ---
57 bounds_mpc = [(0.0, 1.0)] * (Nc * 3)
58 U_warm = np.zeros(Nc * 3)
59 u_prev = np.zeros(3)
60
61 for k in range(N_STEPS):
62     Y_int = Y_ref.copy() # observer parfait (sans mismatch)
63     res = minimize(cost_mpc, U_warm,
64                   args=(Y_int.copy(), t_cur, u_prev.copy()),
65                   method='SLSQP', bounds=bounds_mpc,
66                   options={'maxiter': 80, 'ftol': 1e-6})
67
68     x = res.x
69     current_command[0] = x[0]*P_max
70     current_command[1] = x[1]
71     current_command[2] = x[2]*f_max
72     u_prev = x[:3].copy()
73     # Warm-start : decaler la sequence pour le prochain pas
74     U_warm[::(Nc-1)*3] = x[3:]

```

```
74 U_warm[(Nc-1)*3:] = x[(Nc-1)*3:]
75 # Intégrer la plante (Radau)
76 sol = solve_ivp(serre_plant, (t_cur, t_cur+T_s), list(Y_ref),
77                 method='Radau', rtol=1e-4, atol=1e-6)
78 Y_ref = sol.y[:, -1]
79 t_cur += T_s
```