MINISTÈRE
DE L'ÉDUCATION
NATIONALE,
DE LA JEUNESSE
ET DES SPORTS
Liberté
Égalité
Fraternité

EAE SIN 2

SESSION 2022

AGREGATION
CONCOURS EXTERNE

Section : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR

Option : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR ET INGÉNIERIE INFORMATIQUE

MODÉLISATION D'UN SYSTÈME, D'UN PROCÉDÉ OU D'UNE ORGANISATION

Durée : 6 heures

Calculatrice autorisée selon les modalités de la circulaire du 17 juin 2021 publiée au BOEN du 29 juillet 2021.

L'usage de tout ouvrage de référence, de tout dictionnaire et de tout autre matériel électronique est rigoureusement interdit.

Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.

NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.

Tournez la page S.V.P.

INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie.

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

Concours	Section/option	Epreuve	Matière
EAE	1417A	102	2680

Ce sujet comporte :

- **le sujet et le travail demandé (pages 1 à 24) ;**
- **les annexes (pages 25 à 34) ;**
- **les documents réponses DR1, DR2, DR3, DR4 et DR5.**

Modélisation du système de contrôle d'une pile à combustible

Ce sujet s'appuie sur les articles et ouvrages cités dans la bibliographie page 24.

Analyse structurelle de la pile à combustible

Objectif : Modéliser l'ensemble du système d'une pile à combustible afin de contrôler son alimentation.

En développement intensif ces dernières années, les piles à combustible permettent une production d'énergie propre répondant aux enjeux écologiques d'actualité pour des applications statiques et mobiles.

Le principe de la pile à combustible a été découvert en 1839 par William R. Grove : l'hydrogène et l'oxygène réagissent en produisant de l'électricité, de l'eau et un dégagement de chaleur.

Le coeur de la pile est composé de deux électrodes, l'anode et la cathode, séparées par un électrolyte. Dans une pile à membrane échangeuse de protons (Proton Exchange Membrane Fuel Cells), une des technologies de piles à combustible existantes, ce sont les protons hydratés $\mathrm{H_3O+}$ qui migrent de l'anode vers la cathode, où l'eau est produite.

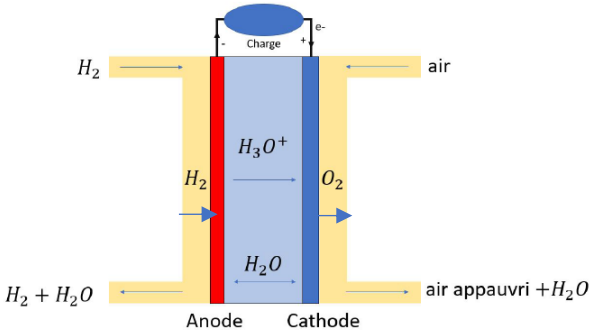


Figure 1 - Principe de fonctionnement d'une cellule élémentaire de pile à combustible

L'analyse ne se focalise pas ici sur la cellule élémentaire illustrée Figure 1, mais sur le système complet de gestion de la pile. Ce système est composé d'un assemblage de cellules élémentaires connectées en série et en parallèle, mais aussi de plusieurs

Tournez la page S.V.P.

B

composants auxiliaires pour former un système de pile à combustible complet comme illustré Figure 2.

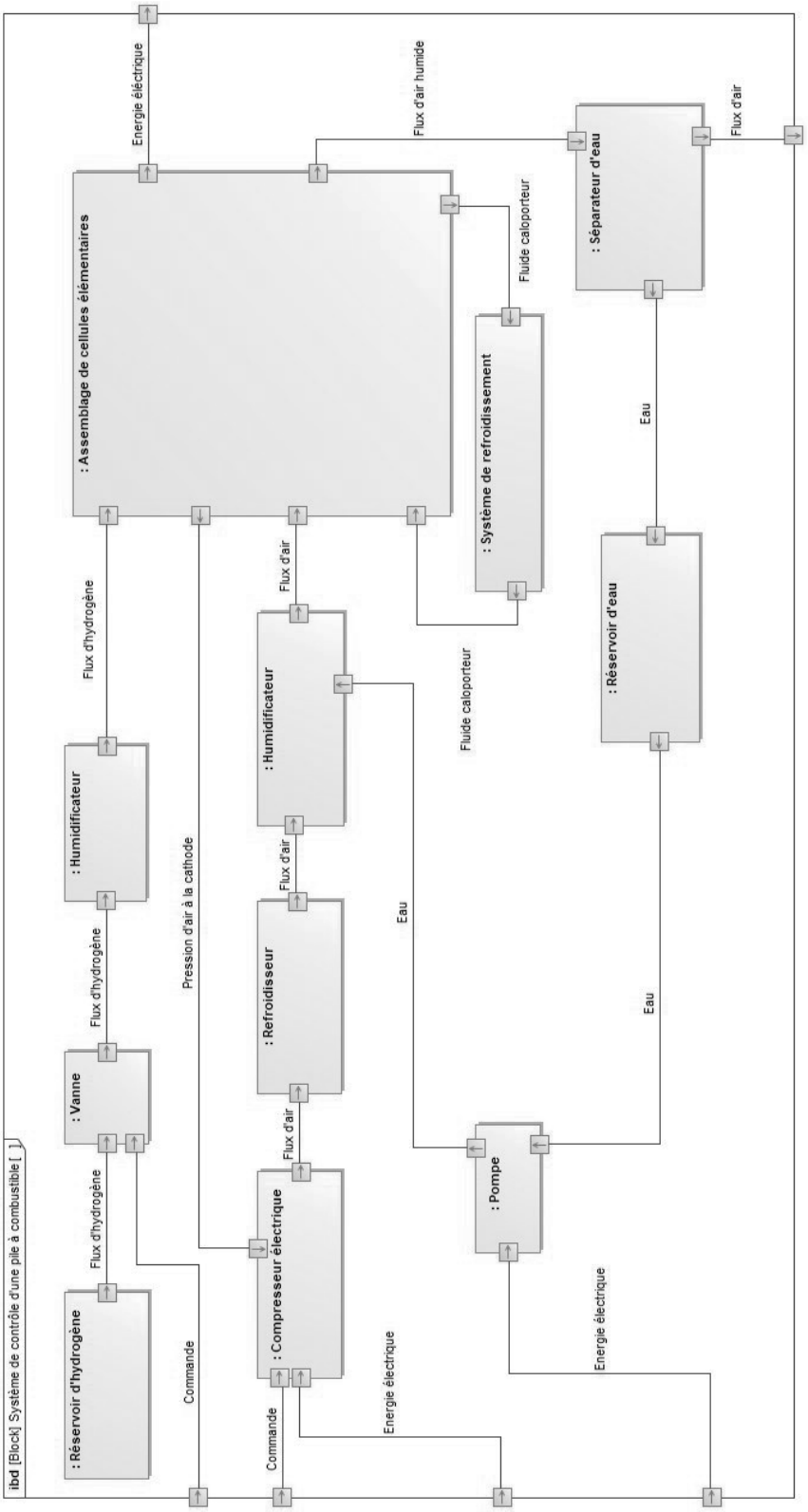


Figure 2 - Diagramme de flux

Un compresseur électrique est employé afin de fournir le flux d'air souhaité à la cathode. Le flux d'air sous pression sortant du compresseur est refroidi puis humidifié

afin d'éviter la déshydratation de la membrane. L'anode est alimentée par un réservoir d'hydrogène sous pression contrôlée par une vanne. Au sein de la pile, l'hydrogène et l'air réagissent et produisent de l'électricité, de l'eau et de la chaleur. La température est contrôlée par un système de refroidissement composé d'un échangeur et d'une pompe permettant la circulation d'un fluide caloporteur. À la sortie de la pile, un séparateur d'eau permet de récupérer la vapeur produite, présente dans le flux d'air, afin d'alimenter l'humidificateur.

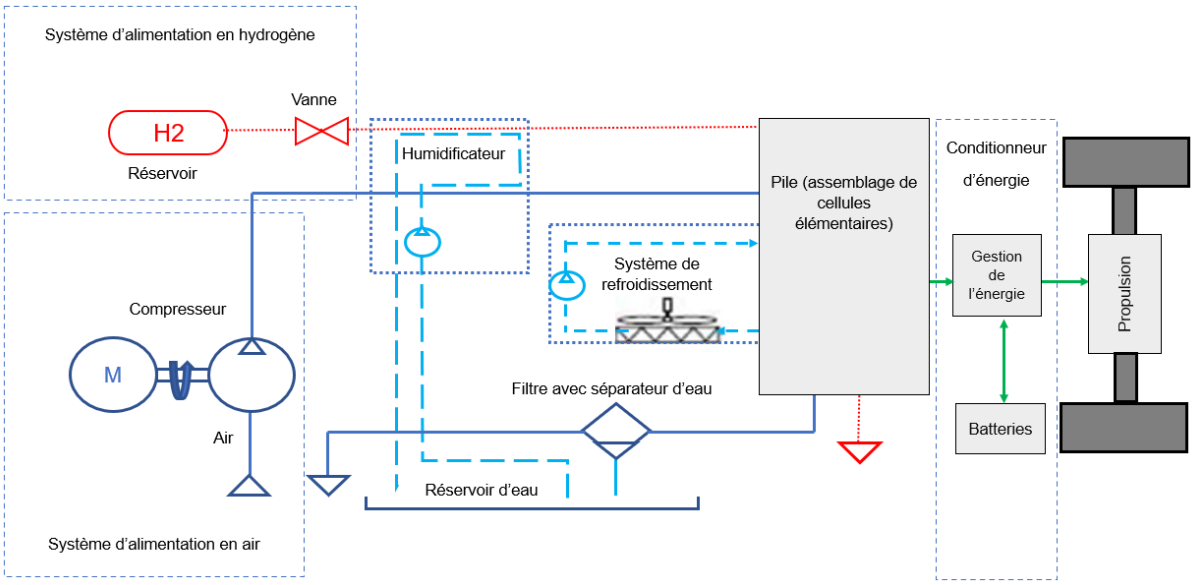


Figure 3 – Schématisation du système complet

Ainsi, comme illustré Figure 3, l'ensemble de ces éléments est organisé en cinq sous-systèmes :

- le système d'alimentation en hydrogène pour l'anode ;
- le système d'alimentation en air pour la cathode ;
- le système d'alimentation en eau servant de liquide de refroidissement ;
- le système d'alimentation en eau de l'humidificateur pour humidifier l'hydrogène et les flux d'air ;
- le conditionneur d'énergie qui permet d'alimenter le moteur.

Pendant le fonctionnement du véhicule, le système est soumis à des variations de charge. Le contrôle du système est alors nécessaire pour maintenir une température optimale, une hydratation de la membrane et une pression désirée des réactifs afin d'éviter une perte d'efficacité et une dégradation de la pile. Les principaux paramètres à contrôler sont les débits d'oxygène et d'hydrogène, les pressions correspondantes, la température ainsi que l'humidité de la membrane. Ceux-ci ne sont pas indépendants.

L'objectif de la première partie du sujet est d'établir un modèle du compresseur afin de contrôler l'alimentation en air du système. À partir de la modélisation du compresseur ainsi que de la modélisation de l'ensemble des autres composants la partie 2 permet de déterminer la commande permettant de fournir le flux suffisant pour assurer des réponses rapides du système tout en minimisant la consommation d'énergie auxiliaire.

Tournez la page S.V.P.

La partie 3 permet de modéliser les transferts de données entre les différents sous-systèmes afin dans la partie 4 d'évaluer la durée de vie des valeurs échangées et de la latence *last-to-first* de la pile à combustible.

Partie 1 Modélisation du compresseur

Objectif : Modéliser le compresseur afin de contrôler l'alimentation en air du système en réduisant les pertes énergétiques.

Dans ce sujet le langage Python est utilisé et l'on considère que la ligne 'import numpy as np' a été exécutée (la documentation numpy est donnée en annexe 1).

L'ensemble de la boucle d'alimentation est décrit Figure 4. Les volumes de cathode et d'anode des multiples cellules de la pile à combustible ainsi que les volumes des collecteurs et tuyaux sont regroupés respectivement en un volume d'anode et un autre de cathode.

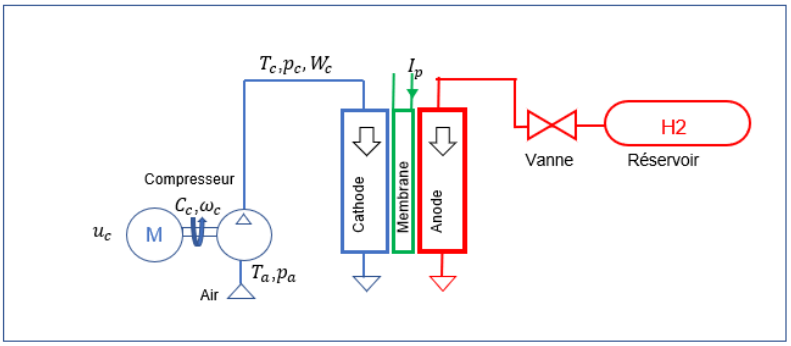


Figure 4 - Sous-système d'alimentation

En entrée du compresseur, l'air est à la température extérieure T_a et à la pression atmosphérique p_a. La tension d'alimentation du moteur du compresseur est noté u_c. La puissance en sortie du moteur (produit du couple moteur C_c et de la vitesse de rotation de l'arbre en sortie ω_c) est noté P_c. En sortie de compresseur le flux d'air W_c est à la température T_c et à la pression p_c. Ce flux d'air va permettre d'alimenter la cathode, tandis que l'anode est alimentée en hydrogène par un réservoir sous pression. I_p est le courant de la pile. L'ensemble des notations ainsi que les unités associées est rappelé Tableau 1.

Notation	Grandeur	Unité
T_a	Température extérieure	°C
p_a	Pression atmosphérique	Pa
u_c	Tension d'alimentation du moteur du compresseur	V
C_c	Couple moteur	$\mathrm{N.m}$
ω_c	Vitesse de rotation de l'arbre en sortie	$\mathrm{rad.s^{-1}}$
P_c	Puissance en sortie du moteur	W
W_c	Flux d'air en sortie de compresseur	$\mathrm{Kg.s^{-1}}$
T_c	Température en sortie de compresseur	°C
p_c	Pression en sortie de compresseur	Pa
I_p	Courant de la pile	A

Tableau 1 : Tableau des notations

Les pertes énergétiques les plus importantes du système sont dues au compresseur permettant l'alimentation en oxygène. Ainsi, la régulation de l'alimentation en oxygène doit être accomplie rapidement et efficacement pour éviter la dégradation de la pile liée à un manque d'oxygène tout en minimisant la puissance perdue.

Le comportement du compresseur peut être déterminé expérimentalement et synthétisé par un ensemble de courbes usuellement appelé cartographie illustrée Figure 5. Ces courbes donnent le rapport de la pression en sortie de compresseur sur la pression atmosphérique $\frac{p_c}{p_a}$ en fonction du débit d'air au travers du compresseur W_c pour une vitesse de rotation du compresseur donnée ω_c.

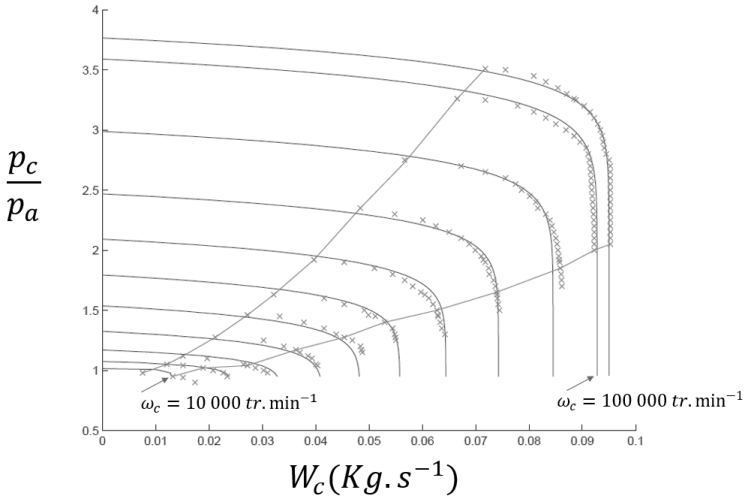


Figure 5 - Cartographie représentant le rapport de pression en fonction du débit massique pour le compresseur. Les mesures expérimentales sont données par des croix. La fonction d'approximation est représentée en continu.

Un extrait de la cartographie du compresseur est représenté Figure 6 pour une seule vitesse de rotation ω_c fixée.

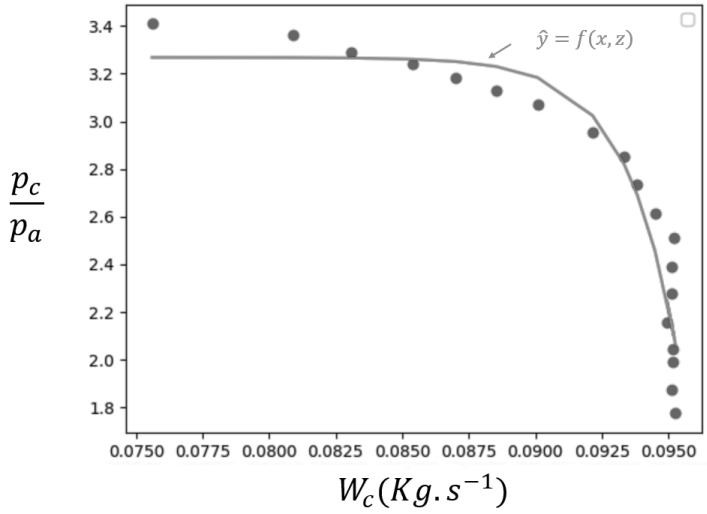


Figure 6 - Extrait de cartographie représentant le rapport de pression en fonction du débit massique. Les mesures expérimentales sont données par les points. La fonction d'approximation est représentée en continu.

Tournez la page S.V.P.

Les différents points de cette cartographie ont été numérisés et saisis dans deux tableaux numpy:

- le tableau `debit` contient le débit massique pour chaque point de mesure en $(Kg.\mathrm{s}^{-1})$;
- le tableau `RPressions` contient le rapport entre la pression d'entrée et de sortie du compresseur $\frac{p_c}{p_a}$ pour chaque point de mesure (sans unité).

Afin de modéliser la partie cartographie, le comportement du compresseur est approximé par une fonction de la forme :

$$\hat{y} = f(x,z) = a * \left(1 - e^{\frac{x-c}{b}}\right) \quad (1)$$

avec a, b, c trois paramètres regroupés dans un vecteur $\mathrm{z} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ que l'on cherche à déterminer. $\hat{y}$ sera alors l'approximation du rapport de pression pour un débit massique x choisi.

Q 1. Définir en Python la fonction $\hat{y} = f(x,z)$ qui prend en arguments x et z, et qui renvoie la valeur de $\hat{y}$ correspondant à l'équation 1.

La classe du solveur Gauss-Newton (permettant de réaliser une régression non linéaire en utilisant une minimisation par une méthode des moindres carrés) est donnée dans le document réponse DR1.

Soit m points de mesure numérotés $i = 0, .., m-1$ de coordonnées (x_i, y_i). Le vecteur x (resp y) contient l'ensemble des abscisses x_i (resp ordonnées y_i) de ces points de mesure.

Le résidu pour un point de mesure $r_i(z) = y_i - f(x_i, z)$ est l'écart entre l'approximation donnée par $f(x_i, z)$ et la valeur mesurée y_i. Le vecteur $r(z)$ contient l'ensemble des résidus r_i de chacun des points.

Dans le cas des moindres carrés non linéaires, l'objectif est alors de minimiser la fonction $g(z)$ telle que :

$$g(z) = \frac{1}{2} r(z)^T r(z) = \frac{1}{2} \sum_{i=0}^{m-1} r_i^2(z)$$

Les lignes de code suivantes permettent d'estimer les coefficients stockés dans z correspondant à cette minimisation pour une fonction donnée f.

```
solver = GNSolver(fit_function=f, max_iter=1000, tolerance_difference=10**(-7))
init_guess = [1., 0.01, 0.01]
z = solver.fit(debit, RPressions, init_guess)
```

Dans la classe Gauss-Newton, la méthode `_calculate_residual` permet de calculer le résidu $r = y - \hat{y}$ pour l'ensemble des valeurs de x. La méthode `get_residual` permet de récupérer ce résidu.

Q 2. Compléter la ligne de code de la méthode `fit` afin d'affecter à la variable `residual` les valeurs du résidu r. Cette ligne symbolisée par un trait horizontal est indiquée Q2 sur le document réponse DR1.

L'erreur quadratique moyenne (noté rmse) est la racine carrée de la moyenne des carrés des valeurs des composantes de $\boldsymbol{r}$: $\boldsymbol{rmse} = \sqrt{\frac{1}{m}\sum_{i=0}^{m-1} \boldsymbol{r}_i^2(\boldsymbol{z})}$.

Q 3. Compléter la ligne de code de la méthode `fit` afin d'affecter à la variable `rmse` la valeur de l'erreur quadratique moyenne. Cette ligne symbolisée par un trait horizontal est indiquée Q3 sur le document réponse DR1. La fonction `sqrt` (racine carrée) peut être utilisée.

Afin de résoudre le problème de minimisation de g, il est nécessaire de déterminer les dérivées premières et secondes de $g(z)$. La dérivée première de $g(z)$ s'écrit $\nabla\, g(z) = \nabla\, r(z)^T r(z)$ avec $\nabla\, r(z)$ notée J la matrice Jacobienne du modèle :

$$J = \nabla\, r(z) = \begin{bmatrix} \frac{\partial r_0(z)}{\partial z_0} & \frac{\partial r_0(z)}{\partial z_1} & \frac{\partial r_0(z)}{\partial z_2} \\ \vdots & \vdots & \vdots \\ \frac{\partial r_{m-1}(z)}{\partial z_0} & \frac{\partial r_{m-1}(z)}{\partial z_1} & \frac{\partial r_{m-1}(z)}{\partial z_2} \end{bmatrix}$$

$$\text{soit } J_{ij} = \frac{\partial r_i(z)}{\partial z_j} \quad j \in \{0,1,2\} \quad i \in \{0,1,..,m-1\}$$

Une approximation numérique de la matrice Jacobienne peut être obtenue avec une différence progressive, qui est définie par :

$$\widetilde{J_{ij}} = \frac{r_i(z + \delta e_j) - r_i(z)}{\delta} \quad j \in \{0,1,2\} \quad i \in \{0,1,..,m-1\}$$

où δ est choisi suffisamment petit, $\mathrm{z} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ le vecteur défini précédemment et où e_j est le j-ième vecteur de la matrice identité de dimension 3.

La structure de la méthode `_calculate_jacobian` est donnée ci-après :

- Calculer le résidu avec les coefficients de z : $R_0 = r(z)$.
- Créer une liste vide J.
- Pour $j \in \{0,1,2\}$ faire
  - Calculer le vecteur des coefficients modifiés $\hat{z} = z + \delta e_j$ avec le pas δ.
  - Calculer le résidu correspondant (vecteur colonne de l'ensemble des résidus) $R = r(\hat{z})$
  - Calculer la différence progressive (vecteur colonne de la matrice Jacobienne) $différence = (R - R_0)/\delta$
  - Ajouter cette dérivée à la liste J.

Tournez la page S.V.P.

Q 4. En utilisant la structure fournie, compléter la méthode `_calculate_jacobian` dans GNSolver qui prend en argument le vecteur des paramètres z et le pas δ et qui renvoie la matrice Jacobienne. Les lignes sont à compléter dans le document réponse DR1 dans le cadre annoté Q4.

La dérivée seconde est $\nabla^2 g(z) = \nabla r(z)^T \nabla r(z) + S(z)$ avec $S(z) = \sum_{i=0}^{m-1} r_i(z) \nabla^2 r_i(z)$ considéré négligeable dans la méthode de Gauss-Newton.

Soit $h(z)$ l'approximation de $g(z)$ au voisinage de z_0, une valeur particulière de z, $h(z) = g(z_0) + \nabla g(z_0)^T (z - z_0)$. L'objectif de la méthode est alors de déterminer z tel que $\nabla h(z) = 0$ soit $\nabla h(z) = \nabla g(z_0) + \nabla^2 g(z_0)^T (z - z_0) = 0$. Ainsi z est déterminé en résolvant $\nabla^2 g(z_0)^T (z - z_0) = -\nabla g(z_0)$.

La résolution est effectuée de manière itérative. On part d'un choix de coefficient de départ z_0 et on construit par récurrence la suite :

$$z_{k+1} = z_k - (\nabla^2 g(z_k)^T)^{-1} \nabla g(z_k)$$

Q 5. Les commentaires de la méthode `fit` permettant d'estimer les paramètres du modèle sont à compléter. Pour cela, indiquer pour chacune des lignes du tableau du document réponse DR2 un commentaire succinct indiquant l'opération réalisée à la ligne de code correspondante.

Q 6. Après avoir expliqué la différence entre les deux tolérances fixées comme critère de convergence, réaliser une analyse critique de cet algorithme.

Le comportement dynamique du compresseur est déterminé par le principe fondamental de la dynamique appliqué à son axe de rotation :

$$J_c \frac{d\omega_c}{dt} = C_m - C_c$$

où J_c représente l'inertie équivalente du rotor du moteur du compresseur et de l'axe du compresseur ramené à l'axe du compresseur, C_m le couple du moteur du compresseur et C_c le couple nécessaire pour faire fonctionner le compresseur dans les conditions désirées.

Le couple du compresseur C_c dépend de la capacité thermique massique de l'air c_a, du rapport de la capacité thermique à pression constante sur celle à volume constant γ, du rendement du compresseur η_c, de sa vitesse de rotation ω_c, de la pression en sortie de compresseur p_c, de la pression atmosphérique p_a, de la température en entrée du compresseur T_a et du flux en sortie du compresseur F_c.

$$C_c = \frac{c_a T_a}{\omega_c \eta_c} \left[\left(\frac{p_c}{p_a} \right)^{\frac{\gamma-1}{\gamma}} - 1 \right] W_c$$

Le couple du moteur du compresseur C_m peut être déterminé à partir du rendement du moteur du compresseur η_m, de sa constante de couple k_c, de sa constante de force contre-électromotrice k_e, de sa résistance R_m, de sa vitesse de rotation ω_c et de sa tension d'alimentation u_c :

$$C_m = \eta_m \frac{k_c}{R_m}(u_c - k_e \omega_c)$$

Q 7. Exprimer $\frac{d\omega_c}{dt}$ en fonction de u_c,ω_c,p_c, des constantes $k_c, k_e, \eta_m, R_m, p_a, \gamma, c_a, T_a, \eta_c, p_a$ et des coefficients a, b, c de l'équation (1). W_c sera pour cela exprimé en fonction de p_c et des constantes à partir de l'équation (1).

Q 8. Linéariser l'expression de $\frac{d\omega_c}{dt}$ autour du point de fonctionnement (u_{c_0}, ω_{c_0}, p_{c_0}).

Le modèle du compresseur peut ainsi être décomposé en deux parties, comme illustré sur le document réponse DR3.

La cartographie permet de déterminer la température de l'air en sortie de compresseur T_c et le flux W_c pour un rapport de pression donné $\frac{p_c}{p_a}$, une température extérieure T_a donnée et une vitesse de rotation ω_c donnée. L'expression de C_c permet alors de définir le couple à exercer par le moteur du compresseur sur son arbre.

La seconde partie permet de modéliser le comportement dynamique du compresseur. Le principe fondamental de la dynamique appliqué à son axe de rotation et l'expression du couple du moteur du compresseur C_m en fonction des caractéristiques du moteur, permettent de déterminer la vitesse de rotation ω_c à partir de C_c et de la tension d'alimentation u_c.

Q 9. Compléter sur le document réponse DR3 la synthèse graphique du modèle du compresseur obtenu dans cette partie en plaçant $\frac{p_c}{p_a}, T_a, u_c, C_c, \omega_c, W_c, T_c$ sur le schéma.

Partie 2 Contrôle du système d'alimentation

Objectif : Élaborer la commande garantissant une alimentation en oxygène suffisante tout en minimisant les pertes énergétiques.

Simulation du système d'alimentation

Après avoir modélisé le compresseur ainsi que le reste des composants du système, le comportement de l'ensemble de la pile représenté Figure 7 est alors défini par les équations d'état suivantes :

$$\begin{cases} \dot{x} = \beta(x, u_c, I_p) \\ x = [m_{O_2}, m_{H_2}, m_{N_2}, \omega_c, p_c, m_c, m_e, p_s]^T \end{cases}$$

Tournez la page S.V.P.

Les différentes grandeurs dépendent du temps, cependant pour ne pas surcharger les notations la variable de temps ne sera pas notée et chacune des variables d'état traduit l'état à un instant t.

L'état du système dépend de $m_{O_2}, m_{H_2}, m_{N_2}, \omega_c, p_c, m_c, m_e, p_s$ respectivement la masse en oxygène, la masse en hydrogène, la masse en diazote, la vitesse de rotation du compresseur, la pression en sortie de compresseur, la masse d'air en sortie de compresseur, la masse d'eau à l'anode et la pression en sortie de la cathode. Ces grandeurs sont regroupées dans le vecteur d'état du système x.

Figure 7 : Représentation des entrées/sorties de la pile

Les mesures possibles sont le débit d'air à travers le compresseur W_c, la pression en sortie de compresseur p_c, et la tension de la pile u_p. Ces mesures sont stockées dans un vecteur $y = [W_c, p_c, u_p]^T$.

La puissance utile P_u est la différence entre la puissance produite par la pile et la puissance perdue nécessaire au fonctionnement des composants auxiliaires, majoritairement due au compresseur.

L'objectif du contrôle est de reconstituer l'oxygène appauvri à la cathode pendant la production d'énergie. Cette tâche doit être accomplie rapidement et efficacement pour prolonger la durée de vie de la pile. L'excès de débit d'air fourni à la pile est indiqué par le rapport d'excès d'oxygène λ_{O_2}, défini comme le rapport de l'oxygène fourni sur l'oxygène utilisé dans la cathode. Un rapport d'excès d'oxygène élevé améliore P_u, cependant, après avoir atteint une valeur optimale de λ_{O_2}, l'augmenter encore entraînera une augmentation excessive de la puissance du compresseur et diminuera ainsi la puissance utile. Des essais ont permis de déterminer qu'une régulation de λ_{O_2} à la valeur de 2 permet un bon compromis pour maximiser la puissance utile du système en garantissant une durée de vie satisfaisante de la pile.

Ainsi, les variables traduisant la performance du système sont la différence entre la puissance utile désirée et actuelle, $e_{P_u} = P_u^{ref} - P_u$ et le rapport en excès d'oxygène λ_{O_2}. Ces variables sont stockées dans le vecteur $z = [e_{P_u}, \lambda_{O_2}]^T$.

L'objectif de la commande est d'obtenir $e_{P_u} = 0$ et $\lambda_{O_2} = 2$ en commandant la tension d'entrée du compresseur u_c. Le courant est considéré comme une perturbation du système.

Le problème linéarisé autour du point de fonctionnement ($e_{P_{u_0}} = 0, \lambda_{O_{2_0}} = 2, I_{p_0} = 180A, u_{c_0} = 160V$) pour l'ensemble des composants s'écrit :

$$\begin{cases} \delta\dot{x} = A\delta x + B_u\delta u_c + B_i\delta I_p \\ \delta y = C_y\delta x + D_{yu}\delta u_c + D_{yi}\delta I_p \\ \delta z = C_z\delta x + D_{zu}\delta u_c + D_{zi}\delta I_p \end{cases} \quad (2)$$

où δx représente la variation de x par rapport à sa valeur au point de fonctionnement x_0.

Q 10. Compléter le tableau du document réponse DR4 avec les dimensions des différentes matrices $A, B_u, B_i, C_y, D_{yu}, D_{yi}, C_z, D_{zu}, D_{zi}$.

Le Tableau 2 donne les caractéristiques physiques d'un ensemble de microcontrôleurs.

Modèle	Architecture	Mémoire flash (ko)	RAM (o)	Vitesse CPU (MIPS)	Tensions de fonctionnement
AT89C51	Intel 8051 - 8 bits	4	128	12	4 à 6 V
ATmega328P	AVR - 8 bits	32	2048	20	1,8 à 5,5 V
ATmega2560	AVR - 8 bits	256	8192	16	1,8 à 5,5 V
PIC 18F4620	PIC - 8 bits	64	4096	10	2 à 5,5 V
PIC 24FJ128GA006	PIC - 16 bits	128	8192	16	2 à 3,6 V
STM32 L051C8	ARM - 32 bits	64	8192	32	1.65 à 3.6 V
STM32 F091VCT6	ARM - 32 bits	256	32768	48	2 à 3,6 V

Tableau 2 - Quelques microcontrôleurs et leurs caractéristiques

Q 11. Chacune des valeurs est stockée en un flottant double précision. Quelle est la taille totale des données en octets nécessaires pour stocker l'ensemble des matrices ? Quels microcontrôleurs peut-on alors employer dans un système embarqué permettant de traiter les données afin de contrôler l'alimentation ?

Afin de le résoudre de façon discrète par la suite, le problème est réécrit en regroupant les différents termes. Le vecteur ΔX comprend l'ensemble des valeurs du vecteur d'état δx, le vecteur ΔU regroupe l'ensemble des données d'entrée $\Delta U = [\delta u_c, \delta I_p]$ et le vecteur ΔY regroupe l'ensemble des valeurs des vecteurs δy et δz : $\Delta Y = \left[\delta y[0], \delta y[1], \delta y[2], \delta z[0], \delta z[1]\right]^{\mathrm{T}}$.

L'évolution de l'ensemble du vecteur d'état ΔX est ainsi exprimée en fonction des données d'entrée ΔU. Les données de sortie ΔY sont déterminées à partir du vecteur d'état ΔX et des données ΔU. Le système matriciel ainsi obtenu à partir du système (2) est donné dans l'équation (3).

$$\begin{cases} \Delta\dot{X} = A\Delta X + B\Delta U \\ \Delta Y = C\Delta X + D\Delta U \end{cases} \quad (3)$$

Q 12. Compléter la fonction Python `CalculeBCD` du document réponse DR5 permettant de construire les matrices B,C et D à partir des matrices $B_u, B_i, C_y, D_{yu}, D_{yi}, C_z, D_{zu}, D_{zi}$.

Une simulation est mise en place afin de déterminer l'évolution de ΔX et ΔY au cours du temps à partir d'une évolution de ΔU donnée. Le temps de la simulation évolue de $t_0 = 0.s$ à $t_{max} = 5.s$ avec un pas de temps $h = 1.e^{-4}$s.

Tournez la page S.V.P.

Q 13. Écrire la fonction Python `IniT` prenant en arguments t_0, t_{max} et h et retournant la liste des temps correspondants.

L'évolution ΔU correspond à une variation d'intensité de $\delta I_p = 20A$ à un instant $t_1 = 1.s$ soit $\delta I_p = 0\ \forall t \in [0, t_1[$ et $\delta I_p = 20A\ \forall t \in [t1, t_{max}[$. La consigne de variation de δu_c lors de la simulation est nulle pour toute la suite du sujet $\delta u_c = 0\ \forall t$.

Q 14. Écrire la fonction Python `IniU` prenant en argument $\delta I_p, t_1, t_{max}$ et h retournant la liste `ListeDeltaU` correspondant à cette consigne au cours de la simulation.

Connaissant pour un indice de temps k les valeurs de ΔX_k et ΔU_k on cherche à déterminer ΔX_{k+1}. Pour cela, la dérivée de $\Delta \dot{X}$ est approximée sur l'intervalle entre le temps à l'itération k et le temps à l'itération $k+1$ par la valeur de cette dérivée à l'instant k. Un schéma d'Euler explicite est ainsi obtenu $\Delta X_{k+1} = \Delta X_k + h * \Delta \dot{X}_k$.

Q 15. À partir du système d'équations (3) et en utilisant le schéma d'Euler explicite donner les expressions de E et F en fonction de A, B et de la période d'échantillonnage h permettant de résoudre le problème de façon itérative en utilisant les équations de récurrence (4). Écrire la fonction Python `CalculeEF` permettant de calculer E et F.

$$\begin{cases} \Delta X_{k+1} = E\Delta X_k + F\Delta U_k \\ \Delta Y_k = C\Delta X_k + D\Delta U_k \end{cases} \quad (4)$$

Q 16. En utilisant la formule de récurrence, définir la fonction Python `Simulation` qui prend comme argument les différentes matrices définies ci-dessus (A, B, C, D) et l'évolution des paramètres d'entrée au cours de la simulation `ListeDeltaU` afin de calculer la liste `ListeDeltaX` (resp `ListeDeltaY`) des valeurs de ΔX (resp ΔY) au cours du temps avec h le pas de temps et n le nombre de points correspondants.

Le résultat obtenu est illustré Figure 8.

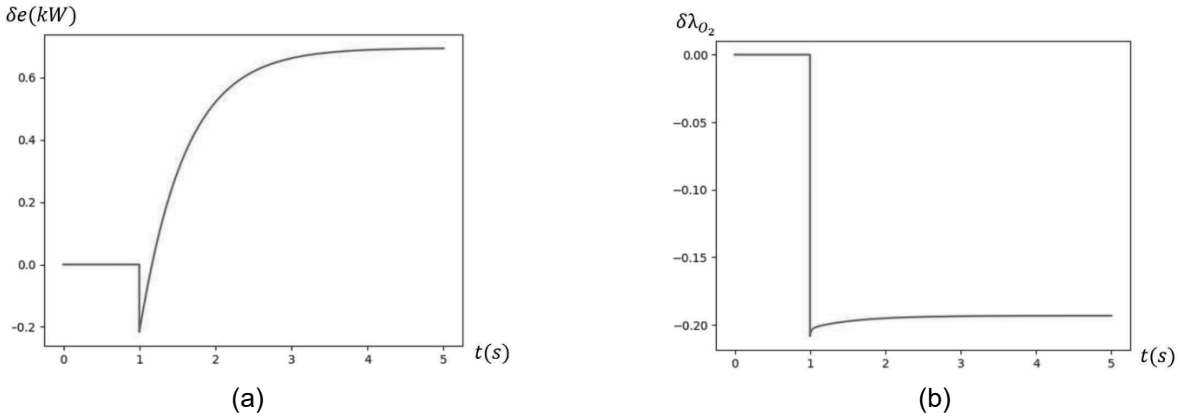


Figure 8 - (a) Évolution de la variation de puissance δe au cours du temps (b) Évolution de $\delta\lambda_{O_2}$ au cours du temps

Q 17. Conclure quant à l'impact d'une perturbation en intensité sur le système.

Boucle ouverte et régulation par anticipation

Le courant qui agit comme une perturbation est mesuré. En utilisant cette mesure, une correction par anticipation peut être mise en place. Cette anticipation est basée sur la recherche de la commande du compresseur qui permet d'obtenir le débit d'air garantissant le bon fonctionnement du système pour la demande de courant correspondante. Des essais expérimentaux sont utilisés afin de trouver la fonction $u_c = f_c(Ip)$ de la boucle d'anticipation qui à partir de l'effet de I_p sur u_c permet d'annuler l'effet de I_p sur $\delta\lambda_{O_2}$ à l'équilibre.

Figure 9 - Système en boucle ouverte avec anticipation

Q 18. Parmi les neuf matrices $A, B_u, B_i, C_y, D_{yu}, D_{yi}, C_z, D_{zu}, D_{zi}$ du système initial (2) quelles sont les matrices à modifier pour prendre en compte cette boucle ? Justifier votre réponse. On ne cherchera pas à donner la valeur des termes correspondants.

La fonction `Simulation` définie précédemment est alors appliquée aux nouvelles matrices A, B, C et D correspondantes avec le même vecteur `ListeDeltaU`. Le résultat obtenu est illustré Figure 10.

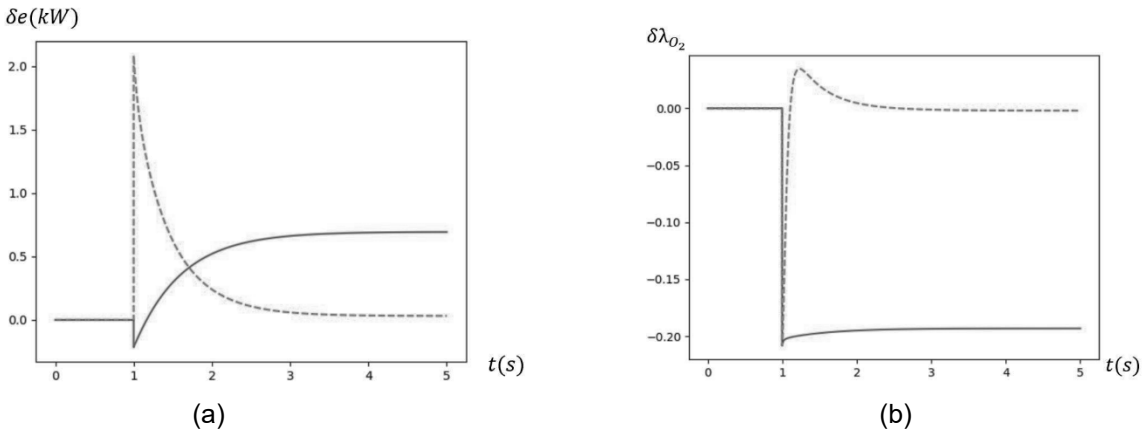


Figure 10 - (a) Évolution de la variation de puissance δe (sans anticipation en trait continu, avec anticipation en pointillés) (b) Évolution de $\delta\lambda_{O_2}$ au cours du temps (sans anticipation en trait continu, avec anticipation en pointillés)

Q 19. Conclure quant à l'intérêt de cette régulation par anticipation. Quels sont les points d'amélioration possibles ?

Tournez la page S.V.P.

Boucle de retour

Une boucle de retour est alors mise en place en utilisant un régulateur quadratique linéaire (LQR).

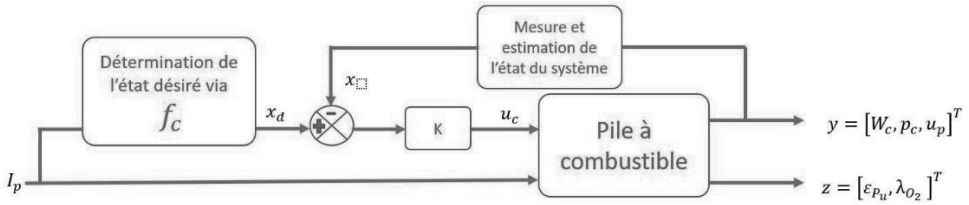


Figure 11 - Boucle de retour

Pour faciliter la lecture les indices Python [i,j] seront utilisés dans les expressions mathématiques suivantes.

L'objectif est de minimiser $\delta\lambda_{O_2} = \delta z_{[1,0]}$ sans utiliser d'entrée excessive. Une fonction de coût est alors mise en place avec les deux matrices de pondération de $\delta z_{[1,0]}$ et δu_c désignées respectivement par Q et R (le choix des valeurs de pondérations de Q et R expriment les préférences du concepteur en termes de contrôle) :

$$Coût = \int_0^\infty \delta z_{[1,0]}^T Q \delta z_{[1,0]} + \delta u_c^T R \delta u_c dt$$

Cependant la perturbation δI_p et la commande δu_c interviennent dans le calcul de $\delta z_{[1,0]}$:

$$\delta z_{[1,0]} = C_{z[1,:]}\delta x + D_{zu[1,0]}\delta u_c + D_{zi[1,0]}\delta I_p$$

Or pour résoudre un problème avec la méthode LQR il est nécessaire de séparer les variables d'état et de contrôle. On pose alors $\delta z'_{[1,0]} = C_{z[1,:]}\delta x$ comme approximation de $\delta z_{[1,0]}$ afin de l'insérer dans la fonction de coût :

$$Coût = \int_0^\infty \left(\delta z'_{[1,0]}{}^T Q \delta z'_{[1,0]} + \delta u_c^T R \delta u_c\right) dt$$

$$= \int_0^\infty \left(\delta x^T C_{[1,:]}^T Q C_{z[1,:]} \delta x + \delta u_c^T R \delta u_c\right) dt$$

Le contrôle optimal δu qui minimise la fonction de coût est alors donné par K le régulateur linéaire quadratique tel que :

$$\delta u_c = -K(\delta x - \delta x_d) \quad \text{avec} \quad K = R^{-1}B^T P$$

où P est la solution de l'équation algébrique de Riccati (ARE) : $PA + A^T P + Q_x - PBR^{-1}B^T P = 0$ avec $Q_x = C_{[1,:]}^T Q C_{z[1,:]}$, Q étant ici une matrice $1 * 1$ avec comme unique composante la valeur du correcteur C que l'on choisit. La variable δx_d est la valeur calculée de l'état désiré du système qui permet d'obtenir $\delta z_{[1,0]} = 0$ en régime permanent pour un δI_p donné.

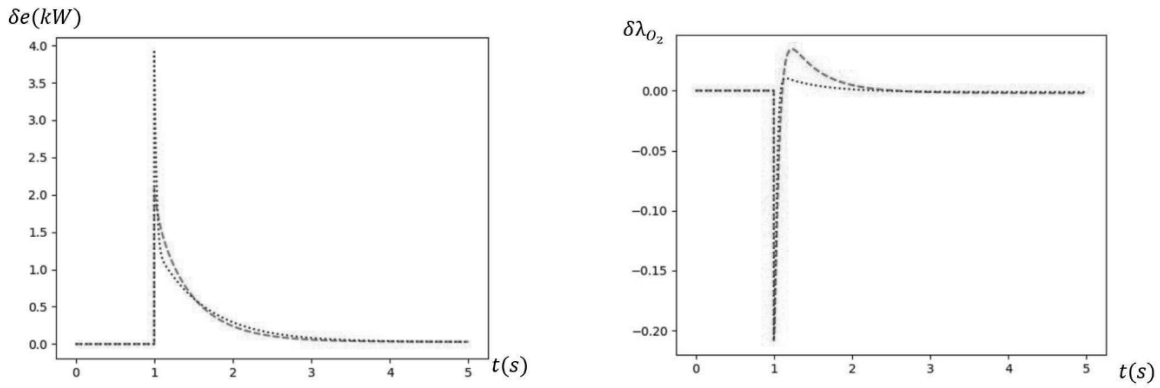
Q 20. Donner les lignes de code permettant de déterminer la matrice P en utilisant `scipy.linalg.solve_continuous_are` (dont l'API est donnée en annexe 2) permettant de résoudre l'équation algébrique de Riccati.

Q 21. En utilisant le système matriciel initial (2) en régime permanent, donner l'expression des matrices G et H du système matriciel $G\begin{bmatrix}\delta x_d \\ \delta u_d\end{bmatrix} + H\delta I_p = 0$ permettant de déterminer δx_d. Ecrire la fonction Python `assemblage` prenant en argument les différentes matrices $A, B_u, B_i, C_z, D_{zu}, D_{zi}$ afin de calculer H et G.

Q 22. Définir une fonction Python `SimulationLQR` qui prend comme argument les différentes matrices $A, B_u, B_i, C_y, D_{yu}, D_{yi}, C_z, D_{zu}, D_{zi}$ (dans lesquelles les termes ont été modifiés pour prendre en compte la boucle d'anticipation conformément à la question 18) et l'évolution des paramètres d'entrée au cours de la simulation `ListeDeltaU` afin de calculer la liste des valeurs de `ListeDeltaY` au cours du temps avec h le pas de temps et n le nombre de points correspondants. La fonction utilisera au maximum les fonctions définies dans les questions précédentes.

La simulation avec cette nouvelle boucle de retour donne alors les résultats illustrés

Figure 12.



(a) (b)

Figure 12 - (a) Évolution de la variation de puissance δe (avec anticipation tracée en pointillés, avec anticipation et boucle de retour tracé avec des points) (b) Évolution de $\delta\lambda_{O_2}$ au cours du temps (avec anticipation tracée en pointillés, avec anticipation et boucle de retour tracé avec des points)

Q 23. Conclure quant à l'apport de cette boucle. Sur quels paramètres peut-on agir pour régler le comportement du système désiré ?

Partie 3 Modélisation des transferts de données d'une pile à combustible par un « Synchronous Dataflow Graph »

Objectif: Modéliser les transferts de données d'une pile à combustible par un formalisme classique.

Un « Synchonous Dataflow Graph » (SDF en abrégé) est un graphe orienté $G = (V, E)$ défini de la manière suivante :

— chaque sommet $i \in V$est associé à un processus et possède un nom complet, un acronyme et une période exprimée en millisecondes ;

— chaque arc $(i,j) \in E$ est associé à un échange de données du sommet i au sommet j qui s'effectue au moyen d'une mémoire partagée. Il est associé à un nom.

Le SDF associé à la pile à combustible est constitué de 7 processus. Il est stocké sous la forme du fichier `pileACombustible.xml` donné en annexe 3. Par exemple, le processus `AirBlock` a pour acronyme AIR et une période de 60ms.

On observe que chaque arc (i,j) est associé à un écrivain (write) qui correspond à i et un lecteur (read) qui correspond à j. Par exemple, l'arc Arc_1 a pour sommet origine (écrivain) PM et destination (lecteur) H2.

Q 24. Lister les noms des 7 processus avec leur acronyme et leur période associée.

Q 25. Représenter le SDF associé à la pile à combustible sous la forme d'un graphe orienté : les sommets de ce graphe sont étiquetés par leur acronyme, les arcs par leur nom.

Pour construire un graphe orienté à partir du fichier xml, nous allons utiliser deux bibliothèques standards Python.

- La récupération des informations du fichier xml est réalisée en utilisant l'API Python `xml.etree.ElementTree`. Cette bibliothèque permet de stocker les informations contenues dans un fichier xml sous la forme d'un arbre. Elle propose aussi des fonctions qui permettent de parcourir cet arbre, comme illustré par les codes suivants.
- La création et la manipulation du SDF se fera en utilisant la bibliothèque `NetworkX` donnée en annexe 4 qui permet de gérer des graphes.

Soit la fonction `AfficheIncidentsArcs` définie de la manière suivante :

```
def AfficheIncidentsArcs(root, nomArc):
    for child in root:
        for demiArc in child:
            if (demiArc.get("name") == nomArc):
                if (demiArc.get("acces") == "write"):
                    print("Le sommet entrant de l'arc " + nomArc + " est "
+ child.get("acronyme"))
                elif (demiArc.get("acces") == "read"):
                    print("Le sommet sortant de l'arc " + nomArc + " est "
+ child.get("acronyme"))
```

On considère alors la suite d'instructions suivantes :

```
import xml.etree.ElementTree as ET
tree = ET.parse( "PileACombustion.xml" )
root = tree.getroot()
AfficheIncidentsArcs(root,"Arc_5")
```

La racine `root` de l'arbre obtenu a 7 fils correspondant aux processus. Les fils de chacun de ces nœuds correspondent aux arcs qui ont le processus correspondant comme extrémité. Par exemple, le nœud correspondant au processus AirBlock à deux fils associés respectivement aux arcs Arc_5 et Arc_6.

Q 26. Donner les affichages obtenus pour la suite d'instructions considérée précédemment.

Soit la fonction `ConstruitSommetsGraphe` qui retourne à partir de l'arbre de racine `root` un graphe orienté sans arc, de sommets les processus étiquetés par leur acronyme et leur période.

Q 27. Donner le code Python de la fonction `ConstruitSommetsGraphe`. Quelle est la complexité de cette fonction ? Justifier votre réponse. On suppose que le rajout d'un sommet à un graphe est en Θ(1).

On souhaite maintenant construire les arcs du graphe. Pour cela, on construit dans un premier temps les listes `ArcIn` et `ArcOut` définies de la manière suivante :

- `ArcIn` contient les couples (u, i) où u est le nom d'un arc et i l'acronyme du processus en entrée.
- `ArcOut` contient les couples (u, i) où u est le nom d'un arc et i l'acronyme du processus en sortie.

Q 28. Donner les 5 premiers éléments des listes ArcIn et ArcOut obtenues dans l'ordre correspondant au fichier `pileACombustible.xml` fourni pour la pile à combustible.

Q 29. Écrire le code de la fonction `ConstruitListeArcs` qui retourne à partir de l'arbre de racine `root` les listes `ArcIn` et `ArcOut`. Quelle est la complexité de cette fonction ? Justifier votre réponse.

Q 30. Écrire la fonction `ConstruitArcs` qui construit les arcs du SDF `mySDF` à partir de l'arbre de racine `root`. Aucun cas d'erreur n'est à considérer. Évaluer la complexité de cette fonction. Justifier votre réponse. On suppose qu'ajouter un arc au graphe est en Θ(1).

On considère par la suite un SDF défini par un graphe orienté $G = (V, E)$ quelconque et une période T_i entière associée à chaque processus $i \in V$.

Pour tout entier $\nu > 0$ et tout processus $i \in V$, le couple $<i, \nu>$ désigne la νième exécution du processus i. On suppose que cette exécution démarre à l'instant $S_i(\nu) = T_i \times (\nu - 1)$ et se termine à $S_i(\nu) + T_i = T_i \times \nu$.

Tout arc $(i, j) \in E$ est associé à un échange d'information du processus i vers le processus j associé à une mémoire partagée notée b_{ij}. Les instants de lecture et d'écriture dans cette mémoire suivent le protocole LET (Logical Execution Time) défini de la manière suivante :

- chaque exécution de i met à jour une nouvelle donnée dans la mémoire b_{ij} une fois cette exécution terminée ; ainsi, à tout instant $t = S_i(\nu) + T_i$ pour $\nu > 0$, le processus i écrit une donnée dans la mémoire b_{ij}, écrasant ainsi la donnée écrite précédemment ;
- chaque exécution de j lit la valeur stockée dans b_{ij} au démarrage de son exécution ; ainsi, à tout instant $t = S_j(\nu)$ pour $\nu > 0$, le processus j lit la donnée stockée dans la mémoire b_{ij}.

Soit un arc $e = (i,j) \in E$. Il existe une dépendance de $< i, \nu_i >$ vers $< j, \nu_j >$ pour deux entiers $(\nu_i, \nu_j) \in (N - \{0\})^2$ si l'exécution $< j, \nu_j >$ récupère dans b_{ij} une donnée émise par $< i, \nu_i >$. On note alors $< i, \nu_i > \rightarrow < j, \nu_j >$.

Q 31. On considère dans cette question les deux processus PM et SS numérotés 1 et 2 de période respective $T_1 = 30ms$ et $T_2 = 20ms$ et la mémoire associée à Arc_2 noté b_{12} qui permet au processus 1 de communiquer des données au processus 2.

- **pour tout entier $\nu > 0$, exprimer les suites $S_1(\nu)$ et $S_2(\nu)$;**
- **donner pour chaque exécution $< 2, \nu_2 >$ avec $\nu_2 \in \{1, \dots, 7\}$, l'exécution du processus 1 $< 1, \nu_1 >$ telle que il y a une dépendance de $< 1, \nu_1 >$ vers $< 2, \nu_2 >$ du fait de b_{12} si cela est possible. Indication : les exécutions $< 2,1 >$ et $< 2,2 >$ sont effectuées respectivement aux instants 0 et $20ms$, il n'y a donc pas de dépendance car $< 1,1 >$ se termine à $t = 30ms$. Par contre, $< 2,3 >$ démarre à l'instant $t = 40ms$, et donc récupère la donnée émise par $< 1,1 >$, d'où $< 1,1 > \rightarrow < 2,3 >$.**

Soit un arc $e = (i,j) \in E$ et $D_e = \{(\nu_i, \nu_j) \in (N - \{0\})^2, < i, \nu_i > \rightarrow < j, \nu_j >\}$. On admet que le couple $(\nu_i, \nu_j) \in D_e$ si :

- l'exécution $< j, \nu_j >$ démarre à la fin de l'exécution de $< i, \nu_i >$ ou après ;
- l'exécution de $< i, \nu_i + 1 >$ se termine strictement après le démarrage de $< j, \nu_j >$.

Q 32. On cherche dans cette question à caractériser l'ensemble D_e.

- **démontrer que $(\nu_i, \nu_j) \in D_e$ si et seulement si $S_i(\nu_i) + T_i \leq S_j(\nu_j) < S_i(\nu_i + 1) + T_i$;**
- **en déduire que $(\nu_i, \nu_j) \in D_e$ si et seulement si $T_j \leq T_j \nu_j - T_i \nu_i < T_i + T_j$;**
- **quelle est l'inégalité associée aux processus 1 et 2 de la question précédente ? Vérifier que les dépendances obtenues dans la question précédente vérifient cette inégalité.**

Q 33. Soit un arc $e = (i,j) \in E$. On pose $K_{ij} = \text{ppcm}(T_i, T_j)$ le plus petit commun multiple de T_i et T_j soit l'entier positif le plus petit divisible à la fois par T_i et T_j. On pose également $n_i = \frac{K_{ij}}{T_i}$ et $n_j = \frac{K_{ij}}{T_j}$.

- **démontrer que, pour tout couple $(v_i, v_j) \in D_e$ et tout entier relatif $p \in Z$, $T_j(v_j + pn_i) - T_i(v_i + pn_j) = T_j v_j - T_i v_i$;**
- **soient alors un couple $(v_i, v_j) \in N - \{0\} \times N - \{0\}$ et une valeur $p \in Z$ telles que $v_i + pn_i > 0$ et $v_j + pn_j > 0$. Déduire de la question précédente que $(v_i, v_j) \in D_e$ si et seulement si $(v_i + p \times n_i, v_j + p \times n_j) \in D_e$;**
- **soient alors les ensembles $\underline{D_e} = D_e \cap (\{1, .., n_i\} \times N - \{0\} \cup N - \{0\} \times \{1, .., n_j\})$ et $D'_e = \{(v_i^0 + pn_i, v_j^0 + pn_j), (v_i^0, v_j^0) \in \underline{D_e}, p \in N\}$. Démontrer que $D_e = \underline{D_e}$;**
- **application numérique : On considère dans cette sous-question les processus PM et SS numérotés 1 et 2 de période respective $T_1 = 30ms$ et $T_2 = 20ms$ et la mémoire associée à l'Arc_2 noté b_{12}. Que valent K_{12}, n_1 et n_2 ? Que vaut $\underline{D_e}$? Caractériser complètement D_e.**

Partie 4 Évaluation de la durée de vie des valeurs échangées et de la latence last-to-first de la pile à combustible

Objectif : Concevoir un algorithme pour calculer la durée de vie maximale d'une valeur échangée et la latence last-to-first de la pile à combustible.

Le « Synchronous Dataflow Graph » (SDF) mis en place dans la partie 3 va être utilisé ici afin d'évaluer sans simulation deux mesures importantes pour une application : la durée de vie maximale d'une valeur, et la latence last-to-first qui est une mesure du temps de réaction du système entre une entrée et la sortie correspondante.

Dans cette partie, on considère que $G = (V, E)$ est un SDF. Soit $e = (i, j)$ un arc de G. La durée de vie d'une valeur (ou la fraîcheur d'une valeur) stockée dans la mémoire b_{ij} entre les exécutions < i, v_i> et < j, v_j> avec $(v_i, v_j) \in D_e$ est définie par $\delta_e(v_i, v_j) = S_j(v_j) - (S_i(v_i) + T_i)$. Par extension, la durée de vie maximale d'une valeur (ou la fraîcheur d'une valeur) stockée dans la mémoire b_{ij} notée $\widehat{\delta_e}$ est le laps de temps maximum entre l'écriture de la donnée dans b_{ij} par une exécution du processus i et sa lecture par une exécution du processus j. Ainsi, $\widehat{\delta_e} = max_{(v_i, v_j) \in D_e}\left(\delta_e(v_i, v_j)\right)$.

Q 34. On considère dans cette question les processus PM et SS numérotés 1 et 2 de période $T_1 = 30ms$ et $T_2 = 20ms$ et la mémoire b_{12} associée à l'Arc_2 qui permet au processus 1 de communiquer des données au processus 2.

- **montrer que $\delta_e(v_1, v_2) = 20v_2 - 30v_1 - 20$ pour $e = (1,2)$;**
- **que vaut $\widehat{\delta}_e$ pour $e = (1,2)$? Justifier votre réponse.**

On admet que l'équation Diophantine $ax + by = 1$ où a et b sont deux entiers naturels non nuls premiers entre eux admet au moins une solution $(x_0, y_0) \in Z^2$.

Q 35. On souhaite démontrer que pour tout arc $e = (i,j) \in E$, $\widehat{\delta}_e = T_i - pgcd(T_i, T_j)$. On rappelle que le pgcd de deux entiers positifs est le plus grand diviseur commun de ces deux entiers.

- **démontrer que la suite $(x_n, y_n) = (x_0 + n \times b, -y_0 + n \times a)$ vérifie l'équation $ax - by = 1$. Le couple$(x_0, y_0) \in Z^2$ est ici solution de l'équation Diophantine $ax + by = 1$;**
- **démontrer que $\widehat{\delta}_e \in \{0, 1, .., T_i - pgcd(T_i, T_j)\}$;**
- **déduire des deux sous-questions précédentes que $\widehat{\delta}_e = T_i - pgcd(T_i, T_j)$.**

Q 36. On souhaite dans cette question calculer la durée maximum de vie sur l'ensemble des mémoires associées au SDF de la pile, soit $\widehat{\delta}_e(G) = max_{e \in E} \widehat{\delta}_e$.

- **calculer $\widehat{\delta}_e$ pour les arcs $e \in E$ du SDF G associé à la pile à combustible. En déduire $\widehat{\delta}_e(G)$ pour le SDF G associé à la pile à combustible ;**
- **compléter le code de la fonction suivante `CalculDureeDeVie` qui renvoie la durée de vie maximum du graphe `mySDF` décrit dans la partie 3. Vous pourrez utiliser la fonction `gcd` de la bibliothèque `numpy` dont la syntaxe est rappelée en annexe 5 ;**

```
def CalculDureeDeVie(mySDF):
    deltaEtoile = 0
    for e in list(mySDF.edges):
        Ti=int(mySDF.nodes[e[0]]['periode'])
        Tj=int(mySDF.nodes[e[1]]['periode'])
        ...
        ...
    return ...
```

- **évaluer la complexité de la fonction en supposant que chaque accès à un sommet ou à un arc se fait en Θ(1). Justifier votre réponse.**

On étend la notion de dépendance aux chemins d'un SDF de la manière suivante : on dit qu'il existe un dépendance de l'exécution $<s, v_s>$ vers $< i, v_i>$ si il existe un chemin du SDF constitué par la suite de sommets $u_1 = s, u_2, .., u_{k-1}, u_k = i$ et la suite d'entiers positifs $v_1 = v_s, v_2, \ldots, v_{k-1}, v_k = v_i$ avec $<s, v_s > \rightarrow < u_2, v_2>$, $<u_2, v_2 > \rightarrow < u_3, v_3>$...et $<u_{k-1}, v_{k-1} > \rightarrow < i, v_i>$. On note alors $<s, v_s > \rightarrow < i, v_i>$.

Par exemple, on considère le SDF G constitué des 3 sommets PM, SS et CS numérotés respectivement 1, 2 et 3 de période respective $T_1 = 30ms$, $T_2 = 20ms$ et $T_3 = 50ms$ et des arcs (1,2) et (2,3).

- on a la dépendance $< 1,2 > \rightarrow < 2,5>$, car $20 \times 5 - 30 \times 2 = 40 \in \{20, \ldots, 40\}$;

- on a également < 2,5 >→< 3,3>, car $50 \times 3 - 20 \times 5 = 50 \in \{50, \ldots, 60\}$.

On en déduit donc la dépendance < 1,2 >→< 3,3>.

Q 37. Montrer que, pour l'exemple ci-dessus, $< \mathbf{1}, \mathbf{4} > \rightarrow < \mathbf{3}, \mathbf{4} >$.

Soit alors $\mu = u_1, u_2, \ldots, u_k$ un chemin élémentaire de G, i.e. qui ne passe pas deux fois par le même sommet. Il ne contient donc pas de circuit. Supposons également la dépendance $<u_1, v_1 > \rightarrow < u_k, v_k>$ associé à μ comme défini dans la question précédente.

On pose $L_\mu(v_1, v_k) = S_{u_k}(v_k) + T_{u_k} - \left(S_{u_1}(v_1) + T_{u_1}\right)$. On note également l'ensemble $D_\mu = \{(v_1, v_k) \in (N - \{0\})^2, < u_1, v_1 > \rightarrow < u_k, v_k >\}$ par extension de D_e, $e \in E$

La latence last-to-first du chemin μ est le temps maximal de transmission d'une donnée depuis la fin d'une exécution de u_1 jusqu'à la fin d'une exécution de u_k qui sont en dépendance. Formellement, cette valeur est définie par $\hat{L}_\mu = max_{(v_1, v_k) \in D\mu}\left(L_\mu(v_1, v_k)\right)$.

Cette valeur est importante à évaluer pour les concepteurs d'une application. En effet, imaginons que le premier sommet u_1 de μ soit associé au processus de supervision de la pile CSM, et le dernier u_k au processus garantissant l'humidification en air HU ; $\hat{L}_\mu$correspond au temps requis garanti pour qu'une donnée émise par le u_1 permette une réaction de u_k. Si l'on souhaite garantir une bonne humidification de la membrane en permanence il est souhaitable que cette latence soit minimum.

Q 38. Montrer que $L_\mu(v_1, v_k) = T_{u_k} v_k - T_{u_1} v_1$. Application numérique : calculer pour l'exemple de la question précédente les valeurs $L_\mu(2, 3)$ et $L_\mu(4, 4)$ pour $\mu = 1, 2, 3$. Ces deux valeurs sont-elles égales ?

Q 39. Soit $\mu = u_1, u_2, \ldots, u_k$ un chemin élémentaire de G.

- **on suppose tout d'abord que μ est un chemin de deux sommets : $k = 2$ et $\mu = u_1, u_2$. Démontrer que $L_\mu(v_1, v_2) = \delta_{e_1}(v_1, v_2) + T_{u_2}$ pour $e_1 = (u_1, u_2)$;**

- **on suppose maintenant que $k > 2$. On pose alors l'arc $e_{k-1} = (u_{k-1}, u_k)$ et les chemins $\mu_k = u_1, \ldots, u_k$ et $\mu_{k-1} = u_1, \ldots, u_{k-1}$. Démontrer que $L_{\mu_k}(v_1, v_k) = \delta_{e_{k-1}}(v_{k-1}, v_k) + T_{u_k} + L_{\mu_{k-1}}(v_1, v_{k-1})$;**

- **on considère dans cette sous-question une suite w_α pour $\alpha \in \{2, \ldots, k\}$ et la suite v_α qui vérifie $v_\alpha = w_\alpha + v_{\alpha-1}$ pour $\alpha \geq 2$ et $v_1 = 0$. Exprimer v_α pour $\alpha \in \{2, \ldots, k\}$ en fonction de la suite w. La démonstration de ce résultat n'est pas requise ;**

- **déduire des questions précédentes que $L_{\mu_k}(v_1, v_k) = \sum_{\alpha=2}^{k}\left(T_{u_\alpha} + \delta_{e_{\alpha-1}}(v_{\alpha-1}, v_\alpha)\right)$.**

Q 40. On souhaite dans cette question, calculer une borne supérieure de la latence last-to-first d'un chemin. Soit $\mu = u_1, u_2, \ldots, u_k$ un chemin élémentaire de *G.*

- **démontrer que $\hat{L}_\mu \leq \sum_{\alpha=2}^{k} T_{u_\alpha} + \sum_{\alpha=2}^{k} \hat{\delta}_{e_{\alpha-1}}$. On rappelle que $e_{\alpha-1} = (u_{\alpha-1}, u_\alpha)$;**
- **en déduire que $\hat{L}_\mu \leq \sum_{\alpha=2}^{k} \left(T_{u_\alpha} + T_{u_{\alpha-1}} - pgcd(T_{u_\alpha}, T_{u_{\alpha-1}}) \right)$;**
- **calculer cette borne supérieure pour le chemin $\mu = 1, 2, 3$ avec les périodes $T_1 = 30ms$, $T_2 = 20ms$ et $T_3 = 50ms$.**

Q 41. On rappelle qu'un graphe est fortement connexe si, pour tout couple de sommets $(u, v) \in V^2$, il existe un chemin de u à v. Une source est un sommet $s \in V$ tel que, pour tout sommet $u \in V$, il existe un chemin de s à u.

- **quelles sont les sources d'un graphe fortement connexe ? Justifier votre réponse ;**
- **est-ce que le SDF associé à la pile à combustible est fortement connexe ? Aucune justification n'est demandée. Quelles sont ses sources ?**

Soit s une source d'un SDF. La latence last-to-first depuis s est le temps maximum de diffusion d'une donnée depuis s à tous les autres sommets du graphe. Plus formellement, si M_s est l'ensemble des chemins élémentaires de sommet de départ s alors la latence last-to-first depuis s est définie par $\hat{L}_s = max_{\mu \in M_s} \hat{L}_\mu$.

On suppose d'autre part dans cette question et la suivante que le graphe considéré $G' = (V, E)$ est sans circuit et de source s. On suppose que tous les arcs $e = (i, j) \in E$ de G' sont valués par $v'(e) = T_i + T_j - pgcd(T_i, T_j)$.

On souhaite programmer le calcul de la borne supérieure définie dans la question 40. L'idée est ici de transformer le graphe G' en un graphe G'' de même structure, mais dont les arcs seront valués par une valeur v'', de sorte à obtenir la latence en appliquant un algorithme de plus court chemin (Bellmann-Ford ici).

On considère ainsi les deux fonctions Python incomplètes suivantes.

- la fonction `ConstruitGSeconde` retourne G'' à partir de G' ;
- la fonction `CalculBorneLatence` retourne la borne supérieure de la latence à partir de G'. Cette fonction appelle `ConstruitGSeconde` et lui applique l'algorithme de Bellmann-Ford.

```
def ConstruitGSeconde(GPrime):
    GSeconde = nx.DiGraph()
    GSeconde.add_nodes_from(GPrime)
    for e in list(GPrime.edges):
        Ti=GPrime.nodes[e[0]]['periode']
        Tj=GPrime.nodes[e[1]]['periode']
        GSeconde.add_edge(e[0],e[1],valeur =...)
    return GSeconde
```

```
def CalculBorneLatence(GPrime):
    if (not nx.is_directed_acyclic_graph(GPrime)):
        return False
    GSeconde = ConstruitGSeconde(GPrime)
    valeursChemins,chemins = nx.single_source_bellman_ford(GSeconde, "PSS",
weight ='valeur')
    ValeurI = min(valeursChemins.values())
    return ...
```

Q 42. Pourquoi ne peut-on pas utiliser l'algorithme de Dijskstra pour calculer les plus courts chemins de G''? Donner les instructions manquantes pour ces deux fonctions.

Q 43. On considère dans cette question le graphe G' extrait de la pile à combustible défini par l'ensemble de sommets $V' = \{PSS, SS, CS, CSM, PM, H2, AIR\} = V$ et l'ensemble des arcs $E' = \{(PSS, SS), (SS, CS), (SS, CSM), (SS, PM), (PM, H2), (PM, AIR)\}$. De plus, $s = PSS$.

- **représenter graphiquement G' sous la forme d'un graphe orienté. Est-ce que G' est sans circuit ? Quelle est la source s de G' ? Aucune justification n'est demandée ;**
- **calculer $v'(e)$ pour les arcs $e \in E'$;**
- **calculer une borne supérieure de la latence last-to-first de G' depuis la source s. Quel est le chemin du graphe G' associé ?**
- **que peut-on en déduire sur le fonctionnement de la pile à combustible ?**

Tournez la page S.V.P.

Bibliographie

Pukrushpan, Jay & Stefanopoulou, Anna & Peng, Huei. (2004). Control of Fuel Cell Power Systems : Principles, Modeling, Analysis, And Feedback Design. Book. Springer. 10.1007/978-1-4471-3792-4

Pukrushpan, Jay & Stefanopoulou, Anna & Peng, Huei. (2002). Modeling and Control for PEM Fuel Cell Stack System. Proceedings of the American Control Conference. 4. 3117 - 3122 vol.4. 10.1109/ACC.2002.1025268.

Hassanaly, Nadine (2009). *Modèle dynamique du système d'alimentation d'air pour le contrôle d'une pile à combustible H2/air de type PEM.* Mémoire. Trois-Rivières, Université du Québec à Trois-Rivières.

Pukrushpan, Jay & Peng, Huei & Stefanopoulou, Anna. (2002). Simulation and Analysis of Transient Fuel Cell System Performance Based on a Dynamic Reactant Flow Model. ASME Int. Mech. Eng. Congress Expo.. 2. 10.1115/IMECE2002-32051.

Niknezhadi, Ali & Allué-Fantova, Miguel & Kunusch, C. & Ocampo-Martinez, Carlos. (2011). Design and implementation of LQR/LQG strategies for oxygen stoichiometry control in PEM fuel cells based systems. Journal of Power Sources. 196. 4277-4282. 10.1016/j.jpowsour.2010.11.059.

Moraal, P.E. & Kolmanovsky, Ilya. (1999). Turbocharger Modeling for Automotive Control Applications. SAE Trans.. 108. 10.4271/1999-01-0908.

Chen, Fengxiang & Yu, Yang & Chen, J.. (2017). Control System Design of Power Tracking for PEM Fuel Cell Automotive Application. Fuel Cells. 17. 10.1002/fuce.201600240.

Cunningham, Joshua & Man, M. & Moore, Robert & Friedman, David. (1999). Requirements for a Flexible and Realistic Air Supply Model for Incorporation into a Fuel Cell Vehicle (FCV) System Simulation. 10.4271/1999-01-2912.

E. A. Lee & D. G. Messerschmitt, Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing, in IEEE Transactions on Computers, vol. C-36, no. 1, pp. 24-35, Jan. 1987, doi: 10.1109/TC.1987.5009446.

Kirsch C.M., Sokolova A. (2012) The Logical Execution Time Paradigm. In: Chakraborty S., Eberspächer J. (eds) Advances in Real-Time Systems. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-24349-3.

Feiertag, N., Richter, K., Nordlander, J., & Jönsson, J. (2008). A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. *RTSS 2009*.

Alix Munier Kordon & Ning Tang. Evaluation of the Age Latency of a Real-Time Communicating System using the LET paradigm. ECRTS 2020, Jul 2020, Modena, Italy.

Annexes

Tournez la page S.V.P.

Annexe 1

Bibliothèque Numpy

An ndarray is a (usually fixed-size) multidimensional container of items of the same type and size. The number of dimensions and items in an array is defined by its shape, which is a tuple of N non-negative integers that specify the sizes of each dimension. The type of items in the array is specified by a separate data-type object (dtype), one of which is associated with each ndarray.

Indexing

ndarrays can be indexed using the standard Python x[obj] syntax, where x is the array and obj the selection. There are three kinds of indexing available: field access, basic slicing, advanced indexing. Which one occurs depends on obj.

Exemple

```
>>> a=np.eye(5,5)
>>> print a
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
```

```
>>> a[:3,:4]=4
>>> a[::3,::4]=5
>>> a[:,3]=6
>>> print a
[[ 5.  4.  4.  6.  5.]
 [ 4.  4.  4.  6.  0.]
 [ 4.  4.  4.  6.  0.]
 [ 5.  0.  0.  6.  5.]
 [ 0.  0.  0.  6.  1.]]
```

numpy.array(object)
Create an array.
Parameters
Object [array_like] : an array, any object exposing the array interface, an object whose __array__ method returns an array, or any (nested) sequence.
Returns : an array object satisfying the specified requirements.
Exemple

```
>>> numpy.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
```

numpy.zeros(shape, dtype=float)
Return a new array of given shape and type, filled with zeros.
Parameters
shape [int or tuple of ints] : shape of the new array, e.g., (2, 3) or 2.
dtype [data-type, optional] : the desired data-type for the array, e.g., numpy.int8. Default is numpy.float64.
Returns : array of zeros with the given shape and dtype.
Exemples

```
>>> numpy.zeros((5,), dtype=int)
array([0, 0, 0, 0, 0])
>>> np.zeros((2, 1))
array([[ 0.],
       [ 0.]])
```

ndarray.transpose( )
Returns a view of the array with axes transposed. For a 2-D array, this is a standard matrix transpose.
Returns : View of a, with axes suitably permuted.
Exemple

```
>>> a = numpy.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.transpose()
array([[1, 3],
       [2, 4]])
```

numpy.matmul(x1,x2)
Matrix product of two arrays.
Parameters
x1, x2 : array_like, input arrays, scalars not allowed.
Returns: y ndarray, the matrix product of the inputs. This is a scalar only when both x1, x2 are 1-d vectors.
Exemple

```
>>> a = numpy.array([[1, 0],
...                  [0, 1]])
>>> b = numpy.array([[4, 1],
...                  [2, 2]])
>>> numpy.matmul(a, b)
array([[4, 1],
       [2, 2]])
```


Tournez la page S.V.P.

scipy.linalg.solve_continuous_are

`scipy.linalg.solve_continuous_are(a, b, q, r, e=None, s=None, balanced=True)` [source]

Solves the continuous-time algebraic Riccati equation (CARE).

The CARE is defined as

$$XA + A^H X - XBR^{-1}B^H X + Q = 0$$

The limitations for a solution to exist are :

- All eigenvalues of A on the right half plane, should be controllable.
- The associated hamiltonian pencil (See Notes), should have eigenvalues sufficiently away from the imaginary axis.

Moreover, if `e` or `s` is not precisely `None`, then the generalized version of CARE

$$E^H XA + A^H XE - (E^H XB + S)R^{-1}(B^H XE + S^H) + Q = 0$$

is solved. When omitted, `e` is assumed to be the identity and `s` is assumed to be the zero matrix with sizes compatible with `a` and `b`, respectively.

Parameters:

a : *(M, M) array_like*
Square matrix

b : *(M, N) array_like*
Input

q : *(M, M) array_like*
Input

r : *(N, N) array_like*
Nonsingular square matrix

e : *(M, M) array_like, optional*
Nonsingular square matrix

s : *(M, N) array_like, optional*
Input

balanced : *bool, optional*
The boolean that indicates whether a balancing step is performed on the data. The default is set to True.

Returns:

x : *(M, M) ndarray*
Solution to the continuous-time algebraic Riccati equation.

Raises:

LinAlgError
For cases where the stable subspace of the pencil could not be isolated. See Notes section and the references for details.

Notes

The equation is solved by forming the extended hamiltonian matrix pencil, as described in [1], $H - \lambda J$ given by the block matrices

```
[ A    0    B ]             [ E   0    0 ]
[-Q  -A^H  -S ] - \lambda * [ 0  E^H   0 ]
[ S^H B^H   R ]             [ 0   0    0 ]
```

and using a QZ decomposition method.

In this algorithm, the fail conditions are linked to the symmetry of the product $U_2 U_1^{-1}$ and condition number of U_1. Here, U is the 2m-by-m matrix that holds the eigenvectors spanning the stable subspace with 2-m rows and partitioned into two m-row matrices. See [1] and [2] for more details.

In order to improve the QZ decomposition accuracy, the pencil goes through a balancing step where the sum of absolute values of H and J entries (after removing the diagonal entries of the sum) is balanced following the recipe given in [3].

New in version 0.11.0.

References

1(1,2) P. van Dooren , “A Generalized Eigenvalue Approach For Solving Riccati Equations.”, SIAM Journal on Scientific and Statistical Computing, Vol.2(2), DOI:10.1137/0902010

[2] A.J. Laub, “A Schur Method for Solving Algebraic Riccati Equations.”, Massachusetts Institute of Technology. Laboratory for Information and Decision Systems. LIDS-R ; 859. Available online : http://hdl.handle.net/1721.1/1301

[3] P. Benner, “Symplectic Balancing of Hamiltonian Matrices”, 2001, SIAM J. Sci. Comput., 2001, Vol.22(5), DOI:10.1137/S1064827500367993

Examples

Given *a*, *b*, *q*, and *r* solve for *x*:

```
>>> from scipy import linalg
>>> a = np.array([[4, 3], [-4.5, -3.5]])
>>> b = np.array([[1], [-1]])
>>> q = np.array([[9, 6], [6, 4.]])
>>> r = 1
>>> x = linalg.solve_continuous_are(a, b, q, r)
>>> x
array([[ 21.72792206,  14.48528137],
       [ 14.48528137,   9.65685425]])
>>> np.allclose(a.T.dot(x) + x.dot(a)-x.dot(b).dot(b.T).dot(x), -q)
True
```

Annexe 3

```
<swModel>
    <task name="PowerModuleManagement" acronyme="PM" periode="30">
        <link name="Arc_1" acces="write">
        </link>
        <link name="Arc_2" acces="write">
        </link>
        <link name="Arc_3" acces="read">
        </link>
        <link name="Arc_4" acces="read">
        </link>
        <link name="Arc_5" acces="read">
        </link>
        <link name="Arc_6" acces="write">
        </link>
    </task>
    <task name="AirBlock" acronyme="AIR" periode="60">
        <link name="Arc_5" acces="write">
        </link>
        <link name="Arc_6" acces="read">
        </link>
    </task>
    <task name="RegulationH2Circuit" acronyme="H2" periode="100">
        <link name="Arc_3" acces="write">
        </link>
        <link name="Arc_1" acces="read">
        </link>
    </task>
    <task name="CoolingSystemBlock" acronyme="CS" periode="50">
        <link name="Arc_12" acces="write">
        </link>
        <link name="Arc_7" acces="read">
        </link>
    </task>
    <task name="CellStatusMonitoring" acronyme="CSM" periode="100">
        <link name="Arc_8" acces="write">
        </link>
        <link name="Arc_9" acces="read">
        </link>
    </task>
     <task name="Humidifier" acronyme="HU" periode="100">
         <link name="Arc_10" acces="write">
         </link>
         <link name="Arc_11" acces="read">
         </link>
     </task>
     <task name="SystemSupervisor" acronyme="SS" periode="20">
         <link name="Arc_4" acces="write">
         </link>

         <link name="Arc_7" acces="write">
         </link>
         <link name="Arc_9" acces="write">
         </link>
         <link name="Arc_11" acces="write">
         </link>
         <link name="Arc_2" acces="read">
         </link>
         <link name="Arc_8" acces="read">
         </link>
         <link name="Arc_12" acces="read">
         </link>
         <link name="Arc_10" acces="read">
         </link>
     </task>
</swModel>
```

Annexe 4

Extrait de la documentation de NetworkX

***class* DiGraph(*incoming_graph_data=None*, ***attr*)** [source]

Base class for directed graphs.

A DiGraph stores nodes and edges with optional data, or attributes.

DiGraphs hold directed edges. Self loops are allowed but multiple (parallel) edges are not.

Nodes can be arbitrary (hashable) Python objects with optional key/value attributes. By convention `None` is not used as a node.

Edges are represented as links between nodes with optional key/value attributes.

Attributes:

Each graph, node, and edge can hold key/value attribute pairs in an associated attribute dictionary (the keys must be hashable). By default these are empty, but can be added or changed using add_edge, add_node or direct manipulation of the attribute dictionaries named graph, node and edge respectively.

```
>>> G = nx.DiGraph(day="Friday")
>>> G.graph
{'day': 'Friday'}
```

Add node attributes using add_node(), add_nodes_from() or G.nodes

```
>>> G.add_node(1, time="5pm")
>>> G.add_nodes_from([3], time="2pm")
>>> G.nodes[1]
{'time': '5pm'}
>>> G.nodes[1]["room"] = 714
>>> del G.nodes[1]["room"]  # remove attribute
>>> list(G.nodes(data=True))
[(1, {'time': '5pm'}), (3, {'time': '2pm'})]
```

Add edge attributes using add_edge(), add_edges_from(), subscript notation, or G.edges.

```
>>> G.add_edge(1, 2, weight=4.7)
>>> G.add_edges_from([(3, 4), (4, 5)], color="red")
>>> G.add_edges_from([(1, 2, {"color": "blue"}), (2, 3, {"weight": 8})])
>>> G[1][2]["weight"] = 4.7
>>> G.edges[1, 2]["weight"] = 4
```

Warning: we protect the graph data structure by making `G.edges[1, 2]` a read-only dict-like structure. However, you can assign to attributes in e.g. `G.edges[1, 2]`. Thus, use 2 sets of brackets to add/change data attributes: `G.edges[1, 2]['weight'] = 4` (For multigraphs: `MG.edges[u, v, key][name] = value`).

Tournez la page S.V.P.

Examining elements of a graph

We can examine the nodes and edges. Four basic graph properties facilitate reporting: `G.nodes`, `G.edges`, `G.adj` and `G.degree`. These are set-like views of the nodes, edges, neighbors (adjacencies), and degrees of nodes in a graph. They offer a continually updated read-only view into the graph structure. They are also dict-like in that you can look up node and edge data attributes via the views and iterate with data attributes using methods `.items()`, `.data('span')`. If you want a specific container type instead of a view, you can specify one. Here we use lists, though sets, dicts, tuples and other containers may be better in other contexts.

```
>>> list(G.nodes)
[1, 2, 3, 'spam', 's', 'p', 'a', 'm']
>>> list(G.edges)
[(1, 2), (1, 3), (3, 'm')]
>>> list(G.adj[1])  # or list(G.neighbors(1))
[2, 3]
>>> G.degree[1]  # the number of edges incident to 1
2
```

parse(*source*, *parser=None*)

Loads an external XML section into this element tree. *source* is a file name or file object. *parser* is an optional parser instance. If not given, the standard `XMLParser` parser is used. Returns the section root element.

write(*file*, *encoding="us-ascii"*, *xml_declaration=None*, *default_namespace=None*, *method="xml"*, ***, *short_empty_elements=True*)

Writes the element tree to a file, as XML. *file* is a file name, or a file object opened for writing. *encoding* [1] is the output encoding (default is US-ASCII). *xml_declaration* controls if an XML declaration should be added to the file. Use `False` for never, `True` for always, `None` for only if not US-ASCII or UTF-8 or Unicode (default is `None`). *default_namespace* sets the default XML namespace (for "xmlns"). *method* is either `"xml"`, `"html"` or `"text"` (default is `"xml"`). The keyword-only *short_empty_elements* parameter controls the formatting of elements that contain no content. If `True` (the default), they are emitted as a single self-closed tag, otherwise they are emitted as a pair of start/end tags.

The output is either a string (`str`) or binary (`bytes`). This is controlled by the *encoding* argument. If *encoding* is `"unicode"`, the output is a string; otherwise, it's binary. Note that this may conflict with the type of *file* if it's an open file object; make sure you do not try to write a string to a binary stream and vice versa.

New in version 3.4: The *short_empty_elements* parameter.

Changed in version 3.8: The `write()` method now preserves the attribute order specified by the user.

Résumé de fonctions utiles :

DiGraph.__init__([incoming_graph_data])	Initialize a graph with edges, name, or graph attributes.
DiGraph.add_node(node_for_adding, **attr)	Add a single node `node_for_adding` and update node attributes.
DiGraph.add_nodes_from(nodes_for_adding, **attr)	Add multiple nodes.
DiGraph.add_edge(u_of_edge, v_of_edge, **attr)	Add an edge between u and v.
DiGraph.add_edges_from(ebunch_to_add, **attr)	Add all the edges in ebunch_to_add.
is_directed_acyclic_graph(G)	Returns True if the graph `G` is a directed acyclic graph (DAG) or False if not.

Nodes

The graph `G` can be grown in several ways. NetworkX includes many graph generator functions and facilities to read and write graphs in many formats. To get started though we'll look at simple manipulations. You can add one node at a time,

```
>>> G.add_node(1)
```

or add nodes from any iterable container, such as a list

```
>>> G.add_nodes_from([2, 3])
```

You can also add nodes along with node attributes if your container yields 2-tuples of the form `(node, node_attribute_dict)`:

```
>>> G.add_nodes_from([
...     (4, {"color": "red"}),
...     (5, {"color": "green"}),
... ])
```


Tournez la page S.V.P.

numpy.gcd

numpy.gcd(*x1*, *x2*, /, *out=None*, *, *where=True*, *casting='same_kind'*, *order='K'*, *dtype=None*, *subok=True*[, *signature*, *extobj*]) = <*ufunc 'gcd'*>

Returns the greatest common divisor of `|x1|` and `|x2|`

Parameters: **x1, x2 :** *array_like, int*

Arrays of values. If `x1.shape != x2.shape`, they must be broadcastable to a common shape (which becomes the shape of the output).

Returns: **y :** *ndarray or scalar*

The greatest common divisor of the absolute value of the inputs This is a scalar if both *x1* and *x2* are scalars.

See also

`lcm`
The lowest common multiple

Examples

```
>>> np.gcd(12, 20)
4
>>> np.gcd.reduce([15, 25, 35])
5
>>> np.gcd(np.arange(6), 20)
array([20,  1,  2,  1,  4,  5])
```

Modèle CMEN-DOC v2 ©NEOPTEC

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Inscription :

(Le numéro est celui qui figure sur la convocation ou la feuille d'émargement)

Né(e) le : ☐☐ / ☐☐ / ☐☐☐☐

(Remplir cette partie à l'aide de la notice)

Concours / Examen : …….. **Section/Spécialité/Série :** ……

Epreuve : …….. **Matière :** …….. **Session :** ……..

CONSIGNES

- *Remplir soigneusement, sur CHAQUE feuille officielle, la zone d'identification en MAJUSCULES.*
- *Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.*
- *Numéroter chaque PAGE (cadre en bas à droite de la page) et placer les feuilles dans le bon sens et dans l'ordre.*
- *Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.*
- *N'effectuer aucun collage ou découpage de sujets ou de feuille officielle. Ne joindre aucun brouillon.*

Document réponse DR1

Tous les documents réponses sont à rendre, même non complétés.

Tournez la page S.V.P.

D

NE RIEN ECRIRE DANS CE CADRE

Document réponse DR1

Questions 2, 3 et 4

```
import logging
from typing import Callable

import numpy as np
from numpy.linalg import pinv

logger = logging.getLogger(__name__)


class GNSolver:
    """
    Gauss-Newton solver.
    Given response vector y, dependent variable x and fit function f,
    Minimize sum(residual^2) where residual = f(x, coefficients) - y.
    """

    def __init__(self,
                 fit_function: Callable,
                 max_iter: int = 1000,
                 tolerance_difference: float = 10 ** (-16),
                 tolerance: float = 10 ** (-9),
                 init_guess: np.ndarray = None,
                 ):
        """
        :param fit_function: Function that needs be fitted; y_estimate =
fit_function(x, coefficients).
        :param max_iter: Maximum number of iterations for optimization.
        :param tolerance_difference: Terminate iteration if RMSE difference
between iterations smaller than tolerance.
        :param tolerance: Terminate iteration if RMSE is smaller than
tolerance.
        :param init_guess: Initial guess for coefficients.
        """
        self.fit_function = fit_function
        self.max_iter = max_iter
        self.tolerance_difference = tolerance_difference
        self.tolerance = tolerance
        self.coefficients = None
        self.x = None
        self.y = None
        self.init_guess = None
        if init_guess is not None:
            self.init_guess = init_guess
```

```
    def fit(self,
            x: np.ndarray,
            y: np.ndarray,
            init_guess: np.ndarray = None) -> np.ndarray:
        """
        Fit coefficients by minimizing RMSE.
        :param x: Independent variable.
        :param y: Response vector.
        :param init_guess: Initial guess for coefficients.
        :return: Fitted coefficients.
        """

        self.x = x
        self.y = y
        if init_guess is not None:
            self.init_guess = init_guess

        if init_guess is None:
            raise Exception("Initial guess needs to be provided")

        self.coefficients = self.init_guess
        rmse_prev = np.inf
        for k in range(self.max_iter):

Q2          residual = _____________________________________________________

            jacobian = self._calculate_jacobian(self.coefficients, step=10
** (-6))
            self.coefficients = self.coefficients -
self._calculate_pseudoinverse(jacobian) @ residual

Q3          rmse = _________________________________________________________

            logger.info(f"Round {k}: RMSE {rmse}")
            if self.tolerance_difference is not None:
                diff = np.abs(rmse_prev - rmse)
                if diff < self.tolerance_difference:
                    logger.info("RMSE difference between iterations smaller
than tolerance. Fit terminated.")
                    return self.coefficients
            if rmse < self.tolerance:
                logger.info("RMSE error smaller than tolerance. Fit
terminated.")
                return self.coefficients
            rmse_prev = rmse
        logger.info("Max number of iterations reached. Fit didn't
converge.")

        return self.coefficients


    def predict(self, x: np.ndarray):
        """
        Predict response for given x based on fitted coefficients.
        :param x: Independent variable.
        :return: Response vector.
        """
        return self.fit_function(x, self.coefficients)
```


Tournez la page S.V.P.

```
    def get_residual(self) -> np.ndarray:
        """
        Get residual after fit.
        :return: Residual (y_fitted - y).
        """
        return self._calculate_residual(self.coefficients)

    def get_estimate(self) -> np.ndarray:
        """
        Get estimated response vector based on fit.
        :return: Response vector
        """
        return self.fit_function(self.x, self.coefficients)
    def _calculate_residual(self, coefficients: np.ndarray) -> np.ndarray:
        y_fit = self.fit_function(self.x, coefficients)
        return y_fit - self.y


    def _calculate_jacobian(self,
                            z: np.ndarray,
                            delta: float = 10 ** (-6)) -> np.ndarray:
        """
        Calculate Jacobian matrix numerically.
        J_ij = d(r_i)/d(x_j)
        """
        R0 = self._calculate_residual(z)

        J = []
        for i, parameter in enumerate(z):
```

Q4		zhat = z.copy() zhat[i] += delta **R =** _____ **difference =** _____ **J**_____

```
        J = np.array(J).T

        return J

    @staticmethod
    def _calculate_pseudoinverse(x: np.ndarray) -> np.ndarray:
        """
        Moore-Penrose inverse.
        """
        return pinv(x.T @ x) @ x.T
```

Modèle CMEN-DOC v2 ©NEOPTEC

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Inscription :

Né(e) le : ☐☐/☐☐/☐☐☐☐

(Le numéro est celui qui figure sur la convocation ou la feuille d'émargement)

(Remplir cette partie à l'aide de la notice)

Concours / Examen : …….. **Section/Spécialité/Série :** ……

Epreuve : …….. **Matière :** …….. **Session :** ……..

CONSIGNES

- *Remplir soigneusement, sur CHAQUE feuille officielle, la zone d'identification en MAJUSCULES.*
- *Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.*
- *Numéroter chaque PAGE (cadre en bas à droite de la page) et placer les feuilles dans le bon sens et dans l'ordre.*
- *Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.*
- *N'effectuer aucun collage ou découpage de sujets ou de feuille officielle. Ne joindre aucun brouillon.*

Documents réponses DR2 à DR5

Tous les documents réponses sont à rendre, même non complétés.

NE RIEN ECRIRE DANS CE CADRE

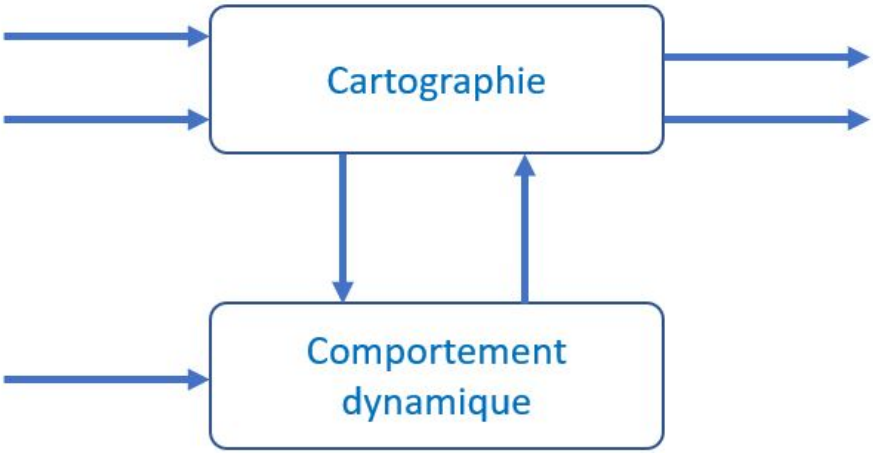
Document réponse DR2

Question Q5

Ligne de la méthode fit	Commentaire proposé
rmse_prev = np.inf	
for k in range(self.max_iter):	
self.coefficients = self.coefficients - self._calculate_pseudoinverse(jacobian) @ residual	
diff = np.abs(rmse_prev - rmse)	
if diff < self.tolerance_difference:	
if rmse < self.tolerance:	

Document réponse DR3

Question Q9



Document réponse DR4

Question Q10

	A	B_u	B_i	C_y	D_{yu}	D_{yi}	C_z	D_{zu}	D_{zi}
Nombre de lignes	8	8							
Nombre de colonnes	8	1							

Tournez la page S.V.P.

Document réponse DR5

Question Q12

```
def CalculeBCD(Bu,Bw,Cz,Cy,Dzu,Dzw,Dyu,Dyw):

    B=np.zeros((8,2))
    C=np.zeros((5,8))
    D=np.zeros((5,2))

    B[:,0]=Bu[:,0]
    B[:,1]=Bw[:,0]

    C[0:2,:]=_________________

    C[2:5,:]=_________________

    D[0:2,0]=_________________

    D[0:2,1]=_________________

    D[2:5,0]=_________________

    D[2:5,1]=_________________


    return(B,C,D)
```