Calculs derrière *PySpecies*

On cherche à résoudre numériquement le système d'équations aux dérivées partielles suivant :

$$\begin{cases} \partial_t u - \Delta(c_1 u + d_{11} u^2 + d_{12} uv) & = u(r_1 - s_{11} u - s_{12} v) \\ \partial_t v - \Delta(c_2 v + d_{21} uv + d_{22} v^2) & = v(r_2 - s_{21} u - s_{22} v) \end{cases}$$

On synthétise ces coefficients dans les matrices suivantes :

$$D := \begin{pmatrix} c_1 & d_{11} & d_{12} \\ c_2 & d_{21} & d_{22} \end{pmatrix} \quad \text{et} \quad R := \begin{pmatrix} r_1 & s_{11} & s_{12} \\ r_2 & s_{21} & s_{22} \end{pmatrix}$$

Schéma d'Euler implicite

On discrétise les concentrations u et v en espace et en temps. Pour alléger les notations, en notant δx et δt les pas spatiaux et temporels considérés, on redéfinit :

$$D \leftarrow \left(\frac{\delta t}{\delta x^2}\right) D \qquad \text{et} \qquad R \leftarrow \delta t \cdot R$$

Étant donné les $(U_k^-)_{0 \le k < K}$ à un instant donné, on cherche à déterminer les $(U_k)_{0 \le k < K}$ à l'instant suivant. Ces valeurs sont reliées par le schéma suivant, pour $k \in \{0, \dots, K-1\}$:

$$\begin{aligned} 0 = U_k - U_k^- &- c_1 \left(U_{k+1} - 2U_k + U_{k-1}\right) - d_{11} \left(U_{k+1}^2 - 2U_k^2 + U_{k-1}^2\right) \\ &- d_{12} \left([UV]_{k+1} - 2[UV]_k + [UV]_{k-1}\right) - r_1 U_k + s_{11} U_k^2 + s_{12} [UV]_k \end{aligned}$$

Si A et B sont deux vecteurs, on définit $A \circ B$ comme le vecteur obtenu en multipliant chacune des coordonnées de A et B terme à terme. On pose également la matrice :

$$J = \begin{pmatrix} & 1 & & 0 \\ & & \ddots & \\ 0 & & & 1 \\ 1 & 0 & & \end{pmatrix}$$

de sorte que dans $\mathbb{Z}/k\mathbb{Z}$, $(U_{k+1})_{1 \le k < K} = JU$, ce qui nous permet de définir des conditions aux bord périodiques et de rassembler les K équations précédentes :

$$f_1(U, V) - U^- = 0$$

Où :

$$\begin{aligned} f_1(U, V) = &\left[(1 - r_1 + 2c_1) I - c_1 (J + J^{-1})\right] \cdot U \\ &+ \left[(s_{11} + 2d_{11}) I - d_{11} (J + J^{-1})\right] \cdot U \circ U \\ &+ \left[(s_{12} + 2d_{12}) I - d_{12} (J + J^{-1})\right] \cdot U \circ V \end{aligned}$$

Soit :

$$\begin{aligned} f_1(U, V) = &- J \left(c_1 U + d_{11} U \circ U + d_{12} U \circ V\right) \\ &+ \left[(1 - r_1 + 2c_1) U + (s_{11} + 2d_{11}) U \circ U + (s_{12} + 2d_{12}) U \circ V\right] \\ &- J^{-1} \left(c_1 U + d_{11} U \circ U + d_{12} U \circ V\right) \end{aligned}$$

En *Python*, multiplier un vecteur v à gauche par la matrice J revient à appliquer la fonction np.roll$(v, 1)$.
On définit X comme la concaténation de U et de V. En inversant 1 et 2 puis U et V dans l'équation précédente, on obtient l'équation analogue en V, et on doit résoudre en résumé :

$$g_{X^-}(X) = 0 \qquad \text{où } g_{X^-}(X) := \begin{pmatrix} f_1(U, V) - U^- \\ f_2(V, U) - V^- \end{pmatrix}$$

Il s'agit de trouver le zéro d'une fonction non linéaire en les coefficients de X.

Méthode de Newton-Raphson

Pour ce faire, on utilise la méthode de Newton dérivée à partir de la formule de Taylor ($X = X^- + \delta X$) :

$$g_{X^-}(X_{\ell+1}) = g_{X^-}(X_\ell) + J_{g_{X^-}}(X_\ell) \cdot \delta X + \mathcal{O}(||\delta X||^2)$$

On initialise $X_0 = X^-$ puis on résoud le système linéaire suivant en δX connaissant X_ℓ :

$$J_{g_{X^-}}(X_\ell) \cdot \delta X = -g_{X^-}(X_\ell)$$

Puis on met à chaque fois à jour $X_{\ell+1} \leftarrow X_\ell + \delta X$ avant de recommencer.
On stoppe la méthode de Newton dès lors que $\|\delta X\| < \varepsilon_r$ où ε_r est un seuil de précision à définir, et l'on retient alors la dernière valeur de X calculée comme celle à l'instant $n+1$. On stoppe la simulation lorsque $\|\delta X\|$ dépasse éventuellement un seuil ε_s, ce qui semble indiquer une instabilité numérique.

Calcul de la jacobienne

On calcule la jacobienne de g par blocs :

$$J_g(X) = \begin{pmatrix} J_U f_1(U,V) & J_V f_1(U,V) \\ J_U f_2(V,U) & J_V f_2(V,U) \end{pmatrix}$$

Or :

$$\begin{aligned} f_1(U,V) = &[(1 - r_1 + 2c_1)I - c_1(J + J^{-1})]U \\ &+ [(s_{11} + 2d_{11})I - d_{11}(J + J^{-1})]U \circ U \\ &+ [(s_{12} + 2d_{12})I - d_{12}(J + J^{-1})]U \circ V \end{aligned}$$

On constate que $J_U U = I$, $J_U U \circ V = \mathrm{diag}\, V$ et $J_U U \circ U = 2\mathrm{diag}\, U$. Ainsi :

$$\begin{aligned} J_U f_1(U,V) = &(1 - r_1 + 2c_1)I - c_1(J + J^{-1}) \\ &+ 2[(s_{11} + 2d_{11})I - d_{11}(J + J^{-1})]\mathrm{diag}\, U \\ &+ [(s_{12} + 2d_{12})I - d_{12}(J + J^{-1})]\mathrm{diag}\, V \end{aligned}$$

En factorisant I, J et J^{-1} dans cette expression, on remarque qu'elle est tridiagonale avec deux termes supplémentaires aux extrémités correspondant aux conditions aux bords :

- Extrémité K : $-(c_1\mathbf{1} + 2d_{11}U + d_{12}V)[-1]$
- Sur-diagonale $(+1)$: $-(c_1\mathbf{1} + 2d_{11}U + d_{12}V)[1:]$
- Diagonale (0) : $(1 - r_1 + 2c_1)\mathbf{1} + 2(s_{11} + 2d_{11})U + (s_{12} + 2d_{12})V$
- Sous-diagonale (-1) : $-(c_1\mathbf{1} + 2d_{11}U + d_{12}V)[:-1]$
- Extrémité $-K$: $-(c_1\mathbf{1} + 2d_{11}U + d_{12}V)[0]$

Puis, de manière analogue :

$$J_V f_1(U,V) = [(s_{12} + 2d_{12})I - d_{12}(J + J^{-1})]\mathrm{diag}\, U$$

Cette matrice possède la même structure :

- Extrémité K : $-d_{12}U_{K-1}$
- Sur-diagonale $(+1)$: $-d_{12}U[1:]$
- Diagonale (0) : $(s_{12} + 2d_{12})U$
- Sous-diagonale (-1) : $-d_{12}U[:-1]$
- Extrémité $-K$: $-d_{12}U_0$

En échangeant 1 et 2 puis U et V, on en déduit la forme complète de J_g. Comme cette matrice contient énormément de termes nuls, on ne stocke que ses diagonales. La fonction MergeDiagonals construit la liste des diagonales de J_g à partir de la liste des diagonales de chacun de ses blocs.

Réduction de la largeur de bande

Pour tirer parti de la structure lacunaire de la matrice J_g, on utilise le module scipy.sparse qui résoud le système linéaire considéré en $O(bN)$ où b est la largeur de bande de la matrice J_g et N sa taille. En l'état, à cause des deux termes extrémaux, la bande passante vaut N, d'où une complexité quadratique décevante. Pour réduire la bande passante de la matrice, on utilise l'algorithme de *Cuthill-McKee inversé* qui permet d'appliquer une permutation à J_g réduisant drastiquement sa bande passante. Puisque cet algorithme ne tient compte que de la structure de la matrice et non de ses valeurs, cette permutation est en fait valable tout au long de la résolution.