Projet Optimisation Numérique

Quentin Jaubertie

January 2017

Table des matières

1 Algorithme de Newton local

1.1 Principe

Cet algorithme minimise la fonction f selon la direction de plus profonde descente $d_k = -\nabla^2 f(x)^{-1}\nabla f(x)$ (avec $\nabla^2 f(x) \in S_n^{++}$).

Le flag de sortie a pour valeur :
- 0 : Nombre d'itérations maximum atteint
- 1 : $\|\nabla f(x_k)\| \leq eps * (\|\nabla f(x_0)\| + tol)$.
- 2 : Suite stationnaire.

Remarques :
- La tolérance **tol** utilisée vaut toujours $\sqrt{eps_{mach}}$
- On utilise toujours la $\|.\|_2$
- La précision eps a pour valeur 10^{-6} (Sauf exception)

1.2 Tests de l'algorithme

1.2.1 Fonction utilisée : $f_1(x_{sol} = [1; 1; 1])$

x_0	epsilon	x^*	N_{itera}	N_{eval}	$\nabla f(x^*)$	flag
10	10^{-6}	1	1	0	$0.2 * 10^{-14}$	1
-3		1		3	$-0.3 * 10^{-14}$	
-2.2		1		2	$-0.4 * 10^{-14}$	
0	10^{-6}	1	1	0	$-0.04 * 10^{-14}$	1
0		1		3	$-0.13 * 10^{-14}$	
0		1		2	$-0.08 * 10^{-14}$	

On remarque que l'algorithme converge en une itération pour ces trois points de départ. **Est ce que l'algorithme converge en une itération quelque soit $x_0 \in \mathbf{R}^n$? ($\mathbf{f} = f_1$)**

Soit $x_0 \in \mathbf{R}^n$, calculons l'itéré $x_1 = x_0 + \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$:

d_0 solution de :

$$\nabla^2 f(x_0) d_0 = -\nabla f(x_0)$$

$$\Leftrightarrow \nabla^2 f(x_0)d_0 = \begin{bmatrix} 6d_1 + 2d_2 + 4d_3 \\ 2d_1 + 8d_2 + 2d_3 \\ 4d_1 + 2d_2 + 6d_3 \end{bmatrix} = \begin{bmatrix} 12 - 6x_1 - 2x_2 - 4x_3 \\ 12 - 2x_1 - 8x_2 - 2x_3 \\ 12 - 4x_1 - 2x_2 - 6x_3 \end{bmatrix}$$

On remarque donc que $d_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - x_0 \implies x_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ soit la solution à notre problème. L'algorithme converge en une itération pour f $=f_1$. De plus, cette fonction est d'ordre 2 (On peut l'assimiler à une parabole). Elle possède donc un unique bassin de convergence ce qui implique la convergence dans ce cas.

1.2.2 Fonction utilisée : $f_2(x_{sol} = [1; 1]$)

Cette fonction est d'ordre 4, ce qui la rend plus complexe que la précédente.
Voici les tests réalisées :

x_0	epsilon	x^*	N_{itera}	N_{eval}	$\nabla f(x^*)$	flag
-1.2	10^{-6}	1	5	[0 ;7 ;6]	$-8.6 * 10^{-6}$	1
1		0.9			$-2.9 * 10^{-11}$	
10	10^{-6}	1	3	[0 ;5 ;4]	$7.2 * 10^{-4}$	1
0		1			$-1.42 * 10^{-3}$	
0	10^{-6}	-Inf	7	[0 ; 9 ; 8]	NaN	2
$0.005+10^{-12}$		Inf			NaN	

On remarque que l'algorithme ne converge pas pour tous les points de départs. **Quelles sont la (les) cause(s) de non convergence ?**

Prenons $x_0^{pert} = \begin{bmatrix} 0 \\ 0.005 + 10^{-12} \end{bmatrix}$:

$$\nabla^2 f(x_0) \approx \begin{bmatrix} 0 & 0 \\ 0 & 200 \end{bmatrix}.$$

La hessienne de f_2 est en pratique non inversible i.e. a un conditionnement très mauvais ce qui empêche le bon déroulement du calcul du pas de Newton

(rcond = 2*10^{-12}). De même, si l'on prend $x_0 = \begin{bmatrix} 0 \\ 0.005 \end{bmatrix}$,la hessienne a un conditionnement infini (rcond = 0) ce qui provoque des divisions par zéro et donc retourne un résultat non fiable.
Remarque : J'utilise **Octave** sur mon ordinateur personnel et ce dernier arrive à calculer la bonne solution avec le deuxième exemple malgré un conditionnement infini.

2 Algorithme des régions de confiance

Cet algorithme permet de minimiser un modèle d'approximation de f : $m_k(x_k + pas)$ tout en gardant le pas dans une région de confiance Δ_k. On met à jour les itérés x_k et Δ_k suivant la décroissance du modèle.
Pour ce projet, le calcul du pas se fait soit avec le pas de Cauchy soit avec l'algorithme de More-Sorensen.
Le flag de sortie vaut :

- 0 : Nombre d'itérations maximum atteint
- 1 : $\|\nabla f(x_k)\| \leq eps * (\|\nabla f(x_0)\| + tol)$.
- 2 : Suite stationnaire : $\|pas\| \leq eps * (\|x_k\| + tol)$.

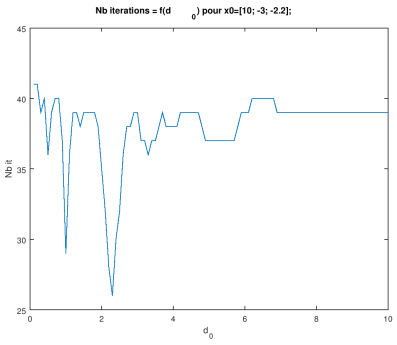
Remarques :

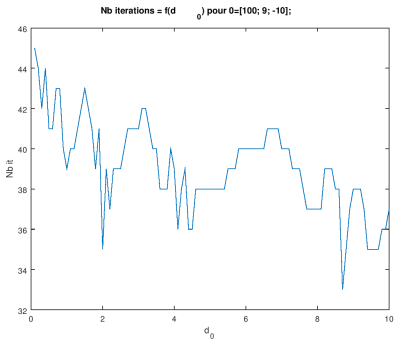
- Modèle d'approximation utilisé :
$$m_k(x_k + s) = f(x_k) + (\nabla f(x_k)|s) + \tfrac{1}{2}(\nabla^2 f(x_k)s|s)$$
- eps = 10^{-6}
- Il faut rester dans un voisinage proche de x_k afin de conserver la validité du modèle.
- Sauf exception, ($\gamma_1, \gamma_2, \eta_1, \eta_2$, Δ_0,Δ_{max}) valent (0.5 ; 2 ; 0.25 ; 0.75, 1, 10^4).

2.1 Résolution avec pas de Cauchy

2.1.1 Fonction f_1

D'après mes calculs, la fonction f_1 appliquée en y est égale à son developpement de Taylor au voisinage de x_k. Ceci signifie donc que nous ne sommes pas obligés de travailler dans un voisinage de x_k.
Une première idée serait de considérer un Δ_0 grand (Par exemple 10^3) afin de ne pas avoir un pas trop petit et de converger plus rapidement. Voici les résultats obtenus en faisant varier Δ_0 pour x_0 = [10 ; -3 ; -2.2] et x_0 = [100 ; 9 ; -10] : (





On voit que l'augmentation de Δ_0 n'implique pas une diminution du nombre d'itérations. On constaste même que le Δ_0 optimal pour l'exemple 1 ≈2.1. Cependant, l'algorithme converge plus lentement vers la solution x^* = [1 ; 1 ; 1] par rapport à **Newton Local**.

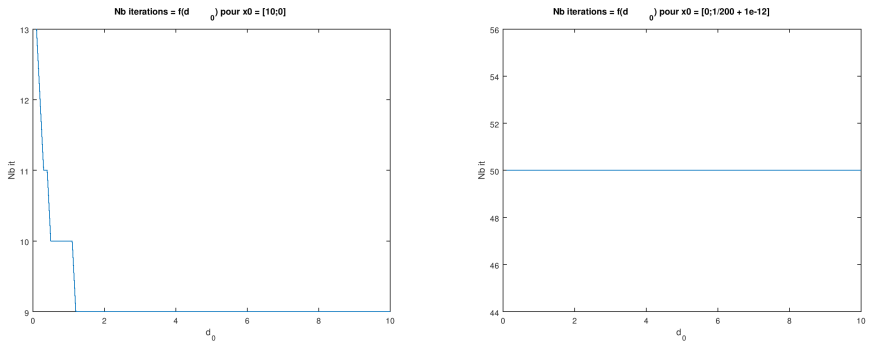
Ensuite, j'ai fait varier le paramètre η_1 sur l'intervalle [0.05 ;0.75] mais ceci n'a eu aucune influence sur le résultat.

L'algorithme des régions de confiance semble moins optimale que l'algorithme de Newton pour f_1 car ce dernier calcule la solution en une itération pour tout x_0.

2.1.2 Fonction f_2

Contrairement à f_1, f_2 diffère de son développement de Taylor à l'ordre 2. Comme précédemment, j'ai repris les tests de **Newton local** en faisant varier Δ_0.

Voici les resultats obtenus avec $x_0 = [10\,;0]$ et $x_0 = [0\,;\, 1/200 + 10^{-12}]$:



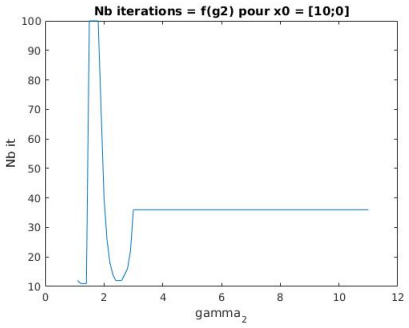
J'ai conservé les mêmes paramètres que pour f_1 avec un nombre maximum d'itérations égal à 50. Ici aussi, l'augmentation de Δ_0 n'implique pas une diminution du nombre d'itérations. Un Δ_0 trop grand ne permet pas de conserver la validité de l'approximation. Pour l'exemple 1, la solution obtenue $x^* = \begin{bmatrix}0.98\\0.97\end{bmatrix}$est proche de solution exacte. L'algorithme converge rapidement (9-10 itérations). Un $\Delta_0 = 2$ est suffisant. On remarque qu'un Δ trop petit augmente le nombre d'itérations car le pas calculé est nécessairement plus petit.
Pour l'exemple 2, cet algorithme permet d'obtenir une solution plus fiable que **Newton local** car on évite le calcul de l'inverse de la hessienne. Cependant, la solution reste "éloignée" de la solution exacte $[1\,;\,1]$:

$$\text{Les solutions trouvées varient entre } x_1^* = \begin{bmatrix}0.92\\0.85\end{bmatrix}$$

$$\text{et } x_2^* = \begin{bmatrix}0.67\\0.48\end{bmatrix}$$

(La solution est plus précise pour $\Delta_0 \approx 3$).

Ensuite, j'ai aussi fait varier η_1 sur $[0.05\,;0.7]$ avec les points $[1\,;2]$ et $[10\,;0]$. Cependant, il n'y a quasiment aucune différence entre les résultats ce qui est étrange (Exemple 1 : $x_1^* = [1.15\,;1.32]$ pour tout η sauf pour η_1=0.7 : $[1.33\,;1.77]$). Pour finir, voici les tests réalisés sur γ_2 :



Contrairement à η_1, γ_2 a un effet important sur le nombre d'itérations car il permet de modifier Δ_k. On voit que pour γ_2^* ≈2.5 ou 1.75, le nombre d'itérations est minimale (≈ 12). Il est donc intéressant, voir nécessaire de se préoccuper de la valeur de γ_2 car elle permet de réduire considérablement le temps d'exécution. On constate même que pour $\gamma \approx 2$, l'algorithme atteint le nombre d'itérations maximum ce qui n'est pas souhaitable. Enfin, on remarque que pour $\gamma_2 \geq 3$, le nombre d'itérations est stationnaire. Pourtant, on aurait pu penser qu'un γ_2 trop petit / trop grand impliquerait une augmentation considérable du nombre d'itérations.

Pour conclure, l'algorithme des régions de confiance est intéressant car il permet d'éviter le calcul de l'inverse de la hessienne et prend en compte toutes les directions donc évite les divisions par zéro. Cependant, il converge moins vite vers la solution qui est, dans certains cas, moins précise. Il peut être très intéressant si nous trouvons les valeurs optimales des paramètres ($\gamma_1, \gamma_2, \eta_1, \eta_2,\ \Delta_0$,$\Delta_{max}$) qui varient suivant chaque fonction. Un Δ_0 grand n'est pas forcément optimal ce qui rend la chose plus ardue.

2.2 Résolution avec More-Sorensen

Voici les resultats des tests avec l'algorithme de More-Sorensen (Les paramètres sont identiques à ceux des tests fournis sur le pas de Cauchy) : Avec f_1 : (Itération de l'algorithme + Itérations de More Sorensen)

x_0	epsilon	x^*	N_{itera}	N_{eval}	$\|\nabla f(x^*)\|$	flag
1	10^{-3}	1	1	2	$6.2 * 10^{-16}$	1
0		1		2		
0		1		2		
10	10^{-3}	1	3+18	4	$1.25 * 10^{-15}$	1
3		1		4		
-2.2		1		4		

Avec f_2 :

x_0	epsilon	x^*	N_{itera}	N_{eval}	$\|\nabla f(x^*)\|$	flag
-1.2	10^{-3}	0.99	34+74	[35 ;20 ;20]	0.02	0
1		0.99				
10	10^{-3}	3.996	5+75	[6 ;6 ;6]	6.01	1
0		15.97				
1	10^{-3}	1	27	[14 ; 10 ; 10]	0.155	1
2		1				

Avec f_1, More-Sorensen est plus efficace que le pas de Cauchy pour les mêmes valeurs de paramètres mais reste moins performant que Newton Local. Avec f_2, l'algorithme se rapproche de la solution $\forall x_0$. On constate que le premier test offre une meilleure solution qu'avec le pas de Cauchy pour le même nombre d'itérations. Cependant, pour $x_0 = \begin{bmatrix}10\\0\end{bmatrix}$, $\|\nabla f_2(x_0)\| \approx 6$ ce qui provoque l'arret de l'algorithme alors que le résultat obtenu ne convient pas ($x_{sol} \approx \begin{bmatrix}3.996\\15.97\end{bmatrix}$). Dans ce cas, il faut augmenter la précision ϵ ($\approx 10^{-6}$). Pour le dernier test, la solution trouvée est meilleure que celle trouvée par le pas de Cauchy mais le nombre d'itérations est plus grand.
Pour conclure, More-Sorensen est dans l'ensemble plus précis mais plus couteux que le pas de Cauchy.

3 Algorithme de Newton non Linéaire

Cet algorithme calcule λ^* tel que $\phi(\lambda^*)=0$ si existence.
Le flag de sortie vaut :
— 0 : Nombre d'itérations maximum atteint
— 1 : $\phi(\lambda) \approx 0$

Remarque :
— Fonctions testées de la forme $\phi : \lambda \to \|s(\lambda)\|^2 - \delta^2$
— On aurait pu calculer la taille de l'intervalle $[\lambda_{min}; \lambda_{max}]$ dans le critère d'arret.

3.1 Tests

Voici les tests réalisés :

$\|s(\lambda)\|^2$	δ	λ_{min}	λ_{max}	λ^*	$\phi(\lambda^*)$	Nb itérations	neval	flag
$\frac{4}{(\lambda-38)^2} + \frac{400}{(\lambda+20)^2}$	0.5	-100	-40	-60	-2.5*10^{-8}	19	[21 ;19]	1
		12	30	21.18	2.0*10^{-9}	3	[5 ;3]	1
		-40	-10	Erreur				Error Précond

La constante δ ne sert qu'à annuler la fonction ϕ. On voit que l'algorithme converge vers la solution si les préconditions sont respectées.

3.2 Améliorations

Il est possible que l'utilisateur rentre des paramètres ne vérifiant pas les conditions. Tout d'abord, je me suis contenté de renvoyer un message d'erreur et de stopper l'algorithme. Voici une proposition d'amélioration : Considérons $(a_i)_{1\leq i\leq n}$ et $(b_i)_{1\leq i\leq n} \in \mathbf{R}^{n^2}$ (Les b_i sont ordonnés).

On suppose que les fonctions utilisées sont de la forme :

$$\phi(\lambda)=\sum_{i=1}^{n} \frac{a_i^2}{(\lambda+b_i)^2} - \delta^2.$$

Cette fonction admet des racines car :
- $\lim_{x\to -b_i} \phi(x) = +\infty$
- $\lim_{x\to \infty} \phi(x) = -\delta^2$

Afin de ne pas trop s'éloigner de l'intervalle de départ, on peut faire :
Si $\lambda_{max} \leq -b_0$ (resp si $\lambda_{min} \geq -b_n$)
alors $\lambda_{max} = -b_0 - \epsilon$ et on fait décroitre λ_{min} jusqu'à avoir $\phi(\lambda_{min}) \leq 0$
(resp $\lambda_{min} = -b_n + \epsilon$ et on fait croitre λ_{max}).

Sinon : $\lambda_{min} = -b_i + -\epsilon$ le plus proche, $\lambda_{max} = \lambda_{min}$ et on fait croître λ_{max} jusqu'à obtenir $\phi(\lambda_{max}) \leq 0$.

Si le pas choisi pour faire (dé)croître λ est grand (> 1) , on est sur de trouver une solution car la fonction est négative à l'infini et positive au voisinage des (b_i). Cependant, il est possible de rater une potentielle racine intermédiaire et donc d'augmenter le nombre d'itérations. Je pense qu'il vaut mieux utiliser un pas grand dans le premier cas et un pas plus petit (au plus 10^{-1}) pour le bloc " Sinon". J'ai conscience que cette méthode n'est pas optimale mais elle permet d'obtenir un résultat. Cependant, il faut avoir accès aux valeurs des b_i ce qui est encombrant.

4 Lagrangien Augmenté

Cet algorithme minimise la fonction f sous la contrainte c. Le flag de sortie vaut :

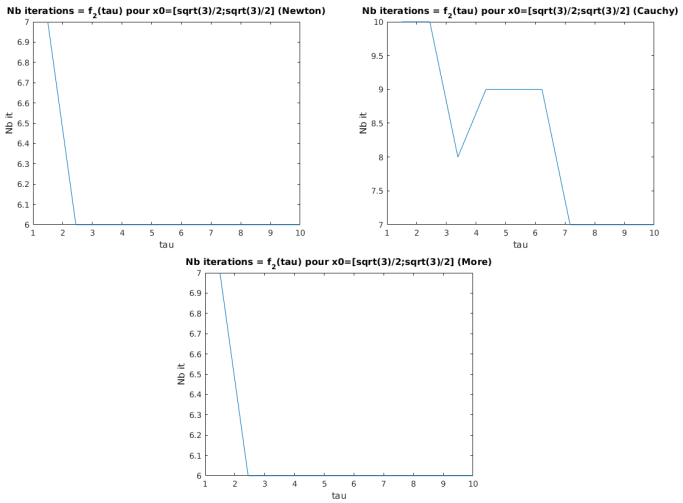
— 0 : Nombre d'itérations maximum atteint
— 1 :$\neg((\|\nabla Lag(x_k)\| < \epsilon_k * (\|\nabla Lag(x_0)\| + tol)) \vee (\|c(x_k)\| < \epsilon_k * (\|c(x_0)\| + tol)))$
— 2 : Suite stationnaire ($\epsilon = 10^{-6} \neq \epsilon_k$)

Une des difficultés est que la dérivée seconde de la contrainte peut être tridimensionnelle. Dans ce cas, on aura : (abus de notation)

$$\lambda_k^T c''(x) = \sum_{i=1}^{n} (\lambda_k(i) * c''^T(:,:,i))$$

4.1 Tests sur f_2 : Influence de τ

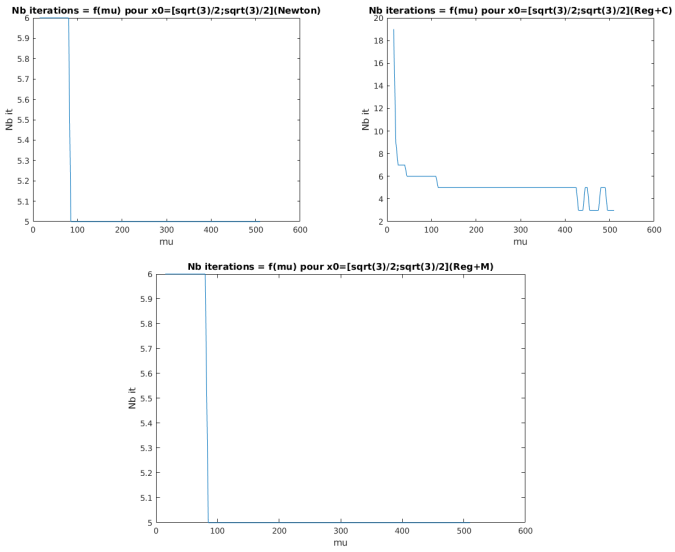
(avec $(\lambda_0, \mu_0, \eta_0, \alpha, \beta) = (1\,;10\,;0.1258925\,;0.1\,;0.9)$)



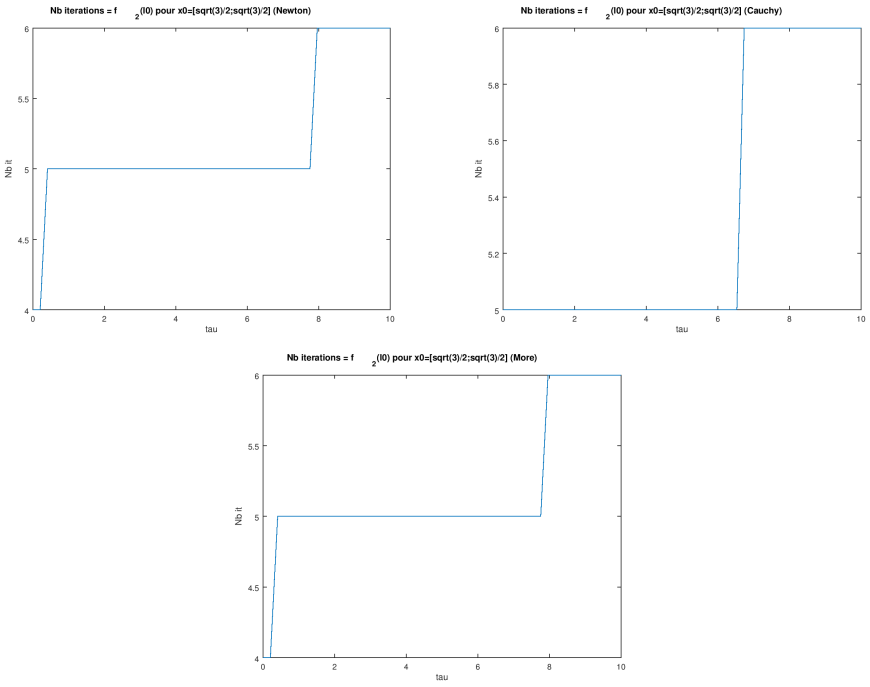
Pour l'exemple 1, la solution $x^* = [0.908\,;\,0.82]$ est trouvée $\forall \tau \geq 2$. On constate que pour $\tau > 1$, la solution est calculée en 6-7 itérations ce qui est convenable. Le rôle de τ est d'augmenter la valeur du μ qui doit être grand donc il vaut avoir un τ conséquent. Pour conclure, une augmentation de τ est, en principe, préférable et permet de trouver une solution convenable tout en réduisant le nombre d'itérations.

4.2 Tests : Valeurs de λ_k et μ_k

En principe, (μ_k) doit être une suite divergente. En effet, plus les μ_k sont grands, plus nous allons minimiser le facteur $\|c(x)\|^2$ qui lui est associé. Au contrainte, les (λ_k) doivent être petits afin de pouvoir minimiser au mieux $\|c(x)\|^2$. En pratique, avec f_2 :



Là aussi, avoir un μ_k élevé est préférable. Dans le cas 2 , il permet de converger en 3 itérations au lieu de 19. Au contraire, voici les tests pour λ_0 (avec μ_0=100) :



L'idéal est d'avoir un λ_0 nul. En effet, le nombre d'itérations semble optimal pour cette valeur.

5 Annexe

- Ajout du critère "suite stationnaire" à Newton Local.
- Calcul itérations internes pour régions de confiance+More-Sorensen et Lagrangien Augmente.
- Ajout programme de tests dynamique.
- Gestion affichage des programmes
- Modification du critere du lagrangien augmente (Suite stationnaire : précision changée)