
```
import autograd as ag
import autograd.numpy as np
```

1 Résolution d'un système

Le système d'équations suivant admet deux solutions réelles.

$$F(a,b) \quad = \quad \left\{ \begin{array}{rcl} a^3\,b - 3\,a^2\,(b-1) + b^2 - 1 & = & 0\,, \\ a^2\,b^2 - 2 & = & 0\,. \end{array} \right.$$

Question 1. Estimer graphiquement les deux solutions en adaptant les commandes du fichier `graphe_R2_dans_R2.py` qui affiche les surfaces définies par deux autres fonctions f et g. Les courbes de niveau $f(a,b) = k$ et $g(a,b) = k$ sont en pointillés sauf les courbes de niveau $f(a,b) = 0$ et $g(a,b) = 0$ qui sont continues. Il faut probablement déplacer un peu la fenêtre d'affichage.

Question 2. Écrire une fonction `Jac_F` paramétrée par a et b, qui retourne la matrice jacobienne de la fonction F.

Question 3. Utiliser les réponses aux deux questions précédentes pour calculer précisément les coordonnées des deux zéros de F par une méthode de Newton.

Question 4. Reprendre la question précédente en utilisant la fonction `jacobian` du paquetage `autograd` à la place de `Jac_F`. Note : pour utiliser `autograd`, il faut transformer F en une fonction paramétrée par un tableau `numpy` (la version `autograd` de `numpy`) contenant a et b :

```
def F (u) :
    a = u[0]
    b = u[1]
```

Question 5. Reprendre la question précédente en dérivant les codes Python suivants, qui évaluent F (voir la section suivante pour la méthode).

```
def F(u) :
    a = u[0]
    b = u[1]
# Pour f(a,b)
    t1 = a ** 2
    t2 = b ** 2
    t3 = (a*b - 3*b + 3)*t1 + t2 - 1
# Pour g(a,b)
    t1 = a ** 2
    t2 = b ** 2
    t4 = t1 * t2 - 2
# Résultat
    return np.array ([t3, t4])
```

2 Introduction à la dérivation automatique (*forward mode*)

La fonction

$$f(a,b) \quad = \quad \sin(a^2) + a^2\, b$$

peut s'évaluer par le code suivant

```
t1 = a*a
t2 = np.sin(t1)
t3 = t1 * b
t4 = t2 + t3
```

Supposons qu'on cherche à calculer son gradient :

$$\nabla f \quad = \quad \begin{pmatrix} \dfrac{\partial f}{\partial a} \\ \dfrac{\partial f}{\partial b} \end{pmatrix} \quad = \quad \begin{pmatrix} 2\,a\,\cos(a^2) + 2\,a\,b \\ a^2 \end{pmatrix}$$

Il suffit d'introduire huit variables

$$\begin{array}{lll} \texttt{dtk\_da} & \text{pour} & \dfrac{\partial t_k}{\partial a} \\ \texttt{dtk\_db} & \text{pour} & \dfrac{\partial t_k}{\partial b} \end{array} \qquad k = 1, 2, \ldots, 4\,.$$

et d'insérer dans le code initial des instructions qui les calculent. Pour cela, il suffit de voir chacune des variables t_k comme une fonction de a et/ou de b et d'appliquer la formule pour la dérivation (partielle) des fonctions composées. Voici ce que cela donne sur l'exemple. On commence par t_1.

```
t1 = a*a
dt1_da = 2*a
dt1_db = 0
```

Pour calculer `dt2_da`, il faut dériver (par rapport à a) $t_2 = \sin(t_1)$. On rappelle la formule pour dériver les fonctions composées. Notons $(f \circ g)(a) = f(g(a))$. On a :

$$(f \circ g)'(a) \quad = \quad f'(g(a))\, g'(a)\,.$$

Sur l'exemple, nous avons $f = \sin$, $g = t_1$ et donc $f' = \cos$. La dérivation est la dérivation partielle par rapport à a donc $g' = \frac{\partial t_1}{\partial a}$. Or nous avons déjà calculé cette quantité et nous l'avons stockée dans `dt1_da`. Nous obtenons donc (le calcul de `dt2_db` est similaire) :

```
t2 = np.sin(t1)
dt2_da = np.cos(t1) * dt1_da
dt2_db = np.cos(t1) * dt1_db
```

Pour calculer `dt3_da`, il faut dériver (par rapport à a) $t_3 = t_1\, b$. On a $f : x \mapsto b\, x$, $g = t_1$ et donc $f' = b$. On obtient :

```
t3 = t1 * b
dt3_da = dt1_da * b
```

Pour calculer `dt3_db`, il faut dériver (par rapport à b) $t_3 = t1\, b$. On applique la formule pour la dérivée d'un produit : $\frac{\partial t_3}{\partial b} = \frac{\partial t_1}{\partial b}\, b + t_1$ qui se traduit par :

```
dt3_db = dt1_db * b + t1
```

Même principe en plus simple pour t_4. Les variables `dt4_da` et `dt4_db` contiennent les deux coordonnées du gradient.

```
t4 = t2 + t3
dt4_da = dt2_da + dt3_da
dt4_db = dt2_db + dt3_db
```