Mécanique des fluides

Section de génie civil

Résolution d'équations implicites

Introduction

Une équation implicite $f(x, y, z, \dots) = 0$ est une équation dont la variable d'intérêt x ne peut pas être isolée. En d'autres mots, il nous est impossible de l'écrire sous forme $x = g(y, z, \dots)$. On ne parvient pas à trouver x par un calcul direct.

Deux exemples récurrents en hydraulique sont le calcul de la hauteur d'eau h pour une charge H donnée et le calcul de la hauteur normale h_n. Pour le premier exemple, on cherche à résoudre

$$H = z + h + \frac{Q^2}{2gS(h)^2} \tag{1}$$

Où la seule inconnue est h. Il n'est possible d'isoler h que si la forme de $S(h)$ le permet. En conséquence, cette équation implicite se résout numériquement. Le deuxième exemple est un cas similaire où il est souvent difficile d'isoler h. Avec la loi de Manning-Strickler, l'équation implicite est

$$Q(h) = KR_H(h)^{2/3}S(h)\sqrt{i}. \tag{2}$$

Trois différentes méthodes de résolution sont présentées ici :

- Le calcul itératif (avec une calculatrice)
- La résolution numérique (méthode itérative également) avec du code
- La méthode graphique

Premièrement, la résolution d'une seule équation est présentée ci-dessous avant de s'intéresser aux systèmes d'équations implicites où on ne parlera que de la méthode numérique.

Résolution d'une équation implicite : Calcul d'une hauteur normale

Prenons l'exemple du calcul de la hauteur normale pour une section trapézoïdale avec une pente des berges à 45°, de largeur de fond $b = 5\,\mathrm{m}$, de rugosité $K = 40\,\mathrm{m}^{1/3}\,\mathrm{s}^{-1}$ pour un débit de $100\,\mathrm{m}^3\,\mathrm{s}^{-1}$ et une pente i de 0,17 %. La section est dessinée sur la figure 1.

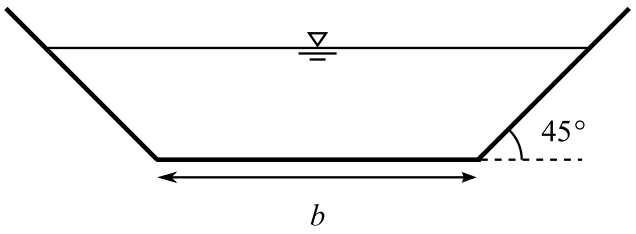

FIGURE 1 – Coupe en travers du canal

La section et le périmètre mouillés s'expriment comme :

$$S = (b + h)h, \quad P = b + 2\sqrt{2}h \tag{3}$$

En substituant les expressions du périmètre et de la section mouillée dans l'équation 4, on arrive à l'équation implicite à résoudre.

$$Q = K\frac{[(b + h_n)h_n]^{5/3}}{(b + 2\sqrt{2}h_n)^{2/3}}\sqrt{i} \tag{4}$$

Calcul itératif

Commençons par exprimer h_n sans l'isoler complètement afin d'avoir équation de la forme $h_n = f(h_n)$:

$$h_n = \frac{1}{b + h_n}\left(\frac{Q}{K\sqrt{i}}\right)^{3/5}\left(b + 2\sqrt{2}h_n\right)^{2/5} \tag{5}$$

Pour la résoudre, on va examiner la suite

$$h_{i+1} = \frac{1}{b + h_i}\left(\frac{Q}{K\sqrt{i}}\right)^{3/5}\left(b + 2\sqrt{2}h_i\right)^{2/5} \tag{6}$$

qui peut soit diverger, soit converger vers un point fixe qui sera une solution. Cela dépend de la forme de l'équation et du choix de la valeur initiale qui doit être proche de la solution réelle. Si la fonction f a une dérivée continue et $|f'(h_n)| < 1$, le point fixe est *attractif* et la suite converge [1].

Pour trouver une bonne solution initiale dans le cas général d'une fonction $g(x)$ monotone autour de sa racine, trouver deux valeurs x_1 et x_2 (que l'on pense proches de la solution) où

$$g(x_1) < 0 < g(x_2) \tag{7}$$

Offre de grandes chances que la racine soit comprise entre x_1 et x_2. Dans le cas $h_n = f(h_n)$, on cherche donc $h_{n1} - f(h_{n1}) < 0 < h_{n2} - f(h_{n2})$.

```
>>> b = 5
>>> Q = 100
>>> K = 40
>>> i = 0.17/100
>>> hn1 = 1
>>> hn1 - 1/(b+hn1)*(Q/K/i^0.5)^(3/5)*(b+2^(3/2)*hn1)^(2/5)
-3.456027717969377
>>> hn2 = 10
>>> hn2 - 1/(b+hn2)*(Q/K/i^0.5)^(3/5)*(b+2^(3/2)*hn2)^(2/5)
6.819951886864983
```

Les conditions sur h_{n1} et h_{n2} sont bien remplies et on peut choisir la solution initiale quelque part entre 1 et 10, prenons $h_n = 5\,\text{m}$. En injectant cette nouvelle valeur dans l'équation 6, on obtient une nouvelle hauteur normale.

```
>>> 5.
>>> 1/(b+ANS)*(Q/K/i^0.5)^(3/5)*(b+2^(3/2)*ANS)^(2/5)
3.8231865032854824
```

Où ANS est la réponse précédente. Cette nouvelle hauteur normale $h_n = 3{,}82\,\text{m}$ est très différente de la valeur précédente $h_n = h_1 = 5\,\text{m}$, ce qui veut dire que la suite n'a pas encore convergé. Il faut réitérer le calcul jusqu'à ce que la différence entre deux résultats consécutifs soit infime.

```
>>> 1/(b+ANS)*(Q/K/i^0.5)^(3/5)*(b+2^(3/2)*ANS)^(2/5)
3.9098553659096287
>>> 1/(b+ANS)*(Q/K/i^0.5)^(3/5)*(b+2^(3/2)*ANS)^(2/5)
4.014359084566256
>>> 1/(b+ANS)*(Q/K/i^0.5)^(3/5)*(b+2^(3/2)*ANS)^(2/5)
3.9824239743546506
>>> 1/(b+ANS)*(Q/K/i^0.5)^(3/5)*(b+2^(3/2)*ANS)^(2/5)
3.9877385996424817
>>> 1/(b+ANS)*(Q/K/i^0.5)^(3/5)*(b+2^(3/2)*ANS)^(2/5)
3.986853561962517
```

Le résultat ne varie plus beaucoup, il a donc convergé et la hauteur normale vaut finalement $h_n = 3{,}99\,\text{m}$.

1. Selon le théorème du point fixe (de Banach), `https://fr.wikipedia.org/wiki/Point_fixe`

On peut aussi employer la dichotomie ou la méthode de Newton. Elles sont décrites dans le GitHub du cours : `https://github.com/cancey/introduction-hydraulique/blob/main/1_hauteur-normale.ipynb`.

Solveur numérique

Il existe de nombreux de solveurs d'équations non linéaires dans différents langages de programmation (Mathematica, MATLAB, Python, etc.). On donne ici un exemple en Python avec la librairie `scipy` dont le module `optimize` contient une pléthore de fonctions utiles à notre problème. On peut notamment employer la méthode de `newton` pour résoudre l'équation 4 [2].

Il faut commencer par importer la fonction d'optimisation et initialiser les variables.

```
from scipy.optimize import newton
b = 5.
K = 40.
Q = 100.
i = 0.17/100
```

Il faut ensuite définir la fonction dont on cherche la racine.

```
def gms_root(hn):
    """Formule de (Gauckler-)Manning-Strickler"""
    S = (b+hn)*hn
    P = b + 2*2**0.5*hn
    return (K * (S/P)**(2/3) * S * i**0.5) - Q  # = 0
```

Puis finalement appliquer la méthode de Newton en y insérant une solution initiale. Cette dernière doit être assez proche de la solution recherchée pour s'assurer de converger vers la bonne solution. Pour une fonction monotone, on peut se référer à l'inégalité 7. Le premier argument à la fonction `newton(fonction_nulle_à_résoudre, solution_initiale)`.

```
hn = newton(gms_root, 5.0)
print(hn)  # 3,99
```

Ce qui donne une hauteur normale $h_n = 3{,}99\,\mathrm{m}$, ce résultat est en accord avec la méthode précédente.

Méthode graphique

C'est la plus simple et la plus sûre quand on sait quelle plage de valeurs observer. De plus, elle permet de voir rapidement s'il existe plusieurs solutions. Il suffit de tracer la fonction $Q(h_n)$, chercher l'intersection avec le débit voulu $Q = 100\,\mathrm{m^3\,s^{-1}}$ et lire l'abscisse correspondante.

2. Une autre méthode populaire de scipy est `fsolve` : `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.root.html#scipy.optimize.root`. Elle permet également de résoudre un système d'équations.

```python
import numpy as np
from matplotlib import pyplot as plt

b = 5.
K = 40.
Q_cible = 100.
i = 0.17/100
hn = np.linspace(0, 10, num=100)
# hn = np.logspace(-1, 1, num=100)

S = (b+hn)*hn
P = b + 2*2**0.5*hn
Q = K*(S/P)**(2/3)*S*i**0.5

plt.axline((hn[0], Q_cible), slope=0, ls="-.", label=r"$Q=100$ m$^\mathrm{3}$/s")
plt.axline((3.99, Q_cible), slope=float("inf"), ls=":", label="$h_n=3.99$ m")
plt.plot(hn, Q, label="GMS")
plt.xlabel("$h_n$")
plt.ylabel("$Q$")
plt.legend()
plt.show()
```

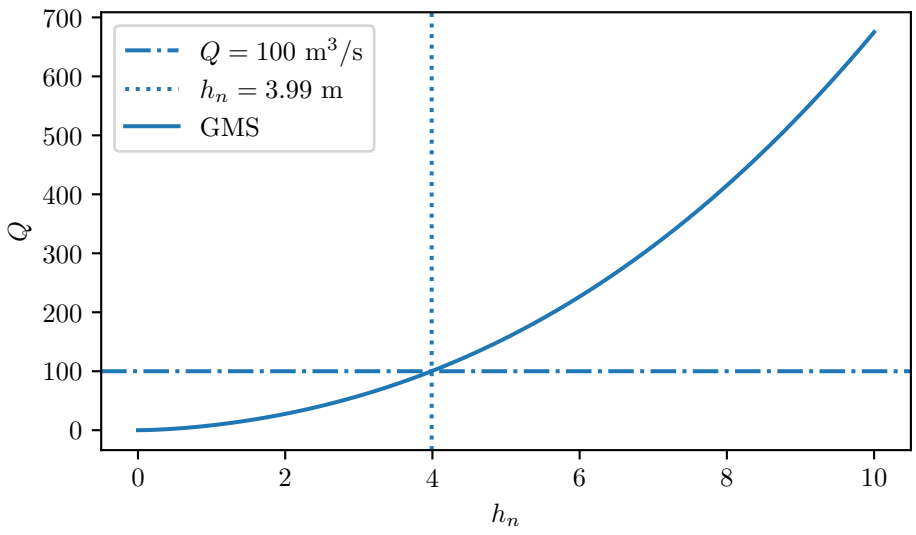

FIGURE 2 – Courbe $Q(h_n)$ pour le canal trapézoïdal.

Le croisement des lignes sur la figure 2 marque la solution et est en accord avec les résultats des méthodes précédentes.

Résolution d'une équation implicite : Hauteur d'eau à partir de la charge.

Quand on veut déterminer la hauteur d'eau après un ressaut ou suite à un seuil, une solution est de partir de la charge connue à l'amont. On se retrouve alors dans le cas de l'équation 1. En y injectant l'expression de la section mouillée en 3, l'équation à résoudre est

$$H = z + h + \frac{Q^2}{2(b+h)^2 h^2 g}. \tag{8}$$

L'importance de la solution initiale

Cette équations a de multiples solutions dont deux sont physiques ($h > 0$). On convergera vers une solution ou vers une autre selon le choix de la solution initiale. Dans le cas du seuil, la solution obtenue est subcritique quand la solution initiale est supérieure à la hauteur critique (minimum de charge spécifique) et supercritique sinon (voir figure 3).

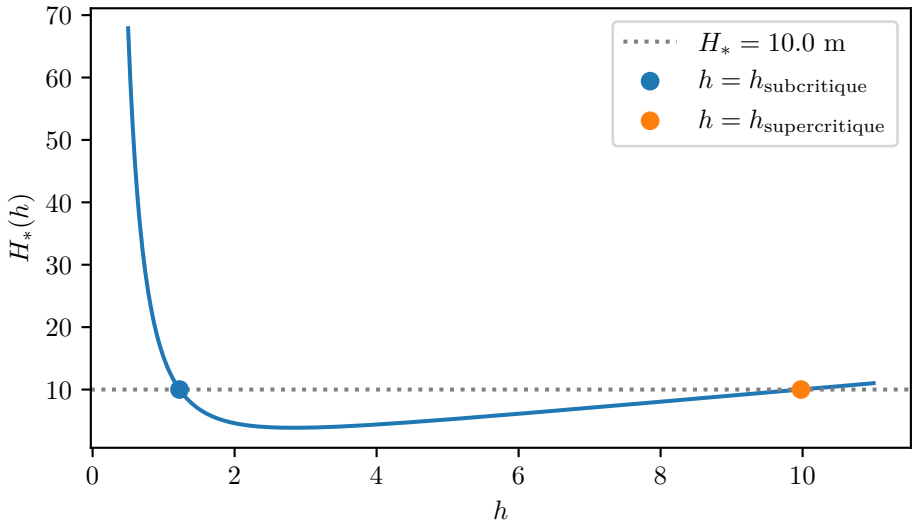

FIGURE 3 – Charge spécifique $H_* = H - z$ en fonction de h. Deux solutions sont obtenues pour deux solutions initiales.

On trouve que pour une charge $H = 10\,\text{m}$, la hauteur peut être de $h = 1{,}22\,\text{m}$ en régime supercritique ou $h = 9{,}98\,\text{m}$ régime subcritique.

Point fixe, dérivée et convergence

Même si le point fixe (solution à $h = f(h)$) à une équation existe, la méthode itérative n'y convergera pas pour toute fonction f car, d'après le théorème du point fixe, il faut que $|f'(h)| < 1$. Essayons d'isoler un h dans l'équation 8, c'est une équation du cinquième degré :

$$(b+h)^2h^3 - (b+h)^2h^2H_* + \frac{Q^2}{2g} = 0, \tag{9}$$

$$\Rightarrow h = \frac{Q}{(b+h_i)\sqrt{2g(H-h_i)}}. \tag{10}$$

Si on essaie de résoudre l'équation 10 par la méthode itérative, on ne trouvera que la solution supercritique car le calcul diverge autour de la solution subcritique. C'est parce que la dérivée de f' est trop grande au voisinage de $h_{\text{subcritique}}$.

Par exemple, avec une solution initiale $h = 1\,\text{m}$,

```
>>> b = 5
>>> H_ = 10.
>>> Q = 100.
>>> g = 9.81
>>> 1.
1.0
>>> Q/(b+_) / (2*g*(H_-_))**0.5
1.2542323360714747
>>> Q/(b+_) / (2*g*(H_-_))**0.5
1.2206116757663512
>>> Q/(b+_) / (2*g*(H_-_))**0.5
1.224856687932349
>>> Q/(b+_) / (2*g*(H_-_))**0.5
1.2243174295231
>>> Q/(b+_) / (2*g*(H_-_))**0.5
1.2243858806671144
```

La solution supercritique $h = 1{,}22\,\text{m}$ est obtenue. Alors que pour une solution initiale $h = 9{,}977\,174\,2\,\text{m}$ (c'est-à-dire extrêmement proche de la solution subcritique),

```
>>> 9.9771742
9.9771742
>>> Q/(b+_) / (2*g*(H_-_))**0.5
9.977175544777532
>>> Q/(b+_) / (2*g*(H_-_))**0.5
9.977468563573513
>>> Q/(b+_) / (2*g*(H_-_))**0.5
10.041940455575565
>>> Q/(b+_) / (2*g*(H_-_))**0.5
(4.487566708955747e-16-7.328752603738753j)
```

On s'éloigne de plus en plus de la solution. Pourtant, le point fixe subcritique existe bel et bien :

```
>>> h = 9.977174193799552
>>> Q/(b+h) / (2*g*(H_-h))^0.5
9.97717419379785
```

On dit que le point fixe supercritique est *attractif* alors que le point fixe subcritique ne l'est pas. Selon la manière d'isoler un des termes h, l'attractivité des points peut changer.

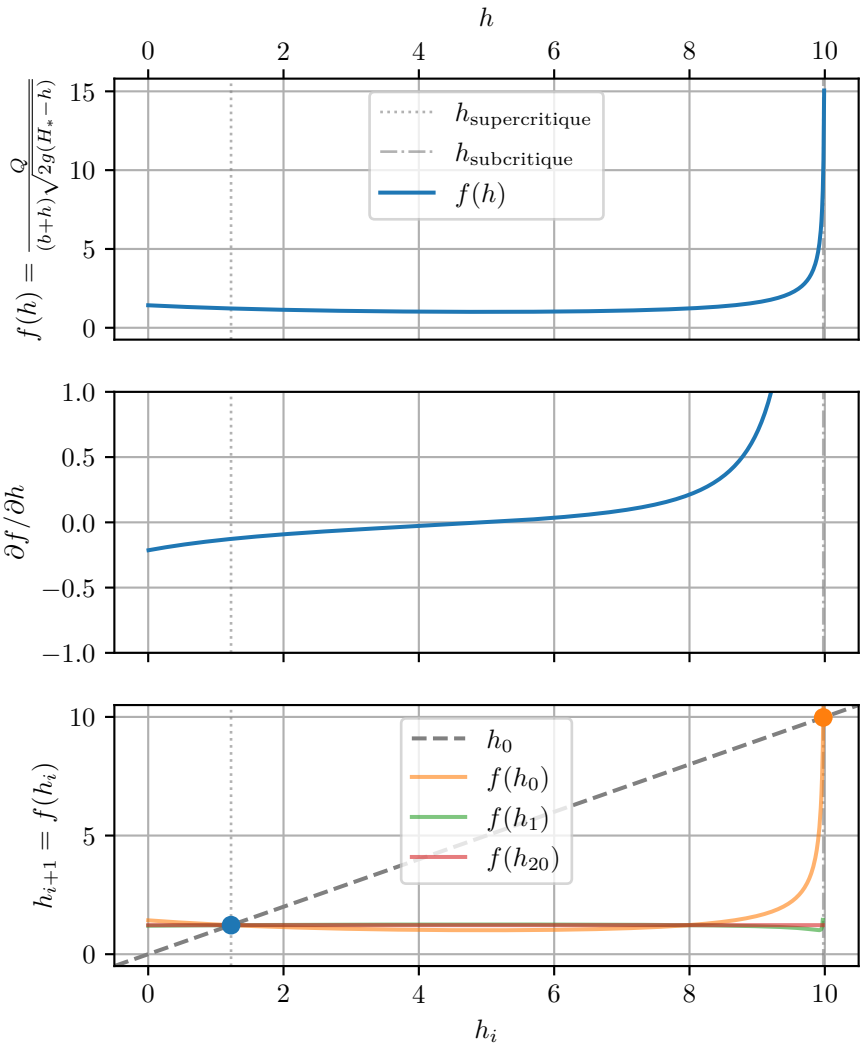

FIGURE 4 – Graphique de la fonction récursive (en haut), sa dérivée (au milieu) et diagramme montrant l'attractivité du point fixe supercritque (en bas).

Résolution d'un système d'équations implicites : Conduites en parallèle

Un glissement de terrain détruit le château d'eau d'un village. Pour avoir accès à l'eau potable, le système est raccordé à celui d'un autre village plus en amont. Deux conduites sont alors en parallèle comme sur la figure 5.

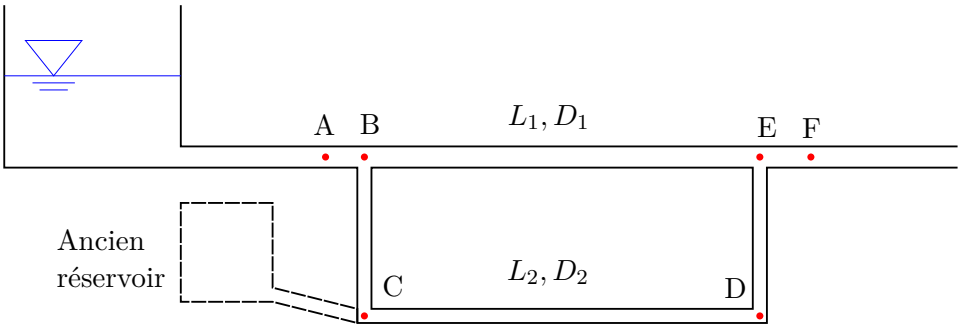

FIGURE 5 – Deux réservoirs connectés à un lac via des conduites en charge.

Connaissant la chute de charge hydraulique ΔH entre le point A et le point F, on cherche à connaitre le débit qui s'écoule naturellement (lorsqu'on ne prélève pas d'eau entre A et F). Pour cela, il faut exprimer la conservation de la charge par les deux conduites. Un système de deux équations implicites se forme quand les pertes de charges singulières et régulières sont prises en compte :

$$\Delta H = \Delta H_{ABEF} = 2\zeta_e \frac{u_A^2}{2g} + f_1 \frac{L_1}{D_1} \frac{u^2}{2g}, \tag{11}$$

$$\Delta H = \Delta H_{ABEF} = 2\zeta_e \frac{u_A^2}{2g} + \left(2\zeta_c + f_2 \frac{L_2}{D_2}\right) \frac{u^2}{2g}, \tag{12}$$

$$\frac{1}{\sqrt{f_i}} = -0,91 \ln\left(0,27 \frac{k_s}{D_i} + \frac{2,51\nu}{\sqrt{f_i} u_i D_i}\right), \quad i = 1,2. \tag{13}$$

Où $u_A = u_1 + u_2 \frac{D_2^2}{D_1^2}$ est la vitesse dans la conduite simple, ζ_e et ζ_c sont les coefficients de perte de charge singulière pour les embranchements et les coudes respectivement. On prendra $\zeta_e = 1,3$ et $\zeta_c = 1,0$. Pour réduire le nombre d'équations, il est possible d'isoler u_i dans la formule de Colebrook (13) et d'injecter l'expression dans les conservations de la charge (11 et 12). On obtient ainsi

$$\Delta H - 2\zeta_e \frac{\left[C(f_1) + C(f_2) D_2^2 / D_1^2\right]^2}{2g} - f_1 \frac{L_1}{D_1} \frac{C(f_1)^2}{2g} = 0, \tag{14}$$

$$\Delta H - 2\zeta_e \frac{\left[C(f_1) + C(f_2) D_2^2/D_1^2\right]^2}{2g} - \left(2\zeta_c + f_2 \frac{L_2}{D_2}\right) \frac{C(f_2)^2}{2g} = 0, \qquad (15)$$

Où C est l'expression de u_i selon l'équation de Colebrook (13) :

$$u_i = C(f_i) = \frac{\nu}{D_i \sqrt{f_i}} \cdot \frac{2,51}{e^{-1/\left(0,91\sqrt{f_i}\right)} - 0,27 k_s/D_i}, \quad i = 1, 2. \qquad (16)$$

De nombreux solveurs de systèmes d'équations existent, chacun avec une ou plusieurs méthodes numériques différentes (par exemple `root`, `fsolve`, `minimize`, `least_squares`... dans `scipy`). On peut utiliser `least_squares` pour minimiser les fonctions 14 et 15 et préciser l'intervalle de valeurs de f_1 et f_2.

```python
import numpy as np
from scipy.optimize import least_squares

# Paramètres du problème
g = 9.81
L1 = 8e3
L2 = 18
D1 = 2.
D2 = 1.
nu = 1.31e-6
ks = 0.01e-3
DH = 5

# Expression de la vitesse en fonction du coefficient de frottement u=C(f)
def colebrook(f, D):
    return (
        nu/D*2.51/(np.sqrt(f)*(np.exp(-(1/(0.91*np.sqrt(f))))-0.27*ks/D))
    )

# Système d'équations à résoudre
def system(f_vector, D=(D1, D2)):

    # Calcul des vitesses d'écoulement à partir du frottement
    f1, f2 = f_vector
    u1 = colebrook(f1, D1)
    u2 = colebrook(f2, D2)

    # Pertes de charge singulières
    uA = u1 + u2*(D1/D2)**2
    DH_B = 1.3*uA**2/(2*g)
    DH_E = 1.3*uA**2/(2*g)
    DH_C = 1.0*u2**2/(2*g)
    DH_D = 1.0*u2**2/(2*g)

    # Pertes de charge régulières
    DH1 = f1*L1/D1 * u1**2/(2*g) + DH_B+DH_E
    DH2 = f2*L2/D2 * u2**2/(2*g) + DH_B+DH_C+DH_D+DH_E

    return DH1 - DH, DH2 - DH

x0 = 0.09, 0.09  # Solution initiale
result = least_squares(
        system,
        x0=x0,
        bounds=((0.008, 0.008), (0.1, 0.1))
```

```
    )
f1, f2 = result.x
u1 = colebrook(f1, D1)
u2 = colebrook(f2, D2)
print(f"{f1 = :.5f}, {f2 = :.5f}")
print(f"{u1 = :.5f}, {u2 = :.5f}")
```

On peut aussi employer une méthode graphique en traçant les lignes de niveau des surfaces $\Delta H_1(f_1, f_2)$ et $\Delta H_2(f_1, f_2)$ où elles valent ΔH. Il est ainsi plus aisé de trouver la ou les racines du problème comme à la figure 6.

```
from matplotlib import pyplot as plt

f1_sol, f2_sol = f1, f2

# On observe une grande plage de valeurs possibles (abaque de Moody)
f1 = np.logspace(np.log10(0.008), np.log10(0.1), num=1000)
f2 = np.logspace(np.log10(0.008), np.log10(0.1), num=1000)

# Il faut générer une grille pour chaque variable
F1, F2 = np.meshgrid(f1, f2)
DH1, DH2 = np.array(system((F1, F2))) + DH

plt.imshow(DH1, extent=(f1[0], f1[-1],f2[0],f2[-1]), origin="lower")
plt.colorbar()
plt.loglog()
plt.xlabel("$f_1$")
plt.ylabel("$f_2$")
plt.title(r"$\Delta H_1$")
plt.gca().set_aspect("auto")

plt.figure()
plt.imshow(DH2,extent=(f1[0],f1[-1],f2[0],f2[-1]),origin="lower")
plt.colorbar()
plt.loglog()
plt.xlabel("$f_1$")
plt.ylabel("$f_2$")
plt.title(r"$\Delta H_2$")
plt.gca().set_aspect("auto")

plt.figure(layout="tight", figsize=)
plt.scatter(f1_sol, f2_sol, label="Solution", zorder=np.inf)
c1 = plt.contour(F1, F2, DH1, levels=[DH], colors=["r"])
c2 = plt.contour(F1, F2, DH2, levels=[DH], colors=["g"])
plt.clabel(c1, c1.levels, fmt=r"$\Delta H_1 = \Delta H$")
plt.clabel(c2, c2.levels, fmt=r"$\Delta H_2 = \Delta H$")
plt.clabel(c1, c1.levels)
plt.clabel(c2, c2.levels)
plt.loglog()
plt.ylim(1.06e-2, 1.15e-2)
plt.xlabel("$f_1$")
plt.ylabel("$f_2$")
plt.legend()
plt.gca().set_aspect("auto")
plt.show()
```

La démarche pour résoudre des systèmes est plus laborieuse, il vaut mieux réduire le système afin de résoudre un minimum d'équations numériquement.

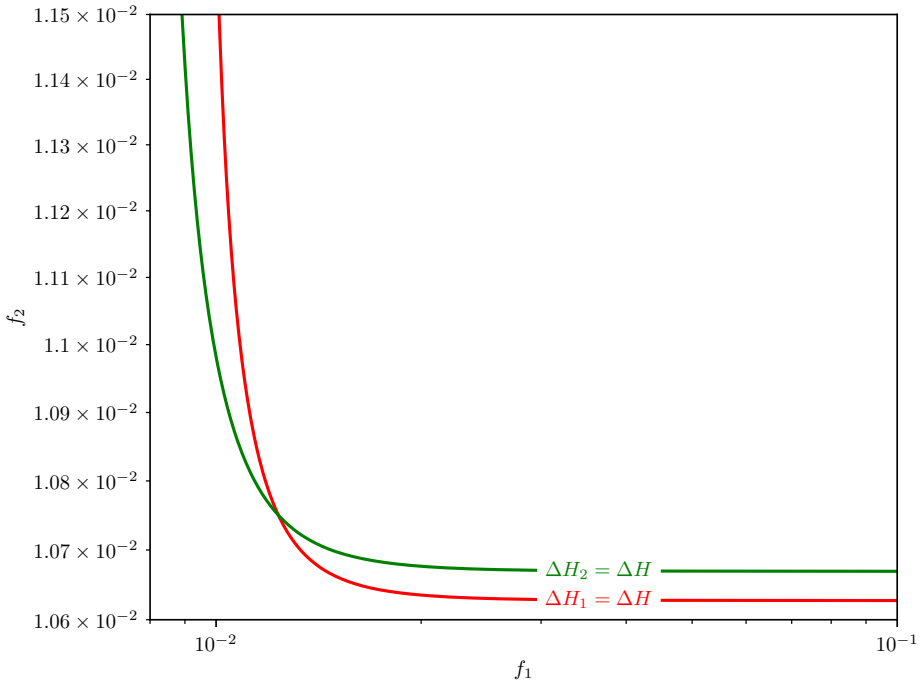

FIGURE 6 – Lignes de niveau des foctions ΔH_1 et ΔH_2. Les deux courbes se croisent en la solution.

Par exemple, en explicitant les variables u_1 et u_2 dans les équations 14 et 15, deux équations sont sorties du système. Réduire le nombre de variables permet également de converger plus sûrement.