Hydraulic engineering and infrastructures

Civil Engineering Section

Solving implicit equations

October 20, 2025

Introduction

An implicit equation $f(x, y, z, \dots) = 0$ is one in which the variable of interest x cannot be isolated — in other words, it cannot be written as $x = g(y, z, \dots)$. It's impossible to directly compute x.

Two common examples in hydraulics are determining the water depth y for a given total head H and computing the normal depth y_n. For example, we seek to solve:

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

where the only unknown is y. Isolating y is only possible if the form of $S(y)$ allows it. Otherwise, this implicit equation must be solved numerically. The second case involves (Gauckler-)Manning(–Strickler)'s law:

$$Q(y) = \frac{1}{n} R_H(y)^{2/3} S(y) \sqrt{i}. \tag{2}$$

Three different solving methods are presented:

- Iterative calculation (by calculator)
- Numerical solving (also iterative) using code
- Graphical method

The resolution of a single implicit equation is presented first, followed by systems of implicit equations, for which only numerical methods are discussed.

Solving a Single Implicit Equation: Normal Depth Calculation

Example: compute the normal depth for a trapezoidal channel with 45° side slopes, bottom width $b = 5\,\mathrm{m}$, roughness $n = 0.025\,\mathrm{s/m^{1/3}}$, discharge $Q = 100\,\mathrm{m^3/s}$, and slope $S = 0.17\,\%$. A cut of the profile is drawn on Figure 1.

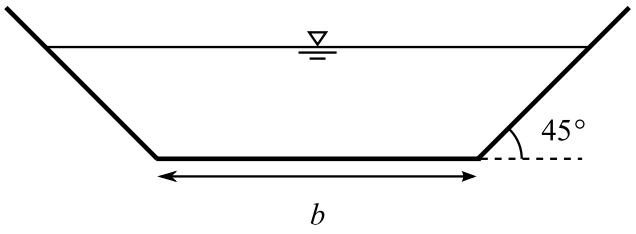

Figure 1: Cross-section of the channel

The wet perimeter and area are expressed as follows :

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

Substituting into Manning–Strickler's Equation 4 gives.

$$Q = \frac{1}{n}\frac{[(b + y_n)y_n]^{5/3}}{(b + 2\sqrt{2}y_n)^{2/3}}\sqrt{S} \tag{4}$$

Iterative calculation with a simple calculator

Let's start by rewriting y_n as a function of itself ($y_n = f(y_n)$):

$$y_n = \frac{1}{b + y_n}\left(\frac{nQ}{\sqrt{S}}\right)^{3/5}\left(b + 2\sqrt{2}y_n\right)^{2/5} \tag{5}$$

To solve this equation, we can examine the following sequence :

$$y_{i+1} = \frac{1}{b + y_i}\left(\frac{nQ}{\sqrt{S}}\right)^{3/5}\left(b + 2\sqrt{2}y_i\right)^{2/5} \tag{6}$$

This sequence may converge to a fixed point (the solution) or diverge depending on the initial value. If f has a continuous derivative and $|f'(y_n)| < 1$, the fixed point is attractive and the sequence converges.

A good starting point is found by identifying two values x_1 and x_2 such that :

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

This ensures that a solution lies between x_1 and x_2. In the case of $y_n = f(y_n)$, we're looking for $y_{n1} - f(y_{n1}) < 0 < y_{n2} - f(y_{n2})$.

```
>>> b = 5
>>> Q = 100
>>> n = 0.025
>>> S = 0.17/100
>>> yn1 = 1
>>> yn1 - 1/(b+yn1)*(n*Q/S^0.5)^(3/5)*(b+2^(3/2)*yn1)^(2/5)
-3.456027717969377
>>> yn2 = 10
>>> yn2 - 1/(b+yn2)*(n*Q/S^0.5)^(3/5)*(b+2^(3/2)*yn2)^(2/5)
6.819951886864983
```

The conditions over y_{n1} and y_{n2} are respected and the initial solution can be set somewhere in between. Let's take $y_n = 5\,\mathrm{m}$. Inserting this new value in Equation 6, a new normal depth is obtained.

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

Where ANS is the previous result. The new value $y_n = 3.82\,\mathrm{m}$ is very different from the former value, meaning that the sequence has not yet converged. The computation needs some more iterations to minimize that gap between consecutive values.

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

The result has stabilized and the final normal depth is $y_n = 3.99\,\mathrm{m}$.

Several other methods can be used to solve those equations such as Newton-Raphson's method or dichotomy.

Numerical Solver

Many nonlinear equation solvers exist in programming languages (Mathematica, MATLAB, Python, etc.). Here is an example in Python, using the `scipy` library's optimization module, which includes many functions useful for our problem. We can use Newton's method to solve Equation 4 [1].

First import the optimization function and initialize the variables.

```
from scipy.optimize import newton
b = 5.
n = 0.025
Q = 100.
S = 0.17/100
```

The define the function to solve for.

```
def gms_root(yn):
    """(Gauckler-)Manning(-Strickler) formula"""
    A = (b+yn)*yn
    P = b + 2*2**0.5*yn
    return (A/P)**(2/3) * A * S**0.5 - n*Q  # = 0
```

And finally apply Newton's method with an initial solution. It should sufficiently close to the solution to ensure the correct solution is found. To that end, we can refer to the inequality 7 for monotonous functions. The arguments to the function are `newton(null_function_to_solve, initial_solution)`.

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

Which yields a normal depth $y_n = 3.99\,\mathrm{m}$, agreeing with the results from the previous method.

Graphical Method

This is the simplest and safest method when we know the approximate range of values. It also helps us see whether multiple solutions exist. We simply plot the function $Q(y_n)$ and find where it intersects the desired discharge $Q = 100\,\mathrm{m}^3/\mathrm{s}$. The corresponding abscissa gives y_n.

```
#!/usr/bin/env python
import numpy as np
from matplotlib import pyplot as plt

g = 9.81
b = 5.
n = 0.025
Q_target = 100.
S = 0.17/100
yn = np.linspace(1e-1, 10, num=10_0000)

B = b + 2*yn
```

[1] Another popular `SciPy` method is `fsolve`, which can also solve systems of equations.

```
A = (b+yn)*yn
P = b + 2*2**0.5*yn
Q = 1/n*(A/P)**(2/3)*A*S**0.5
Qcr = S*np.sqrt(g*A/B)

ynQ = yn[np.abs(Q-Q_target).argmin()]

plt.figure(figsize=(5, 3))
plt.axline((Q_target, yn[0]), slope=np.inf, ls="-.", label=r"$Q=100$ m$^\
    mathrm{3}$/s")
plt.axline((Q_target, ynQ), slope=0, ls=":", label=f"$y_n={ynQ:.2f}$ m")
plt.plot(Q, yn, label="GMS")
plt.ylabel("$y_n$")
plt.xlabel("$Q$")
plt.legend()
plt.show()
```

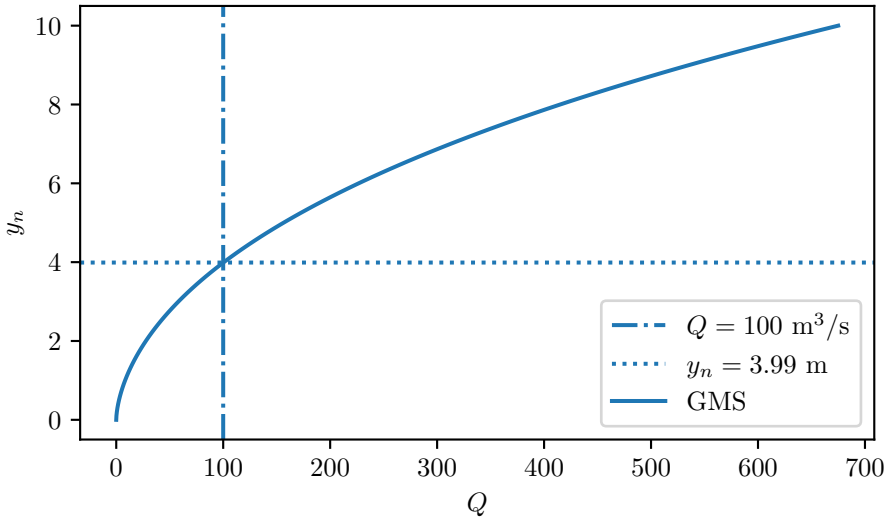

Figure 2: $Q(y_n)$ curve for the trapezoidal channel.

The intersection point in the plot (Figure 2) gives the solution $y_n = 3.99\,\mathrm{m}$, matching the previous methods.

Solving an Implicit Equation: Water Depth from specific energy

When we want to determine the water depth y after a hydraulic jump or over a weir, we can start from the known upstream total head H. The equation to solve (using Equation 3 for the cross-section) becomes Equation 1:

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

Importance of the Initial Value

This equation can have multiple solutions, two of which are physically meaningful ($y > 0$). The result we converge to depends on the initial guess. For a weir, the solution is supercritical when the initial solution is below the critical depth and subcritical if not (see Figure 3).

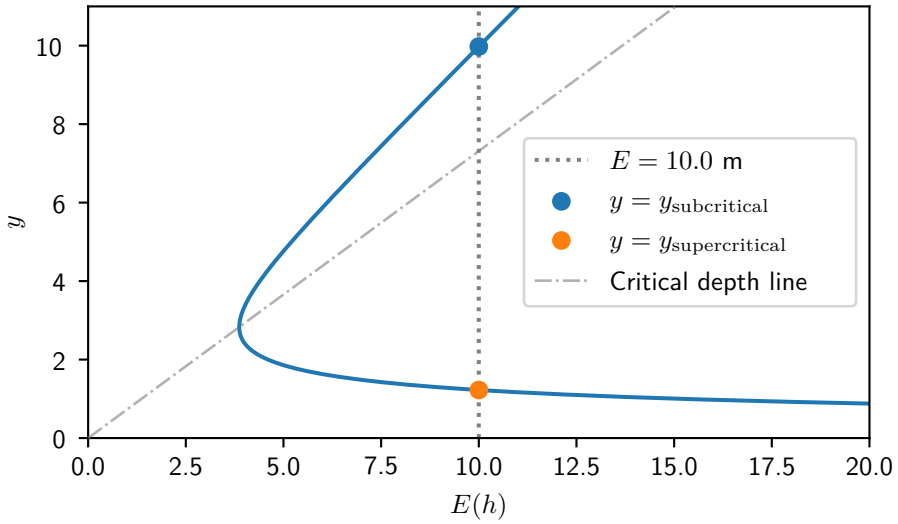

Figure 3: Specific energy $E(y) = H(y) - z$ as a function of y. Changing the initial solution yielded a different solution.

We finally find that for a specific energy of $H(y) = 10\,\mathrm{m}$, the flow depth can either be $y = 1.22\,\mathrm{m}$ (supercritical) or $y = 9.98\,\mathrm{m}$ (subcritical).

Fixed Point, Derivative, and Convergence

Even though a fixed point (solution to $y = f(y)$) exists, the iterative methods do not always converge. This depends on the derivative of the function. According to Banach's fixed-point theorem, the convergence requires the derivative to be smaller than unity ($|f'(y)| < 1$).

Rewriting Equation 8 yields a fifth-degree polynomial :

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

Which can be rearranged to :

$$\Rightarrow y = \frac{Q}{(b+y_i)\sqrt{2g(E-y_i)}}. \tag{10}$$

When trying to solve this equation with the iterative method, only the supercritical solution can be found. This is because the derivative f' of the function around the subcritical solution $y_{\text{subcritical}}$ is too large.

For example, with the initial solution $y = 1\,\text{m}$,

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

The supercritical solution $y = 1.22\,\text{m}$ is obtained. On the other hand, taking the initial value $y = 9.977\,174\,2\,\text{m}$ (extremely close to the subcritical solution),

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

The value of y moves away from the solution, even though the fixed-point exists :

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

The supercritical fixed-point is qualified as *attractive* while the subcritical one is not. The attractiveness of a fixed-point depends on the function f (so it depends on how we isolated the unknown y) through the magnitude of the derivative f'.

Solving a System of Implicit Equations: Parallel Pipes

A landslide destroys a village's water tank. To supply drinking water, the system is connected to another upstream reservoir, resulting in two parallel pipes, as shown in Figure 5.

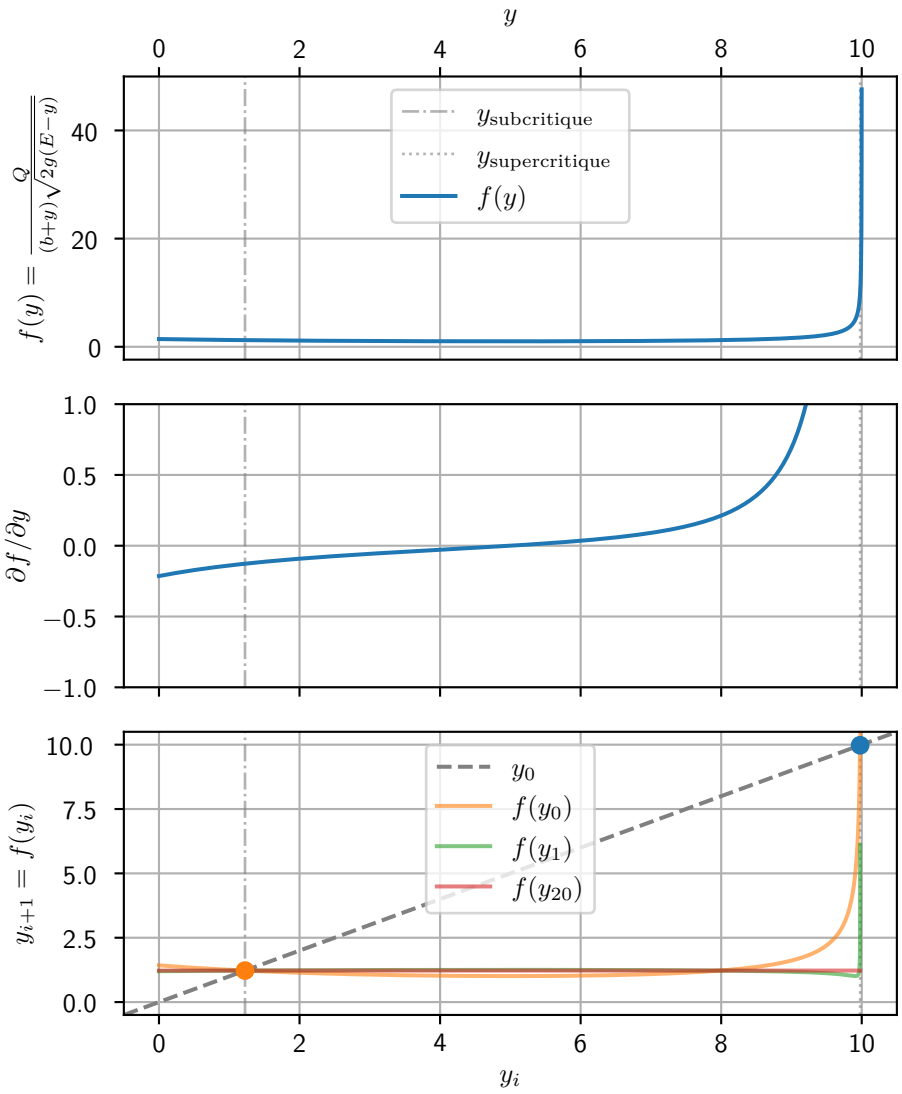

Figure 4: Plot of the recursive function (on top), its derivative (middle) and attractiveness of the points (bottom) where all points above $y_{\text{subcritical}}$ diverge while the rest converge to $y_{\text{supercritical}}$.

Knowing the head loss ΔH between points A and F, we want to find the natural discharge (no withdrawal between A and F).

We express the head loss along both paths; a system of two implicit equations arises when both major and minor losses are included:

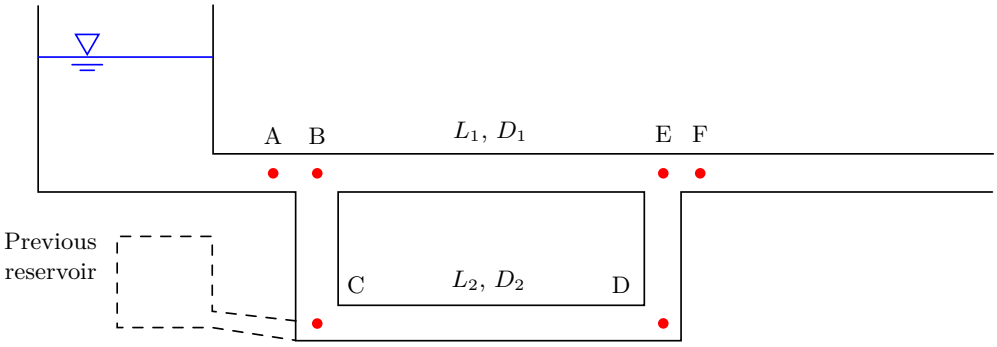

Figure 5: Two reservoirs connected by a branched pipe.

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

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

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

Where $V_A = V_1 + V_2(D_2^2/D_1^2)$ is the flow speed in the single pipe, $K_e = 1.3$ and $K_c = 1.0$ are the minor head loss coefficients for branching and elbows. The system of equations is then as follows :

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

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

Where C is a shorthand notation for the flow speed found through Colebrook's equation (13):

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

Many solvers for systems of equations exist and use different methods (`root`, `fsolve`, `minimize`, `least_squares`... are examples from `scipy`). Let's use `least_squares` to take advantage of the bounds support and minimize Equation 14 along with Equation 15.

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

# Problem's parameters
g = 9.81
L1 = 8e3
L2 = 18
D1 = 2.
D2 = 1.
nu = 1.31e-6
eps = 0.01e-3
DH = 5

# Expressing the flow speed as a function of the friction coefficient 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*eps/D))
    )

# System of equations to solve
def system(f_vector, D=(D1, D2)):

    # Computing the flow speeds
    f1, f2 = f_vector
    V1 = colebrook(f1, D1)
    V2 = colebrook(f2, D2)

    # Minor head losses
    uA = V1 + V2*(D1/D2)**2
    DH_B = 1.3*uA**2/(2*g)
    DH_E = 1.3*uA**2/(2*g)
    DH_C = 1.0*V2**2/(2*g)
    DH_D = 1.0*V2**2/(2*g)

    # Major head losses
    DH1 = f1*L1/D1 * V1**2/(2*g) + DH_B+DH_E
    DH2 = f2*L2/D2 * V2**2/(2*g) + DH_B+DH_C+DH_D+DH_E

    return DH1 - DH, DH2 - DH

x0 = 0.09, 0.09  # Initial solution
result = least_squares(
        system,
        x0=x0,
        bounds=((0.008, 0.008), (0.1, 0.1))
    )
f1, f2 = result.x
V1 = colebrook(f1, D1)
V2 = colebrook(f2, D2)
print(f"{f1 = :.5f}, {f2 = :.5f}")  # f1 = 0.01234, f2 = 0.01075
print(f"{V1 = :.5f}, {V2 = :.5f}")  # V1 = 0.30042, V2 = 1.42525
```

It is also possible to draw contours and thus find the solution(s) as shown in Figure 6.

```python
from system2 import f1 as f1_sol, f2 as f2_sol, DH, system
import numpy as np
from matplotlib import pyplot as plt

# We test out a wide range of values (Moody diagram)
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)

# A grid of values has to be generated for both variables
F1, F2 = np.meshgrid(f1, f2)
DH1, DH2 = np.array(system((F1, F2))) + DH

plt.figure(layout="tight")
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, manual=[(0.0309, 0.01067)], fmt=r"$\Delta H_1 =
    \Delta H$")
plt.clabel(c2, c2.levels, manual=[(0.0309, 0.01063)], fmt=r"$\Delta H_2 =
    \Delta H$")
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()
```

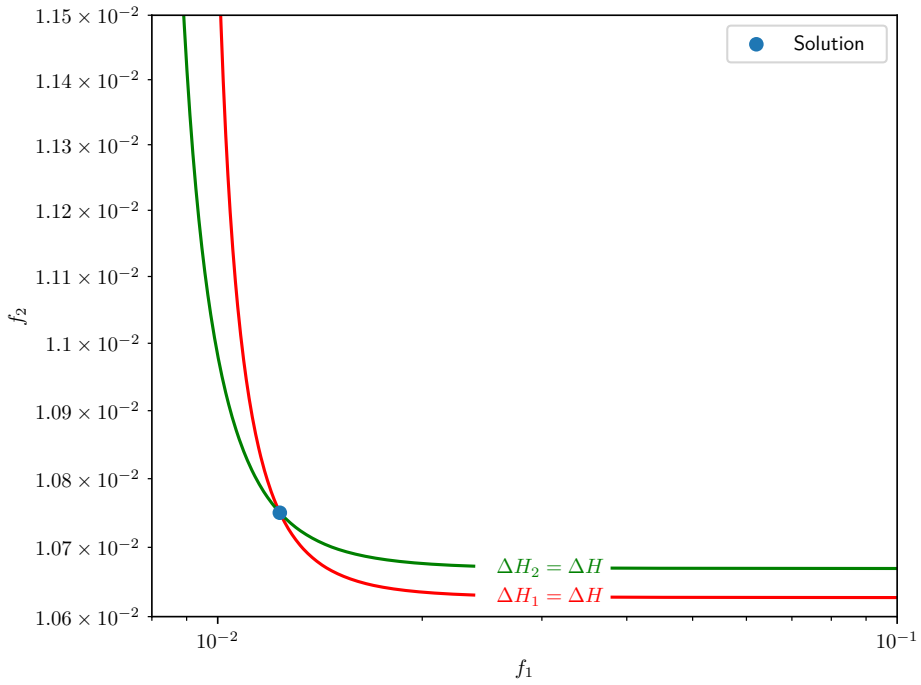

Figure 6: Contour lines for functions ΔH_1 and ΔH_2. Both curves meet at solutions of the problem.

Solving systems of equations can be computationally demanding As a first step, it is a good idea to check whether the system can be reduced. For example, inserting Equation 16 into 14 and 15 removes two equations from the system and convergence is obtained much quicker.