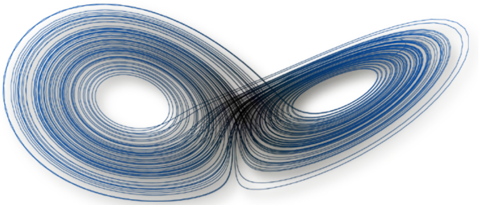


Introduction au calcul scientifique

Jean-Christophe Loiseau

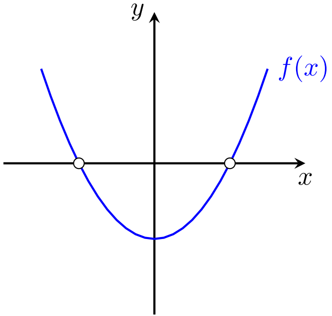
jean-christophe.loiseau@ensam.eu
Laboratoire DynFluid
Arts et Métiers, France.

Recherche de racines

De nombreux problèmes d'ingénierie nécessitent de trouver les valeurs de x telles que

$$f(x) = 0$$

avec $x \in \mathbb{R}$ et $f : \mathbb{R} \to \mathbb{R}$. Les valeurs de x satisfaisant cette équations ont appelées les **racines de la fonction f**.



Recherche de racines

Quelques exemples

Les pertes de charge dans une conduite peuvent être modélisées par

$$\Delta p = f_D \frac{\rho V^2}{2} \frac{L}{D}$$

où f_D est la coefficient de friction de Darcy-Weisbach permettant de prendre en compte la rugosité des parois des la conduite.

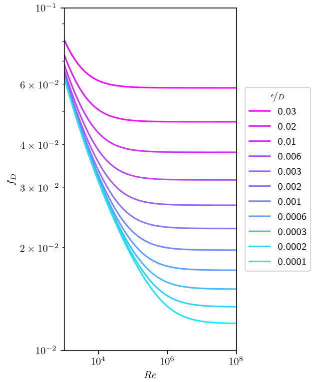
Recherche de racines

Quelques exemples

Pour une taille caractéristique ϵ des rugosités, ce coefficent est solution de l'équation empirique de Colebrook

$$\frac{1}{\sqrt{f_D}} = 2\log_{10}\left(\frac{\epsilon}{3.7D} + \frac{2.51}{Re\sqrt{f_D}}\right).$$

Cette équation n'admet pas de solution analytique. Ses racines doivent alors être calculées numériquement.



Recherche de racines

Quelques exemples

Une possible équation d'état pour un gaz non-idéal est donnée par celle de van der Walls

$$p - \frac{RT}{V-b} + \frac{a}{V^2} = 0$$

En deça d'une température critique, un équilibre liquide-vapeur peut exister. Le volume molaire V de chaque phase est alors déterminé en calculant les racines de l'équation ci-dessus.

Recherche de racines

Fonctions polynômiales

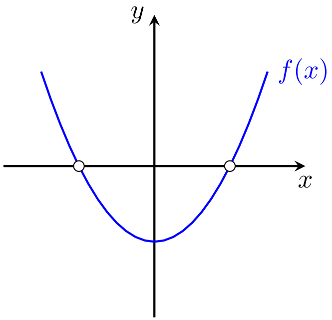
Soit $f : \mathbb{R} \to \mathbb{R}$ une fonction quadratique

$$f(x) = x^2 + ax + b$$

avec a et b des constantes (réelles ou complexes). Ses racines sont données par

$$x = \frac{-a \pm \sqrt{\Delta}}{2}$$

avec $\Delta = a^2 - 4b$ le discriminant.



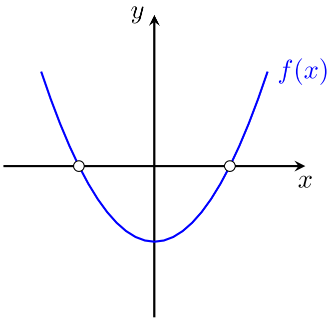
Recherche de racines

Fonctions polynômiales

On peut montrer que les racines de f sont les valeurs propres de la matrice compagnon

$$\boldsymbol{C} = \begin{bmatrix} 0 & -b \\ 1 & -a \end{bmatrix}.$$

Cette équivalence existe pour n'importe quelle fonction polynômiale à une variable.



Recherche de racines

Fonctions polynômiales

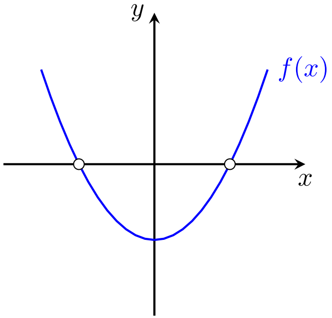
Soit $f : \mathbb{R} \to \mathbb{R}$ une fonction polynomiale donnée par

$$f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1 x + c_0.$$

Ses racines sont les valeurs propres de la matrice

$$\boldsymbol{C} = \begin{bmatrix} 0 & 0 & \cdots & 0 & -c_0 \\ 1 & 0 & \cdots & 0 & -c_1 \\ 0 & 1 & \cdots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -c_{n-1} \end{bmatrix}.$$

Il est important de noter que l'on suppose $c_n = 1$.



Recherche de racines

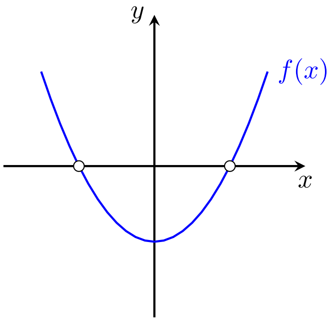
Fonctions polynômiales

```python
from numpy.polynomial import Polynomial

# --> Coefficients du polynome f(x) = x**2 - 1.
coefs = [-1, 0, 1]

# --> Creation du polynome.
p = Polynomial(coefs)

# --> Calcul des racines.
roots = p.roots()
```



Recherche de racines

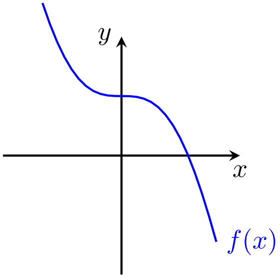
Fonctions non-polynômiales

Pour une fonction $f : \mathbb{R} \to \mathbb{R}$ non-polynômiale, il est rare que les racines puissent être écrites de façon analytique. Il est alors nécessaire de recourir à des méthodes numériques pour les approximer.

Exemple : Trouvez les racines de la fonction

$$f(x) = a - x + b\sin(x)$$

avec a et b des constantes données par le problème.

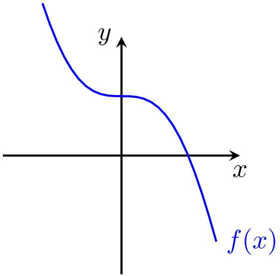


Recherche de racines

Graphe de la fonction

Pour une fonction $f : \mathbb{R} \to \mathbb{R}$, l'approche la plus simple pour voir si il existe des racines dans l'intervalle $[a, b]$ consiste à tracer le graphe de la fonction $f(x)$.

Cette méthode ne permet pas techniquement de calculer les racines mais uniquement de visualiser si il en existe. Elle nous permet donc d'estimer *a priori* combien de racines il y'a et quelles sont leurs valeurs.



Recherche de racines

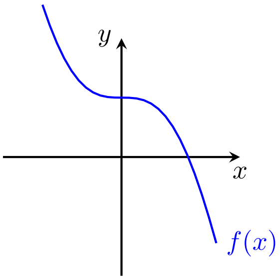
Graphe de la fonction

```
import numpy as np
import matplotlib.pyplot as plt

# --> Definition de la fonction
f = lambda x: np.pi/4 - x + np.sin(x)

# --> Definition de l'intervalle
a, b = -np.pi, np.pi
x = np.linspace(a, b, 1024)

# --> Trace le graphe de la fonction.
plt.plot(x, f(x))
plt.axhline(0, c='black') # Ligne y=0.
plt.show()
```



Recherche de racines

Méthodes numériques

Il existe de nombreux algorithmes pour calculer les racines d'une fonction $f : \mathbb{R} \to \mathbb{R}$. Chacun d'eux repose sur différentes hypothèses concernant f (e.g. continuité, dérivabilité, etc) conduisant à plus ou moins d'efficacité et/ou de robustesse.

Recherche de racines

Méthodes numériques

Dans la suite, nous discuterons des trois méthodes les plus populaires, à savoir :

- la méthode de la bissection (ou dichotomie),
- la méthode de Newton-Raphson,
- la méthode de la sécante.

Elles sont toutes disponibles dans `scipy.optimize` via l'interface unifiée `root_scalar`.

Recherche de racines

Méthode de la bissection

Théorème de Bolzano : Pour toute fonction continue $f : [a, b] \to \mathbb{R}$ telle que $f(a)$ et $f(b)$ soient de signe opposé, il existe au moins un réel $c \in [a, b]$ tel que $f(c) = 0$.

Recherche de racines

Méthode de la bissection

Supposons que les conditions pour que le théorème de Bolzano soit applicable sont vérifiées. Il est alors possible d'approximer numériquement une racine de $f(x)$ à l'aide l'algorithme ci-dessous.

```
while abs(b-a) < tol:
    c = (a+b) / 2
    if (f(a) * f(b) < 0):
        b = c
    else:
        a = c
```

C'est ce que l'on appelle **méthode de la bissection** ou **méthode de la dichotomie**.

Recherche de racines

Méthode de la bissection

A chaque étape, l'erreur d'approximation de la racine est divisée par un facteur 2. Après n itérations, on a donc

$$e_n = \frac{|b-a|}{2^{n+1}}.$$

La méthode de la bissection jouit d'une grande robustesse mais converge **linéairement** et peut donc nécessiter beaucoup d'itérations avant d'arriver au résultat.

Recherche de racines

Méthode de la bissection

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

# --> Definition de la fonction.
f = lambda x : np.pi/4 - x + np.sin(x)

# --> Recherche des racines.
sol = root_scalar(f, bracket=[0, np.pi], method="bisect")

# --> Extrait differentes informations.
x = sol.root # Racine obtenue.
niter = sol.iterations # Nombre d'iterations.
nfev = sol.functions_calls # Nombre d'appels de f(x).
```

La méthode de la bissection repose uniquement sur l'hypothèse de continuité et n'utilise comme information que le signe de $f(x)$ aux bornes des intervalles (d'où sa lente convergence).

Si l'on suppose que $f : \mathbb{R} \to \mathbb{R}$ est également dérivable, alors il est possible de designer un algorithme plus efficace : la **méthode de Newton-Raphson**.

Recherche de racines

Méthode de Newton-Rapshon

Supposons que l'on ait une bonne estimation x_0 de la racine que l'on cherche. Au voisinage de x_0, on peut alors écrire

$$f(x) \simeq f(x_0) + f'(x_0)\,(x - x_0)\,.$$

Posons $g_0(x) = f(x_0) + f'(x_0)\,(x - x_0)$ et cherchons sa racine.

Recherche de racines

Méthode de Newton-Raphson

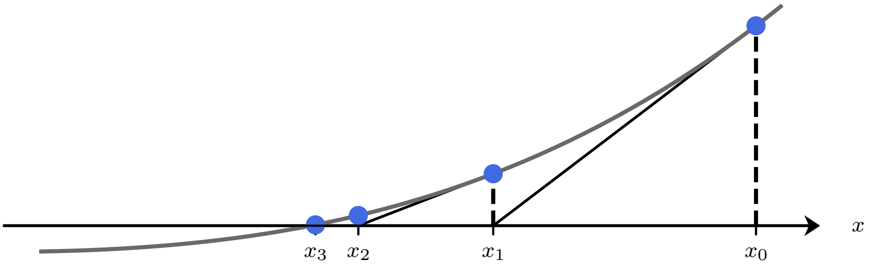
Il est facile de montrer que la racine de $g_0(x)$ est donnée par

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Si x_0 est une bonne estimation de la racine de $f(x)$, alors x_1 tend à être encore meilleure. On peut ensuite répéter ce processus à l'infini et montrer que la convergence est quadratique.

Recherche de racines

Méthode de Newton-Raphson



Recherche de racines

Méthode de Newton-Raphson

Si l'on écrit ce processus en pseudocode, on obtient alors l'algorithme ci-dessous.

```
while abs(f(x)) < tol:
    x = x - f(x) / fprime(x)

```

En pratique, le code est légèrement plus compliqué puisque l'on doit vérifier que $f'(x)$ n'est pas nulle. On peut également ajouter une limite au nombre d'itérations.

Recherche de racines

Méthode de Newton-Raphson

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

# --> Definition de la fonction et de sa derivee.
f = lambda x : np.pi/4 - x + np.sin(x)
df = lambda x: -1 + np.cos(x)

# --> Recherche des racines.
sol = root_scalar(
    f,              # Fonction f(x).
    fprime=df,      # Derivee de la fonction.
    x0=np.pi/2,     # Point de depart.
    method="newton"
)


```

Recherche de racines

Méthode de Newton-Raphson

```python
import numpy as np
from scipy.optimize import root_scalar
from scipy.optimize import approx_frpime

# --> Definition de la fonction et de sa derivee.
f = lambda x : np.pi/4 - x + np.sin(x)
df = lambda x : approx_fprime(x, f, 1e-6)

# --> Recherche des racines.
sol = root_scalar(
    f,                # Fonction f(x).
    fprime=df,        # Derivee de la fonction.
    x0=np.pi/2,       # Point de depart.
    method="newton"
)

```

Recherche de racines

Méthode de la sécante

La méthode de Newton-Raphson est sans doute la méthode la plus efficace pour calculer des racines. Elle nécessite néanmoins la connaissance de $f'(x)$.

Malheureusement, dans certaines situations, $f'(x)$ ne peut être calculée analytiquement et son approximation peut être extrêmement coûteuse en terme de temps de calcul.

Recherche de racines

Méthode de la sécante

En partant de deux itérations successives x_0 et x_1, une alternative est alors de remplacer le calcul de $f'(x)$ par

$$f'(x) \simeq \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

Ceci conduit alors à la **méthode de la sécante**.

Recherche de racines

Méthode de la sécante

On obtient alors la relation de récurrence suivante

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n).$$

Il est important de noter que x_0 et x_1 ne doivent pas nécessairement encadrée la racine recherchée (même si cela peut être préférable).

Recherche de racines

Méthode de la sécante

En supposant que a et b aient été correctement initialisés, l'algorithme correspondant est présenté ci-dessous.

```
while abs(f(x)) < tol:
    x = x - (b-a) / (f(b)-f(a)) * f(b)
    a, b = b, x
```

Tout comme pour la méthode de Newton, on ajoutera en pratique également une condition sur le nombre maximum d'itération.

Recherche de racines

Méthode de la sécante

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

# --> Definition de la fonction.
f = lambda x : np.pi/4 - x + np.sin(x)

# --> Recherche des racines.
sol = root_scalar(f, x0=0, x1=np.pi, method="secant")

# --> Extrait differentes informations.
x = sol.root # Racine obtenue.
niter = sol.iterations # Nombre d’iterations.
nfev = sol.functions_calls # Nombre d’appels de f(x).
```

Recherche de racines

Résumé

Méthode	Bissection	Sécante	Newton-Raphson
Hypothèses sur $f(x)$	Continue	Continue	Continue et dérivable
Initialisation	a et b tels que $f(a)f(b) < 0$	x_0 et x_1	x_0
Nécessite	$f(x)$	$f(x)$	$f(x)$ et $f'(x)$
Convergece	Lente	Moyenne	Rapide
Robustesse	Très robuste	Moyennement robuste	Peu robuste