La recherche dichotomique

Yassine . T

Table des matières

1 Introduction : pourquoi ces techniques ?

Trouver rapidement une information ou résoudre une équation est un besoin central en informatique et en mathématiques appliquées. Deux idées fondatrices reviennent sans cesse :

— **Diviser pour régner** : réduire l'espace de recherche par dichotomie, ce qui mène à la recherche dichotomique en tableaux triés et à la bisection pour les zéros de fonctions.

— **Linéariser localement** : approcher une fonction par sa tangente pour obtenir une mise à jour efficace – c'est la méthode de Newton (Newton–Raphson).

Nous relions ici ces idées : d'abord le contexte historique, puis la théorie mathématique (avec Newton en détail), et enfin une mise en œuvre algorithmique illustrée en Python.

2 Origines historiques

2.1 Racines de la dichotomie

L'idée de couper un intervalle en deux remonte à l'Antiquité. En analyse, la bisection s'appuie sur le théorème des valeurs intermédiaires (Bolzano, Cauchy) : si f est continue et change de signe sur $[a, b]$, un zéro appartient à cet intervalle ; on répète alors la division par deux.

2.2 Des tables numériques à la *binary search*

Avec l'essor des tables et des index (mathématiques, astronomiques, puis informatiques), la recherche efficace dans des données triées s'est imposée : la recherche dichotomique réduit logarithmiquement le nombre de comparaisons.

2.3 Newton, Raphson et l'itération tangentielle

La **méthode de Newton** (XVII$^{\text{e}}$) émerge de l'idée d'utiliser la tangente pour approcher un zéro de f. Raphson formalise l'itération ; l'analyse moderne en donne la théorie de convergence. C'est l'archétype du fait de remplacer un problème non linéaire par des linéarisations locales successives.

3 Partie mathématique : la méthode de Newton

3.1 Principe (rappel)

On cherche une racine α de $f : I \to \mathbb{R}$ avec $f(\alpha) = 0$. À partir d'une approximation x_n proche de α, on linéarise f en x_n :

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n).$$

Le zéro de cette tangente donne l'itération de Newton :

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

On notera l'opérateur de Newton $N_f(x) := x - \frac{f(x)}{f'(x)}$. C'est un schéma de *point fixe* : si $f'(\alpha) \neq 0$, alors $N_f(\alpha) = \alpha$. De plus $N_f'(\alpha) = 0$ (voir ci-dessous), ce qui annonce la convergence quadratique.

3.2 Théorème de convergence quadratique locale

Hypothèses. Soit $f \in C^2(I)$ et $\alpha \in I$ telle que $f(\alpha) = 0$ et $f'(\alpha) \neq 0$ (*zéro simple*). Alors il existe un voisinage U de α tel que $f'(x) \neq 0$ pour tout $x \in U$.

Constantes locales. On se fixe $U = B(\alpha, r)$ (une boule ouverte) sur laquelle

$$m_1 := \inf_{x \in U} |f'(x)| > 0, \qquad M_2 := \sup_{x \in U} |f''(x)| < \infty.$$

(On peut toujours choisir $r > 0$ assez petit pour garantir ces bornes.)

Énoncé. Il existe $\rho \in (0, r]$ tel que si $x_0 \in B(\alpha, \rho)$, la suite (x_n) de Newton est bien définie, reste dans $B(\alpha, \rho)$ et vérifie, pour tout $n \geq 0$,

$$|x_{n+1} - \alpha| \;\leq\; C\,|x_n - \alpha|^2, \qquad C := \frac{M_2}{2\,m_1}.$$

En particulier, $x_n \to \alpha$ avec **convergence quadratique** (l'erreur est approximativement quadratée à chaque itération).

Idée-clé. Posons $e_n := x_n - \alpha$. Une identité exacte montre

$$e_{n+1} = \frac{f''(\xi_n)}{2\,f'(x_n)}\,e_n^2 \quad \text{pour un certain } \xi_n \in (\alpha, x_n),$$

d'où la borne quadratique dès lors que f'' est bornée et f' est *uniformément* non nul autour de α.

4 Recherche dichotomique : théorie et preuves

4.1 Principe et invariant

Pour un tableau trié T, on maintient *gauche* et *droite* de sorte que :

$$\text{si } x \text{ est présent, alors } x \in T[\mathit{gauche}..\,\mathit{droite}].$$

À chaque étape, on prend $m = \lfloor (\mathit{gauche} + \mathit{droite})/2 \rfloor$ et on supprime la moitié impossible.

4.2 Correction (esquisse)

- **Initialisation** : $[0, n-1]$ contient toutes les positions.
- **Conservation** : si $T[m] \neq x$, le tri impose un seul côté possible ; mise à jour de la borne correspondante.
- **Terminaison** : longueur d'intervalle décroissante ; quand $\mathit{gauche} > \mathit{droite}$, plus de position candidate.

4.3 Complexité

Nombre d'itérations $\leq \lceil \log_2(n) \rceil + 1$; temps $\mathcal{O}(\log n)$; espace $\mathcal{O}(1)$ en itératif.

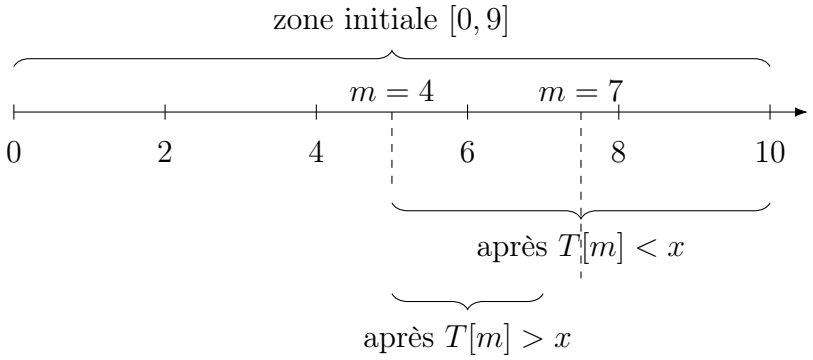

FIGURE 1 – Réduction d'intervalle (vue par indices) pour la recherche dichotomique.

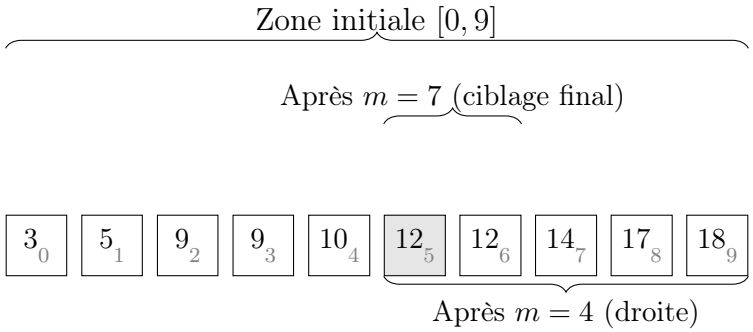

FIGURE 2 – Tableau trié annoté : valeurs (dans les cases), indices (en bas à droite), et étapes de réduction.

5 Algorithmique : pseudocodes et variantes

5.1 Schéma explicatif

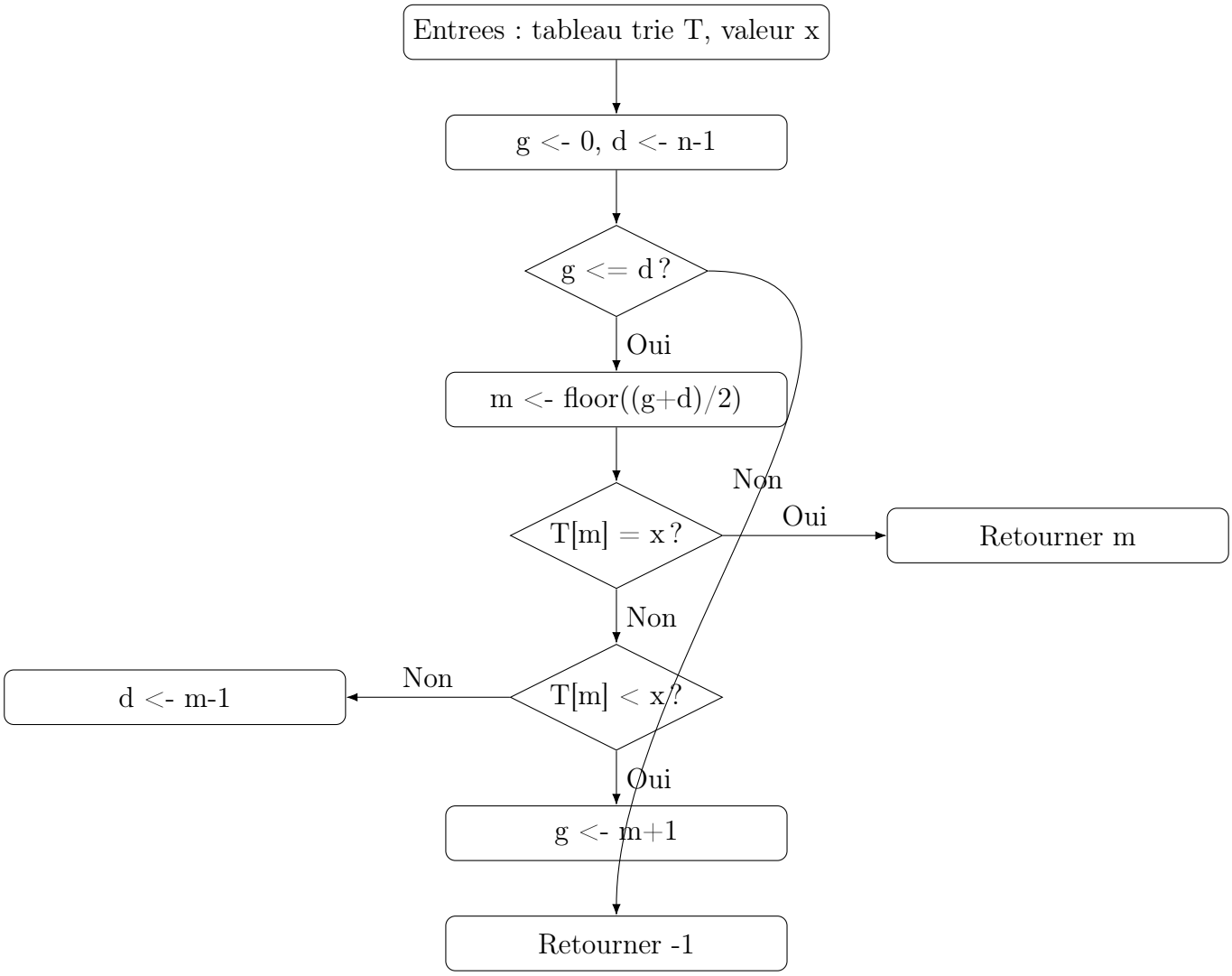

FIGURE 3 – Organigramme de la recherche dichotomique (flots lisibles, sans warning TikZ).

5.2 Version standard (itérative)

```
ALGORITHME RechercheDichotomique(T trie, x)
  gauche <- 0
  droite <- n(T) - 1
  TANT QUE gauche <= droite FAIRE
    m <- floor((gauche + droite) / 2)
    SI T[m] = x ALORS
      RETOURNER m
    SINON SI T[m] < x ALORS
      gauche <- m + 1
    SINON
      droite <- m - 1
    FINSI
  FIN TANT QUE
```

```
  RETOURNER -1
FIN
```

5.3 Première et dernière occurrence

```
ALGORITHME PremiereOccurrence(T trie, x)
  gauche <- 0 ; droite <- n(T) - 1 ; rep <- -1
  TANT QUE gauche <= droite FAIRE
    m <- floor((gauche + droite) / 2)
    SI T[m] >= x ALORS
      SI T[m] = x ALORS rep <- m FINSI
      droite <- m - 1
    SINON
      gauche <- m + 1
    FINSI
  FIN TANT QUE
  RETOURNER rep
FIN
```

5.4 Point d'insertion (type bisect_left)

```
ALGORITHME PointInsertion(T trie, x)
  g <- 0 ; d <- n(T)
  TANT QUE g < d FAIRE
    m <- floor((g + d) / 2)
    SI T[m] < x ALORS g <- m + 1
    SINON d <- m
  FIN TANT QUE
  RETOURNER g
FIN
```

6 Implémentations Python (robustes et commentées)

Recherche dichotomique standard

```python
def binary_search(tab, x):
    # Retourne l'indice de x dans tab (trie) ou -1 si absent.
    g, d = 0, len(tab) - 1
    while g <= d:
        m = (g + d) // 2  # Milieu
        if tab[m] == x:
            return m
        elif tab[m] < x:
            g = m + 1
        else:
            d = m - 1
    return -1
```

Première occurrence, dernière occurrence et point d'insertion

```
def first_occurrence(tab, x):
    g, d, rep = 0, len(tab) - 1, -1
    while g <= d:
        m = (g + d) // 2
        if tab[m] >= x:
            if tab[m] == x:
                rep = m
            d = m - 1
        else:
            g = m + 1
    return rep

def last_occurrence(tab, x):
    g, d, rep = 0, len(tab) - 1, -1
    while g <= d:
        m = (g + d) // 2
        if tab[m] <= x:
            if tab[m] == x:
                rep = m
            g = m + 1
        else:
            d = m - 1
    return rep

def insertion_point(tab, x):
    g, d = 0, len(tab)
    while g < d:
        m = (g + d) // 2
        if tab[m] < x:
            g = m + 1
        else:
            d = m
    return g  # position pour inserer x
```

Méthode de la bisection (zéro de fonction)

```
def bisection(f, a, b, eps=1e-12, itmax=100):
    # Trouve un zero sur [a,b] si f(a)*f(b) < 0, par bisection.
    fa, fb = f(a), f(b)
    if fa * fb > 0:
        raise ValueError("f(a) et f(b) doivent etre de signes opposes."
           )
    for _ in range(itmax):
        m = 0.5 * (a + b)
        fm = f(m)
        if abs(fm) <= eps or 0.5 * (b - a) <= eps:
            return m
        if fa * fm < 0:
            b, fb = m, fm
        else:
```

```
            a, fa = m, fm
    return 0.5 * (a + b)
```

Méthode de Newton

```
def newton(f, fp, x0, eps=1e-12, itmax=100):
    # Methode de Newton avec arret sur petit residu ou petit pas.
    x = x0
    for _ in range(itmax):
        fx = f(x)
        fpx = fp(x)
        if fpx == 0:
            raise ZeroDivisionError("Derivee nulle: Newton impossible."
                )
        x_next = x - fx / fpx
        if abs(x_next - x) <= eps and abs(fx) <= eps:
            return x_next
        x = x_next
    return x
```

7 Conclusion générale

Nous avons relié trois aspects fondamentaux de l'algorithmique et du calcul numérique :

1. **Dichotomie** — L'idée de réduire l'espace de recherche par deux à chaque étape fonde à la fois la recherche dichotomique dans les tableaux triés et la bisection pour l'approximation de zéros. Elle garantit la *correction* (grâce au tri ou au TVI) et une *complexité* en temps $\mathcal{O}(\log n)$ en contexte discret.
2. **Linéarisation** — La méthode de Newton (Newton–Raphson) illustre la puissance de l'approximation tangentielle : sous hypothèses standard ($f \in C^2$, zéro simple, bon point initial), la *convergence est quadratique*, offrant des gains spectaculaires par rapport à la bisection (linéaire) — au prix d'une moindre robustesse si l'initialisation est mauvaise ou si $f'(\alpha) = 0$.
3. **Algorithmique** — Invariants, preuves de correction et analyses de complexité structurent les implémentations. En pratique, on combine souvent une *phase sûre* (quelques itérations de bisection) pour cadrer la solution, puis *Newton* pour accélérer la convergence.

Ces techniques constituent des briques fondamentales, autant pour l'algorithmique (recherche efficace, structures triées) que pour le calcul scientifique (résolution d'équations non linéaires, optimisation numérique). Les schémas fournis (réduction des bornes, organigramme, itérations tangentielle) doivent servir de *repères visuels* pour justifier, expliquer et déboguer les implémentations.

Sources et références

Références

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 3e éd., MIT Press, 2009. (Algorithmes de recherche, preuves par invariants, complexités.)

[2] D. E. Knuth, *The Art of Computer Programming, Vol. 3 : Sorting and Searching*, 2e éd., Addison-Wesley, 1998. (Analyse classique de la binary search et des structures triées.)

[3] J. Bentley, *Programming Pearls*, 2e éd., Addison-Wesley, 2000. (Erreurs classiques de la binary search, méthodologie et tests.)

[4] R. L. Burden, J. D. Faires, *Numerical Analysis*, 10e éd., Cengage, 2015. (Méthodes de bisection et de Newton, preuves et vitesses de convergence.)

[5] K. E. Atkinson, *An Introduction to Numerical Analysis*, 2e éd., Wiley, 1989. (Convergence locale de Newton, zéros multiples, choix de x_0.)

[6] J. Stoer, R. Bulirsch, *Introduction to Numerical Analysis*, 3e éd., Springer, 2002. (Encadrements, stabilité, raffinements de Newton et variantes.)

[7] A.-L. Cauchy, *Cours d'analyse de l'École Royale Polytechnique*, 1821. (Fondements de l'analyse ; contexte historique du TVI et des méthodes d'approximation.)

[8] *Python Standard Library — `bisect` module*, `https://docs.python.org/3/library/bisect.html` (Point d'insertion, `bisect_left`/`bisect_right`, bonnes pratiques.)

[9] *Python `math` module*, `https://docs.python.org/3/library/math.html` (Comparaisons numériques, précision flottante et bonnes pratiques.)

Crédits des schémas. Schémas réalisés par moi mème avec TikZ (`arrows.meta`, `positioning`, `shapes.geometric`, `decorations.pathreplacing`).