Simulations numériques : Résoudre $f(x) = 0$

Objectif : L'objectif de ce tp est de rendre l'élève capable :

- d'écrire en langage **Python** les algorithmes liés à la résolution d'un problème stationnaire de dimension 1 (du type résoudre $f(x) = 0$)
- de tracer l'évolution du nombre d'itérations nécessaire pour résoudre un problème en fonction de l'erreur désirée et déterminer les vitesses de convergences.

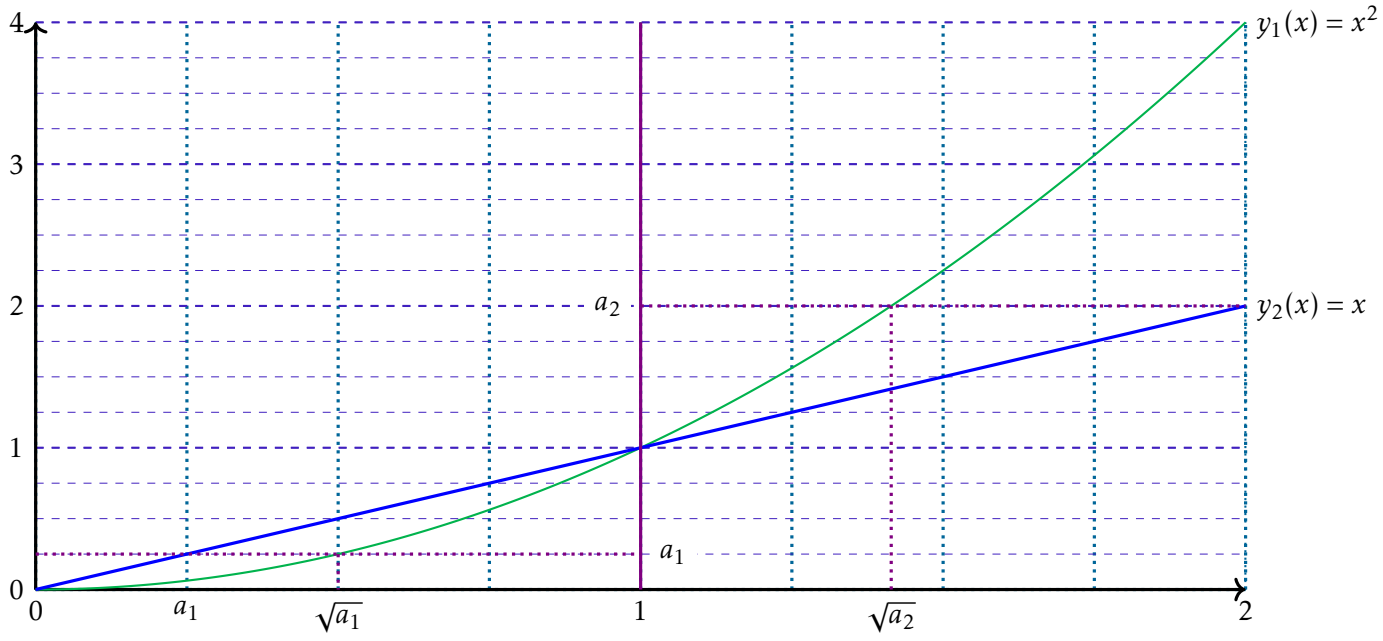

Figure 1 – Fonctions $y_1(x) = x^2$ et $y_2(x) = x$

1 Stratégies de résolutions

1.1 Paramètres

Pour l'exercice, on prendra $a = 2$ et un seuil de convergence $\varepsilon = 10^{-8}$.

Le test de convergence s'effectue, selon les méthodes, parfois sur les images ($f(x)$), parfois sur les antécédents (x). Pour que le test de convergence ait du sens sur les images en terme de nombre de chiffres significatifs, la fonction objectif est normée. On pose alors :

$$f_{obj}(x) = \frac{x^2}{a} - 1$$

Le critère de convergence est alors satisfait

- sur les images quand $\left|f_{obj}(x)\right| \leq \varepsilon$
- sur les antécédents quand $|x - r| \leq \varepsilon'$, si r est la solution.

1.2 Méthode par dichotomie

1.2.1 Principe

Sur une domaine de définition où la fonction f est continue et coupe l'axe des abscisses, rechercher par dichotomie la solution x de l'équation $f(x) = 0$ revient à diviser l'intervalle en deux, garder celui qui contient la solution et recommencer, jusqu'à ce que l'erreur normée soit inférieure au seuil de tolérance.

1.2.2 Réalisation

Si $f_{obj}(x_{gn}).f_{obj}(x_{dn}) < 0$, comme f_{obj} est strictement monotone sur $\mathbb{R}^+$, lorsqu'on coupe l'intervalle $[x_{gn}, x_{dn}]$ en 2 en

$$x_{n+1} = \frac{x_{gn} + x_{dn}}{2} \quad \text{alors}$$

- si $f_{obj}(x_{n+1}).f_{obj}(x_{gn}) < 0$ alors $r \in \left[x_{gn}, x_{n+1}\right]$
- si $f_{obj}(x_{n+1}).f_{obj}(x_{gn}) > 0$ alors $r \in [x_{n+1}, x_{dn}]$
- si $f_{obj}(x_{n+1}).f_{obj}(x_{gn}) = 0$ alors $r = x_{n+1}$

Q - 1 : *Construire une fonction* `Dichotomie` *ayant pour entrée une fonction* `f`*, les bornes de l'intervalle de recherche* `xg` *et* `xd` *et la tolérance* `eps`*. Le test d'arrêt se fera sur les antécédents* ($\left|x_{gn} - x_{dn}\right| < \varepsilon$).

Q - 2 : *Déterminer à l'aide des fonctions précédentes la racine carrée de a.*

Remarque : pour les bornes, on pourra remarquer, d'après la Fig 1, que :

- si $a > 1$ alors $\sqrt{a} \in [1, a]$
- si $a < 1$ alors $\sqrt{a} \in [a, 1]$

1.3 Méthode de la corde de Lagrange

1.3.1 Principe

Au lieu de scinder l'intervalle initial $[x_{gn}, x_{dn}]$ en deux intervalles de même longueur, la méthode de la corde de Lagrange cherche à valoriser la borne la plus proche de la solution en cherchant l'intersection x_{n+1} entre l'axe des abscisses et la corde reliant le point $\left(x_{gn}, f(x_{gn})\right)$ au point $(x_{dn}, f(x_{dn}))$.

L'intervalle $[x_{gn}, x_{dn}]$ est alors coupé en x_{n+1} tel que : $x_{n+1} = \dfrac{x_{dn}.f(x_{gn}) - x_{gn}.f(x_{dn})}{f(x_{gn}) - f(x_{dn})}$.

1.3.2 Réalisation

Q - 3 : *Adapter le programme pour la méthode par dichotomie à la méthode de la corde de Lagrange avec pour test d'arrêt* $\left|f_{obj}(x_n)\right| < \varepsilon$

Q - 4 : *Observer le nombre d'itérations avant convergence pour les deux méthodes avec* $a \in \{0,4; 0,7; 4; 7; 20; 60\}$ *et avec pour test d'arrêt* $|f(x_n)| < \varepsilon$.

1.4 Méthode de Newton

1.4.1 Principe

La méthode de Newton utilise la monotonie de la fonction f au voisinage de r, pour calculer à partir d'un premier candidat x_0, une solution approchée par le développement limité à l'ordre 1 de la fonction f autour de

ce candidat. En effet, si la fonction f est de classe $\mathcal{C}^1$ sur $\mathbb{R}^+$:

$0 = f(r) = f(x_0) + f'(x_0).(r - x_0) + o(r - x_0)$

Ne connaissant pas r, on cherche alors x_1 solution du problème approché dans lequel on néglige $o(r - x_0)$:

$$f(x_1) \approx 0 = f(x_0) + f'(x_0).(x_1 - x_0) \quad \Rightarrow \quad x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Le processus est donc répété en remplaçant x_0 par x_1 jusqu'à ce que le critère de convergence soit satisfait.

1.4.2 Réalisation

Q - 5 : *Déterminer $f'_{obj}(x)$ puis construire une fonction retournant $f'_{obj}(x)$ à partir des données x et a.*

Q - 6 : *Après avoir initialisé le problème en posant comme premier candidat $x_0 = 1$, construire la boucle de récurrence avec pour test d'arrêt $\left|f_{obj}(x_n)\right| < \varepsilon$.*

1.4.3 Algorithme

Algorithm 1 Méthode de Newton

```
1:  xi ← x0
2:  fx ← f(xi)
3:  tant que |fx| > ε faire
4:      fpx ← f'(xi)
5:      si fpx=0 alors
6:          break
7:      fin si
8:      xi ← xi − fx/fpx
9:      fx ← f(xi)
10: fin tant que
11: renvoi: xi
```

1.5 Méthode de la fausse position

1.5.1 Principe

La méthode de Newton, oblige à connaître l'expression de $f'(x)$ (ou la matrice Jacobiene dans le cas des fonctions de plusieurs variables), ce qui peut parfois être lourd à exprimer et prendre beaucoup de temps de calcul.

Comme la méthode de Newton, la méthode de la fausse position utilise un développement limité à l'ordre 1 de la fonction. Cependant, le calcul de la dérivé n'est pas effectué. La méthode de la fausse position utilise alors une estimation de la dérivée basée sur les deux points précédents :

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

1.5.2 Réalisation

Q - 7 : *En posant $x_0 = 1$ et $x_1 = 2$, adapter le programme basé sur la méthode de Newton à la méthode de la fausse position.*

Q - 8 : *Observer le nombre d'itérations nécessaires pour obtenir la convergence des deux méthodes précédentes.*

1.6 Méthode de Schröder d'ordre 3

1.6.1 Principe

Les méthode de Schröder d'ordre n sont basées sur des développements limités de f à l'ordre $n-1$. Ainsi dans le cas d'une méthode de Schröder d'ordre 3, la relation de récurrence entre x_n et x_{n+1} est :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{f''(x_n).f^2(x_n)}{2.f'^3(x_n)}$$

Q - 9 : *A partir de la relation de récurrence précédente, adapter le programme basé sur la méthode de Newton à la méthode de Schröder d'ordre 3*

1.7 Bibliothèques

Les codes **Python** et **Scilab** ont des fonctions optimisées pour calculer rapidement la solution du problème $f(x) = 0$.

Méthode	**Python**
Dichotomie	**scipy.optimize.bisect**(fct,borne inf,borne sup)
Newton	**scipy.optimize.newton**(fct,candidat 1, fct dérivée)

Pour **Python** et **Scilab** , la fonction **fsolve** (**scipy.optemize.fsolve**), permet de résoudre le problème $f(x) = 0$, même lorsque x n'est plus un scalaire mais un vecteur. Il convient alors de fournir la matrice jacobienne (matrice des dérivées partielles de la fonction) pour permettre à l'algorithme de tourner.

Q - 10 : *Tester la fonction **fsolve** au problème posé en introduction.*

2 Vitesse de convergence

Q - 11 : *Éditer un programme permettant de comparer le nombre d'itérations et le temps de calcul de chacune des méthodes présentées dans le Tp.*

Remarque : on pourra utiliser le module **time** avec la commande `time()`, comme décrit *ici*.
La vitesse de convergence est caractérisée par les paramètres K et p caractérisant la relation de récurrence liée à l'erreur :

$$\varepsilon_{n+1} = \mathrm{K}.\varepsilon_n^p \quad \Rightarrow \quad \ln(\varepsilon_{n+1}) = \ln(\mathrm{K}) + p.\ln(\varepsilon_n)$$

Q - 12 : *Déterminer les paramètres* K *et* p *en traçant* $\ln(\varepsilon_{n+1})$ *en fonction de* $\ln(\varepsilon_n)$ *et en début la vitesse de convergence de chaque méthode.*