TP Problèmes stationnaires CORRIGÉ

Pour importer les modules numpy et pyplot :

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

1 Exercice 1

1) La fonction cosinus est disponible dans le module numpy :

```
def f(x):
    return(np.cos(x)-x/4)
```

2)

```
intervalle=np.linspace(0,2,100)
plt.plot(intervalle,f(intervalle))
plt.show()
```

Augmenter le nombre de points ne donne pas de changement perceptible sur le dessin affiché.

2 Exercice 2

a)

```
def dichotomie(f,a,b,epsilon,maxiter):
    u=min(a,b)
    v=max(a,b)
    fu=f(u)
    fv=f(v)
    k=0 #nombre d'itérations
    while (k<maxiter and (v-u)>epsilon):
        w=(u+v)/2
        fw=f(w)
        if (fu>0 and fw>0) or (fu<0 and fw<0):
            u=w
            fu=fw
        else:
            v=w
            fv=fw
        k=k+1
    return((u+v)/2,k)
```

b) La précision machine sur les flottants est de l'ordre de 2^{-64} , on choisit donc $\text{maxiter}=64$ (toute valeur plus grande n'aura pas de sens) :

```
[in] dichotomie(f,0,2,1e-6,64)
[out] (1.2523531913757324,21)
```

En notant α la solution théorique de l'équation, il a suffi de 21 itérations pour obtenir

$$\alpha = 1,252353 \pm 10^{-6}$$

3 Exercice 3

a) Il s'agit de calculer les termes de la suite récurrente $u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}$. Nous utilisons ici la dérivée numérique de f , ie l'approximation :

$$f'(u_n) \approx \frac{f(u_n + h) - f(u_n - h)}{2h}$$

qui est valable pour h suffisamment petit (non nul). Pour tenir compte de la précision sur les flottants en machine (de l'ordre de 10^{-15}) et de la précision souhaitée (10^{-6} ici, choisissons par exemple $h = 10^{-12}$ pour cet exercice.

```
def newton(f,a,b,u0,epsilon,maxiter):
    h=1e-12
    k=0 #nombre d'itérations
    u=u0
    fu=f(u)
    while k<maxiter and abs(fu)>epsilon:
        fpu=(f(u+h)-f(u-h))/(2*h) #derivee numerique
        u=u-fu/fpu
        fu=f(u)
        k=k+1
    return(u,k)
```

b1) On choisit par exemple $u_0 = 1$ (mais toute valeur de $[0, 2]$ convient...). On sait que la méthode de Newton converge rapidement (convergence quadratique, cf cours), inutile de prendre une trop grande valeur de maxiter. Voici le résultat obtenu pour une précision en ordonnée de 10^{-10} :

```
[in] newton(f,0,2,1,1e-10,30)
[out] (1.2523532340027765 , 4)
```

On peut constater la rapidité de cet algorithme : il a suffi de 4 itérations pour obtenir la précision

$$|f(u_4)| \leq 10^{-10}$$

La valeur u_4 est au moins aussi précise que celle obtenue par dichotomie (cf exercice 2) qui avait demandé 21 itérations!

b2) Adaptons un peu la fonction "newton" pour afficher les valeurs successives de u :

```
def newton2(f,a,b,u0,maxiter):
    h=1e-12
    k=0 #nombre d'itérations
    u=u0
    fu=f(u)
    print(u,"iteration =",k)
    while k<maxiter:
        fpu=(f(u+h)-f(u-h))/(2*h) #derivee numerique
        u=u-fu/fpu
        fu=f(u)
        k=k+1
        print(u,"iteration =",k)
```

Nous obtenons

```
[in] newton2(f,0,2,1,5)
[out] 1 iteration = 0
      1.26596254011 iteration = 1
      1.25237760717 iteration = 2
      1.2523532364 iteration = 3
      1.252353234 iteration = 4
      1.252353234 iteration = 5
```

Il a suffi de 3 itérations pour obtenir une précision de 10^{-6} sur le nombre cherché.

4 Exercice 4

a) On représente graphiquement l'algorithme proposé (quelques valeurs de c , avec leur construction géométrique) à partir du graphe de la fonction f : laissé en exercice
b1)

```
def lagrange(f,a,b,epsilon,maxiter):
    u=a
    v=b
    fu=f(u)
    fv=f(v)
    w=(u*fv-v*fu)/(fv-fu) #intersection de la corde et de l'axe des abscisses
    fw=f(w)
    k=0 #nombre d'itérations
    while (k<maxiter and (v-u)>epsilon):
        if (fu>0 and fw>0) or (fu<0 and fw<0):
            u=w
            fu=fw
        else:
            v=w
            fv=fw
        w=(u*fv-v*fu)/(fv-fu)
        fw=f(w)
        k=k+1
    return(w,k)
```

b2) Pour notre exemple nous trouvons

```
[in] lagrange(f,0,2,1e-6,64)
[out] (1.2523532340025887,12)
```

soit 12 itérations pour obtenir une valeur approchée à 10^{-6} près de α

c1) Nous constatons que la méthode de Lagrange est un peu plus efficace que la méthode de dichotomie (12 itérations au lieu de 21 pour la précision souhaitée). Cela dit on constate que la précision réelle obtenue est en fait bien meilleure que 10^{-6} . Ainsi avec la fonction dichotomie on trouve facilement la valeur approchée

$$\alpha = 1,2523532340 \pm 10^{-10}$$

Il doit donc être possible d'améliorer l'algorithme précédent en changeant simplement la condition d'arrêt.

c2) Notons $(w_0, w_1, \dots)$ les valeurs successives de la variable w calculées dans la méthode de Lagrange. On définit de même des suites (u_n) et (v_n) pour les variables u et v . Comme proposé nous allons remplacer la condition d'arrêt « $v_n - u_n \leq \epsilon$ » par

$$|w_n - w_{n-1}| \leq \epsilon$$

Il suffit d'adapter la fonction lagrange de manière à enregistrer deux valeurs successives de w :

```
def lagrange2(f,a,b,epsilon,maxiter):
    u=a
    v=b
    fu=f(u)
    fv=f(v)
    w=(u*fv-v*fu)/(fv-fu) #intersection de la corde et de l'axe des abscisses
    fw=f(w)
    wp=a #valeur precedente de w, initialisee à a par convention
    k=0 #nombre d'itérations
    while (k<maxiter and abs(wp-w)>epsilon):
        if (fu>0 and fw>0) or (fu<0 and fw<0):
            u=w
            fu=fw
        else:
            v=w
            fv=fw
        wp=w
        w=(u*fv-v*fu)/(fv-fu) #nouvelle valeur de w
        fw=f(w)
        k=k+1
    return(w,k)
```

c3) Nous obtenons

```
[in] lagrange2(f,0,2,1e-6,64)
[out] (1.2523532322484774,5)
```

Avec 5 itérations, on obtient quand même une précision de 10^{-6} avec la méthode de Lagrange.

5 Exercice 5

Nous utilisons la condition d'arrêt commune $|v - u| \leq \epsilon$ pour un intervalle $[u, v]$ de valeurs proches de α .

- Pas de problème pour les méthodes de dichotomie et de Lagrange (première version), nous avons déjà obtenu la bonne précision avec respectivement 21 et 12 itérations
- Il nous faut modifier un peu l'algorithme de Newton pour avoir une bonne condition d'arrêt. Par exemple :

```
def newtonbis(f,a,b,u0,epsilon,maxiter):
    h=1e-12
    k=0 #nombre d'itérations
    u=u0
```

```

fpu=(f(u+h)-f(u-h))/(2*h) #derivee numerique
v=u-f(u)/fpu #correspond à u_1
fv=f(v)
while k<maxiter and abs(v-u)>epsilon:
    u=v
    fpv=(f(v+h)-f(v-h))/(2*h) #derivee numerique
    v=v-fv/fpv
    fv=f(v)
    k=k+1
return(v,k)

```

On trouve la bonne précision en 3 itérations :

```

[in] newtonbis(f,0,2,1,1e-6,30)
[out] (1.2523532340027765,3)

```

Évaluons la complexité de ces algorithmes, en fonction de la précision souhaitée ϵ , pour la fonction f et les valeurs $a = 0$, $b = 2$.

- Ces trois algorithmes font tous appel à la fonction f ; cette fonction est suffisamment simple pour que la complexité de ces appels soit supposée constante
- Par conséquent La complexité de ces algorithmes est de l'ordre de $\mathcal{O}(n)$, où n est le nombre d'itérations effectuée.
- La valeur de n est différente d'un algorithme à l'autre :

* Dichotomie : À chaque itération la longueur de l'intervalle $[u, v]$ est divisée par 2, et l'algorithme s'arrête lorsque $v - u \leq \epsilon$, donc n est le plus petit entier tel que $\frac{b-a}{2^n} \leq \epsilon$, ie $n \approx \log_2\left(\frac{1}{\epsilon}\right)$ (convergence linéaire). Pour $\epsilon = 10^{-6}$ on retrouve $n \approx 20$

* Lagrange : n est difficile à calculer en général. Pour notre exemple, notons u_n, v_n, w_n les suites de valeurs obtenues au cours de l'algorithme. On constate assez facilement (faire un dessin !) que $u_k = 0$ et $v_{k+1} = w_k$ pour tout entier k . Donc

$$w_k = \frac{u_k f(v_k) - v_k f(u_k)}{f(v_k) - f(u_k)} = g(w_{k-1})$$

où $g(x) = \frac{-x}{f(x)-1} = \frac{x}{1-\cos x + \frac{x}{4}}$. Ainsi (w_k) est une suite récurrente, qui converge vers α , et qui peut être estimée avec la formule des accroissements finis ; sans rentrer dans les détails on aura

$$w_n = \mathcal{O}(|g'(\alpha)|^n)$$

On vérifie ici que $g'(\alpha) \approx 0,4987$. C'est à peine mieux que la méthode de dichotomie (pour laquelle $w_n = \frac{b-a}{2^n} \approx (0.5)^n$)... Heureusement on a vu que modifier la condition d'arrêt améliore sensiblement la complexité de la méthode de Lagrange

* Newton : d'après le cours, la méthode de Newton a une convergence quadratique, donc $n = \mathcal{O}\left(\log_2\left(\log_2\left(\frac{1}{\epsilon}\right)\right)\right)$. Pour $\epsilon = 10^{-6}$ on obtient $n \approx 4.3$. La méthode de Newton est la plus efficace sur cet exemple.