TP2 – Méthode de Newton

Théorie. La méthode de Newton (dite souvent Newton-Raphson en dimension $d \geq 1$) s'applique à la résolution d'un système d'équations $F(x) = 0$, où $F : \mathbb{R}^d \to \mathbb{R}^d$ est de classe $\mathcal{C}^1$. Elle est définie par l'itération récurrente dans $\mathbb{R}^d$:

$$x_{n+1} = x_n - [dF(x_n)]^{-1} F(x_n).$$

La convergence est garantie lorsque la donnée initiale x_0 est suffisamment proche de la solution recherchée x^* de $F(x) = 0$ à condition que l'application différentielle $dF(x^*) \in \mathcal{L}(\mathbb{R}^d, \mathbb{R}^d)$ soit inversible. La convergence est alors quadratique, i.e.

$$\exists C > 0,\ \forall n \in \mathbb{N}, \quad \|x_{n+1} - x^*\| \leq C\|x_n - x^*\|^2.$$

Ainsi quelques itérations suffisent pour obtenir une précision redoutable.

But du TP. *Il s'agit de mettre en œuvre la méthode de Newton et d'en estimer la vitesse de convergence numériquement dans différents situations élémentaires, puis de traiter des cas d'application plus riches.*

Consignes d'organisation. *Mise en place de l'environnement de travail :*

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

Mise en œuvre.

- ☐ Programmer la méthode de Newton pour une fonction F, de différentielle dF qui sera supposée calculable explicitement.
  Distinguer les cas monodimensionnel et multidimensionnel.
  Pour cela, écrire une fonction `def Newton(x0,N,f,df)` qui, pour une donnée initiale `x0`, contient la boucle principale `x=x-f(x)/df(x)` dans laquelle `f` est la fonction étudiée et `df` sa dérivée.

1. Tester la méthode sur l'exemple monodimensionnel $F_1(x) = e^x - e$, illustrer graphiquement la vitesse de convergence puis la quantifier.

(i) On commencera par le cas théorique où le nombre de pas `N` est imposé. La commande de sortie `return x` renvoie le `N` ième itéré.
Tester ce programme avec la fonction `def f1(x): return np.exp(x)-np.e`
et sa dérivée `def df1(x)`
en partant des données `x0=-1, N=10`.

(ii) Illustrer par des graphiques.

(a) Tracer la suite des itérés sur le graphe de f :

```
 x0=-1,f=f1,df=df1,eps=1e-6
tabx,tabcrit,it=Newton4(x0,f,df,eps)
tabxeff=tabx[:it] #   tableau effectif des itérés
tabcriteff=tabcrit[:it] # tableau effectif des images des itérés
t=np.linspace(-2,5,100)
yt=f1(t)
plt.plot(t,yt,tabxeff,tabcriteff)
plt.show()
```

(b) Tracer l'évolution de la suite des erreurs $(e_k)_{k\geq 0}$ sous la forme d'une relation entre e_{k+1} et e_k à chaque pas k : on définit l'erreur au pas `it` comme la différence entre `tabxeff[it]` et l'approximation `tabxeff[-1]` qui est aussi la dernière valeur calculée dans le tableau des itérés.

```
 err=tabxeff-tabxeff[-1]#   Tableau des erreurs
errx=err[1:-1]# Suite des e_k
erry=err[2:] # Suite des e_{k+1}
plt.plot(errx,erry,errx[0],erry[0],'o',errx[-1],erry[-1],'*')
plt.show()
```

(b) Comparer avec le carré :

```
errx2=errx
lx=len(errx)
for i in range(lx):
    errx2[i]=errx[i]**2
plt.plot(errx2,erry,errx2[0],erry[0],'o',errx2[-1],erry[-1],'*')
plt.show()
```

(c) Utiliser la fonction Logarithme :

```
lerrx=np.log(errx[:-1])
lerry=np.log(erry[:-1])
print(lerrx)
print(lerry)
plt.plot(lerrx,lerry,lerrx[0],lerry[0],'o',lerrx[-1],lerry[-1],'*')
plt.show()
```

(iii) Ecrire des versions ultérieures de `Newton` plus réalistes :

(a) `def Newton2(x0,N,f,df)`: qui renvoie la liste des itérés : `return liste_x`

(b) `def Newton3(x0,N,f,df)`: qui renvoie les listes des itérés et de leurs images respectives `fx=f(x)` : `return liste_x,return liste_fx`

(c) `def Newton4(x0,f,df,eps)`: qui remplace le nombre de pas imposé a priori par un critère d'arrêt : `while crit > eps and it < itmax:`
où `crit=np.abs(f(x))`, `itmax=20` est le nombre de pas maximum imposé en début de programme, `it` est le nombre de pas initialisé à `it=0`, incrémenté à chaque itération au moyen de la commande `it+=1`. La commande de sortie `return liste_x,liste_crit,it` renvoie la liste des itérés `liste_x`, la liste des valeurs des critères `liste_crit`, le nombre de pas `it` nécessaire pour atteindre la précision donnée en entrée `eps`.

2. Utiliser la méthode de Newton pour la résolution du système polynomial suivant

$$\begin{cases} x^2 + y^2 = 2 \\ x^2 - y^2 = 1 \end{cases}$$

Illustrer de nouveau la convergence une première fois en traçant les itérés dans le plan (x, y), et ensuite en quantifiant l'erreur et l'ordre de convergence. Tester plusieurs initialisations.

(i) Ecrire la version multidimensionnelle de Newton `def Newton5(x0,f,df,eps)` en tenant compte des modifications suivantes :

(a) le critère de sortie reste inchangé à condition de prendre
`crit=np.linalg.norm(f(x))`

(b) La boucle principale devient :
`x=x-np.linalg.solve(df(x),f(x))`

(c) Dans un premier temps, la commande de sortie reste : `return liste_crit,liste_it,it`
Tracer l'évoluion du critère `crit` comme dans le cas monodimensionnel.

(d) Ecrire la version multidimensionnelle de Newton `def Newton6(x0,f,df,eps)` qui calcule les itérés successifs.

(i) Pour cela, la boucle principale :
`x=x-np.linalg.solve(df(x),f(x))`

est complétée par la mémorisation des composantes de x :

```
liste_x[it]=x[0]
liste_y[it]=x[1]
```

La commande de sortie devient :

```
return liste_x,liste_y,liste_crit,liste_it,it
```

(ii) Tracer les itérés sur le graphe représentant l'ensemble des solutions comme intersection du cercle $x^2 + y^2 = 2$ et de l'hyperbole $x^2 - y^2 = 1$.

```
x0=np.array([-3,5]), f=f2, df=df2, eps=1e-6 # Données initiales
t=np.linspace(0,2*np.pi,100)
r=np.sqrt(2)
xt=r*np.cos(t)
yt=r*np.sin(t) # Le cercle
s=np.linspace(-2,2,600)
xs=np.cosh(s)
ys=np.sinh(s)# L'hyperbole
tabx,taby,tabcrit,tabit,it=Newton6(x0,f,df,eps)
tabxeff=tabx[:it]# Tableau effectif des abscisses
tabyeff=taby[:it]# Tableau effectif des ordonnées
plt.plot(xt,yt,xs,ys,-xs,ys,tabxeff,tabyeff)# Tracé des itérés
plt.show()
```

(iii) Evolution de l'erreur : procéder comme dans le cas monodimensionnel avec la nouvelle définition de `err` :

```
err=np.zeros(it)
for i in range(it):
    err[i]=np.linalg.norm(np.array([tabxeff[i]-tabxeff[-1],
                                    tabyeff[i]-tabyeff[-1]]))
```

◇ Sur ce même exemple, on déterminera numériquement une relation entre le nombre d'itérations nécessaire à la convergence pour une donnée initiale $x_0 = (2^{-p}, 1)$ et le nombre entier p).

◇ Lorsque $x_0 = (2^{-10}, 1)$, on pourra enfin tracer le ratio $\ln(\|x_{n+1} - x^\star\|)/\ln(\|x_n - x^\star\|)$ en fonction de n et interpréter les résultats.

3. Déterminer l'ordre de convergence dans le cas de la recherche des zéros de $F_2(x) = (e^x - e)^2$ et $F_3(x) = (e^x - e)^{5/2}$.

☐ Programmer la méthode de Newton modifiée définie par l'itération :

$$x_{n+1} = x_n - p[dF(x_n)]^{-1}F(x_n).$$

où p est la « multiplicité » du zéro recherché.

◇ Reprendre l'étude de convergence de la question **3** par cette méthode modifiée.

☐ Programmer la méthode de quasi-Newton dans laquelle l'application différentielle $dF(x)$ est remplacée par l'approximation suivante, dépendant du réel $h > 0$:

$$\widetilde{dF}(x) : y \mapsto \sum_{i=1}^{d} \frac{1}{h}(f(x + he_i) - f(x))y_i.$$

4. Tester cette méthode sur les exemples traités précédemment, pour différentes valeurs de h, et déterminer l'ordre de convergence effectif.

5. Trouver une approximation des valeurs $\lambda \in [0, 10]$ telles que $\displaystyle\int_0^1 \sin(\lambda e^{x^2})\,\mathrm{d}x = 1/2$.

6. On souhaite obtenir une approximation de la solution $u \in \mathcal{C}^2([0,1])$ du problème différentiel aux limites (dont on admettra l'existence) :

$$u(0) = 1, \quad u(1) = 2, \quad \frac{d^2}{dx^2}(uv) = u^2,\ x \in [0,1], \quad v(x) = 1 + \cos^2(\pi x).$$

Pour ce faire, on se donne un entier $N \geq 2$ et on pose $V_n = 1 + \cos^2(\pi \frac{n-1}{N-1})$ pour $1 \leq n \leq N$. Le problème continu est approché par une formulation aux différences finies sur $[0,1]$. L'inconnue est un vecteur $U_n \in \mathbb{R}^N$ solution de

$$U_1 = 1, \quad U_N = 2, \quad N^2(U_{n+1}V_{n+1} - 2U_nV_n + U_{n-1}V_{n-1}) = U_n^2,\ 2 \leq n \leq N-1.$$

Résoudre ce système approché pour N fixé suffisamment grand.

7. (Extrait du TP précédent)
On considère le problème aux limites suivant :

$$\begin{cases} -u''(x) + a(x)u(x) = f(x), \\ u(0) = u(1) = 0, \end{cases}$$

où les fonctions $a \geq 0$ et f sont données. Pour résoudre ce problème, on introduit le problème de Cauchy dépendant du paramètre α :

$$\begin{cases} -u_\alpha''(x) + a(x)u_\alpha(x) = f(x), \\ u_\alpha(0) = 0 \text{ et } u_\alpha'(0) = \alpha, \end{cases}$$

Ce dernier problème peut être résolu à l'aide d'une méthode d'intégration des équations différentielles ordinaires. Il suffit de trouver α tel que $u_\alpha(1) = 0$, ce qui n'est pas difficile puisque l'application $\varphi : \alpha \mapsto u_\alpha(1)$ est affine !

Appliquer la même méthode au problème non-linéaire

$$\begin{cases} -u''(x) + a(x)u^3(x) = f(x), \\ u(0) = u(1) = 0, \end{cases}$$

pour lequel l'application $\varphi : \alpha \mapsto u_\alpha(1)$ est non-linéaire. On pourra alors utiliser la méthode de Newton ou une variante pour résoudre le problème de tir.