TP3: Résolution d'équations non linéaires

October 12, 2016

Objectifs

1. Méthodes de résolution d'équations (scalaires) non linéaires
2. Utilisation des commandes PYTHON liées aux fonctions

Méthodes classiques de résolution

Dans votre dossier "AN_TP", créer un dossier TP3-NOM et lancer PYTHON dans ce répertoire.

Localisation des racines

Le but ici est de localiser les quatres premières racines positives de la fonction

$$F(x) = \cos(x)\cosh(x) + 1$$

1. Créer un fichier "localisation.py" où vous écrirez vos commandes PYTHON
2. Créer un vecteur X qui contiendra les abscisses des points à tracer, allant de 0 à 13 avec un pas h qui sera saisie par l'utilisateur de votre programme (vous, un collègue, votre chargé de TP).
3. Créer un vecteur Y contenant les images des points X par la fonction F.
4. Tracer le graphe de F sans oublier les axes et les légendes
5. Donner une valeur approchée (à 10^{-1} près) des 4 premières racines positives de F
6. Utiliser les boites noires de PYTHON (en utilisant les librairies adéquates) pour obtenir les 4 premières racines positives de F

Résolution par la méthode de dichotomie

On cherche à avoir une valeur plus précise de la première racine positive de F.

Rappel de cours: On part de l'intervalle $[a_0, b_0]$ dans lequel la fonction F change *une seule fois* de signe et on calcule la valeur de F au point $m = \dfrac{a_0 + b_0}{2}$

- Soit $F(a_0)F(m) < 0$ et dans ce cas la racine appartient à $[a_0, m]$
- Soit $F(b_0)F(m) < 0$ et dans ce cas, la racine appartient à $[m, b_0]$

On localise donc la racine dans un nouvel intervalle $[a_1, b_1]$ de longueur moitié et on recommence le procédé.

A faire

1. Créer un fichier "dichotomie.py" où vous écrirez votre programme
2. Initialiser les valeurs de $a_0, F(a_0)$ et $b_0, F(b_0)$. Vous pourrez utiliser une fonction pour calculer les valeurs de la fonction F en différents points.
3. Faire une boucle "while" avec un test d'arrêt approprié pour calculer de manière itérative les bornes a_k, b_k en faisant les tests avec la commande "if" sans oublier de mettre un compteur pour savoir le nombre d'itérations effectuées.
4. Afficher le résultat

Résolution par la méthode de Newton

Rappel de cours: On part de la valeur X_0 et on calcule $X_1 = X_0 - \dfrac{F(X_0)}{F'(X_0)}$ puis X_2, etc.

A faire

1. Créer un fichier newton.py où vous écrirez vos commandes PYTHON
2. Définir la fonction F, la fonction F' sachant que $\cosh' = \sinh$
3. Utiliser la méthode de Newton pour trouver une valeur approchée de la première racine positive de F. On partira de la valeur entière immédiatement supérieurs à la première racine.
4. Illustrer la vitesse de convergence de l'algorithme en traçant $\dfrac{\log|X_{k+1} - X^*|}{\log|X_k - X^*|}$ où X^* est la première racine de F et $(X_k)_{k\in\mathbb{N}}$ les itérées de l'algorithme de Newton.

Question: donner le nombre d'itérations nécessaire pour obtenir X^* à 10^{-12} près

5. Remplacer la valeur initiale X_0 par la valeur enitère immédiatement inférieure à la première racine: on constate que l'algorithme ne converge pas

Question: Rechercher par tatonnement le domaine de convergence de l'algorithme vers X^*

Résolution par la méthode de Newton avec dérivation numérique

Rappel de cours: Le calcul de la dérivée F' est ici relativement aisé mais ce n'est pas toujours le cas. Il faut alors évaluer *numériquement* une valeur approchée de la dérivée. Pour $h > 0$ "assez petit" (voir TP1 pour un choix judicieux de h), on a:

$$F'(X) \approx \frac{F(X+h) - F(X)}{h}$$

A faire

1. Créer un fichier "newtonD.py" où vous écrirez vos commandes PYTHON
2. Définir une fonction DF permettant de calculer une valeur approchée de F'
3. Programmer la méthode de Newton "approchée" en remplaçant $F'(X_k)$ par $DF(X_k, h)$ dans l'algorithme classique.
4. Refaire les questions de la section précédente en prenant $h = 0.01$ (illustrer en particulier la vitesse de convergence

Question: Combien d'itérations sont nécessaires pour avoir une valeur approchée à 10^{-12} de X^*

Exercices de synthèse

Exercice 1 On cherche à maximiser la fonction

$$L(\alpha) = \frac{l_2}{\sin(\pi - \gamma - \alpha)} + \frac{l_1}{\sin(\alpha)}$$

avec $l_1 = 2, l_2 = 2.5$ et $\gamma = \frac{3\pi}{5}$.

1. Montrer que α est solution de l'équation

$$l_2 \frac{\cos(\pi - \gamma - \alpha)}{\sin^2(\pi - \gamma - \alpha)} - l_1 \frac{\cos(\alpha)}{\sin^2(\alpha)} = 0.$$

2. Ecrire l'algorithme de Newton permettant de résoudre l'équation

3. Programmer l'algorithme de Newton pour calculer une valeur approchée de α avec une tolérance $\varepsilon = 10^{-9}$. On partira de la valeur approchée $\alpha_0 = 0.6$.

Exercice 2 On considère l'équation $\tan(x) = \tanh(x)$

1. Soit $k \in \mathbb{Z}$. On note $I_k =]-\frac{\pi}{2} + k\pi, , \frac{\pi}{2} + k\pi[$. Montrer qu'il existe une solution $x_k \in I_k$ unique à l'équation $\tan(x) = \tanh(x)$.

2. Que vaut x_0?

3. Calculer une valeur approchée de x_1 à 10^{-2} près à l'aide de la méthode de dichotomie.

4. Calculer une valeur approchée de x_1 à 10^{-9} près à l'aide de la méthode de Newton.

Exercice 3 Notons P le polynôme défini par

$$P(x) = 2097152\,x^7 - 4161536\,x^6 + 2731008\,x^5 - 755904\,x^4 + 94488\,x^3 - 5334\,x^2 + 127\,x - 1$$

1. Tracer le graphe de la fonction P

2. Programmer la méthode de Newton à pas double pour calculer la plus grande racine réelle de P en partant de $x_0 = 100$. Mettre en place un test pour vérifier que les itérées ne dépassent pas la plus grande racine: quand c'est le cas, basculer sur la méthode de Newton classique.

3. Mettre en place un algorithme pour calculer les trois racines suivantes (voir le polycopié de cours).