TP 3 : Résolution d'équations non linéaires

Résumé

Attention, veuillez lire l'intégralité du TP avant de commencer. Notez également que ce TP fait l'objet de 8h de formation. Ainsi, veillez à bien noter où vous vous êtes arrêté et assurez un suivi de vos avancées.

L'objectif de ce TP est dans un premier temps de trouver la racine du polynôme $P(x) = x^3 + 4x^2 - 10$ se trouvant dans l'intervalle $[0, 5]$. Puis dans un second temps de comparer les différentes méthodes qui nous permettent de l'obtenir.

1 Préparation

Vous allez tout d'abord créer un nouveau répertoire, contenant un nouveau script principal. Puis, à l'aide de la fonction '*lookfor*' ou d'Internet, cherchez la fonction qui retourne les racines d'un polynôme sous Matlab.

Utilisez cette fonction pour définir quelle est la véritable valeur que l'on souhaite trouver. Stockez cette valeur dans une variable, elle vous permettra de calculer l'erreur plus tard.

Créez la fonction *function* ***[y]*** = *func(**x**)* retourne un vecteur **y** correspondant aux valeurs de $P(x)$ pour un vecteur **x** donné.

Enfin, dans la fenêtre de commande, saisissez la commande *format long*. À l'aide de la documentation, déterminez l'effet de cette commande.

2 Méthode de dichotomie

La méthode de dichotomie est l'une des plus simples en terme d'implémentation. Ainsi :

1. À partir de l'algorithme du cours, créez la fonction suivante :

```
function [xfinal] = dichotomic_func( a , b,tol, iterMax)
%  Fonction de dichotomie qui execute l algorithme de dichotomie sur l'intervalle [a,b] pour trouver la racine présente dans cet intervalle
%
%  * Entree :
%      -> a - Int - Borne inferieure de l intervalle
%      -> b - Int - Borne superieure de l intervalle
%      -> tol - Float - critere d arret
%      -> iterMax - Int - Maximum d iterations de notre algorithme
%
%  * Sortie :
```

```
11 %       -> xfinal - Float - L approximation de notre
       racine
```

2. Testez cette fonction pour différentes valeurs d'intervalle et assurez-vous de sa convergence vers la bonne valeur.

3 Méthode de trichotomie

Vous avez vu en CM-TD que la méthode de dichotomie est assez lente à converger, une des manières d'accélérer la convergence est d'augmenter le nombre de points intermédiaires. Ainsi, nous définirons deux points c1 et c2 situés respectivement au $\frac{1}{3}$ et $\frac{2}{3}$ de l'intervalle et nous effectuerons la même manipulation que la dichotomie pour remanier notre intervalle.

1. À partir du raisonnement ci-dessus, implémentez la fonction avec la signature suivante :

```
1  function [xfinal] = dichotomic2_func(a , b,tol,
       iterMax)
2  %   Fonction de trichotomie qui execute l algorithme
       de dichotomie sur l'intervalle [a,b] pour
       trouver la racine présente dans cet intervalle
3  %
4  %   * Entree :
5  %       -> a - Int - Borne inferieure de l
       intervalle
6  %       -> b - Int - Borne superieure de l
       intervalle
7  %       -> tol - Float - critere d arret
8  %       -> iterMax - Int - Maximum d iterations de
       notre algorithme
9  %
10 %   * Sortie :
11 %       -> xfinal - Float - L approximation de notre
       racine
```

2. Testez cette nouvelle fonction et assurez-vous de sa convergence vers la bonne valeur.
3. Par comparaison avec la méthode de dichotomie, donnez la vitesse de convergence exacte de cette méthode. Peut-on constater une différence d'un point de vue informatique? Cela pourrait-il poser un souci sur la durée?

Maintenant que vous avez deux méthodes à votre disposition, vous pouvez commencer à entamer des réflexions quant à une stratégie de comparaisons. Pour ce TP, nous vous demanderons de prendre en compte :

— L'erreur entre la véritable valeur et la valeur estimée ;
— Le nombre d'itérations pour obtenir un résultat suffisant ;
— La vitesse d'exécution moyenne sur 100 exécutions (voir la fonction '*clock*') ;

NB : Étant donné que ces problèmes ne dépendent d'aucuns phénomènes aléatoires, on considère que l'erreur et le nombre d'itérations n'évoluent pas pour un algorithme donné dans des conditions données. La vitesse d'exécution en revanche dépend de phénomène aléatoire inhérent à votre machine et aux composants qui la constituent : d'où la nécessité de faire une moyenne.

4. Modifiez vos fonctions afin qu'elles retournent le paramètre de sortie '*nbIter*' représentant la dernière itération calculée ;
5. Modifiez vos fonctions afin qu'elles prennent en entrée le paramètre '*trueValue*' représentant la valeur de la racine recherchée et retournent le paramètre '*err*' représentant l'erreur d'approximation à chaque itération ;
6. Rajoutez quelques lignes de commentaires afin de rendre votre code clair pour d'autres utilisateurs.
7. Modifiez votre script en ajoutant une partie qui s'occupera de calculer le temps moyen d'exécution de vos deux algorithmes.

À présent que vous avez testé ces deux fonctions, nous allons utiliser une fonction plus poussée de Matlab. Dans l'état actuel, si je vous demande de changer de fonction à étudier (autre polynôme, fonction trigonométrique,...), vous devrez modifier manuellement le fichier '*func.m*'. Or, le but d'un algorithme est de pouvoir se généraliser à plusieurs fonctions, car le concept mathématique est le même. Ainsi, nous souhaiterions changer une nouvelle fois la signature de nos algorithmes de dichotomie, pour qu'elles contiennent en paramètre une fonction spécifique ('*fun*'). Pour ce faire :

8. À l'aide de votre mémo Matlab, trouvez une méthode pour passer une fonction en paramètre d'entrée de vos fonctions.
9. Faites en sorte que ce paramètre '*fun*' soit appelé à la place de '*func*' dans vos fonctions.
10. Testez vos algorithmes de dichotomie et trichotomie pour différentes fonctions dont vous connaissez les racines afin de vous assurer du bon fonctionnement de l'ensemble.

L'entièreté de ces parties est importante pour les prochaines séances de TP. Si vous n'êtes pas parvenus jusqu'ici à la fin de la première séance, n'hésitez pas à retravailler ces parties en guise de préparation.

4 Méthode du point fixe

La méthode du point fixe est un bon entraînement de fusion entre l'univers des mathématiques et de l'informatique. La première étape est purement mathématiques :

1. À l'aide des exercices réalisés en TD, proposez 3 ou 4 équations qui peuvent être utilisées dans l'algorithme du point fixe pour trouver une de ses racines.

Une fois ces trois-quatre équations en main, vous allez passer à l'étape de l'informatique :

2. Créez pour chaque fonction, une fonction similaire à *fun(**x**)*.
3. À l'aide des différents éléments amenés lors du TP et du TD, implémentez la fonction qui calculera la racine par la méthode du point fixe. Elle présentera la signature ci-dessous.

```
function[xfinal,nbIter, err]=fixedPoint_func(fun,p0
    ,iterMax,tol,trueValue)
%  Fonction d iteration du point fixe (fixed point
    iteration)
%
%  * Entree :
%      -> fun - handle - Pointeur de fonction a
    traiter
%      -> p0 - Float - initial approximation
%      -> tol - Float - critere d arret
%      -> iterMax - Int - Maximum d iterations de
    notre algorithme
%      -> trueValue - Float - veritable valeur de
    la racine
%
%  * Sortie :
%      -> xfinal - Float - L approximation de notre
     racine
%      -> nbIter - Int - Nombre d iterations
    necessaire pour trouver la bonne valeur approchee
%      -> err - [Float] - Valeur de l erreur entre
    l element calcule et la veritable valeur
```

4. Testez l'algorithme pour les différentes fonctions et pour différents points d'initialisation. Que pouvez-vous remarquer pour certaines fonctions ? Est ce normal ?
5. Choisissez la fonction la plus stable, selon vous (demandez confirmation auprès du tuteur) et commentez les autres tests.
6. Testez maintenant cet algorithme pour la fonction : $x - \frac{x^3+4*x^2-10}{(3*x^2+8*x)}$ pour différents points d'initialisation. Que remarquez-vous ? Cette méthode vous rappelle-t-elle quelque chose ?
7. Avant de passer à l'algorithme suivant, faites comme pour les algorithmes de dichotomie et ajoutez dans le script une partie qui calcule la vitesse moyenne d'exécution.

5 Méthode de Newton

1. Créez une variable '*deriv*' qui aura le même rôle que '*fun*' sauf qu'elle retournera les valeurs de la dérivée.
2. À l'aide de '*funcderiv*' et des éléments vus en TD, implémentez à présent la méthode de Newton qui présente la signature suivante :

```
function[xfinal,nbIter, err] = newton_func(fun,deriv
    ,p0,iterMax,tol,trueValue)
%  Fonction de Newton
%
%  * Entree :
%      -> fun - handle - Pointeur de fonction a
    traiter
%      -> deriv - handle - Pointeur sur la derivee
    de fonction a traiter
%      -> p0 - Float - premiere approximation
%      -> tol - Float - critere d arret
%      -> iterMax - Int - Maximum d iterations de
    notre algorithme
%      -> trueValue - Float - veritable valeur de
    la racine
%
%  * Sortie :
%      -> xfinal - Float - L approximation de notre
     racine
%      -> nbIter - Int - Nombre d iterations
    necessaire pour trouver la bonne valeur approchee
%      -> err - [Float] - Valeur de l erreur entre
    l element calcule et la veritable valeur
```

6 Méthode de la sécante

La méthode de la sécante permet de combler un des défaut de la méthode de Newton qui nécessite que l'on connaisse la dérivée.

1. À l'aide des éléments vu en TD, implémentez à présent la méthode de la sécante qui présente la signature suivante :

```
function[xfinal,nbIter, err] = secante_func(fun,a,b,
    iterMax,tol,trueValue)
%  Fonction de la secante
%
%  * Entree :
%      -> fun - handle - Pointeur de fonction a
    traiter
%      -> a - Float - Borne inf de l intervalle de
    recherche
```

```
%       -> b - Float - Borne sup de l intervalle de recherche
%       -> iterMax - Int - Maximum d iterations de notre algorithme
%       -> tol - Float - critere d arret
%       -> trueValue - Float - veritable valeur de la racine
%
%   * Sortie :
%       -> xfinal - Float - L approximation de notre racine
%       -> nbIter - Int - Nombre d iterations necessaire pour trouver la bonne valeur approchee
%       -> err - [Float] - Valeur de l erreur entre l element calcule et la veritable valeur
```

7 Méthode de la fausse position

La méthode de la fausse position apporte un élément supplémentaire sur la méthode de Newton et la méthode de la sécante. En effet, ces deux méthodes ne garantissent pas l'encadrement de la racine. On peut citer en contre-exemple la méthode de dichotomie où l'on sait que la racine est toujours dans l'intervalle grâce au test $f(x_0).f(x_1) < 0$. En résumé, la méthode de la fausse position est identique à la méthode de la sécante excepté que l'on effectue la condition $f(x_1).f(x_new) < 0$ pour redéfinir les bornes. Nous nous assurons ainsi, la conservation de la racine entre les bornes.

1. À l'aide de ces éléments, de vos réflexions personnelles et de vos recherches, implémentez la fonction suivante :

```
function[xfinal,nbIter, err] = falsePos_func(fun,a,b,iterMax,tol,trueValue)
%   Fonction de la fausse position
%
%   * Entree :
%       -> fun - handle - Pointeur de fonction a traiter
%       -> a - Float - Borne inf de l intervalle de recherche
%       -> b - Float - Borne sup de l intervalle de recherche
%       -> iterMax - Int - Maximum d iterations de notre algorithme
%       -> tol - Float - critere d arret
%       -> trueValue - Float - veritable valeur de la racine
%
%   * Sortie :
```

```
13 %       -> xfinal - Float - L approximation de notre
       racine
14 %       -> nbIter - Int - Nombre d iterations
       necessaire pour trouver la bonne valeur approchee
15 %       -> err - [Float] - Valeur de l erreur entre
       l element calcule et la veritable valeur
```

8 Comparaisons de méthode

À l'aide des différents éléments que vous avez développés tout au long des TP, vous allez à présent résumer l'ensemble de vos travaux sous forme d'un court compte rendu contenant les réponses aux différentes questions ainsi qu'une figure et un tableau contenant les résultats d'expérience. Une mise en page élégante serait apprécié et prise en compte dans la note finale. Concernant les modalités d'expérimentation, vous allez fixer les valeurs suivantes :

- $a = 0$
- $b = 5$
- $tolerance = 1e-3$
- $nbMaxIter = 100$
- $initSecant = 0||fonctiondevotrechoix$
- $initNewton = 5$

1. Représentez sur une même figure l'ensemble des erreurs que vous avez obtenues pour chaque technique sur une échelle logarithmique ('*log10(err)*'). Arrangez-vous pour faire apparaître clairement chaque résultat.
2. Complétez le tableau suivant :

	Résultat (7-8 ch.)	Nb Itérations	Vitesse moyenne	Puissance erreur
Dichotomie				
Trichotomie				
Point fixe				
Newton				
Sécante				
Fausse position				

3. Concluez.

9 Exercice complémentaire : Méthode de Steffensen

La méthode de Steffensen est issu d'un long travail de réflexion : on utilise le critère de Aitken pour optimiser au maximum la vitesse de convergence. Cependant, elle a aussi des conditions à respecter.

1. À l'aide de vos recherches personnelles, expliquez en quoi la méthode de Steffensen est intéressante et implémentez la fonction suivante :

```
function[xfinal,nbIter, err]=stef_func(fun,p0,
    iterMax,tol,trueValue)
%   Fonction de Steffensen
%
%   * Entree :
%       -> fun - handle - Pointeur de fonction a
    traiter
%       -> deriv - handle - Pointeur sur la derivee
    de fonction a traiter
%       -> p0 - Float - premiere approximation
%       -> tol - Float - critere d arret
%       -> iterMax - Int - Maximum d iterations de
    notre algorithme
%       -> trueValue - Float - veritable valeur de
    la racine
%
%   * Sortie :
%       -> xfinal - Float - L approximation de notre
     racine
%       -> nbIter - Int - Nombre d iterations
    necessaire pour trouver la bonne valeur approchee
%       -> err - [Float] - Valeur de l erreur entre
    l element calcule et la veritable valeur
```

2. Vous ajouterez cette méthode à votre travail et agrandirez le tableau de résultats en conséquence.