Université de Caen
UFR Sciences
L3 Maths

1er semestre 2023–2024
Portail MIM
Calcul formel

TP noté du 9 octobre 2023

Les notes et corrigés des TP précédents sont autorisés. La consultation de sites web et les communications entre étudiantes et étudiants sont interdits. On importera les modules `math` et `matplotlib`, à l'exclusion de tout autre module. On utilisera `abs(x) < prec` pour tester `x == 0`, avec `prec = 1e-15` (précision machine). Toute fonction mathématique sera définie comme une procédure Python.

Exercice 1. On rappelle que la méthode de Newton pour chercher une annulation de g est une méthode de point fixe pour la fonction $\Phi(x) = x - \frac{g(x)}{g'(x)}$, c'est-à-dire qu'elle consiste à calculer les premiers termes de la suite récurrente $x_{n+1} = \Phi(x_n)$.
Dans cet exercice on va étudier la méthode de Halley pour rechercher une annulation d'une fonction f strictement croissante. Elle consiste à appliquer la méthode de Newton à la fonction $g : x \mapsto f(x)/\sqrt{f'(x)}$. Pour cela on a besoin de connaître f, f' et f''.

1. Au brouillon, expliciter la fonction Φ obtenue en remplaçant g par $f/\sqrt{f'}$. *Le résultat sera utilisé pour la question suivante, on ne demande pas de le noter dans le script. On notera qu'il peut s'écrire sans racine carrée.*
2. Écrire une fonction `Halley(f,g,h,x0,p,N)` qui recherche un zéro de la fonction `f` par cette nouvelle méthode, en partant du point `x0`. On suppose que les arguments `g` et `h` passés à la fonction contiennent respectivement f' et f''. Le paramètre `p` représente la précision recherchée. La fonction retournera le couple (x, n), où n est le plus petit nombre d'itérations tel que $|f(x_n)| < p$. Si le nombre d'itérations dépasse la valeur du paramètre `N`, la procédure retournera `False`. On essaiera de minimiser les appels aux fonctions passées en argument.
3. Utiliser la fonction `Halley` pour calculer une approximation de la racine cubique de 4. On utilisera les paramètres $p = 10^{-12}$, $N = 100$ et $x_0 = 2$. Vérifier que le résultat obtenu est bien la racine cubique de 4.
   Faire le même test en partant de $x_0 = -2$. Que se passe-t-il ? Adapter la fonction `Halley` pour éviter qu'elle provoque l'interruption du script dans ce genre de cas.
4. On souhaite comparer la fonction `Halley` avec la fonction `Newton` classique (rappelée dans le script). Afficher le nombre d'itérations nécessaires, avec `Halley` et avec `Newton`, pour calculer une approximation de la racine cubique de 4 comme ci-dessus, en partant des valeurs $x_0 = 1, 2, 3, \dots, 9$. Conclusion ? La fonction `Halley` a-t-elle cependant des inconvénients ? *Répondre en commentaire dans le script.*

Exercice 2. Pour $n \in \mathbb{N}^*$ fixé on considère les polynômes de Bernstein $B_0, \dots, B_n$ donnés par la formule $B_k(t) = \binom{n}{k} t^k (1-t)^{n-k}$. Soit $P = (p_0, \dots, p_n)$ une liste de $n+1$ points du plan, $p_i = (x_i, y_i) \in \mathbb{R}^2$. Pour tout $t \in [0, 1]$ on considère le point

$$b(t) = \sum_{k=0}^{n} B_k(t) p_k \in \mathbb{R}^2.$$

On obtient ainsi une courbe paramétrée $b : [0, 1] \to \mathbb{R}^2$, appelée courbe de Bézier aux points de contrôle P. On peut vérifier qu'on a $B_k(t) \geq 0$ et $\sum_{k=0}^{n} B_k(t) = 1$ pour tout $t \in [0, 1]$, ainsi $b(t)$ est pour tout t un barycentre des points $p_0, \dots, p_n$ donnés. De plus on a $b(0) = p_0$, $b(1) = p_n$: la courbe relie le premier point donné au dernier. En général elle ne passe *pas* par les points intermédiaires.

1. Écrire une procédure `B(n,k,t)` qui calcule $B_k(t)$. On incrémentera un compteur global `mult` pour compter le nombre total de multiplications effectuées (il faut $l - 1$ multiplication pour calculer t^l). On utilisera la fonction `math.comb(n,k)` qui calcule le coefficient binômial $\binom{n}{k}$ sans incrémenter `mult` (on peut imaginer que ces coefficients sont stockés dans une table).
2. Écrire une procédure `Bezier(P,t)` qui calcule pour `t`$\in [0, 1]$ le point $b(t)$ de la courbe paramétrée associée à une liste de points `P = [[x0,y0], [x1,y1], ..., [xn,yn]]` passée en argument. On incrémentera le compteur global `mult` là où c'est nécessaire.
3. Tracer la courbe de Bézier associée à la liste de points `P = [[0,0], [0,2], [2,0], [2,4]]`.
   On placera une centaine de points. On remplira les listes `X`, `Y` des abscisses et ordonnées des points à placer en utilisant la procédure `Bezier(P,t)` puis on utilisera la librairie `matplotlib` pour placer et relier ces points.

suite au verso

Un autre algorithme est possible pour calculer $b(t)$ en se basant sur l'interprétation barycentrique évoquée précédemment. Partant d'une liste de $n+1$ points $(p_0, \ldots, p_n)$, on considère pour $k = 0, \ldots, n-1$ le point q_k égal au barycentre de p_k et p_{k+1}, avec coefficients respectifs $1-t$ et t. On obtient ainsi une nouvelle liste de points $(q_0, \ldots, q_{n-1})$. On recommence cette opération jusqu'à arriver à une liste contenant un seul point z_0. Alors ce point z_0 est le point $b(t)$ recherché (on ne demande pas de le démontrer). C'est l'algorithme de De Casteljau.

4. Écrire une procédure `DeCasteljau(P,t)` qui calcule $b(t)$ à l'aide de ce nouvel algorithme, toujours en incrémentant le compteur `mult`. On privilégiera une mise en œuvre itérative qui consomme moins de mémoire, mais une procédure récursive est également possible.
5. En reprenant la liste `P` de la question 3, calculer $b(0.4)$ de deux manières, en utilisant `Bezier` et `DeCasteljau`. Vérifier qu'on obtient le même résultat et comparer le nombre de multiplications effectuées dans les deux cas.
6. Calculer en fonction du nombre de points $N = n+1$ le nombre de multiplications effectuées par chacune des deux méthodes. Commenter. *On répondra en commentaire dans le script.*