X Informatique MP/PC 2013 — Corrigé

Ce corrigé est proposé par Arnaud Borde (École polytechnique) ; il a été relu par Simon Billouet (ENS Cachan) et Benjamin Monmege (ENS Cachan).

Cette épreuve est commune aux filières PC et MP option sciences de l'ingénieur et ne compte que pour l'admission. Ce corrigé a été rédigé en Maple car ce langage est le plus utilisé en prépa, même s'il est plus destiné au calcul formel qu'à la programmation. D'autres langages comme C++, Caml, Java et Python étaient autorisés. Les fonctions écrites n'utilisant que la syntaxe de base, vous pourrez sans difficulté les réécrire dans le langage de votre choix.

Le sujet propose l'étude des points fixes de fonctions à domaine fini, en se basant sur les ensembles $\mathsf{E}_n = \{0, \dots, n-1\}$. Il est composé de deux parties.

- La première s'intéresse au cas général et aux notions d'attracteur principal et de temps de convergence.
- La seconde examine le cas particulier des fonctions croissantes, d'abord au sens usuel du terme puis généralisé avec une relation d'ordre binaire quelconque. Le sujet se termine avec l'exemple de la divisibilité en tant que relation d'ordre.

Cette année, quatre questions ne demandent pas d'écrire de code. Deux d'entre elles (les questions 11 et 12) consistent en des démonstrations de mathématiques et les deux autres demandent de justifier la complexité d'un code.

C'est un sujet plus difficile que les années précédentes ; en particulier, les questions demandant d'écrire des algorithmes de complexité logarithmique peuvent être considérées comme hors-programme. En dehors de ces questions, il est à la portée d'un élève ayant un minimum préparé cette épreuve spécifique à ce concours. Une bonne maîtrise de la programmation récursive permet de gagner du temps sur un certain nombre de questions. L'énoncé dans son ensemble est clair et les questions bien détaillées. On regrettera juste la remarque sur le type de passage par paramètre, qui en plus d'être elle aussi hors-programme, est inutile pour la plupart des candidats, cette distinction n'existant pas en Maple.

Rédiger du code sur papier est un exercice auquel il convient d'accorder un soin particulier. Un code (sur papier ou sur ordinateur) est très rarement juste dès le début : un brouillon s'impose donc pour éviter de multiples ratures et rajouts. De même, afin de faciliter la lecture du code par le correcteur, il est important de l'indenter et de l'espacer afin d'en dégager la structure (quelles sont les parties qui le composent, à quelles boucles appartiennent les instructions). Il ne faut pas non plus hésiter à commenter (avec # sous Maple) son code, surtout s'il est long, et à expliquer les grandes lignes de sa démarche.

Indications

Partie I. Recherche de point fixe : cas général

1 On peut s'arrêter dès que l'on rencontre un point fixe.

2 Parcourir le tableau et tenir à jour un compteur.

3 Le code peut être écrit soit avec une boucle `for` soit de manière récursive.

4 Combiner les codes des questions 2 et 3.

5 Remarquer qu'une fonction qui admet un attracteur principal n'a qu'un seul point fixe et que cet attracteur principal est atteint en au plus n itérations pour chaque entier.

6 Utiliser la formule de récurrence donnée par l'énoncé.

7 Construire récursivement grâce à la formule de l'énoncé un tableau contenant tous les temps de convergence.

Partie II. Recherche efficace de points fixes

8 La condition de croissance d'une fonction f représentée par un tableau `t` est donnée par $t[i] \leqslant t[i+1]$ pour $0 \leqslant i \leqslant n-2$.

9 Procéder par dichotomie sur le domaine E_n en utilisant l'assertion de l'énoncé.

10 Un algorithme logarithmique a pour particularité le fait que doubler la taille du tableau rajoute un nombre fixe d'instructions (correspondant à un « tour de boucle »).

11 Appliquer k fois la croissance de f.

12 Ne pas oublier que 1 divise tous les entiers et en particulier tous les x_m.

13 Utiliser le résultat de la démonstration précédente.

14 Regarder comment évolue la suite des itérés de 1 par f.

Dans Maple, contrairement à la plupart des autres langages de programmation, les indices de tableaux commencent par défaut à 1 et non à 0. Il est préférable de préciser la convention que vous adoptez pour les indices car le sujet suppose qu'ils débutent à 0. Les rapports du concours des années précédentes recommandent d'ailleurs de suivre l'énoncé plutôt que les conventions du langage utilisé, quitte à écrire un code qui n'est pas compilable/exécutable. C'est ainsi que ce corrigé a été rédigé.

L'énoncé travaillant beaucoup avec des tableaux, il y a deux opérations courantes qui reviennent souvent : obtenir la taille d'un tableau et initialiser un tableau de taille donnée avec une valeur donnée. L'énoncé nous indique l'existence d'une fonction `allouer`, qui permet uniquement d'allouer la mémoire nécessaire à un tableau de taille n donnée (ce qui est surtout utile dans d'autres langages que Maple, comme C ou Java). Comme elle ne garantit rien sur l'initialisation du tableau, une boucle d'initialisation est nécessaire après son appel. Par exemple pour définir un tableau `t` de taille `n` rempli de zéros, on écrit (car les indices d'un tableau de taille n vont de 0 à $n-1$) :

```
t := allouer(n);
for i from 0 to n-1 do
  t[i]:=0;
od;
```

On peut aussi utiliser l'expression `[seq(0, i=0..n-1)]` pour initialiser un tableau de taille `n`. Pour ce corrigé, nous avons choisi la fonction `nops()` pour obtenir le nombre d'éléments d'un tableau. Les années précédentes, une fonction `taille` était supposée exister, sur le modèle d'`allouer`.

I. Recherche de point fixe : cas général

1 Il s'agit ici de parcourir le tableau et de retourner `vrai` dès que l'on rencontre un point fixe. `faux` est retourné si l'on a parcouru l'ensemble du tableau sans trouver de point fixe. Une boucle `for` est utilisée pour parcourir le tableau et l'on met à profit le fait qu'une instruction `RETURN()` arrête immédiatement la procédure.

```
admet_point_fixe:=proc(t)
  local i;
  for i from 0 to nops(t)-1 do
    # on teste si i est un point fixe
    if t[i] = i then
      RETURN(vrai);
    fi;
  od;
  # on ne passe ici que si on a parcouru l'ensemble
  # du tableau sans trouver de point fixe
  RETURN(faux);
end;
```

2 Cette question est très similaire à la précédente, la différence étant qu'au lieu de retourner `vrai` lorsque l'on rencontre un point fixe, un compteur `pts_fixes`, initialisé à 0, est incrémenté.

```
nb_points_fixes:=proc(t)
  local i, pts_fixes;
  pts_fixe := 0;
    for i from 0 to nops(t)-1 do
    if t[i] = i then
      pts_fixes := pts_fixes + 1;
    fi;
  od;
  RETURN(pts_fixes);
end;
```

3 Traitons cette question avec un code récursif. En effet, les fonctions f^k vérifient la relation de récurrence : pour tout $x \in \mathsf{E}_n$

$$f^0(x) = x \qquad \text{et} \qquad \forall k \geqslant 1, \quad f^{k+1}(x) = f(f^k(x))$$

La première ligne représente la condition d'arrêt du code.

```
itere := proc(t, x, k)
  if k = 0 then
    RETURN(x);
  else
    RETURN(itere(t, t[x], k-1));
  fi;
end;
```

> On peut aussi écrire un code itératif en appliquant k fois la fonction f, c'est-à-dire le tableau `t`, à l'aide d'une boucle `for`.
>
> ```
> itere := proc(t, x, k)
>   local i, res;
>   res := x;
>   for i from 1 to k do
>     res := t[res];
>   od;
>   RETURN(res);
> end;
> ```

4 Modifions le code de la procédure `nb_point_fixes` de la question 2 en remplaçant le test `t[i] = i` par le même test mais sur l'itérée : `itere(t, i, k) = i`.

```
nb_points_fixes_iteres:=proc(t, k)
  local i, pts_fixes;
  pts_fixe := 0;
  for i from 0 to nops(t)-1 do
    if itere(t, i, k) = i then
      pts_fixes := pts_fixes + 1;
    fi;
  od;
  RETURN(pts_fixes);
end;
```