TP3 : Familiarisation avec MATLAB et le travail sur les tableaux

Le langage interprété MATLAB se fonde, on le verra, comme son nom l'indique (MATrixLABoratory), sur le travail à partir de tableaux (matérialisés en termes mathématiques par des matrices). L'objectif de cette troisième séance de TP est de poursuivre la familiarisation avec ce logiciel au travers précisément de ce travail sur les tableaux. On s'initiera aussi à la représentation visuelle avec les instructions `plot` (représentation graphique des fonctions d'une variable réelle), `mesh` ou `surf` (représentation graphique en 3D des fonctions de deux variables réelles), `image`, `imagesc`, `imshow` (visualisation des tableaux sous forme d'images, la brilliance des pixels traduisant la taille des entrées, ce de manière inversement proportionnelle : blanc = grande entrée numérique, noir=petite entrée numérique). Vous serez aussi amenés avec les commandes du graphisme telles `colormap`.

Une initiation à MATLAB (et Scilab) en `pdf` (malheureusement non interactive) est disponible sur le lien

`http://www.math.u-bordeaux1.fr/~ayger/initMS.pdf`

Téléchargez la depuis ce lien et sauvez la copie dans votre répertoire `TPMATLAB`. Vous imprimerez ultérieurement ce fichier pour le consulter à tête reposée.

Ouvrez pour l'instant MATLAB en tapant sous votre terminal la commande `matlab` : vous devez disposer de quatre espaces (que vous pouvez visualiser ou non en utilisant les onglets `Desktop` et `Window` sur le bandeau) :

- *Command Window* (l'espace pour vous le plus important) où vous allez taper le plus souvent vos instructions (derrière de prompt `>>`) et voir s'afficher les résultats au terme de leur exécution [1]. Il est utile d'exploiter le fait que la machine garde en mémoire les instructions déjà effectuées (ceci est bien utile lorsque vous voulez taper une instruction similaire à une instruction déjà effectuée : tapez les premières lettres et vous verrez apparaitre l'ancien modèle que vous pourrez alors corriger).
- *Command History* : c'est ici que s'affice l'historique des commandes. C'est une fenêtre que vous pouvez fermer pour ne pas encombrer votre écran de console.
- *Current Directory* : vous voyez dans cet espace le contenu du répertoire dans lequel vous vous êtes placé (qui devrait être toujours le répertoire `TPMATLAB` crée lors du TP1). C'est en particulier dans ce répertoire que se trouveront

1. Attention : mettre `;` derrière une instruction vous évite de voir l'affichage (c'est exactement le contraire de ce qui se passe avec `Maple12!`) ; ne rien mettre derrière vous fait au contraire courir le risque de voir s'afficher un résultat, ce qui n'est en règle générale pas souhaitable, surtout lorsque les variables en jeu sont de grands tableaux.

tous les fichiers `.m` correspondant à des `script` (par exemple commençant par `[.,., ...,.] = function (.,...,.)`) que vous aurez à écrire ou à utiliser, les fichiers `.mat` correspondant aux sessions ou aux variables que vous sauverez éventuellement, ainsi que les fichiers de données numériques (tableaux de nombres, images `jpeg`, *etc.*) préalablement téléchargés.
- *Workspace* : ici se trouvent recensées les variables actuellement affectées dans votre présente session de travail.

Les commandes `Unix` restent valables sous l'environnement `MATLAB` ; reportez vous au fascicule de TP 2011-2012 (F. Pazuki), page 12, pour un rappel de ces commandes avec les touches `Ctrl + [...]`.

Quelques tests préliminaires. Au contraire de `Maple12` (qui est un logiciel interprété de calcul symbolique ou formel), donc entrainé aux calculs *sans pertes* lorsque les variables d'entrée sont des entiers, des rationnels, ou des polynômes à coefficients entiers, rationnels, voire algébriques, `MATLAB` est un logiciel interprété de calcul scientifique, où les calculs numériques impliquent en général des nombres réels ou complexes déclarés en flottants (`floating`, soit -c'est le cas par défaut- en double précision `double`, ou bien en simple précision `single`) et sont donc tributaires des arrondis inévitables liés à l'erreur machine (en accord toutefois avec la norme du standard `IEEE754` (*cf.* les sections 1.2.3 et 1.2.4 du polycopié de cours). Testez les commandes suivantes :

```
>> eps
>> realmax
>> realmin
```

A l'aide de `help` (exemple : tapez `help eps`), analysez le sens de ces variables modifiables ; comment interpréter le résultat de la commande

```
>> eps(x)
```

lorsque `x` est un nombre flottant (faites le test avec `eps(pi)`, avec `eps(1)`, `eps(1/2)`, `eps(`$10^{(20)}$ `+ pi)`, `eps(`10^7`+pi)`). Conclusion ? En quoi `eps(x)` peut il être considéré comme un indicateur de la précision au niveau du flottant `x` ? Le logiciel n'affiche par défaut que quatre chiffres représentatifs pour les nombres réels flottants (en double précision) ; pour en voir 16 (comme cela est théoriquement possible, aux arrondis près, *cf.* la valeur de `eps(x)`), il faut préalablement taper l'instruction :

```
>> format long
```

Testez ceci sur des exemples (par exemple sur les variables non effacables `i`, `j`, `sqrt(-1)`). Attention : `j` est le `i` des physiciens, ce n'est pas ici le `exp(2*i*pi/3)` des mathématiciens !

EXERCICE 1 (Manipulation de tableaux numériques sous `MATLAB`). Les commandes `rand (M,N)` et `randn (M,N)` génèrent des matrices à `M` lignes et `N` colonnes dont les entrées sont des réalisations de variables aléatoires réelles (mutuellement indépendantes) suivant respectivement :
- en ce qui concerne la fonction `rand`, la loi uniforme sur $[0, 1]$ (chaque entrée est un nombre flottant aléatoire entre 0 et 1, tous les choix étant équiprobables) ;
- en ce qui concerne la fonction `randn`, la loi normale (moyenne 0 et écart type égal à 1) sur $\mathbb{R}$.

(1) Générez de telles matrices suivant les instructions :

```
>> A = rand(10,8);
>> A
>> Adouble = [A A];
>> Adouble
>> B = randn(12,9);
>> B
>> Bdouble = [B;B];
>> Bdouble
>> AA = A(1:2:10,1:2:8);
>> AA
>> BB = B(1:3:8,2:2:9);
>> ONES = ones (7,5);
>> ONES
>> ZEROS = zeros (10,7);
>> ZEROS
>> EYE = eye (10,5);
>> EYE
>> AAA = flipud (A);
>> AAA
>> BBB = fliplr (B);
>> BBB
```

Analysez l'opération qu'exécute chacune de ces instructions. Qu'exécute en particulier l'instruction `Mat1=Mat(1:pas1:M,1:pas2:N)` si `Mat` est un tableau numérique à `M` lignes et `N` colonnes, `pas1` et `pas2` étant des entiers respectivement entre 1 et `M` et 1 et `N` ?

(2) Construisez une matrice `Apad` à 16 lignes et 16 colonnes en bordant la matrice `A` générée à la question (1) de manière symétrique (des deux côtés) par des blocs de zéros [2]. Faites ensuite la même chose avec cette fois des blocs de 1.

(3) Créez une matrice à 19 lignes et 15 colonnes dont les lignes et les colonnes sont celles de la matrice `A` générée à la question (1), mais séparées cette fois entre elles par des lignes et des colonnes de zéros.

(4) Quelle est la méthode la plus judicieuse (au niveau du temps d'exécution) pour déclarer sous `MATLAB` les matrices :

$$U = \begin{pmatrix} 0 & 0 & 3 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 8 \\ 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 8 & 0 \\ 1 & 0 & 0 & 0 & 0 & 5 \end{pmatrix} \qquad V = \begin{pmatrix} 1 & 1 & 3 & 1 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 & 8 \\ 1 & 2 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 8 & 1 \\ 1 & 1 & 1 & 1 & 1 & 5 \end{pmatrix} \quad ?$$

Déclarez ces deux matrices de la manière que vous jugerez la plus efficace possible (au niveau du temps d'exécution).

Sauvez votre session dans votre répertoire `TPMATLAB` (celui dans lequel vous êtes précisément en train de travailler) avec l'instruction `save TP3exo1`. Vérifiez que

2. Pareille opération, fort utile dans la pratique, soit pour compléter une matrice rectangulaire en une matrice carrée pour des calculs ultérieurs, soit pour s'accorder une ≪ marge de sécurité ≫, telle un ≪ cadre ≫, autour d'un tableau ou d'une image, s'appelle le *zeropadding*.

vous avez bien ainsi créé un fichier `TP3exo1.mat` dans votre répertoire `TPMATLAB`. Faites ensuite l'instruction `clear all` pour rendre à nouveau vierge tout votre *Workspace*. Vérifiez que c'est bien le cas. Que se passe t'il si vous donnez les instructions :

```
>> load TP3exo1
>> who
```

Faites `clear all` à nouveau.

EXERCICE 2 (décomposition de Haar d'une image). Voici encore un exercice prétexte à la manipulation de tableaux sous `MATLAB`. Vous y verrez aussi l'étroite relation entre tableaux de nombres (donc matrices) et images. Téléchargez depuis le site `http://www.math.u-bordeaux1.fr/~yger/initiationMATLAB` le fichier (de données) `imageTP3exo2`. Une fois ceci fait, « chargez » ce fichier dans votre *Workspace* en tapant sous la *Command Window* les instructions :

```
>> load imageTP3exo2
>> I=imageTP3exo2;
```

Grâce à la commande `size`, vérifiez que la variable ainsi déclarée `I` est un tableau à 256 lignes et 256 colonnes [3]. Testez les valeurs de quelques entrées `I(i,j)` pour voir qu'il s'agit en fait d'une matrice dont les entrées sont des nombres entiers positifs (mais considérés ici comme des nombres flottants en double précision, comme la réponse `1` à l'instruction `isfloat(I)` vous le prouvera, vérifiez le).

(1) Visualisez la matrice `I` de deux manières :
   - en utilisant la visualisation en 3D suivant `mesh(I)` ;
   - en utilisant la visualisation en 2D suivant `image(I)`, `imagesc(I)` (voire aussi `imshow(I,[])` si vous êtes au CREMI, *cf.* plus loin).

   Quelle est, pour ce type de matrice `I`, de ces deux visualisations, celle qui est la plus adaptée ? Quel est l'effet de la commande

   ```
   >> II = imrotate(I,angle);
   ```

   (où `angle` désigne la valeur en flottant d'un angle entre 0 et 360 degrés exprimé en flottant) ? [4]. On utilisera l'une des routines du type `image` mentionnées plus haut pour visualiser l'image `II` ainsi qu'une instruction convenable (faire `help axis`) pour que l'affichage respecte la taille (ici carrée) de l'image. Quelle est la difficulté liée à la réalisation mathématique d'une telle transformation de matrice (pensez au passage entre le repérage cartésien dans une grille de pixels et le repérage polaire) ? Que se passe t-il en particulier lorsqu'un pixel (i, j) tourne d'un certain angle ? Tombe t'on encore sur un pixel de la grille cartésienne ?

(2) Ouvrez un fichier `.m` vierge en utilisant l'onglet *File* du bandeau. Vous allez essayer dans ce fichier de rédiger un code, dont la première instruction (déclaration des entrées et sorties de la fonction) sera

   ```
   function [RR,DH,DV,DO] = Haar (ipt)
   ```

3. Notez que 256 est une puissance de 2, ce qui n'est, on le verra plus tard, nullement un hasard en ce qui concerne les formats d'images `jpeg`.

4. Cette instruction `imrotate` bien utile n'est malheureusement pas présente dans la bibliothèque de `Scilab`.

qui est censé calculer, étant donnée une matrice de flottants `ipt` de taille $(2^N, 2^N)$, les quatre matrices, de taille $(2^{N-1}, 2^{N-1})$, dont les entrées sont respectivement les nombres

$$\begin{aligned}
\texttt{RR}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)+\texttt{ipt}(2i-1,2j)+\texttt{ipt}(2i,2j-1)+\texttt{ipt}(2i,2j)}{2}\\
\texttt{DH}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)+\texttt{ipt}(2i-1,2j)-\texttt{ipt}(2i,2j-1)-\texttt{ipt}(2i,2j)}{2}\\
\texttt{DV}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)-\texttt{ipt}(2i-1,2j)+\texttt{ipt}(2i,2j-1)-\texttt{ipt}(2i,2j)}{2}\\
\texttt{DO}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)-\texttt{ipt}(2i-1,2j)-\texttt{ipt}(2i,2j-1)+\texttt{ipt}(2i,2j)}{2}
\end{aligned}$$

Testez votre code en l'exécutant avec comme matrice `ipt` la matrice `I` :

```
>> [RR, DH, DV, DO] = Haar(I);
```

Affichez avec `image` (ou mieux `imagesc`) les quatre matrices `RR, DH, DV, DO`. Pour le rendu des couleurs et du graphisme, vous pouvez jouer avec la commande `colormap` (exemples les instructions `colormap hot`,`colormap jet`, `colormap pink`, `colormap jet`, *etc.*, faire `help colormap` pour décider). Pour un meilleur rendu graphique, vous pouvez aussi faire appel à l'instruction

```
>> imshow(MATRICE,[ ])
```

Cette commande est dans la bibliothèque du *Toolbox* `Image Processing` (disponible sous `MATLAB` au CREMI). Justifiez la terminologie : `RR` = *Résumé*, `DH` = *Détails Horizontaux*, `DV` = *Détails Verticaux*, `DO` = *Détails Obliques*. Recommencez le traitement de la matrice `I` cette fois sur la matrice `RR`. Qu'observez vous sur les quatre images de taille $(64, 64)$ ainsi obtenues ? Poursuivez si vous voulez pour obtenir quatre images de taille (32,32) en décomposant le nouveau résumé. La décomposition que vous êtes en train d'effectuer ici s'appelle *décomposition de Haar* : elle permet d'≪ isoler ≫ les ≪ structures ≫ cohérentes des détails (ou ≪ accidents ≫) d'une image digitalisée. Sauvez votre fichier `.m` sous le nom `Haar` si vous ne l'avez pas encore fait et fermez le.

(3) Ouvrez un nouveau fichier vierge `.m` depuis l'onglet *File* du bandeau. Rédigez dans ce fichier un code (correspondant encore à une fonction) dont la première instruction sera

```
function II = HaarInverseAux (I1,I2,I3,I4)
```

qui, étant donnée quatre matrices `I1,I2,I3,I4` de taille $(2^{N-1}, 2^{N-1})$, fabrique une matrice `II` de taille $(2^N, 2^N)$ telle que

$$\texttt{II}(2i-1,2j-1)=\texttt{I1}(i,j),\ \texttt{II}(2i-1,2j)=\texttt{I2}(i,j)$$
$$\texttt{II}(2i,2j-1)=\texttt{I3}(i,j),\ \texttt{II}(2i,2j)=\texttt{I4}(i,j)$$

pour toute paire d'entiers (i, j) entre 1 et 2^{N-1}. Sauvez ce nouveau fichier `.m` sous le nom `HaarInverseAux`. Vous pourrez être amenés à utiliser ce code comme code auxiliaire par la suite.

(4) Ouvrez un nouveau fichier `.m` où vous réaliserez un code (toujours une fonction) dont la première instruction sera cette fois

`function I=HaarInverse (RR,DH,DV,DO)`

qui, étant données quatre matrices `RR, DH, DV, DO`, toutes de même taille $(2^{N-1}, 2^{N-1})$, reconstitue une matrice I de taille $(2^N, 2^N)$ de manière à ce que ce code exécute l'opération inverse de celle exécutée par la fonction `Haar`. Validez votre résultat en utilisant la matrice `I` de la question 1. Sauvez votre fichier `.m` comme `HaarInverse`.

EXERCICE 3 (Réalisation d'une fonction `in line`). Une fonction `g` d'une variable (réelle ou complexe) peut être déclarée sous MATLAB par la commande

```
g = inline('x^2-1','x');
```

(1) Déclarez sur ce modèle la fonction

$$x \in \mathbb{R} \mapsto \frac{\sin(\pi x)}{\pi x},$$

dite aussi *sinus cardinal*. Représentez ensuite le graphe de cette fonction sur $[-10, 10]$, le pas d'échantillonnage étant choisi pour les abscisses égal à $1/100$. Utilisez pour cela la commande `plot`. Quel est l'effet de l'instruction `plot(x,y,'r')` par rapport à l'instruction `plot(x,y,'k')`, ou à la simple instruction `plot(x,y)` ? Déclarez la nouvelle fonction

$$x \in \mathbb{R} \mapsto \frac{\sin(\pi x)}{\pi(\sqrt{x^2+1})}.$$

En utilisant `hold` (faire `help hold`), affichez le graphe de cette seconde fonction (toujours au dessus de $[-10, 10]$ avec le même pas) sur la même figure que précédemment, mais avec une autre couleur que le graphe précédent.

(2) On considère maintenant les deux fonctions, cette fois de deux variables,

$$F : (r, \theta) \longmapsto r^5 \cos(5\theta) + r^3(\cos(3\theta))/2 + 1$$
$$G : (r, \theta) \longmapsto r^5 \sin(5\theta) + r^3(\sin(3\theta))/2$$

(on notera qu'il s'agit de la partie réelle et de la partie imaginaire de la fonction polynomiale $z \mapsto z^5 + z^3/2 + 1$, exprimées toutes deux en coordonnées polaires). Ouvrez un fichier `.m` (que vous dénommerez, en le sauvant, `TP3exo3`) et dans lequel, sur les deux premières lignes, vous déclarerez ces deux fonctions `F` et `G`, fonctions des variables `r` et `theta`, sur le modèle suivant lequel vous avez déclaré $x \mapsto \sin(x)/x$ à la première question. Sur les lignes qui suivent, rédigez les instructions qui conduisent :

- à l'évaluation des fonctions F, G et $\sqrt{F^2+G^2}$ sur le maillage `r=0:2/100:2`, `theta=0:pi/100:pi` de $[0, 2] \times [0, \pi]$ (de pas $2/100$ en abscisse et $\pi/100$ en ordonnée). Attention ! pensez à déclarer l'une des deux variables `r` ou `theta` (à vous de décider laquelle) en ligne et l'autre en colonne, car il faut que `F(r,theta)` soit une matrice carrée de taille ici $(101, 101)$, et non un scalaire !
- à l'affichage en trois dimensions (suivant la commande `mesh`), dans cet ordre, sur trois figures successives, des graphes des trois fonctions $F, G, \sqrt{F^2+G^2}$ au dessus de la grille carrée définie plus haut ;
- avec l'instruction finale `axis([0 pi 0 2 0 1])` (faire `help axis` pour en comprendre la syntaxe), à un zoom sur la dernière figure obtenue, à savoir celle du graphe de $\sqrt{F^2+G^2}$, permettant ainsi

de localiser (au moins grossièrement) les trois zéros du polynôme $P(X) = X^5 + X^3/2 + 1$ situés dans le demi plan $\operatorname{Im} z \geq 0$ du plan complexe.

En utilisant l'onglet *Desktop* dans le bandeau du fichier `TP3exo3` que vous venez de remplir, lancez l'exécution des instructions rédigées dans ce code. Localisez les valeurs approchées des modules et des arguments des trois racines de $P(X)$ de partie imaginaire positive ou nulle. Que peut-on dire des deux racines manquantes (dans tout le plan complexe cette fois) ?

Nettoyez votre *Workspace* avec `clear all` (après avoir vérifié que votre fichier `TP3exo3` a bien été sauvé comme un fichier `.m` dans votre répertoire `TPMATLAB`).

EXERCICE 4 (Maillages et évaluation de fonctions sur un maillage). Faites `help meshgrid` et analysez l'exemple qui est proposé sous ce `help` pour l'évaluation de la fonction

$$(x, y) \mapsto x \exp(-x^2 - y^2)$$

au dessus d'un maillage de $[-2, 2]^2$ avec un pas de .2 suivant les deux axes de coordonnées.

(1) Sur le même modèle que dans l'exemple ainsi défini, construire 3 variables `X,Y,Z` de manière à ce que la commande

```
>> surf (X,Y,Z)
```

corresponde à l'affichage en 3D du graphe de la fonction

$$(x, y) \mapsto \operatorname{sinc}(x^2 + y^2)$$

(où sinc représente le sinus cardinal, voir la question 1 de l'exercice 3) au dessus du maillage régulier de $[-2, 2]^2$ de pas .1 (suivant les deux axes). Quelle est la taille des trois variables `X,Y,Z` ? Réalisez cette même représentation graphique avec cette fois l'instruction `mesh` au lieu de `surf`.

(2) Ouvrez un nouveau fichier `.m` que vous sauverez dans votre répertoire de travail `TPMATLAB` comme `MaillagePolaire`. Rédigez sur ce fichier le code d'une fonction

```
function Z=MaillagePolaire(r1,r2,K)
```

qui retourne, étant donnés deux nombres flottants positifs `0<r1<r2` et un entier `K>0`, la matrice `Z` de taille `(K+1,K+1)` dont l'entrée (i, j) correspond au nombre complexe de module et argument respectivement :

$$r(i, j) = \texttt{r1} + \frac{i-1}{\texttt{K}}\,\texttt{(r2-r1)}, \quad \theta(i, j) = -\pi + 2\pi\,\frac{j-1}{\texttt{K}}$$

(vous utiliserez ici encore l'instruction `meshgrid` sur le modèle de ce que vous avez fait à la question 1).

Après vous être assuré que votre fichier `MaillagePolaire.m` avait bien été sauvé, faites l'instruction `clear all` pour libérer votre *Workspace*.

EXERCICE 5 (méthode de Newton dans le champ complexe).

(1) Ouvrez le fichier `Newton.m` et sauvez le (toujours comme fichier `.m`) sous le nom `Newton1`. Modifiez ce fichier de manière à ce que la boucle `DO` s'arrête automatiquement dès que `abs(x(k+1)-x(k)) <= eps` ou que le nombre d'itérations dépasse strictement un seuil `N` fixé, et que la fonction

`function [x,err,Nb] = Newton1(init,N)`

ainsi construite valide (lorsque `Nb <N` et `err=0`) la valeur approchée `x` de l'unique zéro réel de $X^5+X^3/2+1$ obtenue, l'erreur finalement commise [5] `err=abs(x(Nb+1)-x(Nb))` au terme de l'exécution du code, ainsi que le nombre d'itérations `Nb` (nécessairement inférieur ou égal à `N`) effectuées réellement lors de l'exécution de la boucle pour parvenir (si toutefois l'algorithme y parvient dans les limites du nombre maximal `N` d'itérations imparti) au premier cran tel que `abs(x(Nb+1)-x(Nb))<eps`. Testez votre nouveau code avec `init=-1` et le seuil garde fou `N=20`. Quelle valeur de `Nb` trouvez vous ?

(2) Testez votre nouvelle fonction `Newton1` en prenant cette fois `init=1+i` (`i=sqrt(-1)`) evec `N=100`, puis `init=-1+i` avec `N=100`. Qu'observez vous ? Déduisez en les valeurs approchées des cinq racines (complexes) du polynôme $X^5+X^3/2+1$.

(3) Modifiez si nécessaire votre fonction `Newton1` pour qu'elle puisse admettre comme variable d'entrée une variable `init` qui soit cette fois une matrice de nombres complexes et non plus un nombre complexe. Vérifiez (mathématiquement) que les cinq racines du polynôme $X^5+X^3/2+1$ sont dans la couronne du plan complexe $\{0.8 \leq |z| \leq 2\}$ et que le polynôme dérivé $5X^4+3X^2/2$ ne s'annule pas dans cette couronne. Lancez

```
>> [Z,err,Nb] = Newton1(MaillagePolaire(.8,2,100),100);
```

(la fonction `MaillagePolaire` a été construite à l'exercice 4). Retrouvez à partir de l'affichage (avec `imagesc(X)`, `imshow(X,[])` ou `mesh`(X), appliqué à une matrice `X` qui sera `abs(Z)` ou `angle(Z)`) la position (en polaire) des cinq racines complexes du polynôme $X^5+X^3/2+1$ (*cf.* la question 2 de l'exercice 3).

5. Ce devrait en principe être 0 pour la machine lorsque `Nb<N` ; si ce n'est pas le cas, ceci signifie que l'algorithme n'a pu aboutir au terme du nombre garde-fou `N` d'itérations imparti *a priori*.