Algorithmes numériques – Rapport

Polynôme caractéristique, valeurs propres et vecteurs propres

Axel Delsol, Pierre-Loup Pissavy

Janvier 2014

Table des matières

1 Préambule

1.1 Structure du programme

Nous avons conçu un programme principal avec menus, présenté sous la forme suivante :

```
Menu principal : Polynôme caractéristique, valeurs propres et
  vecteurs propres

Choisir le mode de saisie de la matrice :
1- Utiliser le générateur de matrices
0- Entrer manuellement les valeurs
(Saisie des valeurs de la matrice...)

(Affichage de la matrice correspondante...)
Quelle résolution utiliser ?
1- Méthode de Leverrier
2- Méthode de Leverrier améliorée
3- Méthode des Puissances Itérées
9- Nouvelle série de points (Menu principal)
0- Quitter
Votre choix :
```

FIGURE 1.1 – Apercu : Menu Principal

Au lancement, le programme demande la saisie des valeurs, qu'il stocke dans un tableau, puis affiche le menu. Après chaque méthode, il est possible de réutiliser le jeu de données (chaque méthode qui doit modifier les valeurs utilise un duplicata).

Le menu principal est codé dans le fichier source `main.c`. Les méthodes sont codées dans des fichiers individuels sauf les méthodes de Leverrier qui sont réunies. Les prototypes des fonctions sont écrits dans les headers correspondants. La liste de tous ces fichiers est présentée figure 1.3.

Le stockage des valeurs se fait en double précision (type `double`, 64 bits) afin d'obtenir des résultats suffisamment précis.

De plus, les méthodes de Leverrier utilisent une structure de polynôme (composée du degré et des coefficients), présentée figure 1.2, pour faciliter la compréhension du code.

```
4 typedef struct polynome
5 {
6   int d; //degree
7   double* poln; //coefficients
8 } polynome;
```

FIGURE 1.2 – Code : Structure de Polynôme

```
leverrier.c
main.c
matrices.c
polynome.c
puissancesIt.c
useful.c
generateur.h
leverrier.h
matrices.h
polynome.h
puissancesIt.h
useful.h
makefile
```

FIGURE 1.3 – Apercu : Arborescence des fichiers `C` et `makefile`

1.2 Compilation et logiciels utilisés

La compilation est gérée par un `makefile`.
Le compilateur utilisé est `GCC`. Il suffit de taper `make` pour lancer la compilation, puis `./main` pour lancer le programme.
Pour nettoyer les fichiers temporaires, il faudra taper `make clean`.
Ce `makefile` permet également de générer ce rapport ainsi que quelques fichiers qui y sont intégrés.

Afin de vérifier les résultats, nous avons utilisé le logiciel de calcul mathématique Maple.

1.3 Jeux de tests

Nous avons choisi d'effectuer les tests des méthodes de Leverrier et des puissances itérées sur les mêmes matrices. Les matrices utilisées sont celles du TP sur les méthodes de résolution de systèmes d'équations, de dimension 3 à 15. Ceci nous permet de calculer l'image de la valeur propre (donnée par la puissance itérée) par le polynôme caractéristique obtenu avec les méthodes de Leverrier. On peut ainsi contrôler les résultats puisque normalement, l'image d'une valeur propre par le polynôme caractéristique de la matrice associée est égale à 0.

Note : Les résultats sont tronqués à 10^{-6} pour des raisons de lisibilité mais les calculs ont été effectués avec la précision de la machine.

$$\text{Creuse à 70\%} \begin{pmatrix} 6.000000 & 5.000000 & 5.000000 & 2.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 9.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 6.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \end{pmatrix} \tag{1}$$

$$\text{A bord} \begin{pmatrix} 1.000000 & 0.500000 & 0.250000 & 0.125000 & 0.062500 & 0.031250 & 0.015625 & 0.007812 \\ 0.500000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.250000 & 0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.125000 & 0.000000 & 0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.062500 & 0.000000 & 0.000000 & 0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.031250 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 & 0.000000 & 0.000000 \\ 0.015625 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 0.007812 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{pmatrix} \tag{2}$$

$$\text{Ding Dong} \begin{pmatrix} 0.200000 & 0.333333 & 1.000000 \\ 0.333333 & 1.000000 & -1.000000 \\ 1.000000 & -1.000000 & -0.333333 \end{pmatrix} \tag{3}$$

$$\text{Franc} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 0 & 0 & 3 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 0 & 0 & 0 & 4 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 0 & 0 & 0 & 0 & 5 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 0 & 0 & 0 & 0 & 0 & 6 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 8 & 8 & 8 & 8 & 8 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 9 & 9 & 9 & 9 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 10 & 10 & 10 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 11 & 11 & 11 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 & 12 & 12 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12 & 13 \end{pmatrix} \tag{4}$$

$$\text{Hilbert}\begin{pmatrix}
1.000000 & 0.500000 & 0.333333 & 0.250000 & 0.200000 & 0.166667 & 0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 \\
0.500000 & 0.333333 & 0.250000 & 0.200000 & 0.166667 & 0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 \\
0.333333 & 0.250000 & 0.200000 & 0.166667 & 0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 \\
0.250000 & 0.200000 & 0.166667 & 0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 \\
0.200000 & 0.166667 & 0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 \\
0.166667 & 0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 \\
0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 & 0.047619 \\
0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 & 0.047619 & 0.045455 \\
0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 & 0.047619 & 0.045455 & 0.043478 \\
0.100000 & 0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 & 0.047619 & 0.045455 & 0.043478 & 0.041667 \\
0.090909 & 0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 & 0.047619 & 0.045455 & 0.043478 & 0.041667 & 0.040000 \\
0.083333 & 0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 & 0.047619 & 0.045455 & 0.043478 & 0.041667 & 0.040000 & 0.038462 \\
0.076923 & 0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 & 0.047619 & 0.045455 & 0.043478 & 0.041667 & 0.040000 & 0.038462 & 0.037037 \\
0.071429 & 0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 & 0.047619 & 0.045455 & 0.043478 & 0.041667 & 0.040000 & 0.038462 & 0.037037 & 0.035714 \\
0.066667 & 0.062500 & 0.058824 & 0.055556 & 0.052632 & 0.050000 & 0.047619 & 0.045455 & 0.043478 & 0.041667 & 0.040000 & 0.038462 & 0.037037 & 0.035714 & 0.034483
\end{pmatrix} \tag{5}$$

$$\text{KMS avec } p = 0.25 \begin{pmatrix}
1.000000 & 0.250000 & 0.062500 & 0.015625 \\
0.250000 & 1.000000 & 0.250000 & 0.062500 \\
0.062500 & 0.250000 & 1.000000 & 0.250000 \\
0.015625 & 0.062500 & 0.250000 & 1.000000
\end{pmatrix} \tag{6}$$

$$\text{Lehmer} \begin{pmatrix}
1.000000 & 0.500000 & 0.333333 & 0.250000 & 0.200000 & 0.166667 \\
0.500000 & 1.000000 & 0.666667 & 0.500000 & 0.400000 & 0.333333 \\
0.333333 & 0.666667 & 1.000000 & 0.750000 & 0.600000 & 0.500000 \\
0.250000 & 0.500000 & 0.750000 & 1.000000 & 0.800000 & 0.666667 \\
0.200000 & 0.400000 & 0.600000 & 0.800000 & 1.000000 & 0.833333 \\
0.166667 & 0.333333 & 0.500000 & 0.666667 & 0.833333 & 1.000000
\end{pmatrix} \tag{7}$$

$$\text{Lotkin} \begin{pmatrix}
1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 \\
0.500000 & 0.333333 & 0.250000 & 0.200000 & 0.166667 & 0.142857 & 0.125000 \\
0.333333 & 0.250000 & 0.200000 & 0.166667 & 0.142857 & 0.125000 & 0.111111 \\
0.250000 & 0.200000 & 0.166667 & 0.142857 & 0.125000 & 0.111111 & 0.100000 \\
0.200000 & 0.166667 & 0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 \\
0.166667 & 0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 \\
0.142857 & 0.125000 & 0.111111 & 0.100000 & 0.090909 & 0.083333 & 0.076923
\end{pmatrix} \tag{8}$$

$$\text{Moler} \begin{pmatrix}
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 3 & 1 & 1 & 1 & 1 & 1 & 1 \\
-1 & 0 & 1 & 4 & 2 & 2 & 2 & 2 & 2 \\
-1 & 0 & 1 & 2 & 5 & 3 & 3 & 3 & 3 \\
-1 & 0 & 1 & 2 & 3 & 6 & 4 & 4 & 4 \\
-1 & 0 & 1 & 2 & 3 & 4 & 7 & 5 & 5 \\
-1 & 0 & 1 & 2 & 3 & 4 & 5 & 8 & 6 \\
-1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 9
\end{pmatrix} \tag{9}$$

2 Polynôme caractéristique

2.1 Méthode de Leverrier

2.1.1 Présentation

La méthode de Leverrier permet de déterminer le polynôme caractéristique d'une matrice $A \in M_{n,n}(\Re)$ c'est-à-dire le déterminant $|A - \lambda I_n| = P(\lambda)$ où I_n est la matrice identité. Le but est donc de déterminer les coefficients a_i du polynôme $P(\lambda) = a_n + a_{n-1}\lambda + \cdots + a_0\lambda^n$.

Pour les trouver, on définit d'abord $S_p = \mathrm{Tr}(A^p)$ puis on applique les identités de Newton à l'aide de la formule suivante :

$$\forall p \in \{1, \cdots, n\}, \qquad \begin{cases} a_0 = (-1)^n \\ a_p = -\dfrac{\sum_{i=0}^{p-1} a_i \cdot S_{p-1}}{p} \end{cases}$$

2.1.2 Programme

```
9  void leverrier(double** A, int n)
10 {
11   int i, j;
12   double tempo;
13   double* coeffs=(double*) malloc ((n+1)*sizeof(double)); //tableau coeffs
14   double* traces=(double*) malloc(n*sizeof(double)); //tableau traces
15   double** tmp;
16   polynome* p;
17
18   //On remplit notre tableau contenant les traces
19   for (i=1; i<=n; i++)
20   {
21     tmp=puissanceMatrice(A, n, i);
22     traces[i-1]=trace(tmp, n);
23     for (j=0; j<n; j++)
24     {
25       free(tmp[j]);
26     }
27     free(tmp);
28   }
29
30   //On remplit le tableau des coefficients
31   for(i=0; i<=n; i++)
32   {
33     coeffs[i]=0;
34   }
35   coeffs[n]=pow(-1.0,n);
36   for(i=1; i<=n; i++)
37   {
38     for(j=0;j<i;j++)
39     {
40       coeffs[n-i] = coeffs[n-i] - coeffs[n-j]*traces[i-j-1];
41     }
42     coeffs[n-i] = coeffs[n-i]/i;
43   }
```

```
44    p=creerPoly(n+1, "tableau", coeffs);
45    menuAffichage(p);
46
47    //liberation memoire
48    //libération du tableau de trace
49    free(traces);
50    //libération du polynome
51    free(p->poln);
52    free(p);
53  }
```

FIGURE 2.1 – Code : leverrier.c

2.2 Méthode de Leverrier améliorée

2.2.1 Présentation

La méthode de Leverrier améliorée consiste à appliquer un algorithme itératif calculant deux matrices A et B pour obtenir l'un après l'autre les coefficients a_1 à a_n.
On a $a_0 = (-1)^n$, et on calcule les autres coefficients a_k de la manière suivante :
Pour $k = 1 \cdots n$, $\quad A_k = B_{k-1}A; \quad a_k = \frac{1}{k}\mathrm{Tr}(A_k); \quad B_k = A_k - a_k I \quad$ avec $B_0 = I$.

2.2.2 Programme

```
55  void leverrierA(double** A, int n)
56  {
57    int i, j;
58    double tempo;
59    double** tmp;
60    double* coeffs=(double*) malloc ((n+1)*sizeof(double)); //tableau coeffs
61    double** Ak=(double**) malloc (n*sizeof(double*));
62    double** B=(double**) malloc (n*sizeof(double*));
63    double** I=(double**) malloc (n*sizeof(double*));
64    polynome* p;
65
66    for (i=0; i<n; i++)
67    {
68      Ak[i]=(double*) malloc (n*sizeof(double));
69      I[i]=(double*) malloc (n*sizeof(double));
70      B[i]=(double*) malloc (n*sizeof(double));
71    }
72
73    //Création matrice identité, initialisation de B et copie de A
74    for (i=0; i<n; i++)
75    {
76      for (j=0; j<n; j++)
77      {
78        Ak[i][j]=A[i][j];
79        if (i==j)
80        {
81          I[i][j]=1;
82          B[i][j]=1;
83        }
84        else
85        {
86          I[i][j]=0;
87          B[i][j]=0;
88        }
89      }
90    }
91
92    //On remplit le tableau des coefficients
93    for(i=0; i<=n; i++)
94    {
95      coeffs[i]=0;
96    }
97
```

```
98    coeffs[0]=pow(-1,n);
99
100
101   for(i=1; i<=n; i++)
102   {
103     Ak=produitMatriciel(B,A,n,n,n);
104     coeffs[i]=trace(Ak,n)/i;
105     tmp=produitSMatriciel(I,n,n,coeffs[i]);
106     for(j=0;j<n;j++)
107     {
108       free(B[j]);
109     }
110     free(B);
111     B=difference(Ak,tmp,n,n);
112     for(j=0;j<n;j++)
113     {
114       free(Ak[j]);
115       free(tmp[j]);
116     }
117     free(tmp); free(Ak);
118   }
119
120   //On inverse le tableau des coefficients
121   for(i=0;i<=((n+1)/2)-1;i++)
122   {
123     if ((n%2) == 0)
124     {
125       coeffs[i]=-coeffs[i];
126     }
127     tempo = coeffs[i];
128     coeffs[i] = coeffs[n-i];
129     coeffs[n-i] = tempo;
130   }
131
132   p=creerPoly(n+1, "tableau", coeffs);
133   menuAffichage(p);
134
135   //liberation memoire
136   for (i=0; i<n; i++)
137   {
138     free(B[i]);
139   }
140   free(B);
141   //libération du polynome
142   free(p->poln);
143   free(p);
144 }
```

FIGURE 2.2 – Code : leverrier.c

2.3 Résultats des tests

Système analysé	Polynôme caractéristique obtenu
(1)	**Polynôme Leverrier :** $P(x) \approx +6.000000 \cdot x^4 - 1.000000 \cdot x^5$ **Polynôme Leverrier amélioré :** $P(x) \approx +6.000000 \cdot x^4 - 1.000000 \cdot x^5$
(2)	**Polynôme Leverrier :** $P(x) \approx 0.666687 - 6.000122 \cdot x + 23.000305 \cdot x^2 - 49.333740 \cdot x^3$ $+65.000305 \cdot x^4 - 54.000122 \cdot x^5 + 27.666687 \cdot x^6 - 8.000000 \cdot x^7 + 1.000000 \cdot x^8$ **Polynôme Leverrier amélioré :** $P(x) \approx -0.666687 + 6.000122 \cdot x - 23.000305 \cdot x^2$ $+49.333740 \cdot x^3 - 65.000305 \cdot x^4 - 54.000122 \cdot x^5 + 27.666687 \cdot x^6 - 8.000000 \cdot x^7$ $-1.000000 \cdot x^8$
(3)	**Polynôme Leverrier :** $P(x) \approx -1.896296 + 2.311111 \cdot x + 0.866667 \cdot x^2 - 1.000000 \cdot x^3$ **Polynôme Leverrier amélioré :** $P(x) \approx -1.896296 + 2.311111 \cdot x + 0.866667 \cdot x^2$ $-1.000000 \cdot x^3$
(4)	**Polynôme Leverrier :** $P(x) \approx -1884.003497 - 58.681818 \cdot x + 3081.727273 \cdot x^2$ $-49621.000000 \cdot x^3 + 406120.000000 \cdot x^4 - 1693692.000000 \cdot x^5 + 3506217.000000 \cdot x^6$ $-3506217.000000 \cdot x^7 + 1693692.000000 \cdot x^8 - 406120.000000 \cdot x^9 + 49621.000000 \cdot x^{10}$ $-3081.000000 \cdot x^{11} + 91.000000 \cdot x^{12} - 1.000000 \cdot x^{13}$ **Polynôme Leverrier amélioré :** $P(x) \approx 1.000000 - 91.000000 \cdot x + 3081.000000 \cdot x^2$ $-49621.000000 \cdot x^3 + 406120.000000 \cdot x^4 - 1693692.000000 \cdot x^5 + 3506217.000000 \cdot x^6$ $-3506217.000000 \cdot x^7 + 1693692.000000 \cdot x^8 - 406120.000000 \cdot x^9 + 49621.000000 \cdot x^{10}$ $-3081.000000 \cdot x^{11} + 91.000000 \cdot x^{12} - 1.000000 \cdot x^{13}$
(5)	**Polynôme Leverrier :** $P(x) \approx -0.000000 - 0.000000 \cdot x - 0.000000 \cdot x^2$ $-0.000000 \cdot x^3 - 0.000000 \cdot x^4 + 0.000000 \cdot x^5 + 0.000000 \cdot x^6 - 0.000000 \cdot x^7$ $+0.000000 \cdot x^8 - 0.000000 \cdot x^9 + 0.000000 \cdot x^{10} - 0.000277 \cdot x^{11} + 0.050664 \cdot x^{12}$ $-0.931768 \cdot x^{13} + 2.335873 \cdot x^{14} - 1.000000 \cdot x^{15}$ **Polynôme Leverrier amélioré :** $P(x) \approx 0.000000 + 0.000000 \cdot x + 0.000000 \cdot x^2$ $+0.000000 \cdot x^3 + 0.000000 \cdot x^4 + 0.000000 \cdot x^5 + 0.000000 \cdot x^6 + 0.000000 \cdot x^7$ $+0.000000 \cdot x^8 - 0.000000 \cdot x^9 + 0.000000 \cdot x^{10} - 0.000277 \cdot x^{11} + 0.050664 \cdot x^{12}$ $-0.931768 \cdot x^{13} + 2.335873 \cdot x^{14} - 1.000000 \cdot x^{15}$

Système analysé	Polynôme caractéristique obtenu
(6)	**Polynôme Leverrier :** $P(x) \approx 0.823975 - 3.625488 \cdot x + 5.804443 \cdot x^2$ $-4.000000 \cdot x^3 + 1.000000 \cdot x^4$ **Polynôme Leverrier amélioré :** $P(x) \approx -0.823975 + 3.625488 \cdot x - 5.804443 \cdot x^2$ $-4.000000 \cdot x^3 - 1.000000 \cdot x^4$
(7)	**Polynôme Leverrier :** $P(x) \approx 0.020052 - 0.379769 \cdot x + 2.563758 \cdot x^2$ $-7.716667 \cdot x^3 + 10.591667 \cdot x^4 - 6.000000 \cdot x^5 + 1.000000 \cdot x^6$ **Polynôme Leverrier amélioré :** $P(x) \approx -0.020052 + 0.379769 \cdot x - 2.563758 \cdot x^2$ $+7.716667 \cdot x^3 + 10.591667 \cdot x^4 - 6.000000 \cdot x^5 - 1.000000 \cdot x^6$
(8)	**Polynôme Leverrier :** $P(x) \approx 0.000000 + 0.000000 \cdot x + 0.000000 \cdot x^2 + 0.000032$ $\cdot x^3 + 0.022474 \cdot x^4 + 0.586254 \cdot x^5 + 1.955134 \cdot x^6 - 1.000000 \cdot x^7$ **Polynôme Leverrier amélioré :** $P(x) \approx 0.000000 + 0.000000 \cdot x + 0.000000 \cdot x^2$ $+0.000032 \cdot x^3 + 0.022474 \cdot x^4 + 0.586254 \cdot x^5 + 1.955134 \cdot x^6 - 1.000000 \cdot x^7$
(9)	**Polynôme Leverrier :** $P(x) \approx 1.000000 - 29133.000000 \cdot x + 77688.000000 \cdot x^2$ $-88575.000000 \cdot x^3 + 56073.000000 \cdot x^4 - 21375.000000 \cdot x^5 + 4956.000000 \cdot x^6$ $-666.000000 \cdot x^7 + 45.000000 \cdot x^8 - 1.000000 \cdot x^9$ **Polynôme Leverrier amélioré :** $P(x) \approx 1.000000 - 29133.000000 \cdot x + 77688.000000 \cdot x^2$ $-88575.000000 \cdot x^3 + 56073.000000 \cdot x^4 - 21375.000000 \cdot x^5 + 4956.000000 \cdot x^6$ $-666.000000 \cdot x^7 + 45.000000 \cdot x^8 - 1.000000 \cdot x^9$

2.4 Comparaisons

On peut remarquer des différences entre les deux méthodes.

Tout d'abord, les polynômes sont différents dans le cas des puissances paires. La méthode de Leverrier nous donne les polynômes caractéristiques mathématiques exacts (c'est à dire que le coefficient du plus grand monome est $(-1)^n$) alors que la méthode de Leverrier améliorée nous donne l'opposé du polynôme précédent. Cependant, pour l'objectif du TP, ceci n'est pas important car il n'affecte pas les valeurs propres de la matrice.

De plus, les complexités des algorithmes sont différentes. En effet, la méthode de Leverrier est en $O(n^5)$ (donc coûteuse) alors que la version améliorée est en $O(n^3)$. Cette différence se ressent au niveau du système (4) où les premiers termes des polynômes sont différents. Cet écart est dû aux imprécisions de calcul de la machine entraînant un résultat erroné pour la méthode de Leverrier.

Enfin, on peut noter qu'il est possible de faire partiellement la méthode de Leverrier en parallèle lors du calcul des puissances et traces des matrices alors que la méthode de Leverrier améliorée est un processus itératif où on a toujours besoin de connaître les valeurs précédentes pour continuer.

3 Valeurs propres et vecteurs propres

3.1 Méthode des puissances itérées

3.1.1 Présentation

La méthode des puissances itérées est un algorithme itératif permettant de trouver la plus grande valeur propre d'une matrice et un vecteur propre.

1. On commence par choisir un vecteur initial x_0 non nul (nous avions initialement choisi de prendre le vecteur unité pour tous nos tests, mais nous avons finalement laissé le choix à l'utilisateur),
2. On calcule x_{i+1} à l'aide de la relation de récurrence $x_{i+1} = A \cdot x_i$,
3. On arrête l'algorithme quand les vecteurs x_i et x_{i+1} sont proches (nous avons choisi de les comparer à l'aide de leur norme euclidienne).

Ceci est possible uniquement si les valeurs propres sont toutes différentes. En effet, la méthode s'appuie sur le fait que la limite du quotient d'une valeur propre sur la plus grande tend vers 0. On peut alors émettre l'hypothèse que si les valeurs propres sont très proches alors le nombre d'itérations pour obtenir un résultat est important. Cette hypothèse sera testée en dernière partie de test.

3.1.2 Programme

```
9
10 void puissancesIt(double** A, int n, double precision)
11 {
12   int i, cpt=0;
13   double ecart = 1.;
14   double** x1 = (double**) malloc (n*sizeof(double*));
15   double** x2;
16   double** x3;
17   double** xt;
18   double** xtx;
19   //Initialisation de x1
20   printf("Vecteur initial :\n");
21   x1 = creerRemplirMatrice(n,1);
22   cleanBuffer();
23 //   version initiale : vecteur unité
24 //   for (i=0; i<n; i++)
25 //   {
26 //     x1[i] = (double*) malloc (sizeof(double));
27 //     x1[i][0] = 1.;
28 //   }
29   while (ecart > precision)
30   {
31     cpt++;
32     x2 = produitMatriciel(A,x1,n,n,1); //x2 = A x1
33     diviserComposantes(x2,n);
34     x3 = difference(x2, x1, n, 1); //x3 = x2-x1
35     ecart = norme(x3, n); //norme(x2-x1)-> écart
36     //libération mémoire
37     for (i=0; i<n; i++)
38     {
39       free(x3[i]);
```

```
40        free(x1[i]);
41      }
42      free(x3);
43      free(x1);
44      x1 = x2; //xi = x(i+1)
45    }
46    printf("Nombre d'itérations : %d\n", cpt);
47    printf("Approximation du vecteur propre :\n");
48    for (i=0; i<n; i++)
49    {
50      printf("x[%d]=%f\n", (i+1), x1[i][0]);
51    }
52    xt = transpose(x1, n, 1);
53    x2 = produitMatriciel(xt, A, 1, n, n); //xt A
54    x3 = produitMatriciel(x2, x1, 1, n, 1); //xt A x
55    xtx = produitMatriciel(xt, x1, 1, n, 1); //xt x
56    printf("Approximation de la valeur propre maximum : %f\n", x3[0][0]/xtx[0][0]);
57    //libération mémoire
58    for (i=0; i<n; i++)
59    {
60      free(x1[i]);
61    }
62    free(xt[0]); free(x2[0]); free(x3[0]); free(xtx[0]);
63    free(xt); free(x2); free(x3); free(xtx); free(x1);
64  }
```

FIGURE 3.1 – Code : puissancesIt.c

3.2 Résultats des tests

Remarque : Les tests ont été effectués deux fois : une en précision 10^{-5} et une en 10^{-9} afin d'observer l'impact de la précision sur les résultats. Afin de pouvoir comparer les résultats, nous avons utilisé le vecteur unité comme vecteur initial.

3.2.1 Résultats avec une précision 10^{-5}

Système analysé	Itérations	Vecteur	Valeur	Image par Leverrier	Image par Leverrier amélioré
(1)	3	$\begin{pmatrix} 1.000000 \\ 0.000000 \\ 0.000000 \\ 0.000000 \\ 0.000000 \end{pmatrix}$	6.000000	0.000000	0.000000
(2)	43	$\begin{pmatrix} 73.898567 \\ 63.999989 \\ 31.999995 \\ 15.999997 \\ 7.999999 \\ 3.999999 \\ 2.000000 \\ 1.000000 \end{pmatrix}$	1.577333	0.000000	-590.987420
(3)	365	$\begin{pmatrix} 0.353044 \\ -1.549648 \\ 1.000000 \end{pmatrix}$	1.569365	0.000000	0.000000

Système analysé	Itérations	Vecteur	Valeur	Image par Leverrier	Image par Leverrier amélioré
(4)	33	$\begin{pmatrix} 1.557371 \\ 3.071013 \\ 4.454695 \\ 5.584562 \\ 6.354832 \\ 6.694680 \\ 6.581598 \\ 6.048019 \\ 5.178865 \\ 4.099349 \\ 2.954549 \\ 1.884488 \\ 1.000000 \end{pmatrix}$	35.613867	-3143394623488.0	-3143394623488.0
(5)	11	$\begin{pmatrix} 7.243207 \\ 4.511415 \\ 3.408464 \\ 2.779831 \\ 2.364104 \\ 2.065033 \\ 1.837827 \\ 1.658468 \\ 1.512781 \\ 1.391800 \\ 1.289546 \\ 1.201862 \\ 1.125759 \\ 1.059028 \\ 1.000000 \end{pmatrix}$	1.845928	0.000000	0.000000

Système analysé	Itérations	Vecteur	Valeur	Image par Leverrier	Image par Leverrier amélioré
(6)	19	$\begin{pmatrix} 1.000000 \\ 1.442995 \\ 1.442995 \\ 1.000000 \end{pmatrix}$	1.466563	-0.000000	-25.234334
(7)	11	$\begin{pmatrix} 0.567822 \\ 0.899130 \\ 1.074934 \\ 1.131370 \\ 1.098183 \\ 1.000000 \end{pmatrix}$	3.601212	-0.000000	-3705.380262
(8)	8	$\begin{pmatrix} 8.229502 \\ 2.893044 \\ 2.067110 \\ 1.622324 \\ 1.340532 \\ 1.144649 \\ 1.000000 \end{pmatrix}$	2.223362	-0.000002	-0.000002
(9)	8	$\begin{pmatrix} -0.199475 \\ 0.008623 \\ 0.216349 \\ 0.414722 \\ 0.595166 \\ 0.749881 \\ 0.872178 \\ 0.956770 \\ 1.000000 \end{pmatrix}$	25.131788	0.283203	0.283203

3.2.2 Résultats avec une précision 10^{-9}

Système analysé	Itérations	Vecteur	Valeur	Image par Leverrier	Image par Leverrier amélioré
(1)	3	$\begin{pmatrix} 1.000000 \\ 0.000000 \\ 0.000000 \\ 0.000000 \\ 0.000000 \end{pmatrix}$	6.000000	0.000000	0.000000
(2)	64	$\begin{pmatrix} 73.898579 \\ 64.000000 \\ 32.000000 \\ 16.000000 \\ 8.000000 \\ 4.000000 \\ 2.000000 \\ 1.000000 \end{pmatrix}$	1.577333	-0.000000	-590.987420
(3)	587	$\begin{pmatrix} 0.353046 \\ -1.549652 \\ 1.000000 \end{pmatrix}$	1.569365	-0.000000	-0.000000
(4)	54	$\begin{pmatrix} 1.557373 \\ 3.071016 \\ 4.454700 \\ 5.584567 \\ 6.354838 \\ 6.694686 \\ 6.581603 \\ 6.048024 \\ 5.178869 \\ 4.099351 \\ 2.954550 \\ 1.884488 \\ 1.000000 \end{pmatrix}$	35.613861	-334626816.0	-334626816.0

Système analysé	Itérations	Vecteur	Valeur	Image par Leverrier	Image par Leverrier amélioré
(5)	17	$\begin{pmatrix} 7.243208 \\ 4.511416 \\ 3.408464 \\ 2.779831 \\ 2.364104 \\ 2.065033 \\ 1.837827 \\ 1.658468 \\ 1.512781 \\ 1.391800 \\ 1.289546 \\ 1.201862 \\ 1.125759 \\ 1.059028 \\ 1.000000 \end{pmatrix}$	1.845928	0.000000	0.000000
(6)	34	$\begin{pmatrix} 1.000000 \\ 1.443000 \\ 1.443000 \\ 1.000000 \end{pmatrix}$	1.466563	-0.000000	-25.234334
(7)	19	$\begin{pmatrix} 0.567820 \\ 0.899129 \\ 1.074933 \\ 1.131370 \\ 1.098182 \\ 1.000000 \end{pmatrix}$	3.601212	0.000000	-3705.380262

Système analysé	Itérations	Vecteur	Valeur	Image par Leverrier	Image par Leverrier amélioré
(8)	12	$\begin{pmatrix} 8.229502 \\ 2.893044 \\ 2.067110 \\ 1.622324 \\ 1.340532 \\ 1.144649 \\ 1.000000 \end{pmatrix}$	2.223362	-0.000000	-0.000000
(9)	14	$\begin{pmatrix} -0.199475 \\ 0.008623 \\ 0.216349 \\ 0.414722 \\ 0.595166 \\ 0.749880 \\ 0.872177 \\ 0.956769 \\ 1.000000 \end{pmatrix}$	25.131788	0.000488	0.000488

3.2.3 Test de l'hypothèse du lien entre les itérations et les valeurs propres

Dans ces exemples, les valeurs propres de la matrice sont 1.000001 et 1.000000 et le polynôme caractéristique est $P(x) = 1.000001 - 2.000001 \cdot x + 1.000000 \cdot x^2$.

Les systèmes analysés sont les suivants :

$$\begin{pmatrix} 2.000001 & -1.000001 \\ 1.000000 & 0.000000 \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} 1.000001 & 1.000000 \\ 0.000000 & 1.000000 \end{pmatrix} \tag{2}$$

Système analysé	Itérations	Vecteur	Valeur
(1)	1	$\begin{pmatrix} 1.000000 \\ 1.000000 \end{pmatrix}$	1.000000
(2)	Non terminé	Non déterminé	Non déterminé

3.3 Comparaisons

La méthode des puissances donne globalement des résultats corrects et précis.

On peut noter que le nombre d'itérations est assez peu prévisible. On peut seulement dire que le nombre d'itérations :

- ne dépend pas de la taille de la matrice puisque le système (3) a été résolu en 365 itérations alors que le système (5) a été résolu en 11,
- ne dépend pas des valeurs propres. Dans la présentation, nous avons supposé que plus les valeurs propres étaient proches de la plus grande, plus le nombre d'itérations était grand. Or le système (1) du dernier jeu de test montre que ce n'est pas toujours le cas puisque les valeurs propres diffèrent de 0.000001 et l'algorithme s'effectue en 1 itération.
- dépend de la précision voulue. En effet, on peut noter que le nombre d'itérations entre les précisions 10^{-5} et 10^{-9} a fortement augmenté.

Cependant, tous les tests n'ont pas donné des résultats fiables.

Le système (2) nous donne une image de la valeur propre erronée alors que le polynôme caractéristique de la matrice est juste (il a été testé avec un logiciel mathématique).
Ensuite, la valeur propre du système (4) n'est pas une valeur propre de la matrice (aussi vérifié à l'aide d'un logiciel mathématique) car les polynômes caractéristiques sont corrects.
Enfin, les systèmes (1) et (2) du dernier jeu de test montrent deux choses.

1. L'algorithme ne donne pas les mêmes résultats alors que les racines des polynômes sont les mêmes (puisque ce sont les mêmes polynômes)
2. Les résultats ne sont pas obtenus comme le montre le système (2) où ils sont faux pour le système (1) car la plus grande valeur propre n'est pas 1.000000 mais 1.000001.

4 Conclusion

Nous pourrons noter que ces algorithmes offrent des résultats intéressants, mais que le conditionnement des systèmes reste indispensable pour obtenir des valeurs correctes et convaincantes.

En effet, nous avons pu remarquer en choisissant d'autres vecteurs initiaux pour la méthode des Puissances Itérées que les résultats variaient. Ainsi par exemple, pour la matrice A suivante, on obtenait un calcul impossible (Not a Number) avec le vecteur unité, mais un résultat correct avec le vecteur b.

$$A = \begin{pmatrix} -0.666667 & 0.000000 & 1.333333 \\ 0.277778 & 2.000000 & 0.111111 \\ 0.166667 & 0.000000 & -0.333333 \end{pmatrix} \qquad b = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \text{Valeur propre maximum : 2.}$$

Ce phénomène peut s'expliquer de la manière suivante : si la matrice est symétrique, la méthode est convergente avec le vecteur unité. N'ayant pas systématiquement des matrices symétriques, cela explique certaines erreurs dans nos résultats. C'est la raison pour laquelle nous avons finalement choisi de laisser le choix du vecteur initial à l'utilisateur car la méthode peut converger avec d'autres vecteurs.

Ainsi, on obtiendra des valeurs correctes pour le système (2) du dernier jeu de tests avec pour vecteur initial $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

On peut finalement remarquer que ces algorithmes restent simples et peuvent être améliorés pour devenir plus génériques et fonctionner pour un plus large éventail de systèmes.