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
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.

```c
typedef struct polynome
{
  int d; //degree
  double* poln; //coefficients
} polynome;
```

Figure 1.2 – Code : Structure de Polynôme

```
generateur.c
leverrier.c
main.c
matrices.c
pile.c
polynome.c
puissancesIt.c
useful.c
file.h
generateur.h
leverrier.h
matrices.h
pile.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.

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 du polynôme caractéristique obtenu dans 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.

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

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

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

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

$$\text{Franc}\begin{pmatrix}
1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 \\
1.000000 & 2.000000 & 2.000000 & 2.000000 & 2.000000 & 2.000000 & 2.000000 & 2.000000 & 2.000000 & 2.000000 & 2.000000 & 2.000000 & 2.000000 \\
0.000000 & 2.000000 & 3.000000 & 3.000000 & 3.000000 & 3.000000 & 3.000000 & 3.000000 & 3.000000 & 3.000000 & 3.000000 & 3.000000 & 3.000000 \\
0.000000 & 0.000000 & 3.000000 & 4.000000 & 4.000000 & 4.000000 & 4.000000 & 4.000000 & 4.000000 & 4.000000 & 4.000000 & 4.000000 & 4.000000 \\
0.000000 & 0.000000 & 0.000000 & 4.000000 & 5.000000 & 5.000000 & 5.000000 & 5.000000 & 5.000000 & 5.000000 & 5.000000 & 5.000000 & 5.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 5.000000 & 6.000000 & 6.000000 & 6.000000 & 6.000000 & 6.000000 & 6.000000 & 6.000000 & 6.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 6.000000 & 7.000000 & 7.000000 & 7.000000 & 7.000000 & 7.000000 & 7.000000 & 7.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 7.000000 & 8.000000 & 8.000000 & 8.000000 & 8.000000 & 8.000000 & 8.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 8.000000 & 9.000000 & 9.000000 & 9.000000 & 9.000000 & 9.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 9.000000 & 10.000000 & 10.000000 & 10.000000 & 10.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 10.000000 & 11.000000 & 11.000000 & 11.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 11.000000 & 12.000000 & 12.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 12.000000 & 13.000000
\end{pmatrix} \tag{4}$$

$$\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}$$

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 polynome $P(\lambda) = a_n + a_{n-1}\lambda + \cdots + a_0\lambda^n$.

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

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

2.1.2 Programme

```c
void leverrier(double** A, int n)
{
  int i, j;
  double tempo;
  double* coeffs=(double*) malloc ((n+1)*sizeof(double)); //tableau coeffs
  double* traces=(double*) malloc(n*sizeof(double)); //tableau traces
  double** tmp;
  polynome* p;

  //On remplit notre tableau contenant les traces
  for (i=1; i<=n; i++)
  {
    tmp=puissanceMatrice(A, n, i);
    traces[i-1]=trace(tmp, n);
    for (j=0; j<n; j++)
    {
      free(tmp[j]);
    }
    free(tmp);
  }

  //On remplit le tableau des coefficients
  for(i=0; i<=n; i++)
  {
    coeffs[i]=0;
  }
  coeffs[n]=pow(-1.0,n);
  for(i=1; i<=n; i++)
  {
    for(j=0;j<i;j++)
    {
      coeffs[n-i] = coeffs[n-i] - coeffs[n-j]*traces[i-j-1];
    }
    coeffs[n-i] = coeffs[n-i]/i;
  }
```

```
  p=creerPoly(n+1, "tableau", coeffs);
  menuAffichage(p);

  //liberation memoire
  //libération du tableau de trace
  free(traces);
  //libération du polynome
  free(p->poln);
  free(p);
}
```

FIGURE 2.1 – Code : leverrier.c

2.2 Méthode de Leverrier améliorée

2.2.1 Présentation

2.2.2 Programme

```
void leverrierA(double** A, int n)
{
  int i, j;
  double tempo;
  double** tmp;
  double* coeffs=(double*) malloc ((n+1)*sizeof(double)); //tableau coeffs
  double** Ak=(double**) malloc (n*sizeof(double*));
  double** B=(double**) malloc (n*sizeof(double*));
  double** I=(double**) malloc (n*sizeof(double*));
  polynome* p;

  for (i=0; i<n; i++)
  {
    Ak[i]=(double*) malloc (n*sizeof(double));
    I[i]=(double*) malloc (n*sizeof(double));
    B[i]=(double*) malloc (n*sizeof(double));
  }

  //Création matrice identité, initialisation de B et copie de A
  for (i=0; i<n; i++)
  {
    for (j=0; j<n; j++)
    {
      Ak[i][j]=A[i][j];
      if (i==j)
      {
        I[i][j]=1;
        B[i][j]=1;
      }
      else
      {
        I[i][j]=0;
        B[i][j]=0;
      }
    }
  }

  //On remplit le tableau des coefficients
  for(i=0; i<=n; i++)
  {
    coeffs[i]=0;
  }

  coeffs[0]=pow(-1,n);


  for(i=1; i<=n; i++)
  {
    Ak=produitMatriciel(B,A,n,n,n);
```

```
    coeffs[i]=trace(Ak,n)/i;
    tmp=produitSMatriciel(I,n,n,coeffs[i]);
    for(j=0;j<n;j++)
    {
      free(B[j]);
    }
    free(B);
    B=difference(Ak,tmp,n,n);
    for(j=0;j<n;j++)
    {
      free(Ak[j]);
      free(tmp[j]);
    }
    free(tmp); free(Ak);
  }

  //On inverse le tableau des coefficients
  for(i=0;i<=((n+1)/2)-1;i++)
  {
    if ((n%2) == 0)
    {
      coeffs[i]=-coeffs[i];
    }
    tempo = coeffs[i];
    coeffs[i] = coeffs[n-i];
    coeffs[n-i] = tempo;
  }

  p=creerPoly(n+1, "tableau", coeffs);
  menuAffichage(p);

  //liberation memoire
  for (i=0; i<n; i++)
  {
    free(B[i]);
  }
  free(B);
  //libération du polynome
  free(p->poln);
  free(p);
}
```

Figure 2.2 – Code : leverrier.c

2.3 Résultats des tests

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

2.4 Comparaisons

3 Valeurs propres et vecteurs propres

3.1 Méthode des puissances itérées

3.1.1 Présentation

3.1.2 Programme

```
void puissancesIt(double** A, int n, double precision)
{
  int i, cpt=0;
  double ecart = 1.;
  double** x1 = (double**) malloc (n*sizeof(double*));
  double** x2;
  double** x3;
  double** xt;
  double** xtx;
  //Initialisation de x1 -> vecteur unité
  for (i=0; i<n; i++)
  {
    x1[i] = (double*) malloc (sizeof(double));
    x1[i][0] = 1.;
  }
  while (ecart > precision)
  {
    cpt++;
    x2 = produitMatriciel(A,x1,n,n,1); //x2 = A x1
    diviserComposantes(x2,n);
    x3 = difference(x2, x1, n, 1); //x3 = x2-x1
    ecart = norme(x3, n); //norme(x2-x1)-> écart
    //libération mémoire
    for (i=0; i<n; i++)
    {
      free(x3[i]);
      free(x1[i]);
    }
    free(x3);
    free(x1);
    x1 = x2; //xi = x(i+1)
  }
  printf("Nombre d'itérations : %d\n", cpt);
  printf("Approximation du vecteur propre :\n");
  for (i=0; i<n; i++)
  {
    printf("x[%d]=%f\n", (i+1), x1[i][0]);
  }
  xt = transpose(x1, n, 1);
  x2 = produitMatriciel(xt, A, 1, n, n); //xt A
  x3 = produitMatriciel(x2, x1, 1, n, 1); //xt A x
  xtx = produitMatriciel(xt, x1, 1, n, 1); //xt x
  printf("Approximation de la valeur propre maximum : %f\n", x3[0][0]/xtx[0][0]);
  //libération mémoire
  for (i=0; i<n; i++)
  {
    free(x1[i]);
  }
  free(xt[0]); free(x2[0]); free(x3[0]); free(xtx[0]);
```

```
    free(xt); free(x2); free(x3); free(xtx); free(x1);
}
```

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.

3.2.1 Résultats avec une précision 10^{-5}

Système analysé	Itération	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	Itération	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

3.2.2 Résultats avec une précision 10^{-9}

Système analysé	Itération	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	Itération	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

3.3 Comparaisons

4 Conclusion