DIF Clément

LOXOL Nicolas

Compte rendu TP : Calcul des Valeurs propres

1. Rappel des méthodes :

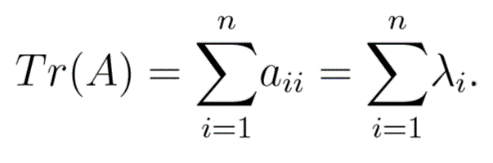
Avant de rentrer directement dans l’explication des méthodes ; il est important de rappeler certaines définitions :

Valeur propre : Soit λ une valeur réelle ; alors λ est valeur propre de f un endomorphisme de E dans E s’il existe u appartenant à E non nul tel que f(u) = λu.

Vecteur propre : Soit u non appartenant à E, E non vide. Alors u est un vecteur propre de l’endomorphisme f s’il existe λ une valeur réelle tel que f(u) = λu.

Polynôme caractéristique : Un polynôme caractéristique d’un endomorphisme f : E → E est un polynôme appartenant à Rn[X], noté et défini par Cf(X) = det (MB(f) - XIdE) où MB(f) est la matrice de f dans une base B de E quelconque.

Trace d’une matrice :



Remarques :

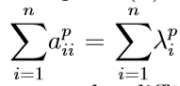
- Il est très facile de passer d’un endomorphisme a une matrice et réciproquement.

- Rn[X] est l’ensemble des polynômes de degré inférieur ou égal à n.

- IdE est l’endomorphisme identité sur l’espace vectoriel E.

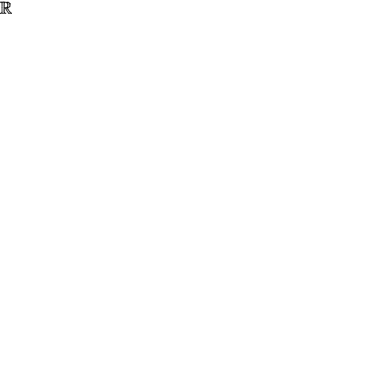
Méthode de Leverrier basique :

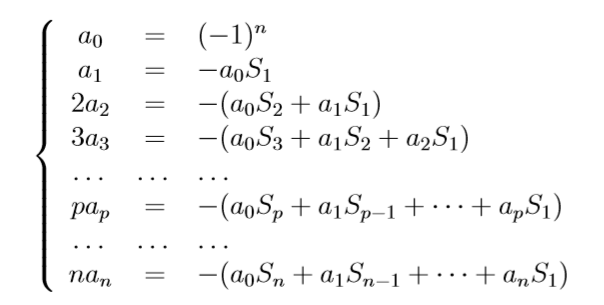
Soit CA(λ) le polynôme caractéristique associé à la matrice A.

On a CA(λ) = |A − λIn| = an + an−1λ + an−2λ2 + ··· + a0λn

Et on note Sp = Tr(Ap) =

La méthode de Leverrier permet de déterminer les coefficients du polynôme caractéristique précédemment défini.

D’après la propriété suivante issue du cours, Apx = λpx ∀p ‎∈  ∀x ‎∈ n, les identités de Newton nous permettent de relier les traces des différentes puissances de A aux coefficients du polynôme caractéristique de la manière suivante :



On obtient alors le résultat exact de chaque coefficient ai de CA(λ) ∀i = 0, …, n.

Remarque :

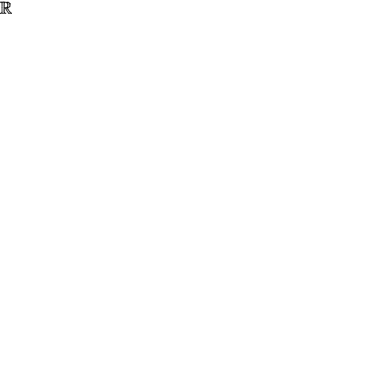
Cette méthode de Leverrier basique sera comparée en termes d’efficacité et de stockage, dans le programme, avec une méthode de Leverrier améliorée.

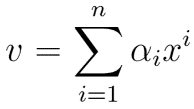
Méthode des puissances itérées :

Soit une matrice carrée A.

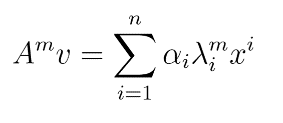
Cette méthode permet de calculer la plus grande valeur propre en valeur absolue du spectre (ensemble des valeurs propres) de la matrice A.

On suppose que toutes les valeurs propres sont distinctes. Des vecteurs propres xi associés respectivement aux valeurs propres λi sont alors linéairement indépendants.

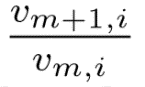
Un vecteur v ∈ n peut ainsi s’écrire



Si on multiplie m fois cette égalité par A, on obtient :



Si le spectre de A est tel que |λ1| > |λ2| > ··· > |λn| alors pour m très grand, le rapport tend vers 0 pour i = 2, ..., n.

Notons vm+1 = Amv, alors d'après ce qui précède, lorsque m est grand, vm+1 = Amv tend vers α1x1λm1.

Par conséquent, le rapport des deux vecteurs successifs tend vers λ1, ∀i = 1, ..., n.

On en déduit le processus itératif suivant :

* Choisir un vecteur v1 initial.
* Pour k ≥ 1, calculer vk+1 = Akv1 = Avk
* Arrêter lorsque que l‘on obtient : pour toute paire de composantes i et j de ces vecteurs.

Un vecteur propre x1 associé à la valeur propre λ1 peut être calculé en utilisant l'équation suivante : Ax1 = λ1x1

1. Présentation du programme :

Notre programme est constitué de plusieurs parties distinctes :

* Les « *includes* » afin d’utiliser les fonctions de la bibliothèque standard.
* Une structure « optimisation » et un typedef qui nous servira à comparer l’efficacité des méthodes Leverrier et évaluer la complexité temporelle et spatiale de chaque méthode en fonction de différents facteurs comme l’ordre de la matrice, la nature de la matrice, etc.
* Les prototypes repartis en plusieurs catégories : les méthodes définies précédemment, les opérations appliquées aux matrices, les fonctions d’allocation dynamique, les fonctions de génération de matrice et une fonction annexe.
* La fonction main ou l’on injecte les jeux d’essais c’est-à-dire la matrice, la taille de celle-ci, la précision ainsi que la méthode de calcul souhaitée.
* Les définitions des fonctions présentées dans les prototypes :

*methode\_leverrier\_base 🡪* Applique la méthode de Leverrier de base

*methode\_leverrier\_amelioree 🡪* Applique la méthode de Leverrier améliorée

*methode\_puissance 🡪* Applique la méthode des puissances

*generer\_identite 🡪* Génère la matrice identité

*copier\_matrice 🡪* Copie une matrice vers une autre

*puissance\_matrice 🡪* Calcule la puissance d’une matrice

*multiplier\_matrices 🡪* Multiplie deux matrices

*multiplier\_matrice\_scalaire 🡪* Multiplie un matrice par un scalaire

*additionner\_matrices 🡪* Additionne deux matrices

*calcule\_norme 🡪* Calcule la norme d’un vecteur

*calcule\_trace 🡪* Calcule la trace d’une matrice

*allouer\_memoire\_matrice 🡪* Allocation dynamique d’une matrice

*liberer\_memoire\_matrice🡪* Libération de la mémoire allouée pour une matrice

*afficher\_matrice 🡪* Affiche une matrice

*generer\_matrice\_creuse 🡪* Génère un matrice creuse

*generer\_matrice\_bord 🡪* Génère un matrice de Bord

*generer\_matrice\_ding\_dong 🡪* Génère un matrice de Ding Dong

*generer\_matrice\_de\_franc 🡪* Génère un matrice de Franc

*generer\_matrice\_de\_hilbert 🡪* Génère un matrice de Hilbert

*generer\_matrice\_kms 🡪* Génère un matrice de kms

*generer\_matrice\_de\_lotkin 🡪* Génère un matrice de Lotkin

*generer\_matrice\_de\_moler 🡪* Génère un matrice de Moler

*min 🡪* Calcul du minimum entre deux entiers

1. Présentation des jeux d’essais :

Nous avons choisi d’utiliser les matrices du TP1 comme jeux d’essais (creuse, de Bord, de Ding Dong, de Franc, de Hilbert, kms, de Lotkin et de Moler).

Pour la méthode de Leverrier, nous ferons varier la nature de la matrice ainsi que sa taille.

Pour la méthode des puissances, nous rajouterons comme paramètre à faire varier, la précision c’est-à-dire l’ordre de grandeur de la différence entre deux itérations.

Remarque :

Ce que l’on appelle « nature de la matrice » est en fait le type de cette dernière parmi ceux énoncés plus haut (kms, creuse, de Bord, etc.).

1. Commentaires des jeux d’essais :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Taille | Leverrier | | Leverrier améliorée | |
| Nombre de clocks processeur | Nombre d’octets alloués | Nombre de clocks processeur | Nombre d’octets alloués |
| 2 | 4 | 124 | 3 | 172 |
| 3 | 6 | 156 | 3 | 220 |
| 4 | 11 | 188 | 5 | 268 |
| 50 | 12 762 062 | 1 660 | 31 325 | 2 476 |

|  |  |  |
| --- | --- | --- |
| Taille | Méthode des puissances | |
| Nombre de clocks processeur | Nombre d’octets alloués |
| 2 | 23 | 48 |
| 3 | 29 | 56 |
| 4 | 31 | 64 |
| 50 | 1 991 | 432 |
| Précision | Méthode des puissances | |
| Nombre de clocks processeur | Nombre d’octets alloués |
| 10-1 | 26 | 56 |
| 10-2 | 28 | 56 |
| 10-3 | 30 | 56 |
| 10-10 | 33 | 56 |

On constate que la méthode de Leverrier améliorée est plus efficace que la méthode de Leverrier classique. En effet, le temps nécessaire à la résolution est bien plus élevé pour cette dernière et cela est d’autant plus vrai lorsque la taille de la matrice augmente. Le nombre d’octets nécessaires est, quant à lui, environ 1,5 fois plus élevé pour la méthode améliorée.

Concernant la méthode des puissances itérées, lorsque la taille de la matrice augmente, les clocks et les octets nécessaires augmentent en proportion. Cependant lorsque la précision augmente d’une manière notable, le temps de résolution augmente que très peu et les octets nécessaires au fonctionnement de la méthode restent exactement identiques.

Lors des tests, nous avons également remarqué que la nature des matrices n’influe pas sur le temps d’exécution ni sur le nombre d’octets nécessaires pour la résolution des méthodes de Leverrier et de la méthode des puissances.