

Projet du Calcul Parallèle

Problèmes aux valeurs propres symétriques

Ce rapport est un document explicatif du code attaché ainsi de l'idée générale du projet. Ce projet nommé "Problèmes aux valeurs propres symétriques : Réduction au tri diagonale."

Ce rapport contient essentiellement une explication de l'idée de projet, une simulation de l'exécution du code ainsi qu'une comparaison entre le code séquentiel et parallèle.

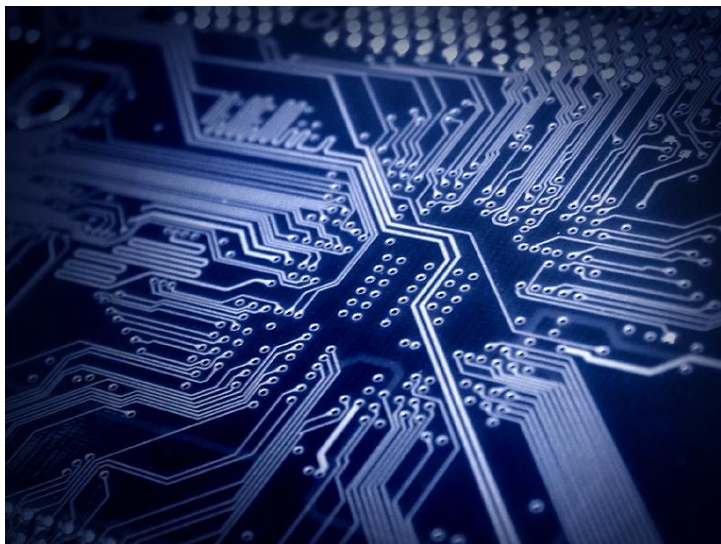
Ce rapport est réalisé en 11/06/2022 par EL HARRARI ANAS de la filière AI.

Aperçu

L'algèbre linéaire est un langage universel qui sert à décrire de nombreux phénomènes en

mécanique, électronique, et économie. C'est l'une des trois branches majeures des mathématiques appliquées au traitement des données, avec les probabilités et l'analyse. L'algèbre linéaire est utilisée par *tous* les composants logiciels, et dans une certaine mesure matérielle, de tous les équipements informatiques que vous utilisez.

L'algèbre linéaire apporte les outils qui permettent de travailler sur des matrices, qu'il s'agisse d'outils théoriques pour connaître



Ce projet traite un de ces problèmes, c'est celui de la résolution des systèmes linéaires symétriques en utilisant une des méthodes les plus importantes en ce domaine : la réduction à la forme triangulaire. La réduction à la forme triangulaire des matrices est un grand sujet de recherche en algèbre linéaire. Pour une matrice quelconque, il n'est pas aussi évident de s'assurer de la réduction à la forme triangulaire d'une matrice ou de la transformer en cette forme. Au cours de ce Projet nous allons nous limiter aux matrices carrées symétriques.

certaines propriétés spécifiques aux données qu'on observe.

Avoir une bonne connaissance de l'algèbre linéaire fournit de puissants outils pour résoudre des systèmes d'équations à plusieurs équations et à plusieurs variables.

De plus, l'algèbre linéaire, notamment avec l'utilisation des vecteurs, permet de bien représenter et de comprendre le monde physique qui nous entoure.

L'algèbre linéaire est d'une grande importance en physique. De nombreuses théories sont basées sur des approximations au premier ordre qui introduisent des relations linéaires entre les paramètres. C'est pourquoi les physiciens comme les numériciens ont consacré de grands efforts, depuis les débuts de l'informatique, à la mise au point d'algorithmes et de programmes efficaces pour résoudre cette classe de problèmes.

Dans la pratique, pour les systèmes comportant un grand nombre d'équations, à partir de $N = 20$ environ, il peut se produire des dégénérescences fortuites par suite d'accumulation d'erreurs d'arrondi au cours du calcul. On doit alors effectuer les calculs en double précision même lorsque la simple précision est satisfaisante pour la connaissance des inconnues.

Cela impose un grand problème en face du mathématicien ainsi que les programmeurs surtout après un développement incroyable au niveau de la machine et leurs capacités et performances.

[Tapez ici]

El Harrari ANAS

Problème aux valeurs propres symétriques : Réduction au tri diagonal



Outils utilisés en ce Projet :

- Langage de Programmation C++
- Logiciels QT
- Des packages mathématiques
- Le langage Matlab

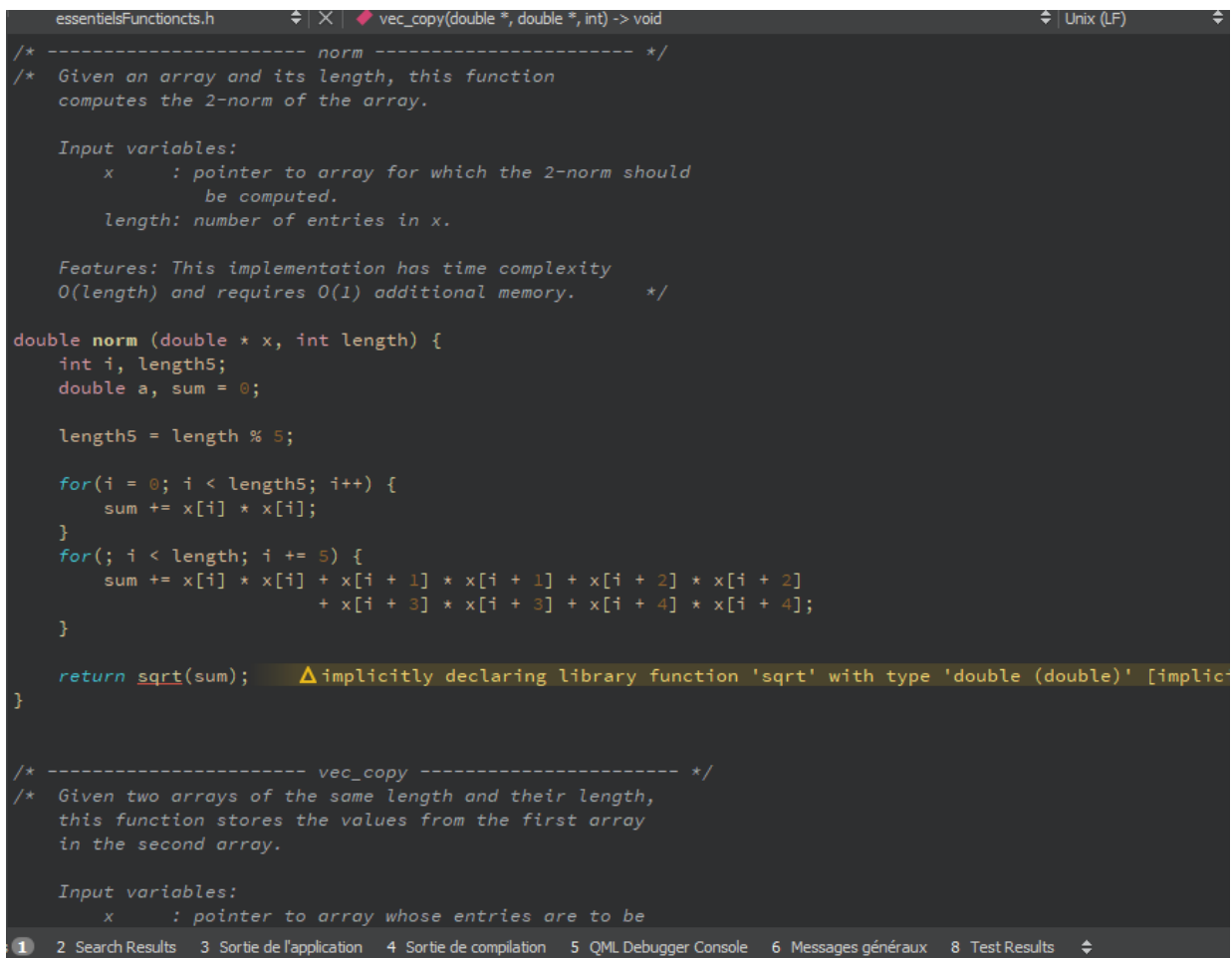
Pourquoi QT ?

Certes, ce projet a pour but de résoudre un des problèmes d'algèbre linéaire les plus connus. Cependant, le but principal reste d'appliquer les théories et les connaissances de calcul parallèle dans des sujets concrets et pratiques. QT offre un ensemble des classes et des bibliothèques intéressantes qui servent à achever ce but. Des classes telles que (qthread, qmaths, qDthread...etc) ainsi que des fonctions mises en scène afin de faciliter l'implémentation des bases du Calcul parallèle.

QT supporte plusieurs langages de programmation (comme C/C++, c' , python. Etc). Cependant, suite aux règles données, en cahier de charge du projet, nous allons se limiter dans notre travail au langage c++.

Code Séquentiel vs Code Parallélisé

Dans cette partie, nous allons discuter le code écrit afin de répondre au sujet du projet. Ensuite, nous allons réaliser une comparaison entre le code dans sa version séquentiel et parallélisé afin de montrer la différence au niveau les performances et rapidité d'exécution.



```
essentialsFunctioncts.h  X  vec_copy(double *, double *, int) -> void  Unix (LF)

/* ----- norm ----- */
/* Given an array and its length, this function
   computes the 2-norm of the array.

   Input variables:
       x      : pointer to array for which the 2-norm should
                be computed.
       length: number of entries in x.

   Features: This implementation has time complexity
   O(length) and requires O(1) additional memory.  */

double norm (double * x, int length) {
    int i, length5;
    double a, sum = 0;

    length5 = length % 5;

    for(i = 0; i < length5; i++) {
        sum += x[i] * x[i];
    }
    for(; i < length; i += 5) {
        sum += x[i] * x[i] + x[i + 1] * x[i + 1] + x[i + 2] * x[i + 2]
              + x[i + 3] * x[i + 3] + x[i + 4] * x[i + 4];
    }

    return sqrt(sum);  Δ implicitly declaring library function 'sqrt' with type 'double (double)' [implicit
}

/* ----- vec_copy ----- */
/* Given two arrays of the same length and their length,
   this function stores the values from the first array
   in the second array.

   Input variables:
       x      : pointer to array whose entries are to be
```

[Tapez ici]

El Harrari ANAS

Problème aux valeurs propres symétriques : Réduction au tri diagonal

```

n: number of columns of A.
*/

void householder (double ** a, double ** v, int m, int n) {
    int i, j;
    double vnorm, vTa, vpartdot;

    for(i = 0; i < n; i++) {
        /* set v[i] equal to subvector a[i][i : m] */
        partialvec_copy(a[i], v[i], m - i, i);

        /* vpartdot = ||v[i]||^2 - v[i][0] * v[i][0]; since vpartdot
           is unaffected by the change in v[i][0], storing this value
           prevents the need to recalculate the entire norm of v[i]
           after updating v[i][0] in the following step */
        vpartdot = partialdot_product(v[i], v[i], m - i, i);

        /* set v[i][0] = v[i][0] + sign(v[i][0]) * ||v[i]|| */
        if(v[i][0] < 0) {
            v[i][0] -= sqrt(v[i][0] * v[i][0] + vpartdot);
        }
        else {
            v[i][0] += sqrt(v[i][0] * v[i][0] + vpartdot);
        }

        /* normalize v[i] */
        vnorm = sqrt(v[i][0] * v[i][0] + vpartdot);
        scalar_div(v[i], vnorm, m - i, v[i]);

        for(j = i; j < n; j++) {
            /* set a[j][i:m] = a[j][i:m] - 2 * (v[i]^T a[j][i:m]) * v[i] */
            vTa = subdot_product(a[j], v[i], m - i, i);
            vTa *= 2;
            partialscalar_sub(v[i], vTa, m - i, i, a[j]);
        }
    }
}

```

Afin de répondre au sujet de ce projet, nous allons parcourir un des méthodes les plus communs pour tri diagonaliser une matrice. C'est la méthode de Householder. Dans les systèmes linéaires symétriques, nous nous sommes assurés que notre Matrice est une matrice carrée ainsi que diagonalisable et tridiagonalisable. Au cours de ce projet, nous allons même se bénéficier de cette tridiagonalisation afin d'effectuer la décomposition QR. Une décomposition basé sur La tri diagonalisation du matrice principale afin d'arriver à un produit d'une matrice triangulaire supérieure et une matrice parfaitement diagonalisable. En outre, nous sommes même convaincus que les valeurs propres de notre systèmes sont réelles et de même signes. Ce code-là est composé principalement de deux fichiers, un qui contient les fonctions indispensables pour appliquer le tri diagonalisation de la matrice ainsi que la décomposition QR. Le deuxième fichier contient principalement la réponse au sujet du projet ainsi qu'une normalisation des vecteurs de la matrice Q dans le but de donner un sens au travail réalisé au cours de ce projet. La nature, purement mathématiques du projet, nécessite une forte connaissance en algèbre linéaire et bilinéaire ainsi que le calcul matriciel et les espaces vectoriels. Le code commence par demander au utilisateur la taille de la matrice appartient au système ainsi qu'une vérification qu'elle est bien carrée et par la suite réaliser le traitement nécessaire pour arriver au résultat souhaité. Ce résultat est illustré par la capture suivante :

[Tapez ici]

El Harrari ANAS

Problème aux valeurs propres symétriques : Réduction au tri diagonal

```
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Enter the dimension m (where A is a m by n matrix): 4
Enter the dimension n (where A is a m by n matrix): 4
A =
    1    0    0    0
    2    1    0    0
    3    2    1    0
    4    3    2    1

R =
-5.47723 -3.65148 -2.00832 -0.730297
 0  0.816497  0.816497  0.408248
 0  5.55112e-17  0.547723  0.365148
 0  1.11022e-16      0 -0.408248

v[0] =  0.768952  0.237433  0.356149  0.474865
v[1] = -0.760308  0.248695  0.600069
v[2] = -0.930176 -0.367113
v[3] =      1

Numerical verification that v_1, ..., v_4 are normalized:
||v[1]|| = 1, ||v[2]|| = 1, ||v[3]|| = 1, ||v[4]|| = 1.

Appuyez sur <ENTRÉE> pour fermer cette fenêtre...
-
```

Performance du code Parallélisé :

Taille de la matrice	Code Séquentiel (temps d'exécution en ms)	Code Parallélisé (temps d'exécution en ms)
4X4	3	0.96
10X10	15	0.98
100X100	189	1
1000X1000	360000	56
10000X100000	INF	388

Réalisation au-delà du C++ (EX : Matlab) :

```
Editor - C:\Users\Uemf\Downloads\Calcul-Parallel\untitled.m
untitled.m  x  +
1  function B=untitled(A)
2      [n,m]=size(A);
3      if(n==m)
4          for i=1:n-2
5              a11=A(i,i);
6              a=A(i+1:end,i);
7              if(isequal(a,zeros(n-1,i)))
8                  break
9              end
10             U=A(i+1:end,i+1:end);
11             k=sqrt(a'*a);
12             v=a-k*eye(n-i,1);
13             alpha=v'*v;
14             p=eye(n-i)-2*v*v'/alpha;
15             A(i:end,i:end)=[a11,k*eye(n-i,1)';k*eye(n-i,1),p*U*p];
16 end
```

```
Command Window
B =
    3.0000    8.0623    0
    8.0623   11.8462    2.2308
         0    5.2308    1.1538
fx >> |
```

Ces langages mènent à un résultat rapide et facile ou le développeur n'est pas obligé de parcourir les différentes notions théories des maths ainsi que de la programmation. Cependant un tel programme est lourd au niveau de la mémoire et son exécution au niveau du processeur.

[Tapez ici]

El Harrari ANAS

Problème aux valeurs propres symétriques : Réduction au tri diagonal

Conclusion :

Les problèmes d'algèbre linéaire étaient toujours un sujet de la discussion entre les mathématiciens. Un sujet qui influence différents domaines

On est jusqu'à maintenant à la recherche d'une solution plus performante et parfaite à un tel type de problème.

La solution présentée et développée en ce projet reste un des meilleures solutions à nos jours pour traiter les problèmes des valeurs propres symétriques et une opportunité intéressante afin de développer et pratiquer les théories et les connaissances parcourus en Calcul Parallèle.