Miniprojet d'Algorithmique et méthodes de programmation

Jimmy Hester, Guilhem Saurel

1*er juin* 2011

Table des matières

1 Algorithme

En ce qui concerne l'algorithme de calcul, nous avons décidé de fixer le nombre d'itération non pas de manière arbitraire mais en imposant une précision minimale sur la convergence de la valeur propre : l'algorithme s'arrête lorsque la différence entre la valeur propre d'une itération et celle de la suivante est inférieure à 0.001. Un nombre d'itérations maximal a été fixé en cas de non convergence de la suite afin d'éviter le bouclage infini. Les itérations se font 20 par 20 pour réduire le nombre de calculs (les conditions de continuation de l'algorithme ne sont vérifiées que toutes les 20 itérations).

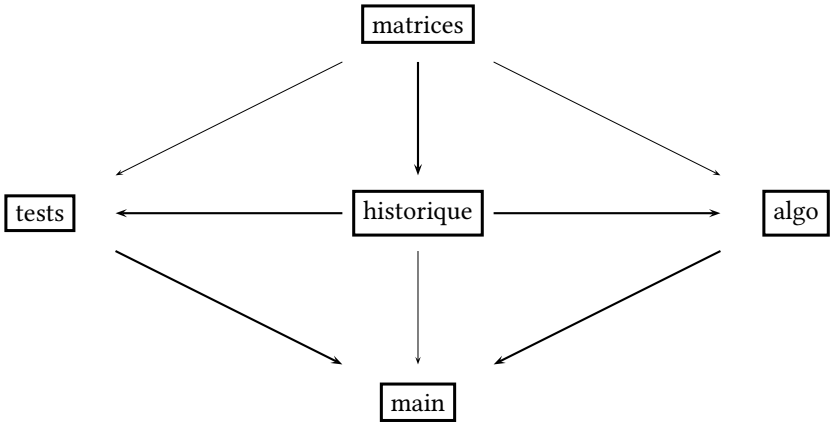
On vérifie ensuite que la norme de x n'est pas nulle (ou très petite) pour ne pas diviser par 0 pendant le calcul de u1 et pour les calculs suivants . On cherche ensuite un composante non nulle de x pour en extraire la valeur propre sachant qu'elles ne peuvent pas toutes êtres nulles puisque la norme de x a été testée non nulle. Si le nombre d'itérations est trop élevé ou que la norme de x devient trop petite, le programme conclut brutalement (plus ou moins justement) qu'il n'y a pas de valeur propre.

2 Découpage des fichiers

En ce qui concerne le découpage de fichiers, on a préféré regrouper toutes les fonctions similaires ensemble. Par exemple, il parait plus logique de mettre ensemble tous les constructeurs et destructeurs, et toutes les méthodes d'affichage pour les complexes, les vecteurs et les trois types de matrices, plutot que de les séparer en trois ou cinq fichiers .cpp différents. Cependant, puisque le fichier .h doit avoir le même nom que le .cpp, on ne peut pas éclater les définitions des différents types de méthodes (par exemple un "affichage.cpp" et un "constructeurs.cpp") car cela impliquerait que la définition de la classe serait explosée dans plusieurs .h différents, ce qui n'est pas très lisible. On a donc seulement des fichiers .h et .cpp pour l'algorithme, l'historique et les tests, qui sont traités chacuns de leur côté, et tout le reste part dans matrice.h et matrice.cpp.

Mais peut être que la gestion des complexes aurait mérité une séparation.

On pourrait schématiser les dépendances des fichiers entre eux de la manière suivante :



3 Tests unitaires

Afin de vérifier le bon fonctionnement des classes et des méthodes, nous avons programé quelques tests censés nous avertir lorsque nous cassions quelque chose. Cela a en effet servi à plusieurs reprises !

De plus, nous avions toujours gardé la possibilité de ne pas encombrer l'affichage avec ces tests à l'aide d'un seul booléen.

4 Détails techniques qui nous ont marqués

Nous avons fait le choix de travailler avec des classes plutôt que des structures, au début essentiellement parcequ'il y avait moins de texte à écrire, et que ça ne changeait rien. Puis petit à petit, nous nous sommes rendus compte que la plupart de nos fonctions pouvaient en fait passer pour des méthodes de nos classes, et ça semble donc au final beaucoup plus clair. Cependant, nous n'avons pas tenu compte du principe d'encapsulation.

On initialise les variables lors de la création avec des parenthèses, pour bien faire la différence entre le constructeur d'affectation et l'opérateur =, même si en pratique ça ne change rien : ce n'est que pour ne pas le perdre de vue. En effet, généralement, si l'on n'y réfléchi pas, utiliser l'opérateur = au lieu du constructeur d'affectation peut engendrer deux affectations consécutives, la première étant inutile . De plus, dans notre cas, ne pas bien faire la différence a longtemps empêché nos destructeurs de fonctionner correctement, ce qui engendrait naturellement une fuite mémoire. Mais maintenant, valgrind est sans appel : `total heap usage : 224 allocs, 224 frees, 16,764 bytes allocated` [...] `All heap blocks were freed -- no leaks are possible`

Notre compilateur étant verbeux, il nous fait remarquer à chaque compilation que l'option "%lf" n'est pas standard pour printf et ses dérivés, mais nous n'avons pas pu arriver à afficher des doubles sans avoir de warning.

On a aussi droit à l'alerte `reference to local variable ‘b’ returned [enabled by default]` aux lignes 78 et 85 de matrice.cpp, mais cela ne semble pas trop grave...

Sinon, on aurait pu surcharger l'opérateur << pour afficher plus facilement les matrices, mais le temps à manqué.

5 Ordonnage

Il n'était nulle part mentionné dans la sujet que la matrice creuse un devait respecter un certain ordre. Cependant, on s'est vite rendu compte que la conversion d'une matrice creuse un vers une matrice creuse deux ou qu'un test d'égalité entre deux matrices creuses un serait infiniement plus simple (à coder, mais aussi en nombre d'opérations processeur) si l'on partait du principe que ces matrices étaient lues de gauche à droite et de haut en bas.

Nous avons donc décidé de partir du principe que nous ne manipulerions que des matrices ordonnées, et que si, à un moment, ou a un autre, on pouvait tomber sur une matrice non ordonnée, alors il nous fallait l'ordonner.

Pour cela, nous avons d'abord fait un test, qui nécessite un nombre d'opérations faible (dominé par N_z) avant de se lancer dans l'ordonnage, qui lui demande beaucoup plus.

Cependant, l'algorithme d'ordonnage utilisé est *particulièrement* rudimentaire (le nombre d'opération est dominé par $N^2 N_z$...). Nous pensions utiliser un smoothsort, qui tirerait profit du fait qu'une matrice serait très probablement peu mélangées (pour rappel, c'est un tri de complexité en $O(N \log N)$, ce qui est déjà pas mal, mais si les données sont déjà presque triées, il est de complexité en $O(N)$: le test ne serait alors plus nécessaire).

6 Respect du cahier des charges

Malheureusement, notre algorithme donne des résultats peu convaincants devant ceux fournis par Scilab... Mais nous nous en sommes rendu compte un peu tard.

Il aurait également fallut qu'on fasse des lectures/écritures en binaire, mais puisque nous n'en avons pas vraiment eu besoin, nous l'avons repoussé, jusqu'à qu'il ne soit trop tard... Cependant, l'algorithme était déjà tout prêt, vu que nous l'avions déjà codé pour le TD cinq (si vous tenez à y jeter un oeil, il est là : "http ://bde.enseeiht.fr/ saurelg/cinq.cpp", vers les lignes 80 à 100).

La sauvegarde de l'historique de convergence a elle été implémentée, mais n'est pas utilisée dans notre programme, vu que l'algorithme en lui-même n'a pas l'air de marcher...

Tout le reste est codé et fonctionne.

Enfin, l'implémentation de l'algorithme pour deux types de matrices différents n'a pas été fait, mais vu que nous avons tout le nécessaire pour convertir n'importe quelle matrice en matrice creuse deux, cela semble plutot futile. On remarque également qu'étant donné que tout est fait à partir de surcharges d'opérateurs, et que ces surcharges sont implémentées pour les trois types de matrices, on pourrait ne même pas avoir de conversion à faire.

7 Conclusion

Pour conclure, on peut dire que nous avons certainement passé beaucoup trop de temps sur des détails de perfectionnistes (utilisation mémoire, lisibilité du code, non-redondance, optimisations diverses, définition de méthodes parfois inutilisées, questionnements métaphysiques sur la nécessité de ne pas écraser un fichier déjà présent sur le système, conventions de forme, organisation des fonctions dans les fichiers, tests unitaires, gestion des erreurs et autres asserts, etc.) alors que nous aurions du, au moins, répondre à la question posée, à savoir trouver des valeurs/vecteurs propres. Cependant, il est hors de question de laisser un tel travail à ce point inachevé, qu'il soit rendu ou pas, et il est plutôt probable que tout marche bien le jour de l'oral.

8 Et en bonus track...

L'historique de notre travail depuis le début du miniprojet est disponible sur "http ://github.com/nim65s/scritps/", dans le dossier "miniprojet" (le reste du dépôt n'ayant rien à voir avec du C).

A matrices.h

```
#ifndef MATRICES_H_INCLUDED
#define MATRICES_H_INCLUDED

#include <string>

class complexe {
    public :
        double re;
        double im;

        complexe();
        complexe(double const & a);
        complexe(double const & a, double const & b);
        complexe(complexe const & a);

        complexe & operator+=(complexe const & a);
        complexe & operator-=(complexe const & a);
        complexe & operator*=(complexe const & a);
        complexe & operator/=(complexe const & a);
        complexe & operator*=(double const & x);
        complexe & operator/=(double const & x);

        void afficher() const;
        bool isnull() const;
        double norme() const;
};
bool operator==(complexe const & a, complexe const & b);
bool operator!=(complexe const & a, complexe const & b);
complexe operator*(complexe const & a, complexe const & b);
complexe operator*(complexe a, double const & b);
complexe operator/(complexe const & a, complexe const & b);
complexe operator/(complexe a, double const & b);
complexe operator+(complexe a, complexe const & b);
complexe operator-(complexe a, complexe const & b);
complexe pow(complexe const & a, int const & b);
complexe conjugue(complexe & a);

class vecteur {
    public :
        int n;
        complexe * coef;

        vecteur(int const & dim);
        vecteur(vecteur const & v);
        vecteur & operator=(vecteur const & a);
        ~vecteur();

        void afficher() const;
        double norme() const;
};
bool operator==(vecteur const & a, vecteur const & b);
bool operator!=(vecteur const & a, vecteur const & b);
vecteur operator*(vecteur const & v, double const & x);
vecteur operator*(vecteur v, complexe const & a);
vecteur operator/(vecteur v, double const & x);

class matricecreusedeux {
    public :
        int m;
        int n;
        int nz;
        complexe * vals;
        int * j;
        int * II;

        matricecreusedeux(int const & lig, int const & col, int const & nzv);
        matricecreusedeux(matricecreusedeux const & A);
        matricecreusedeux & operator=(matricecreusedeux const & A);
        ~matricecreusedeux();

        void afficher() const;
        int ecrire() const;
        int ecrire(std ::string const & file) const;
};
bool operator==(matricecreusedeux const & A, matricecreusedeux const & B);
bool operator!=(matricecreusedeux const & A, matricecreusedeux const & B);
vecteur operator*(matricecreusedeux const & M, vecteur const & v);
matricecreusedeux liredeux(bool const & comp);
matricecreusedeux liredeux(std ::string const & file, bool const & comp);

class matricecreuseun {
```

```
    public :
        int m;
        int n;
        int nz;
        int * i;
        int * j;
        complexe * coef;

        matricecreuseun(int const & lig, int const & col, int const & nzv);
        matricecreuseun(matricecreuseun const & A);
        matricecreuseun & operator=(matricecreuseun const & A);
        ~matricecreuseun();

        void afficher() const;
        bool estenbordel() const;
        matricecreuseun ordonne() const;
        matricecreusedeux versdeux() const;
        int ecrire() const;
        int ecrire(std ::string const & file) const;
};
bool operator==(matricecreuseun A, matricecreuseun B);
bool operator!=(matricecreuseun const & A, matricecreuseun const & B);
vecteur operator*(matricecreuseun const & M, vecteur const & v);
matricecreuseun lireun(bool const & comp);
matricecreuseun lireun(std ::string const & file, bool const & comp);

class matricepleine {
    public :
       int m;
       int n;
       int nz;
       //complexe * coef; TODO : coef[m]? coef[m][n]?
       complexe coef[10][10];

       matricepleine();
       matricepleine(int const & lig, int const & col, int const & nzv);
       matricepleine(matricepleine const & A);
       matricepleine & operator=(matricepleine const & A);
       ~matricepleine();

       void afficher() const;
       matricecreuseun versun() const;
       matricecreusedeux versdeux() const;
};
bool operator==(matricepleine const & A, matricepleine const & B);
bool operator!=(matricepleine const & A, matricepleine const & B);
vecteur operator*(matricepleine const & A, vecteur const & v);

#endif
```

B matrices.cpp

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cmath>
#include <assert.h>
#include "matrices.h"

using namespace std;

/*****************************************************************************************************************
 *                                                  Complexes                                                    *
 *****************************************************************************************************************/

bool operator==(complexe const & a, complexe const & b) {{{ // ------------------------------------ Égalité de complexes
    if(a.re == b.re && a.im == a.im) return true;
    return false;
}}}

bool operator!=(complexe const & a, complexe const & b) {{{ // ------------------------------- Différence de complexes
    return !(a==b);
}}}

complexe operator+(complexe a, complexe const & b) {{{ // ---------------------------------------- Addition de complexes
    a.re += b.re;
    a.im += b.im;
    return a;
}}}

complexe operator-(complexe a, complexe const & b) {{{ // ---------------------------------- Soustraction de complexes
```

```
    a.re -= b.re;
    a.im -= b.im;
    return a;
}}}

complexe conjugue(complexe a) {{{ // ------------------------------------------------------- Conjugaison de complexes
    a.im = -a.im;
    return a;
}}}

complexe operator*(complexe const & a, complexe const & b) {{{ // ------------------------- Multiplication de complexes
    // (v+iw)(x+iy) = vx - wy + i(vy +wx)
    complexe c;
    c.re = a.re*b.re-a.im*b.im;
    c.im = a.re*b.im+a.im*b.re;
    return c;
}}}

complexe operator*(complexe a, double const & b) {{{ // ------------------------- Multiplication de complexe et de réel
    a.re *= b;
    a.im *= b;
    return a;
}}}

complexe operator/(complexe const & a, complexe const & b) {{{ // ------------------------------- Division de complexes
    // (v+iw)/(x+iy) = ((v+iw)(x-iy))/(x²+y²)
    return a*conjugue(b)/(pow(b.re,2)+pow(b.im,2));
}}}

complexe operator/(complexe a, double const & b) {{{ // ---------------------------- Division d'un complexe par un réel
    a.re /= b;
    a.im /= b;
    return a;
}}}

complexe & complexe ::operator+=(complexe const & a) {{{ // --------------------------- Addition raccourcie de complexes
    re += a.re;
    im += a.im;
    return *this;
}}}

complexe & complexe ::operator-=(complexe const & a) {{{ // ------------------------ Soustraction raccourcie de complexes
    re -= a.re;
    im -= a.im;
    return *this;
}}}

complexe & complexe ::operator*=(complexe const & a) {{{ // ---------------------- Multiplication raccourcie de complexes
    complexe b;
    b.re = re*a.re-im*a.im;
    b.im = re*a.im+im*a.re;
    return b;
}}}

complexe & complexe ::operator/=(complexe const & a) {{{ // --------------------------- Division raccourcie de complexes
    complexe b;
    b.re = (re*a.re-im*a.im)/(pow(a.re,2)+pow(a.im,2));
    b.im = -(re*a.im+im*a.re)/(pow(a.re,2)+pow(a.im,2));
    return b;
}}}

complexe & complexe ::operator*=(double const & x) {{{ // -------------- Multiplication raccourcie de complexe et de réel
    re *= x;
    im *= x;
    return *this;
}}}

complexe & complexe ::operator/=(double const & x) {{{ // ----------------- Division raccourcie d'un complexe par un réel
    re /= x;
    im /= x;
    return *this;
}}}

double complexe ::norme() const {{{ // ------------------------------------------------------------- Norme d'un complexe
    return sqrt(pow(re,2)+pow(im,2));
}}}

bool complexe ::isnull() const {{{ // ------------------------------------------------------------ Nullité d'un complexe
    if(re == 0 && im == 0) return true;
    return false;
}}}

complexe pow(complexe const & a, int const & k) {{{ // ------------------------------- Puissance entière d'un complexe
```

```
    complexe b = a;
    for(int i(1); i<k; i++) b *= a;
    return b;
}}}

/*****************************************************************************************************************
 *                                          Construteurs et destructeurs                                         *
 *****************************************************************************************************************/

complexe ::complexe()  : re(0), im(0) {{{ // ------------------------------------------------- Constructeur de complexe
}}}

complexe ::complexe(double const & a)  : re(a), im(0) {{{ // ------- Constructeur de complexe à partir d'une partie réelle
}}}

// ------------------------------------------------- Constructeur de complexe à partir de parties réelle et imaginaire
complexe ::complexe(double const & a, double const & b)  : re(a), im(b) {{{
}}}

complexe ::complexe(complexe const & a)  : re(a.re), im(a.im) {{{ // ------------------ Constructeur de complexe de copie
}}}

vecteur ::vecteur(int const & dim)  : n(dim) {{{ // ---------------------------------------------- Constructeur de vecteur
    assert(n != 0);
    coef = new complexe[n];
}}}

vecteur ::vecteur(vecteur const & v)  : n(v.n) {{{ // ---------------------------------- Constructeur de vecteur de copie
    assert(n != 0);
    coef = new complexe[n];
    for(int i(0); i<n; i++) coef[i] = v.coef[i];
}}}

vecteur & vecteur ::operator=(vecteur const & v) {{{ // ----------------------------- Opérateur d'affectation de vecteur
    if(this != &v) {
        n = v.n;
        delete [] coef;
        coef = new complexe(*(v.coef));
    }
    return *this;
}}}

vecteur ::~vecteur() {{{ // ------------------------------------------------------------------------ Destructeur de vecteur
    delete [] coef;
}}}

// ------------------------------------------------------------------------------------------ Constructeur de matrice pleine
matricepleine ::matricepleine(int const & lig, int const & col, int const & nzv)  : m(lig), n(col), nz(nzv) {{{
    assert(n != 0 && m != 0 && nz != 0);
    //TODO coef = new complexe[nzv][nzv];
}}}

// ------------------------------------------------------------------------------- Constructeur de matrice pleine de copie
matricepleine ::matricepleine(matricepleine const & A)  : m(A.m), n(A.n), nz(A.nz) {{{
    assert(n != 0 && m != 0 && nz != 0);
    for(int i(0); i<m; i++) for(int j(0); j<n; j++) coef[m][n] = A.coef[m][n];
    //TODO coef = new complexe[nzv][nzv];
}}}

matricepleine & matricepleine ::operator=(matricepleine const & A) {{{ // ----- Opérateur d'affectation de matrice pleine
    if(this != &A) {
        m = A.m;
        n = A.n;
        nz = A.nz;
        for(int i(0); i<m; i++) for(int j(0); j<n; j++) coef[m][n] = A.coef[m][n];
    }
    return *this;
}}}

matricepleine ::~matricepleine() {{{ // ------------------------------------------------- Destructeur de matrice pleine
}}}

// ------------------------------------------------------------------------------------ Constructeur de matrice creuse un
matricecreuseun ::matricecreuseun(int const & lig, int const & col, int const & nzv)  : m(lig), n(col), nz(nzv) {{{
    assert(m != 0 && n != 0 && nz != 0);
    i = new int[nzv];
    j = new int[nzv];
    coef = new complexe[nzv];
}}}

// ------------------------------------------------------------------------- Constructeur de matrice creuse un de copie
matricecreuseun ::matricecreuseun(matricecreuseun const & A)  : m(A.m), n(A.n), nz(A.nz) {{{
    assert(m != 0 && n != 0 && nz != 0);
```

```
    i = new int[nz];
    j = new int[nz];
    coef = new complexe[nz];
    for (int k(0);k<A.nz;k++) {
        i[k] = A.i[k];
        j[k] = A.j[k];
        coef[k] = A.coef[k];
    }
}}}

// ------------------------------------------------------------------------ Opérateur d'affectation de matrice creuse un
matricecreuseun & matricecreuseun ::operator=(matricecreuseun const & A) {{{
    if(this != &A) {
        m = A.m;
        n = A.n;
        nz = A.nz;
        delete [] i;
        delete [] j;
        delete [] coef;
        i = new int(*(A.i));
        j = new int(*(A.j));
        coef = new complexe(*(A.coef));
    }
    return *this;
}}}

matricecreuseun ::~matricecreuseun() {{{ // ------------------------------------------ Destructeur de matrice creuse un
    delete [] i;
    delete [] j;
    delete [] coef;
}}}

// ------------------------------------------------------------------------------------ constructeur de matrice creuse deux
matricecreusedeux ::matricecreusedeux(int const & lig, int const & col, int const & nzv)  : m(lig), n(col), nz(nzv) {{{
    assert(m != 0 && n != 0 && nz != 0);
    vals = new complexe[nzv];
    j = new int[nzv];
    II = new int[lig+1];
}}}

// ------------------------------------------------------------------------ Constructeur de matrice creuse deux de copie
matricecreusedeux ::matricecreusedeux(matricecreusedeux const & A)  : m(A.m), n(A.n), nz(A.nz) {{{
    assert(m != 0 && n != 0 && nz != 0);
    vals = new complexe[nz];
    j = new int[nz];
    II = new int[m+1];
    for(int i(0); i<nz; i++) {
        vals[i] = A.vals[i];
        j[i] = A.j[i];
    }
    for(int i(0); i<=m; i++) II[i] = A.II[i];
}}}

// ------------------------------------------------------------------------ Opérateur d'affectation de matrice creuse deux
matricecreusedeux & matricecreusedeux ::operator=(matricecreusedeux const & A) {{{
    if(this != &A) {
        m = A.m;
        n = A.n;
        nz = A.nz;
        delete [] vals;
        delete [] j;
        delete [] II;
        vals = new complexe(*(A.vals));
        j = new int(*(A.j));
        II = new int(*(A.II));
    }
    return *this;
}}}

matricecreusedeux ::~matricecreusedeux() {{{ // ------------------------------------ Destructeur de matrice creuse deux
    delete [] vals;
    delete [] j;
    delete [] II;
}}}

/*****************************************************************************************************************************
 *                                                Affichage de matrices                                                      *
 *****************************************************************************************************************************/

void complexe ::afficher() const {{{ // ------------------------------------------------------- Affichage de complexe
    if(im!=0) printf("%5.3e+i%-5.3e ", re, im);
    else printf("%12.11g ", re);
}}}
```

```
void vecteur ::afficher() const {{{ // ------------------------------------------------------------ Affichage de vecteur
    cout << "[ ";
    for(int i(0);i<n;i++) coef[i].afficher();
    cout << "]" << endl;
}}}

void matricepleine ::afficher() const {{{ // ---------------------------------------------- Affichage de matrice pleine
    cout << "⎡ ";
    for(int j(0);j<m;j++) coef[0][j].afficher();
    cout << "⎤" << endl;
    for(int i(1);i<n-1;i++) {
        cout << "⎢ ";
        for(int j(0);j<m;j++) coef[i][j].afficher();
        cout << "⎥" << endl;
    }
    cout << "⎣ ";
    for(int j(0);j<m;j++) coef[n-1][j].afficher();
    cout << "⎦" << endl;
}}}

void matricecreuseun ::afficher() const {{{ // ------------------------------------------ Affichage de matrice creuse un
    cout << "  i  | ";
    for(int k(0);k<nz;k++) printf("%12d ",i[k]);
    cout << endl << "  j  | ";
    for(int k(0);k<nz;k++) printf("%12d ",j[k]);
    cout << endl << "coef | ";
    for(int k(0);k<nz;k++) coef[k].afficher();
    cout << endl;
}}}

void matricecreusedeux ::afficher() const {{{ // ------------------------------------- Affichage de matrice creuse deux
    cout << "vals | ";
    for(int k(0);k<nz;k++) vals[k].afficher();
    cout << endl << "  j  | ";
    for(int k(0);k<nz;k++) printf("%12d ",j[k]);
    cout << endl << " II  | ";
    for(int k(0);k<=m;k++) printf("%12d ",II[k]);
    cout << endl;
}}}

/*****************************************************************************************************************************
 *                                                      Égalités                                                             *
 *****************************************************************************************************************************/

bool operator==(vecteur const & a, vecteur const & b) {{{ // ------------------------------------- Égalité de vecteurs
    if (a.n != b.n) return false;
    for(int i(0);i<a.n;i++) if (a.coef[i] != b.coef[i]) return false;
    return true;
}}}

bool operator!=(vecteur const & a, vecteur const & b) {{{ // ------------------------------------ Différence de vecteurs
    return !(a==b);
}}}

bool operator==(matricepleine const & A, matricepleine const & B) {{{ // ------------------ Égalité de matrices pleines
    if( A.n != B.n || A.m != B.m) return false;
    for(int i(0);i<A.n;i++) for(int j(0);j<A.m;j++) if (A.coef[i][j] != B.coef[i][j]) return false;
    return true;
}}}

bool operator!=(matricepleine const & A, matricepleine const & B) {{{ // --------------- Différence de matirces pleines
    return !(A==B);
}}}

bool operator==(matricecreuseun A, matricecreuseun B) {{{ // --------------------------- Égalité de matrices creuses un
    if (A.m != B.m || A.n != B.n || A.nz != B.nz) return false;
    if (A.estenbordel()) A = A.ordonne();
    if (B.estenbordel()) B = B.ordonne();
    for(int k(0);k<A.n;k++) if(A.i[k] != B.i[k] || A.j[k] != B.j[k] || A.coef[k] != B.coef[k]) return false;
    return true;
}}}

bool operator!=(matricecreuseun const & A, matricecreuseun const & B) {{{ // --------- Différence de matrices creuses un
    return !(A==B);
}}}

bool operator==(matricecreusedeux const & A, matricecreusedeux const & B) {{{ // ------ Égalité de matrices creuses deux
    if (A.m != B.m || A.n != B.n || A.nz != B.nz) return false;
    for (int k(0);k<A.nz;k++) if (A.vals[k] != B.vals[k] || A.j[k] != B.j[k]) return false;
    for (int k(0);k<=A.m;k++) if (A.II[k] != B.II[k]) return false;
    return true;
}}}
```

```
bool operator!=(matricecreusedeux const & A, matricecreusedeux const & B) {{{ // --- Différence de matrices creuses deux
    return !(A==B);
}}}

/*****************************************************************************************************************
 *                                                 Multiplications                                               *
 *****************************************************************************************************************/

vecteur operator*(matricepleine const & M, vecteur const & v) {{{ // ---- Multiplication de matrice pleine et de vecteur
    assert(M.m == v.n);
    vecteur w(M.n);
    for(int i(0);i<M.n;i++) for(int j(0);j<M.m;j++) w.coef[i] += v.coef[j]*M.coef[i][j];
    return w;
}}}

vecteur operator*(matricecreuseun const & M, vecteur const & v) {{{ // Multiplication de matrice creuse un et de vecteur
    assert(M.m == v.n);
    vecteur w(M.n);
    for(int k(0);k<M.nz;k++) w.coef[M.i[k]] += M.coef[k]*v.coef[M.j[k]];
    return w;
}}}

// ------------------------------------------------------------- Multiplication de matrice creuse deux et de vecteur
vecteur operator*(matricecreusedeux const & M, vecteur const & v) {{{
    assert(M.m == v.n);
    vecteur w(M.n);
    int a(0);
    for(int b(0);b<M.n;b++) while(a<M.II[b]) {
        w.coef[b] += M.vals[a]*v.coef[M.j[a]];
        a++;
    }
    return w;
}}}

/*****************************************************************************************************************
 *                                                  Conversions                                                  *
 *****************************************************************************************************************/

matricecreuseun matricepleine::versun() const {{{ // ----------------------- Conversion de matrice pleine vers creuse un
    matricecreuseun A(m, n, nz);
    int cmpt(0);
    for(int i(0);i<m;i++) {
        for(int j(0);j<n;j++) {
            if (!coef[i][j].isnull()) {
                A.i[cmpt] = i;
                A.j[cmpt] = j;
                A.coef[cmpt++] = coef[i][j];
            }
        }
    }
    return A;
}}}

matricecreusedeux matricepleine::versdeux() const {{{ // ----------------- Conversion de matrice pleine vers creuse deux
    matricecreusedeux A(m, n, nz);
    int cmptm(0);
    int cmptnz(0);
    bool yadejaqqchsurlaligne;
    for (int i(0);i<m;i++) {
        yadejaqqchsurlaligne = false;
        for(int j(0);j<n;j++) {
            if(!coef[i][j].isnull()) {
                A.vals[cmptnz] = coef[i][j];
                A.j[cmptnz++] = j;
                if (!yadejaqqchsurlaligne) {
                    yadejaqqchsurlaligne = true;
                    A.II[cmptm++] = cmptnz;
                }
            }
        }
        if (!yadejaqqchsurlaligne) A.II[cmptm++] = 0;
    }
    A.II[cmptm] = A.II[0]+cmptnz;
    return A;
}}}

matricecreusedeux matricecreuseun::versdeux() const {{{ // ------------ Conversion de matrice creuse un vers creuse deux
    matricecreusedeux A(m, n, nz);
    int cmpt(-1);
    for(int a(0);a<nz;a++) {
        A.vals[a] = coef[a];
        A.j[a] = j[a];
```

```
        if (i[a] != cmpt) {
            if (i[a] == cmpt+1) A.II[++cmpt] = a+1;
            else A.II[++cmpt] = 0;
        }
    }
    A.II[++cmpt] = A.II[0]+A.nz;
    return A;
}}}

/*****************************************************************************************************************
 *                                                  Ordonnage                                                    *
 *****************************************************************************************************************/

matricecreuseun matricecreuseun ::ordonne() const {{{ // ---------------------------------------- Fonction d'ordonnage
    matricecreuseun A(m, n, nz);
    int cmpt(0);
    for(int a(0);a<m;a++) {
        for(int b(0);b<n;b++) {
            for(int c(0);c<nz;c++) {
                if(i[c] == a && j[c] == b) {
                    A.i[cmpt] = i[c];
                    A.j[cmpt] = j[c];
                    A.coef[cmpt] = coef[c];
                    cmpt++;
                }
            }
        }
    } // TODO euh... y'a pas moyen de faire ça mieux? N*N*nz opérations :s
    return A;
}}}

bool matricecreuseun ::estenbordel() const{{{ // ------------------------------ Vérification de la nécéssité d'ordonnage
    int a(-1);
    int b(-1);
    for(int c(0);c<nz;c++) {
        if (a < i[c] || ( a == i[c] && b < j[c] )) {
            a = i[c];
            b = j[c];
        }
        else return true;
    }
    return false;
}}}

/*****************************************************************************************************************
 *                                                   Fichiers                                                    *
 *****************************************************************************************************************/

matricecreuseun lireun(string const & file, bool const & comp) {{{ // --------------------- Lecture de matrice creuse un
    FILE * t;
    t = fopen(file.c_str(),"r");
    assert(t!=NULL);
    int m, n, nz;
    complexe coef;
    fscanf(t, "%d %d %d", &m, &n, &nz);
    matricecreuseun M(m, n, nz);
    for(int k(0); k<nz; k++) {
        if(comp) fscanf(t, "%d %d %lf %lf", &M.i[k], &M.j[k], &M.coef[k].re, &M.coef[k].im);
        else fscanf(t, "%d %d %lf", &M.i[k], &M.j[k], &M.coef[k].re);
    }
    fclose(t);
    return M;
}}}

int matricecreuseun ::ecrire(string const & file) const {{{ // --------------------------- Écriture de matrice creuse un
    FILE * t; // TODO Faudrait sécuriser ça
    t = fopen(file.c_str(),"w");
    fprintf(t, "%d %d %d\n", m, n, nz);
    for(int k(0); k<nz; k++) fprintf(t, "%d %d %lf %lf\n", i[k], j[k], coef[k].re, coef[k].im);
    fclose(t);
    return 0;
}}}

matricecreusedeux liredeux(string const & file, bool const & comp) {{{ // --------------- Lecture de matrice creuse deux
    FILE * t;
    t = fopen(file.c_str(),"r");
    assert(t!=NULL);
    int m, n, nz;
    complexe vals;
    fscanf(t, "%d %d %d", &m, &n, &nz);
    matricecreusedeux M(m, n, nz);
    for(int k(0); k<=m; k++) fscanf(t, "%d", &M.II[k]);
    for(int k(0); k<nz; k++) {
```

```
        if(comp) fscanf(t, "%d %lf %lf", &M.j[k], &M.vals[k].re, &M.vals[k].im);
        else fscanf(t, "%d %lf", &M.j[k], &M.vals[k].re);
    }
    fclose(t);
    return M;
}}}

int matricecreusedeux ::ecrire(string const & file) const {{{ // ------------------------ Écriture de matrice creusedeux
    FILE * t; // TODO Faudrait sécuriser ça
    t = fopen(file.c_str(),"w");
    fprintf(t, "%d %d %d\n", m, n, nz);
    for(int k(0); k<=m; k++) fprintf(t, "%d\n", II[k]);
    for(int k(0); k<nz; k++) fprintf(t, "%d %lf %lf\n", j[k], vals[k].re, vals[k].im);
    fclose(t);
    return 0;
}}}

string demandernomdufichier() {{{ // ------------------------------------------------------------ Choix du nom du fichier
    string a, b, c, d, e, f;
    char rep;
    a = "../test.mx";
    b = "../A398.mx";
    c = "../A62.mx";
    d = "../B398.mx";
    e = "../B62.mx";
    f = "En entrer un autre";
    cout << "Quel fichier voulez vous lire?" << endl;
    cout << "a) " << a << endl;
    cout << "b) " << b << endl;
    cout << "c) " << c << endl;
    cout << "d) " << d << endl;
    cout << "e) " << e << endl;
    cout << "f) " << e << endl;
    cout << "==> ";
    cin >> rep;
    string choix; // TODO l'idée était de faire un * choix...
    switch (rep) {
        case 'a' :
            choix = a;
            break;
        case 'b' :
            choix = b;
            break;
        case 'c' :
            choix = c;
            break;
        case 'd' :
            choix = d;
            break;
        case 'e' :
            choix = d;
            break;
        case 'f' :
            cin >> choix;
            break;
        default  :
            assert(false);
            break;
    }
    return choix;
}}}

matricecreuseun lireun(bool const & comp) {{{ // ----------------------- Choix du fichier pour Lecture matrice creuse un
    return lireun(demandernomdufichier(), comp);
}}}

int matricecreuseun ::ecrire() const {{{ // ---------------------------- Choix du fichier pour Écriture matrice creuse un
    string nom;
    cout << "Comment voulez-vous appeler le fichier?" << endl << "==> ";
    cin >> nom;
    return ecrire(nom);
}}}

matricecreusedeux liredeux(bool const & comp) {{{ // ----------------- Choix du fichier pour Lecture matrice creuse deux
    return liredeux(demandernomdufichier(), comp);
}}}

int matricecreusedeux ::ecrire() const {{{ // ------------------------ Choix du fichier pour Écriture matrice creuse deux
    string nom;
    cout << "Comment voulez-vous appeler le fichier?" << endl << "==> ";
    cin >> nom;
    return ecrire(nom);
}}}
```

```
/*****************************************************************************************************************
 *                                                Opérations vecteur-nombre                                      *
 *****************************************************************************************************************/

double vecteur ::norme() const {{{  // ------------------------------------------------------------ Norme d'un vecteur
    double m(0);
        for (int i(0); i<n; i++) m += pow(coef[i].norme(),2);
    return sqrt(m);
}}}

vecteur operator/(vecteur v, double const & x) {{{ // ------------------------------- Division d'un vecteur par un réel
        for (int i(0); i<v.n; i++) v.coef[i] /= x;
    return v;
}}}

vecteur operator*(vecteur v, double const & x) {{{ // ------------------------- Multiplication d'un vecteur et d'un réel
        for (int i(0); i<v.n; i++) v.coef[i] *= x;
    return v;
}}}

vecteur operator *(vecteur v, complexe const & a){{{ // ------------------- Multiplication d'un vecteur et d'un complexe
    for(int i(0); i<v.n; i++) v.coef[i] *= a;
    return v;
}}}

// vim : set foldmethod=marker :
```

C tests.h

```
#ifndef TESTS_H_INCLUDED
#define TESTS_H_INCLUDED

int test_conversions(bool const & afficher);
int test_produits(bool const & afficher);
int test_ordonnage(bool const & afficher);
int test_fichiers(bool const & afficher);
int test_historique(bool const & afficher);

#endif
```

D tests.cpp

```
#include <iostream>
#include <cstdio>
#include <string>
#include "matrices.h"
#include "historique.h"

using namespace std;

int test_conversions(bool const & afficher) {{{
    cout << "\tTest conversions entre matrices" << endl;
    matricepleine B(5, 5, 12);
    B.coef[0][0].re = 1.1;
    B.coef[0][3].re = 4;
    B.coef[1][0].re = 5;
    B.coef[1][1].re = 2.2;
    B.coef[1][3].re = 7;
    B.coef[2][0].re = 6;
    B.coef[2][2].re = 3.3;
    B.coef[2][3].re = 8;
    B.coef[2][4].re = 9;
    B.coef[3][2].re = 11;
    B.coef[3][3].re = 10.1;
    B.coef[4][4].re = 12.7;
    if (afficher) B.afficher();
    if (afficher) cout << endl;

    matricecreuseun C(B.versun());
    if (afficher) C.afficher();
    if (afficher) cout << endl;

    matricecreusedeux D(B.versdeux());
    if (afficher) D.afficher();
    if (afficher) cout << endl;

    matricecreusedeux E(C.versdeux());
```

```
    if (afficher) E.afficher();

    if (D==E) return 0;
    return 1;
}}}

int test_produits(bool const & afficher) {{{
    cout << "\tTest produit pleine*vecteur" << endl;
    matricepleine A(5, 5, 4);
    for (int i(0);i<A.m;i++) A.coef[i][i].re = i;
    if (afficher) A.afficher();
    if (afficher) cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    vecteur v(5);
    for (int i(0);i<5;i++) v.coef[i].re = i;
    if (afficher) v.afficher();
    if (afficher) cout << "========================================================================" << endl;
    vecteur x(A*v);
    if (afficher) x.afficher();

    cout << "\tTest produit creuseun*vecteur" << endl;
    matricecreuseun E(A.versun());
    if (afficher) E.afficher();
    if (afficher) cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    if (afficher) v.afficher();
    if (afficher) cout << "========================================================================" << endl;
    vecteur y(E*v);
    if (afficher) y.afficher();

    cout << "\tTest produit creusedeux*vecteur" << endl;
    matricecreusedeux F(A.versdeux());
    if (afficher) F.afficher();
    if (afficher) cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    if (afficher) v.afficher();
    if (afficher) cout << "========================================================================" << endl;
    vecteur z(F*v);
    if (afficher) z.afficher();

    if (x==y && x==z && !(y!=z)) return 0;
    return 1;
}}}

int test_ordonnage(bool const & afficher) {{{
    cout << "\tTest ordonnage" << endl;
    matricecreuseun A(5, 5, 4);
    for (int i(0);i<4;i++) {
        A.i[i] = i+1;
        A.j[i] = i+1;
        A.coef[i].re = i+1;
    }
    if (afficher) A.afficher();
    if (A.estenbordel()) {
        if (afficher) cout << "A est en bordel..." << endl;
        return 1;
    }
    else if (afficher) cout << "A est en ordre" << endl;

    matricecreuseun B(5, 5, 2);
    B.i[0] = 4;
    B.j[0] = 2;
    B.coef[0].re = 42;
    B.i[1] = 2;
    B.j[1] = 4;
    B.coef[1].re = 24;
    if (afficher) B.afficher();
    if (!B.estenbordel()) {
        if (afficher) cout << "B est en ordre..." << endl;
        return 1;
    }
    else if (afficher) cout << "B est en bordel" << endl;

    matricecreuseun C(B.ordonne());
    if (afficher) C.afficher();
    if (C.estenbordel()) {
        if (afficher) cout << "B est en bordel..." << endl;
        return 1;
    }
    else if (afficher) cout << "B est en ordre" << endl;

    return 0;
}}}

int test_fichiers(bool const & afficher) {{{
    matricecreuseun A(lireun("../test.mx", false));
```

```
    if (afficher) {
        cout << endl << "\tA" << endl;
        A.afficher();
    }
    matricecreuseun B(A.ordonne());
    if (afficher) {
        cout << endl << "\tB" << endl;
        B.afficher();
    }
    B.ecrire("B");
    matricecreusedeux C(B.versdeux());
    if (afficher) {
        cout << endl << "\tC" << endl;
        C.afficher();
    }
    C.ecrire("C");
    matricecreusedeux D(liredeux("C", true));
    if (afficher) {
        cout << endl << "\tD" << endl;
        D.afficher();
    }
    if (D == C) return 0;
    return 1;
}}}

int test_historique(bool const & afficher) {{{
    if (afficher) cout << "TODO" << endl;
    return 0;
}}}

// vim : set foldmethod=marker :
```

E algo.h

```
#ifndef ALGO_H_INCLUDED
#define ALGO_H_INCLUDED

#include "matrices.h"

int algo(matricecreusedeux const & A, bool const & afficher);

#endif
```

F algo.cpp

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cmath>
#include <assert.h>
#include "matrices.h"
#include "historique.h"

using namespace std;

int algo(matricecreusedeux const & A, bool const & afficher) {{{
    cout << "\tDébut de l'algorithme" << endl;
    if (afficher) {
        cout << "Matrice initiale : " << endl;
        A.afficher();
    }
        int i,k(0);
    complexe lambda1(1), temp(2);
    double norme_x;
    vecteur q(A.m), x(A.m), u1(A.m);
    double m(A.m);
    double val(sqrt(1/m));
    for (int i(0); i<A.m; i++) q.coef[i].re = val;
    if (afficher) {
        cout << "Vecteur initial : " << endl;
        q.afficher();
    }

    // L'algorithme s'arrête si la solution a été trouvée ou si le calcul est trop long
    while ((temp-lambda1).norme() > 0.001 && k<2000 ) {
        for (i=0;i<20;i++) { // On fait 20 itérations avant de revérifier le convergence
            x = A*q;
            norme_x = x.norme();
```

```
            q = x/norme_x;
        }
        k += 20;

        if (afficher) {
            cout << "x : " << endl;
            x.afficher();
            cout << "q : " << endl;
            q.afficher();
        }

        if (norme_x>0.000001) { // On vérifie que norme(x) ne devient pas nulle pour ne pas diviser par 0
            x = A*q;
            temp = lambda1;
            // On cherche une composante de x exploitable au sens de l'algorithme
            for (i=0;i<q.n;i++) if (!q.coef[i].isnull()) lambda1=x.coef[i]/q.coef[i];
            u1=q*pow(lambda1/lambda1.norme(),k);
        }
        else {
            printf("Il n'y a pas de valeur propre...\n");
            k=2000;
            return 1;
        }
    }
    if (k < 2000) {
        printf("La plus grande valeur propre est :");
        lambda1.afficher();
        printf("\nUn vecteur propre associe est : ");
        u1.afficher();
    }
    else {
        printf("Il n'y a pas de valeur propre! \n");
        return 1;
    }
    return 0;
}}}
```

G historique.h

```
#ifndef HISTORIQUE_H_INCLUDED
#define HISTORIQUE_H_INCLUDED

#include "matrices.h"

class element {
    public :
        int k;
        double qualite, valeur_propre;
        vecteur vecteur_propre;
        element * next;
        void ajouter_dans_historique() const;
        element mettre_fin(element * liste_convergence);
};

#endif
```

H historique.cpp

```
#include <cstdio>
#include <iostream>
#include <assert.h>
#include "matrices.h"
#include "historique.h"

element element ::mettre_fin(element * liste_convergence) {{{
    next = NULL;
    if (liste_convergence == NULL) return *this;
    else {
        element * temp = liste_convergence;
        while (temp->next != NULL) temp = temp->next;
        temp->next = this;
        return *liste_convergence;
    }
}}}

void element ::ajouter_dans_historique() const {{{
    FILE * fichier = NULL;
    fichier = fopen("historique.txt", "a");
```

```
    assert(fichier != NULL);
    fprintf(fichier, "Iteration %d\n", k);
    fprintf(fichier, "Qualite %f\n", qualite);
    fprintf(fichier, "Lambda %f\n", valeur_propre);
    fprintf(fichier, "Vecteur propre [ ");
    for (int i(0);i<vecteur_propre.n;i++) fprintf(fichier, "%5.3e+i%-5.3e ", vecteur_propre.coef[i].re, vecteur_propre.coef[i].im);
    fprintf(fichier, "]\n\n");
    fclose(fichier);
}}}

// vim : set foldmethod=marker :
```

I main.cpp

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cmath>
#include <assert.h>
#include "matrices.h"
#include "historique.h"
#include "tests.h"
#include "algo.h"

using namespace std;

int main() {
    cout << "\t\tMini Projet" << endl;
    bool afficher = true;
    int a(test_conversions(afficher));
    int b(test_produits(afficher));
    int c(test_ordonnage(afficher));
    int d(test_fichiers(afficher));
    int e(test_historique(afficher));

    int f(algo(lireun("../test.mx", false).versdeux(), afficher));

    if ( a == 0 && b==0 && c==0 && d == 0 && e == 0 && f == 0 ) {
        cout << " OK " << endl;
        return 0;
    }
    else {
        cout << " KO : a=" << a << " | b=" << b << " | c=" << c << " | d=" << d << " | e=" << e << " | f=" << f << endl;
        return 1;
    }
}
```