Corrigé du Contrôle d'Informatique INF 311

Sujet proposé par P. Chassignet et F. Nielsen

6 juillet 2009

Seuls les documents fournis dans le cadre du cours et les notes personnelles sont autorisés.
Durée : 2 heures. Les étudiants EV2 ont le droit à 30 minutes supplémentaires.

Les trois questions de l'exercice 1 sont indépendantes. L'exercice 3 utilise les fonctions écrites à l'exercice 2. L'exercice 4 est indépendant. Un barème pour chaque exercice, pour un total de 23 points, est donné à titre indicatif. Nous donnons le nombre de lignes attendues pour la réponse à chaque question ; si la réponse que vous envisagez est beaucoup plus longue, il est probable qu'il s'agisse d'une méthode trop compliquée. Les questions les plus difficiles sont marquées par **.

On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction.

Exercice 1. Analyse de programmes

barème envisagé : 3 points

(1a) [1 point] On veut calculer la moyenne de n nombres $x_1, ..., x_n$ stockés dans un tableau de réels comme $\bar{x} = \sum_{i=1}^{n} \frac{1}{n} * x_i$. Le code Java ci-dessous est incorrect.

```
public class Exo1a {
  public static double moyenneBug(double[] x) {
    if (x == null || x.length == 0)
      return 0.0;
    double moy = 0.0;
    double weight = 1 / x.length;
    for (int i = 0; i < x.length; i++)
      moy = moy + weight * x[i];
    return moy;
  }

  public static void main(String[] args) {
    double[] t = { 0.0, 1.0, 2.0, 3.0, 4.0 };
    System.out.println(moyenneBug(t));
  }
}
```

Dites ce qu'il affiche, expliquez pourquoi et corrigez ce code le plus simplement possible.
(réponse attendue : quelques lignes)

Solution: Ce programme se compile sans erreur, mais la division `1 / x.length` prend 2 opérandes entiers et il s'agit donc de la division euclidienne. Pour `x.length` > 1 le résultat de la division vaut alors 0 et la fonction retourne finalement 0 (qui sera affiché sous la forme `0.0`).

La correction la plus simple consiste à écrire `double weight = 1.0 / x.length;` ce qui force la division dans le type `double` et produit le résultat attendu (le programme affiche alors `2.0`).

(1b) [1 point] Pour chacune des deux fonctions suivantes, justifiez si elle termine ou non et, si elle termine, dites ce qu'elle retourne.

```
public static int f(int n) {
  if (n >= 0)
    return 1;
  else
    return f(n + 1);
}

public static int g(int n) {
  if (n <= 1)
    return 0;
  else
    return 1 + g(n - 2);
}
```

(réponse attendue : quelques lignes)

Solution: Les deux fonctions terminent quelque soit la valeur de `n`. La fonction `f` retourne toujours `1` . La fonction `g` retourne `0` pour `n` négatif et `n/2` (quotient entier, noté $\lfloor \frac{n}{2} \rfloor$) pour `n` positif.

(1c) [1 point] Dites ce qu'affiche le programme ci-dessous et expliquez pourquoi.

```
public class Exo1c {
  public static void afficher(int[] t) {
    System.out.print(t[0]);
    for (int i = 1; i < t.length; i++)
      System.out.print("␣" + t[i]);
    System.out.println();
  }

  public static void remplir(int[] u, int v) {
    for (int i = 0; i < u.length; i++)
      u[i] = v;
  }
```

```
  public static void remplir2(int[] t, int v) {
    t = new int[t.length];
    remplir(t, v);
  }

  public static void main(String[] args) {
    int[] t = new int[4];
    afficher(t);
    remplir(t, 2);
    afficher(t);
    remplir2(t, 1);
    afficher(t);
  }
}
```

(réponse attendue : une dizaine de lignes)

Solution: Le programme affiche :

```
0 0 0 0
2 2 2 2
2 2 2 2
```

Pour la première ligne, le tableau est initialisé inplicitement avec des zéros. Pour la deuxième ligne, la référence `u` de la fonction `remplir` est la même référence que `t` de `main`. La fonction `remplir` range donc des `2` dans le tableau qui est affiché. La fonction `remplir2` fait mettre des `1` dans un nouveau tableau qui est référencé par son paramètre `t`. Mais cela affecte une copie de la référence qui est passée par `main` et la référence `t` de `main` reste inchangée.

Exercice 2. Espace affine euclidien

barème envisagé : 4 points

Dans cet exercice, on va modéliser un espace de dimension d dont les points et les vecteurs sont représentés comme des tableaux de type `double[]` et de taille d.

Dans les questions qui suivent, on n'a pas à connaître d et les fonctions demandées doivent fonctionner pour des tableaux de taille quelconque. On pourra cependant supposer que lorsqu'une fonction prend plusieurs tableaux en paramètre, alors ceux-ci sont de tailles compatibles.

Toutes les fonctions demandées pour cet exercice seront placées dans une même classe nommée `Espace`.

(2a) [½ point] Écrivez une fonction statique `double[] copie(double[] t)` qui retourne une copie du tableau `t`.

(réponse attendue : quelques lignes)

Solution: voir ci-dessous

(2b) [½ point] Écrivez une fonction statique `double[] vecteur(double[] p1, double[] p2)` qui prend comme arguments deux points p_1 et p_2 et qui retourne un **nouveau** tableau contenant les coordonnées du vecteur de p_1 à p_2 (qui se calcule comme $\vec{v} = p_2 - p_1$).
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

(2c) [1 point] Écrivez une fonction statique `double distance(double[] p1, double[] p2)` qui prend comme arguments deux points p_1 et p_2 et qui retourne la distance euclidienne $\|p_2 - p_1\|$ entre ces deux points.
Vous **devez** utiliser la fonction `vecteur` de la question précédente. Pour calculer un carré, vous utiliserez la fonction `static double sqr(double x) { return x*x; }` qui sera placée aussi dans la classe `Espace`. Enfin, pour calculer la racine carrée, vous utiliserez la fonction `static double sqrt(double x)` de la classe `Math`.
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

(2d) [½ point] Écrivez une fonction statique `double[] scalaire(double k, double[] v)` qui multiplie les coordonnées du vecteur $\vec{v}$ par le nombre k, autrement dit, $\vec{v}$ devient $k.\vec{v}$. Pour un usage plus commode, la fonction retourne la référence du tableau argument `v` ainsi modifié, mais sans recréer de tableau.
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

(2e) [½ point] Écrivez une fonction statique `void translater(double[] p, double[] v)` qui prend comme arguments un point p et un vecteur $\vec{v}$ et modifie p qui devient $p + \vec{v}$.
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

(2f) [1 point] On considère maintenant un ensemble non vide de points dont on veut calculer l'isobarycentre (le centre de masse). Écrivez une fonction statique `double[] barycentre(double[][] pts)` qui prend comme arguments un tableau de points (`pts[i][j]` est la j^{e} coordonnée du i^{e} point) et qui retourne un tableau contenant les coordonnées de leur barycentre. Vous **devez** utiliser les fonctions `scalaire` et `translater` des questions précédentes.
(réponse attendue : quelques lignes)

Solution:

```
// une classe pour manipuler des points et des vecteurs
// sous forme de tableaux
public class Espace {
```

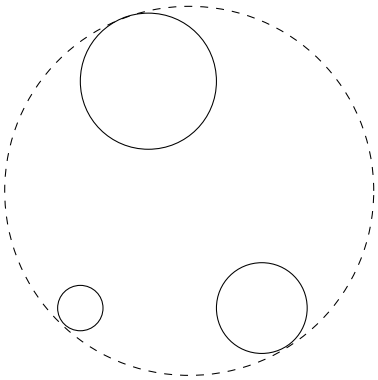
```
// fonction qui fait une copie du tableau t
static double[] copie(double[] t) {
  double[] tt = new double[t.length];
  for (int i = 0; i < t.length; ++i)
    tt[i] = t[i];
  return tt;
}

// fonction qui retourne le vecteur entre les points
// p1 et p2
static double[] vecteur(double[] p1, double[] p2) {
  double[] v = new double[p1.length];
  for (int i = 0; i < v.length; ++i)
    v[i] = p2[i] - p1[i];
  return v;
}

static double sqr(double x) {
  return x * x;
}

// la fonction qui calcule la distance entre les points
// p1 et p2, en utilisant les fonctions vecteur et sqr
static double distance(double[] p1, double[] p2) {
  double[] v = vecteur(p1, p2); // un seul appel
  double d = 0.0;
  for (int i = 0; i < v.length; ++i)
    d += sqr(v[i]);
  return Math.sqrt(d);
}

// la fonction qui multiplie chaque valeur du vecteur v
// par x et retourne le tableau v
static double[] scalaire(double k, double[] v) {
  for (int i = 0; i < v.length; ++i)
    v[i] *= k;
  return v;
}

// la fonction qui translate le point p du vecteur v
static void translater(double[] p, double[] v) {
  for (int i = 0; i < p.length; ++i)
    p[i] += v[i];
}
```

```
  // la fonction qui construit le barycentre d'un tableau
  // de points
  static double[] barycentre(double[][] pts) {
    double[] v = new double[pts[0].length];
    // v contient implicitement des 0
    for (int i = 0; i < pts.length; ++i)
      translater(v, pts[i]);
    return scalaire(1.0 / pts.length, v);
  }

  // une variante, robuste aux cas de l'ensemble vide
  static double[] barycentreBis(double[][] pts) {
    if (pts == null || pts.length == 0)
      return null;
    double[] iso = copie(pts[0]); // copie indispensable
    for (int i = 1; i < pts.length; i++)
      translater(iso, pts[i]);
    return scalaire(1.0 / pts.length, iso);
  }

} // fin de la classe
```

Exercice 3. Les boules

barème envisagé : 7 points

Dans cet exercice, on va considérer des boules de l'espace affine euclidien qui seront définies chacune par un centre et un rayon. La boule de centre c et de rayon r est l'ensemble des points x de l'espace qui vérifient $\|x - p\| \leq r$.
Le but est de calculer la plus petite boule englobante d'un ensemble de boules donné. Il s'agit de la boule de plus petit rayon dans laquelle sont incluses toutes les boules considérées, voir la figure ci-dessous :



Pour cet exercice, vous **devez** utiliser au mieux les fonctions de la classe `Espace` de l'exercice précédent et de la classe `Boule` définie ci-dessous pour ne pas réécrire du code déjà écrit.

(3a) [½ point] Écrivez une classe `Boule` qui permet de définir des boules avec deux champs d'objet :
- `centre` de type `double[]` qui donne les coordonnées du centre de cette boule, et
- `rayon` de type `double` qui est le rayon de cette boule.

Munissez cette classe d'un constructeur `Boule(double[] p, double rr)` qui initialise un objet de ce type.
(réponse attendue : une dizaine de lignes)

Solution: voir ci-dessous

(3b) [½ point] Écrivez une fonction statique `double maxDistance(Boule b, double[] p)` qui retourne la distance maximale entre tout point de la boule `b` et le point de coordonnées `p`.
Cette fonction sera dans la classe `Boule`.
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

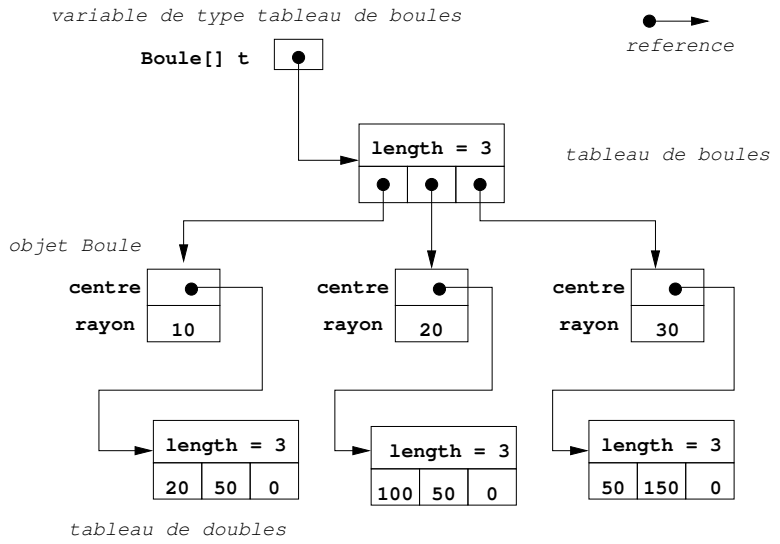
On considère maintenant un ensemble $\mathcal{B}$ de boules, supposé non vide et qui sera décrit comme un tableau `Boule[] boules`.

(3c) [½ point] Écrivez la fonction `main` de la classe `Boule` qui va construire un exemple 3D formé d'un tableau de 3 boules, respectivement : une boule de centre $(20, 50, 0)$ et de rayon 10, une boule de centre $(100, 50, 0)$ et de rayon 20 et une boule de centre $(50, 150, 0)$ et de rayon 30.
(réponse attendue : une dizaine de lignes)

Solution: voir ci-dessous

(3d) [½ point] Dessinez ensuite, avec des boîtes et des flèches, une représentation de la mémoire qui est utilisée pour cet exemple.

Solution:



(3e) [1 point] Écrivez une fonction statique
`Boule bouleMaxDistance(Boule[] boules, double[] p)` qui retourne la référence d'une boule, prise parmi celles qui sont référencées dans le tableau `boules` et dont la distance maximale avec le point de coordonnées `p` est maximale.
Cette fonction sera dans la classe `Boule`.
(réponse attendue : une dizaine de lignes)

Solution: voir ci-dessous

(3f) [1 point] Écrivez une fonction statique
`double[] pointMaxDistance(Boule b, double[] p)` qui retourne les coordonnées d'un point de la boule référencée par `b` qui réalise la distance maximale avec le point de coordonnées `p`.
Cette fonction sera dans la classe `Boule`.
Indications : on supposera que le point p est différent du centre c de la boule et on prendra le point q adéquat sur la droite passant par p et c. En utilisant les fonctions de la classe `Espace`, cette fonction s'écrit sans aucune boucle apparente.
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

(3g) [1 point] Écrivez une fonction statique `double[] barycentre(Boule[] boules)` qui retourne les coordonnées de l'isobarycentre des centres des boules qui sont référencées dans le tableau `boules`.

Cette fonction sera dans la classe `Boule`.
Vous **devez** utiliser la fonction `barycentre` de la classe `Espace` et, pour économiser la mémoire, le tableau qui est passé à cette fonction **doit** partager les coordonnées des points déjà définis pour les boules.
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

Le calcul exact de la plus petite boule englobante est très difficile en grandes dimensions. Toutefois, un algorithme itératif très simple et découvert récemment permet d'approximer la plus petite boule englobante comme suit :

On initialise le centre C_1 comme l'isobarycentre des centres des boules données.

À l'itération $i+1$, on note b_i la boule qui maximise sa distance maximale au centre actuel C_i et p_i le point de b_i qui réalise ce maximum. On fait alors coulisser le centre dans la direction de p_i en posant :

$$C_{i+1} = C_i + \frac{1}{i+1}(p_i - C_i)$$

Si on note respectivement C_* et r_*, le centre et le rayon de la plus petite boule englobante, il a été démontré en 2003 que $\|C_k - C_*\| \leq \sqrt{\frac{1}{k}} r_*$ (et par conséquent le rayon r_k de la boule englobante de centre C_k est au plus $(1 + \sqrt{\frac{1}{k}}) r_*$).

(3h) [2 points] ** Écrivez une fonction statique
`Boule approximationBouleEnglobante(Boule[] boules, int k)` qui retourne une approximation de la plus petite boule englobante en itérant `k` fois la mise à jour du centre.
(réponse attendue : une dizaine de lignes)

Solution:

```
public class Boule {
  double[] centre;
  double rayon;

  // le constructeur
  Boule(double[] p, double r) {
    this.rayon = r;
    this.centre = p;
  }

  // la fonction qui calcule la distance maximale entre
  // un point de la boule b et le point p
  static double maxDistance(Boule b, double[] p) {
```

```
  return Espace.distance(b.centre, p) + b.rayon;
}

// la fonction qui retourne la boule qui maximise
// sa distance maximale avec p
public static Boule bouleMaxDistance(Boule[] boules,
    double[] p) {
  double dMax = 0.0;
  Boule bMax = null;
  for (int i = 0; i < boules.length; i++) {
    double d = maxDistance(boules[i], p);
    if (d > dMax) {
      bMax = boules[i];
      dMax = d;
    }
  }
  return bMax;
}

// la fonction qui retourne le point de la boule b
// qui est le plus e'loigne' du point p
public static double[] pointMaxDistance(Boule b,
    double[] p) {
  double[] pMax = Espace.copie(b.centre);
  double d = Espace.distance(p, b.centre);
  Espace.translater(pMax,
    Espace.scalaire(b.rayon / d,
      Espace.vecteur(p, b.centre)));
  return pMax;
}

// la fonction qui construit le barycentre des centres
// d'un ensemble de boules
public static double[] barycentre(Boule[] boules) {
  double[][] centres = new double[boules.length][];
  for (int i = 0; i < boules.length; i++) {
    centres[i] = boules[i].centre;
  }
  return Espace.barycentre(centres);
}

public static Boule approximationBouleEnglobante(
    Boule[] boules, int k) {
  if ( boules == null || boules.length == 0 )
```

```
      return null;
    double[] centre = barycentre(boules);
    for (int i = 1; i <= k; i++) {
      Boule b = bouleMaxDistance(boules, centre);
      double[] p = pointMaxDistance(b, centre);
      Espace.translater(centre,
        Espace.scalaire(1.0 / (i + 1),
          Espace.vecteur(centre, p)));
    }
    Boule b = bouleMaxDistance(boules, centre);
    double r = maxDistance(b, centre);
    return new Boule(centre, r);
  }

  public static void main(String[] args) {
    Boule[] boules = new Boule[3];
    double[] p0 = { 20.0, 50.0, 0.0 };
    boules[0] = new Boule(p0, 10.0);
    double[] p1 = { 100.0, 50.0, 0.0 };
    boules[1] = new Boule(p1, 20.0);
    double[] p2 = { 50.0, 150.0, 0.0 };
    boules[2] = new Boule(p2, 30.0);
  }

} // fin de la classe
```

Exercice 4. Codage par des listes

barème envisagé : 9 points

On considère un espace affine euclidien de très grande dimension d. On va se placer dans le cas un peu particulier où, pour chaque point ou vecteur, il y a très peu de coordonnées dont la valeur est non nulle. Il est alors intéressant de coder les points et les vecteurs comme des listes qui ne donnent que les coordonnées non nulles.
On utilisera la structure de données suivante :

```
public class ListeCoord {
  int i;
  double v;
  ListeCoord suivant;

  // le constructeur
  ListeCoord(int i0, double v0, ListeCoord s0){
    this.i = i0;
    this.v = v0;
    this.suivant = s0;
  }
}
```

Le champ `i` donne le numéro d'une coordonnée (de 0 à $d-1$). Le champ `v` donne la valeur de la coordonnée associée. Les éléments d'une liste seront dans l'ordre des `i` décroissants.

Par exemple, dans un espace de dimension 7, le point et le vecteur de coordonnées $(0, 0, 0, 1, 2, 0, 3)$ seront représentés par la liste :
$(i = 6, v = 3) \longrightarrow (i = 4, v = 2) \longrightarrow (i = 3, v = 1)$.
Notez aussi que, quelque soit la dimension d, l'origine et le vecteur nul seront toujours représentés par la liste vide. La liste vide est donc un argument ou un résultat valide pour toutes les fonctions demandées ci-dessous.

Pour les questions qui suivent, les fonctions demandées seront toutes placées dans une même classe, dont le nom importe peu.

(4a) [½ point] Écrivez une fonction statique `ListeCoord listeDe(double[] p)` **itérative** qui retourne la liste qui correspond au tableau de coordonnées `p`.
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

(4b) [½ point] Écrivez une fonction statique `double carre(ListeCoord lc)` **itérative** qui retourne le module au carré $\|v\|^2$ pour le vecteur v qui est codé par la liste `lc`.
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

(4c) [½ point] Écrivez une fonction statique `double coordonnee(ListeCoord lc, int i)` **récursive** qui retourne la valeur de la `i`$^{\text{e}}$ coordonnée du point ou vecteur qui est codé par la liste `lc`.
(réponse attendue : quelques lignes)

Solution: voir ci-dessous

(4d) [1 point] Écrivez une fonction statique `ListeCoord annuler(ListeCoord lc, int i)` **récursive** qui modifie la liste donnée par `lc` pour exprimer que la `i`$^{\text{e}}$ coordonnée est mise à la valeur zéro. Cette fonction ne doit pas faire de `new`.
(réponse attendue : une dizaine de lignes)

Solution: voir ci-dessous

(4e) [1 point] Écrivez une fonction statique
`ListeCoord affecter(ListeCoord lc, int i, double v)` **récursive** qui modifie la liste donnée par `lc` pour exprimer que la `i`$^{\text{e}}$ coordonnée prend la valeur `v` (on supposera que $v \neq 0$). Cette fonction doit faire un et un seul `new` dans le cas où la `i`$^{\text{e}}$ coordonnée était précédemment nulle et ne doit pas faire de `new` dans le cas contraire.
(réponse attendue : une dizaine de lignes)

Solution: voir ci-dessous

(4f) [2 points] Écrivez une fonction statique
`double produitScalaire(ListeCoord lc1, ListeCoord lc2)` itérative ou récursive qui retourne le produit scalaire des deux vecteurs donnés par `lc1` et `lc2`. On rappelle que le produit scalaire de deux vecteurs u et v est défini par $< u, v >= \sum_{i=0}^{d-1} u_i.v_i$.
(réponse attendue : une dizaine de lignes)

Solution: voir ci-dessous

(4g) [3 points] ** Écrivez une fonction statique
`ListeCoord difference(ListeCoord lc1, ListeCoord lc2)` **récursive** qui retourne une **nouvelle** liste correspondant à la différence ($lc1 - lc2$) des deux points ou vecteurs donnés par `lc1` et `lc2` et qui ne partage aucune référence avec les listes arguments.
(réponse attendue : une vingtaine de lignes)

Solution: voir ci-dessous

(4h) [½ point] Le carré de la distance entre deux points p et q peut se calculer par $\|q-p\|^2$ ou par $\|p\|^2 + \|q\|^2 - 2 < p, q >$. Écrivez un test qui permet de vérifier, pour deux points de votre choix, que les deux expressions sont égales.
(réponse attendue : quelques lignes)

Solution:

```
public class EspaceListe {

  // construire la liste qui correspond au tableau
```

```
static ListeCoord listeDe(double[] p) {
  ListeCoord l = null;
  for (int i = 0; i < p.length; i++)
    if (p[i] != 0)
      l = new ListeCoord(i, p[i], l);
  return l;
}

// le module du point correspondant
static double carre(ListeCoord lc) {
  double sum = 0.0;
  while (lc != null) {
    sum += lc.v * lc.v;
    lc = lc.suivant;
  }
  return sum;
}

// retourne la valeur de la i-eme coordonnee
static double coordonnee(ListeCoord lc, int i) {
  if (lc == null || i > lc.i) // optimisation
    return 0;
  else if (i == lc.i)
    return lc.v;
  else
    return coordonnee(lc.suivant, i);
}

// annule la valeur de la i-eme coordonnee
static ListeCoord annuler(ListeCoord lc, int i) {
  if (lc == null)
    return null;
  else if (lc.i == i)
    return lc.suivant;
  else if (lc.i < i) // ce cas est seulement
    return lc;       // une optimisation
  else {
    lc.suivant = annuler(lc.suivant, i);
    return lc;
  }
}

// affecte la valeur v (non nulle) a la i-eme coordonnee
static ListeCoord affecter(ListeCoord lc, int i,
```

```
    double v) {
  if (lc == null || lc.i < i)
    return new ListeCoord(i, v, lc);
  else if (lc.i == i) {
    lc.v = v;
    return lc;
  } else {
    lc.suivant = affecter(lc.suivant, i, v);
    return lc;
  }
}

// retourne le produit scalaire
static double produitScalaire(ListeCoord lc1,
    ListeCoord lc2) {
  if (lc1 == null || lc2 == null)
    return 0.0;
  else if (lc1.i == lc2.i)
    return lc1.v * lc2.v
         + produitScalaire(lc1.suivant, lc2.suivant);
  else if (lc1.i < lc2.i)
    return produitScalaire(lc1, lc2.suivant);
  else
    return produitScalaire(lc1.suivant, lc2);
}

// retourne le produit scalaire
static double produitScalaireIteratif(ListeCoord lc1,
    ListeCoord lc2) {
  double res = 0.0;
  while (lc1 != null && lc2 != null) {
    if (lc1.i == lc2.i) {
      res += lc1.v * lc2.v;
      lc1 = lc1.suivant;
      lc2 = lc2.suivant;
    } else if (lc1.i < lc2.i)
      lc2 = lc2.suivant;
    else
      lc1 = lc1.suivant;
  }
  return res;
}

// retourne la difference correspondant a lc1 - lc2
```

```
  static ListeCoord difference(ListeCoord lc1,
      ListeCoord lc2) {
    if (lc1 == null && lc2 == null)
      return null;
    else if (lc1 == null)
      return new ListeCoord(lc2.i, -lc2.v,
        difference(lc1, lc2.suivant));
    else if (lc2 == null)
      return new ListeCoord(lc1.i, lc1.v,
        difference(lc1.suivant, lc2));
    else if (lc1.i == lc2.i) {
      double v = lc1.v - lc2.v;
      if (v == 0.0)
        return difference(lc1.suivant, lc2.suivant);
      else
        return new ListeCoord(lc1.i, v,
          difference(lc1.suivant, lc2.suivant));
    } else if (lc1.i < lc2.i)
      return new ListeCoord(lc2.i, -lc2.v,
        difference(lc1, lc2.suivant));
    else
      return new ListeCoord(lc1.i, lc1.v,
        difference(lc1.suivant, lc2));
  }

  public static void test() {
    double[] p1 = { 0.0, 0.0, 0.0, 1.0, 2.0, 3.0, 0.0 };
    double[] p2 = { 1.0, 0.0, 1.0, 0.0, 2.0, 0.0, 3.0 };
    ListeCoord l1 = listeDe(p1);
    ListeCoord l2 = listeDe(p2);
    double d = carre(difference(l1, l2));
    double dd = carre(l1) + carre(l2)
      - 2 * produitScalaire(l1, l2);
    System.out.println(d - dd);
  }

} // fin de la classe
```