Atelier E2 - Algèbre linéaire et vision 3D

TP2 : Vecteurs en dimension arbitraire

Romain Negrel*

Objectifs

Programmer une classe `Vecteur` pour représenter et manipuler les vecteurs dans l'espace euclidien de dimension n, quel que soit l'entier naturel n.

Liste des compétences à acquérir pendant le TP :

- déclarer un tableau ;
- allouer un tableau ;
- accéder aux éléments d'un tableau ;
- écrire une boucle 'for' simple pour parcourir les indices et les éléments d'un tableau ;
- écrire et programmer un algorithme (i.e., une suite d'instructions) pour :
  - calculer la norme d'un vecteur en dimension arbitraire ;
  - multiplier un vecteur par un scalaire en dimension arbitraire ;
  - normaliser un vecteur en dimension arbitraire ;
  - calculer la somme de deux vecteurs en dimension arbitraire ;
  - calculer le produit scalaire de deux vecteurs en dimension arbitraire ;
  - tester la colinéarité de deux vecteurs en dimension arbitraire ;
  - calculer le produit vectoriel de deux vecteurs en dimension 3 ;
  - tester l'orthogonalité de deux vecteurs en dimension 3 ;
  - tester la coplanarité de trois vecteurs en dimension 3 ;
- utiliser le code-pad sous BlueJ pour vérifier que l'action d'une méthode est conforme ;
- mettre en œuvre une suite de calculs pour constater les limitations des tests d'égalité entre nombres réels représentés avec le type `double`.

Sujet

1. **Travail préliminaire**

   1.1. Téléchargez le fichier '`ATL_2201_TP2.zip`' et enregistrez-le dans le répertoire '`ATL_2201`' créé lors du premier TP.

*d'après un sujet de Jean Cousty et Benjamin Perret

1.2. Sous BlueJ, ouvrez le fichier '`ATL_2201_TP2.zip`' : Menu "Projet" → "Open Project" et sélectionnez le fichier '`ATL_2201_TP2.zip`'.
Attention : pour le TP2, vous travaillez sur un nouveau projet BlueJ. Vous ne repartez pas du projet du TP1.

Important !

Après avoir répondu à une question, cliquez sur le bouton "`Exécuter les ...`" et vérifiez que l'indicateur de la question passe au vert. Si BlueJ indique une croix rouge pour la question que vous venez de résoudre, cela veut dire que votre solution n'est pas correcte ! Si l'indicateur passe au vert, cela veut dire que votre solution est peut-être correcte !

Important !!

Ajoutez des commentaires dans votre code !

2. Création de la classe `Vecteur`

Cette classe devra permettre d'instancier un vecteur en dimension arbitraire, c'est-à-dire, un vecteur défini dans un espace euclidien dont la dimension est choisie au moment de la création du vecteur. En termes de programmation, cela signifie que l'on passera au constructeur (dont la programmation sera abordée aux questions 2.3 et 2.4) de la classe un paramètre indiquant la dimension n de l'espace de définition du vecteur ; ce vecteur comprendra donc n coordonnées.

Aide

Dans ce TP, nous distinguerons procédures et fonctions :

- une fonction retourne une valeur,
- une procédure ne retourne rien et modifie directement l'instance courante.

2.1. Créez dans votre projet une nouvelle classe `Vecteur` (bouton 'New Class' sous BlueJ).
Inspectez le code de la classe `Vecteur` créée par BlueJ. Que contient-il ? Cela correspond-il à un vecteur en dimension arbitraire ? Supprimez les attributs et méthodes qui ne sont pas nécessaires. En utilisant uniquement les types de données vus dans cet atelier jusqu'à présent, est-il possible de déclarer les attributs correspondant à un vecteur en dimension arbitraire ? Pourquoi ?

2.2. Afin de pouvoir définir les attributs d'une classe pour représenter un vecteur en dimension arbitraire, il est possible d'utiliser un tableau. Par exemple, le vecteur dont les coordonnées sont $(0, 1.5, 2)^\top$ sera représenté grâce à un attribut tableau à 3 cases, et l'on retrouvera à l'indice 0 de ce tableau la valeur 0, à l'indice 1 la valeur 1.5 et à l'indice 2 la valeur 2. Après avoir lu cette ressource sur les tableaux, déclarez dans la classe `Vecteur` un attribut pour représenter un vecteur en dimension arbitraire.

Rappel

Les coordonnées des vecteurs sont des nombres réels, et le type `double` est utilisé pour représenter ces nombres réels.

2.3. Ajoutez à la classe `Vecteur` un constructeur à un paramètre de type `int` spécifiant la dimension de l'espace de définition du vecteur et créant une instance du vecteur nul dans cet espace. Attention, la dimension doit être au moins égale à 2, car certaines méthodes (comme `getX` et `getY`) supposent l'existence d'au moins deux coordonnées et servent à visualiser le vecteur dans un plan. Si la dimension demandée est inférieure à 2, vous devez la fixer automatiquement à 2. La signature de ce constructeur est donc :

```
public Vecteur(final int pn)
```

Aide

Pour initialiser les coordonnées du vecteur à zéro, vous pouvez utiliser la méthode `fill` disponible dans la classe `java.util.Arrays`.

2.4. Ajoutez à la classe `Vecteur` un constructeur de copie qui prend en paramètre une autre instance de la classe `Vecteur` et qui crée une copie de celle passée en paramètre. La signature de ce constructeur est donc :

```
public Vecteur(final Vecteur pVecteur)
```

Aide

Pour initialiser les coordonnées du vecteur à zéro, vous pouvez utiliser la méthode `copyOf` disponible dans la classe `java.util.Arrays`.

2.5. Ajoutez à la classe `Vecteur` un accesseur sans paramètre retournant la dimension de l'espace de définition du vecteur. La signature de cet accesseur est donc :

```
public int getDimension()
```

Attention !

Vérifiez la validité de l'indice avant d'accéder ou modifier une coordonnée. Pour l'instant, vous ne connaissez pas encore les exceptions : contentez-vous d'afficher un message d'erreur dans le terminal et soit ne rien faire, soit retourner une valeur par défaut.

2.6. Ajoutez à la classe `Vecteur` un accesseur à un paramètre de type `int` spécifiant l'indice i de la coordonnée à retourner. La signature de cet accesseur est donc :

```
public double getCoordonnee(final int pi)
```

2.7. Ajoutez à la classe `Vecteur` un modificateur à deux paramètres permettant d'affecter une valeur à l'une des coordonnées de l'instance courante de la classe `Vecteur`. Le premier paramètre, de type `int`, spécifie l'indice de la coordonnée à modifier et le second, de type `double`, spécifie la valeur à affecter à cette coordonnée. La signature de ce modificateur est donc :

```
public void setCoordonnee(final int pi, final double pCoordonnee)
```

2.8. Afin de tester votre classe, créez un vecteur en dimension 3 et un vecteur en dimension 4. Invoquez la méthode `getDimension` sur ces deux vecteurs et vérifiez qu'elle retourne

une valeur conforme à la définition. Modifiez ensuite ces deux vecteurs afin de leur affecter les coordonnées $(0, 1.5, 2)^\top$ et $(0, 0, 3, 1)^\top$ respectivement. Enfin, vérifiez grâce à la méthode `getCoordonnee` que les modifications effectuées sont conformes à ce qui est souhaité. Que se passe-t-il lorsque vous invoquez `getCoordonnee(4)` sur les deux vecteurs que vous avez créés ? Expliquez.

3. **Dessiner**

   3.1. Comme vous l'avez vu lors du TP précédent, la classe Plan permet de représenter des vecteurs de l'espace euclidien de dimension 2. Pour visualiser simplement un vecteur en dimension arbitraire n, nous allons visualiser le vecteur en dimension 2 formé des deux premières coordonnées du vecteur en dimension n. Cela revient en quelque sorte à se placer dans le plan défini par les vecteurs $(1, 0, 0, \cdots, 0)^\top$ et $(0, 1, 0, \cdots, 0)^\top$ et à diriger le regard selon une direction orthogonale à ce plan.

   Afin de pouvoir réaliser une telle visualisation avec la classe Plan, ajoutez à la classe `Vecteur` deux accesseurs :

   - nommés `getX` et `getY` ;
   - sans paramètre ; et
   - retournant respectivement la première et la deuxième coordonnée de l'instance courante de la classe `Vecteur`.

   3.2. Créez un `Vecteur` en dimension 3 et un `Vecteur` en dimension 4 et affectez à ces vecteurs les coordonnées $(0, 1.5, 2)^\top$ et $(1, 2, 3, 1)^\top$. Créez un Plan et représentez les deux vecteurs dans ce plan à l'aide de la méthode `dessinerVecteurEn2d` de ce Plan. Que constatez-vous si vous souhaitez dessiner dans ce plan le vecteur $(0, 1.5, 0)^\top$? Est-ce normal ?

Aide

Le code-pad de BlueJ pourra vous être utile. Pour cela, l'option 'show code-pad' du menu 'view' doit être sélectionnée. Le code-pad est un petit terminal grâce auquel il est possible d'exécuter pas-à-pas des instructions en Java. Essayez par exemple d'entrer dans le code pad l'instruction '`Vecteur u = new Vecteur(3);`' puis l'instruction '`Plan p = new Plan()`', et enfin l'instruction '`p.dessinerVecteurEn2d(u);`'.

Remarque

Dans la suite de l'atelier, on s'intéressera à des méthodes de visualisation permettant de simuler l'image obtenue en positionnant arbitrairement l'observateur dans l'espace euclidien à trois dimensions.

4. **Multiplication par un scalaire**

   4.1. Après avoir consulté les éléments d'aide présentés ci-après, ajoutez à la classe `Vecteur` une procédure publique `multiplicationScalaire` à un paramètre qui multiplie l'instance courante du `Vecteur` par le paramètre scalaire.

Aide

Pour pouvoir répéter un bloc d'instructions donné pour chaque coordonnée d'un vecteur, il est possible d'utiliser une structure de contrôle 'boucle for'. Avant de commencer la programmation de la méthode norme, lisez cette ressource sur la boucle for.

4.2. Testez votre méthode sur quelques exemples en vérifiant qu'elle retourne bien une valeur conforme à la définition.

5. **Norme**

5.1. Ajoutez à la classe `Vecteur` une méthode publique `norme` sans paramètre qui retourne la norme du `Vecteur` courant.

Aide

Pour calculer la racine carrée d'un nombre réel (représenté par un `double` en Java), Java fournit la méthode `sqrt` de la classe `Math`. Par exemple, la ligne de code suivante a pour effet d'affecter à `a` la racine carrée de `b` :

```
a = Math.sqrt(b);
```

5.2. Testez votre méthode sur les vecteurs $(1, 1, 1, 1, 1)^\top$, $(2; 0)^\top$ et $(0, 0, 0)^\top$ et vérifiez que son action est conforme à la définition.

6. **Normalisation**

6.1. Ajoutez à la classe `Vecteur` une procédure publique `normaliser` sans paramètre qui normalise l'instance courante de la classe `Vecteur`.

6.2. Testez votre méthode sur les vecteurs $(1, 1, 1, 1, 1)^\top$, $(2; 0)^\top$ et $(0, 0, 0)^\top$ et vérifiez que son action est conforme à la définition.

7. **Somme vectorielle**

Attention !

Il est important de se rappeler qu'il n'est pas possible de réaliser des opérations entre deux vecteurs de tailles différentes. Pour effectuer des opérations comme la somme, le produit scalaire ou encore le produit vectoriel, **les vecteurs doivent être définis dans des espaces de même dimension**.

7.1. Ajoutez à la classe `Vecteur` une procédure publique `sommeVectorielle` à un paramètre qui somme le `Vecteur` passé en paramètre à l'instance courante du `Vecteur`.

7.2. Testez votre procédure sur quelques exemples et vérifiez que son action est conforme à la définition. Que se passe-t-il lorsque vous essayez de sommer deux vecteurs définis dans des espaces de dimensions différentes ?

8. **Produit scalaire**

8.1. Ajoutez à la classe `Vecteur` une fonction publique `produitScalaire` à un paramètre qui retourne le produit scalaire de l'instance courante du `Vecteur` avec le `Vecteur` passé en paramètre.

8.2. Testez votre méthode sur quelques exemples en vérifiant qu'elle retourne bien une valeur conforme à la définition. Que se passe-t-il lorsque vous essayez de calculer le produit scalaire de deux vecteurs définis dans des espaces de dimensions différentes ?

9. **Produit vectoriel en 3 dimensions**

9.1. Ajoutez à la classe `Vecteur` une fonction publique `produitVectoriel3d` à un paramètre de type `Vecteur`. Cette méthode agit de la manière suivante :

- si le `Vecteur` courant et celui passé en paramètre sont définis en dimension 3, alors la méthode retourne un nouveau `Vecteur` qui est le produit vectoriel de l'instance courante par le `Vecteur` passé en paramètre ;
- sinon, la méthode retourne la référence `null` (grâce à l'instruction `return null;`) pour indiquer que l'opération ne peut pas être effectuée.

Rappel

Le produit vectoriel de deux vecteurs $\mathbf{u} = (x_u, y_u, z_u)^\top$ et $\mathbf{v} = (x_v, y_v, z_v)^\top$ d'un espace de dimension 3, noté $\mathbf{u} \wedge \mathbf{v}$, renvoie un vecteur orthogonal au plan $Vect(\mathbf{u}, \mathbf{v})$. On a :

$$\mathbf{u} \wedge \mathbf{v} = \begin{pmatrix} y_u \cdot z_v - y_v \cdot z_u \\ z_u \cdot x_v - z_v \cdot x_u \\ x_u \cdot y_v - x_v \cdot y_u \end{pmatrix}$$

9.2. Testez votre méthode sur quelques exemples en vérifiant qu'elle retourne bien une valeur conforme à la définition. Testez en particulier le vecteur nul.

9.3. Après avoir lu les éléments d'aide présentés ci-dessous, visualisez graphiquement le résultat du produit vectoriel de u par v, où $u = (1, 1, 1)^\top$ et $v = (-1, 1, -1)^\top$.

10. **Orthogonalité**

10.1. Ajoutez à la classe `Vecteur` une fonction publique `estOrthogonal` à un paramètre qui retourne une valeur booléenne indiquant si l'instance courante du `Vecteur` est orthogonale au `Vecteur` passé en paramètre.

10.2. Testez votre méthode sur quelques exemples en vérifiant qu'elle retourne bien une valeur conforme à la définition. Testez en particulier le vecteur nul.

11. **Colinéarité**

11.1. Ajoutez à la classe `Vecteur` une fonction publique `estColineaire` à un paramètre qui retourne un booléen indiquant si l'instance courante du `Vecteur` est colinéaire au `Vecteur` passé en paramètre.

11.2. Testez votre méthode sur les exemples suivants :

- $(1, 2)^\top$ et $(0, 0)^\top$;
- $(3, 4)^\top$ et $(-6, 8)^\top$;
- $(1, 0)^\top$ et $(0, 1)^\top$;
- $(1, 1)^\top$ et $(-4, -4)^\top$.

Si le résultat est conforme pour tous les tests sauf le dernier, ne vous inquiétez pas, nous allons résoudre le problème dans la question 13.

12. **Coplanarité en 3 dimensions**

Aide

Pour résoudre ce problème de coplanarité, il est utile de se rappeler qu'un plan en 3 dimensions peut être défini

- soit par deux vecteurs non colinéaires ;
- soit par un vecteur normal au plan (Un vecteur qui appartient au plan sera orthogonal à ce vecteur normal).

Dans les questions précédentes, nous avons probablement écrit une fonction qui permet d'obtenir ce vecteur normal en utilisant les deux vecteurs définissant le plan. Une fois ce vecteur normal obtenu, il est possible de vérifier la coplanarité d'un autre vecteur en utilisant une propriété géométrique simple.

12.1. Ajoutez à la classe `Vecteur` une fonction publique `estCoplanaire3d` à deux paramètres qui retourne un booléen indiquant si l'instance courante de la classe `Vecteur` est coplanaire aux deux instances de la classe `Vecteur` passées en paramètre. On supposera que les vecteurs mis en jeu lors d'appels à cette méthode sont toujours définis en dimension 3.

12.2. Testez votre méthode sur quelques exemples en vérifiant qu'elle retourne bien une valeur conforme à la définition. Testez en particulier le vecteur nul.

13. **Erreurs d'arrondi**

Dans ce TP, nous ne nous sommes pas souciés des erreurs d'arrondi.

Rappel

Le type `double` n'est qu'une approximation des nombres réels car il faudrait une quantité infinie de mémoire et de puissance de calcul pour pouvoir représenter et manipuler les réels de façon exacte.

13.1. Recopiez dans la classe `Vecteur` la fonction `approche` que vous aviez écrite au TP précédent.

13.2. Dans les fonctions `estOrthogonal`, `estColineaire` et `estCoplanaire3d`, remplacez chaque test d'égalité entre deux variables de type `double` par un appel à la méthode `approche` avec `epsilon_rel = 1e-9` et `epsilon_abs = 1e-5`.

13.3. Vérifiez que la méthode `estColineaire` fonctionne correctement en reprenant les tests de la question 11.2.