Atelier E2 - Algèbre linéaire et vision 3D
TP1 : Vecteurs en dimension 2

Romain Negrel*

Objectifs

Programmer une classe `VecteurEn2d` pour représenter et manipuler les vecteurs dans l'espace euclidien de dimension 2.

Liste des compétences à acquérir pendant le TP:

- déterminer les attributs et écrire le constructeur d'une classe Java pour représenter un vecteur dans l'espace euclidien à deux dimensions ;
- programmer en Java (sous forme de méthodes) les opérations élémentaires (multiplication par un scalaire, somme et produit scalaire) sur les vecteurs de l'espace euclidien à deux dimensions ;
- écrire un programme pour obtenir un vecteur orthogonal à un vecteur donné ;
- écrire un programme pour tester l'orthogonalité et la colinéarité de deux vecteurs en utilisant les opérations vectorielles ;
- résoudre un problème mathématique dont la solution peut être obtenue par manipulation de vecteurs de l'espace à deux dimensions ;
- écrire complètement une classe en Java.

Sujet

1. **Travail préliminaire**

   1.1. Sur votre compte, créez un répertoire '`ATL_2201`'. Télécharger le fichier '`ATL_2201_TP1.zip`' et enregistrez le dans le répertoire '`ATL_2201`'.

   1.2. Sous BlueJ, ouvrez le fichier '`ATL_2201_TP1.zip`' : Menu "Projet" → "Open Project" et sélectionnez le fichier '`ATL_2201_TP1.zip`'.

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 !

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

Important !!

Ajoutez des commentaires dans votre code !

2. **Création de la classe VecteurEn2d**

   2.1. Créez dans votre projet une nouvelle classe `VecteurEn2d` (bouton '`Nouvelle classe`' sous Bluej, type de classe : Standard). Inspectez son code ? Que contient-il ? Cela correspond-il à un vecteur en deux dimension ?

   2.2. Ajoutez à la classe `VecteurEn2d` les attributs nécessaires pour que les objets de la classe `VecteurEn2d` représentent bien un vecteur de l'espace euclidien en dimension 2. Le type `double` sera utilisé pour représenter des nombres réels.

   2.3. Ajoutez à la classe `VecteurEn2d` un constructeur à 2 paramètres de type double spécifiant la valeur du vecteur. La signature de ce constructeur est donc :

```
public VecteurEn2d(final double pX, final double pY)
```

   2.4. Ajoutez à la classe `VecteurEn2d` un constructeur par recopie. La signature de ce constructeur est donc :

```
public VecteurEn2d(final VecteurEn2d pVecteur)
```

   2.5. Ajoutez à la classe `VecteurEn2d` deux accesseurs correspondant aux deux attributs, i.e., deux méthodes publiques retournant la valeur des attributs du `VecteurEn2d` courant et dont les signatures sont :

```
public double getX()
public double getY()
```

   Pour vérifier que cela fonctionne, construisez le vecteur (2,1).

3. **Dessine**

   Sous blueJ, créez un Plan et un `VecteurEn2d`, puis invoquez la méthode `dessinerVecteurEn2d` de ce Plan en lui passant en paramètre l'instance du `VecteurEn2d` que vous venez de créer. Vérifiez que le dessin obtenu est bien conforme au vecteur que vous avez créé.

4. **Norme**

   4.1. Ajoutez à la classe `VecteurEn2d` une méthode publique norme sans paramètre qui retourne la norme du `VecteurEn2d` courant.

Rappel

La formule de la norme d'un vecteur $\mathbf{v}$ de dimension 2 est la suivante :

$$\|\mathbf{v}\| = \sqrt{x^2 + y^2} \tag{1}$$

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 carré de b :

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

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

5. **Multiplication par un scalaire**

5.1. Ajoutez à la classe `VecteurEn2d` une procédure publique `multiplicationScalaire` à un paramètre qui multiplie l'instance courante du `VecteurEn2d` par le paramètre scalaire.

5.2. Testez votre procédure sur quelques exemples et vérifiez que son action est conforme à la définition.

6. **Somme vectorielle**

6.1. Ajoutez à la classe `VecteurEn2d` une procédure publique `sommeVectorielle` à un paramètre qui somme le `VecteurEn2d` passé en paramètre à l'instance courante du `VecteurEn2d`.

6.2. Testez votre procédure sur quelques exemples et vérifiez que son action est conforme à la définition.

7. **Produit scalaire**

7.1. Rappelez la formule du produit scalaire de deux vecteurs (en dimension 2) ?

7.2. Ajoutez à la classe `VecteurEn2d` une fonction publique `produitScalaire` à un paramètre qui retourne le produit scalaire de l'instance courante du `VecteurEn2d` avec le `VecteurEn2d` passé en paramètre.

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

8. **Orthogonalité**

8.1. Comment peut-on, à partir du calcul d'un produit scalaire, déterminer si deux vecteurs sont orthogonaux ?

> **Rappel**
>
> Pour tout couple de deux vecteurs : $(\mathbf{v}, \mathbf{w})$, nous rappelons que le produis scalaire vérifie la propriété suivant :
>
> $$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\| \times \|\mathbf{w}\| \times \cos\left(\widehat{\mathbf{v}, \mathbf{w}}\right),$$
>
> avec $\widehat{\mathbf{v}, \mathbf{w}}$ l'ange entre les deux vecteurs $\mathbf{v}$, $\mathbf{w}$.

8.2. Ajoutez à la classe `VecteurEn2d` une fonction publique `estOrthogonal` à un paramètre qui retourne une valeur `booleene` indiquant si l'instance courante du `VecteurEn2d` est orthogonale au `VecteurEn2d` passé en paramètre.

8.3. Ajoutez à la classe `VecteurEn2d` une fonction `obtenirVectOrthogonal` sans paramètre qui retourne un vecteur orthogonal non nul à l'instance courante du `VecteurEn2d` (sauf si le vecteur courant est nul, auquel cas il retourne également le vecteur nul).

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

8.5. Créez les vecteurs $\mathbf{v} = (0.1, 9)$ et $\mathbf{w} = (-4.5, 0.05)$. Testez l'orthogonalité de ces vecteurs avec la méthode `estOrthogonal`. Le résultat obtenu est il conforme ? Multipliez $\mathbf{v}$ par 0.1 et re-testez l'orthogonalité. Le résultat est-il toujours conforme ? Pourquoi ?

9. **Erreurs d'arrondi**

   9.1. Ajoutez à la classe `VecteurEn2d` une fonction statique `approche` prenant quatre paramètres de type `double a`, `b`, `epsilon_rel` et `epsilon_abs` et retournant `True` si `a` et `b` sont relativement proche ou `False` si non.

   > **Aide**
   >
   > Pour déterminer si deux valeurs sont relativement proche, nous utiliserions la règle suivant :
   >
   > $$\text{approche}_{\epsilon_{\text{rel}},\epsilon_{\text{abs}}}(a,b) = \begin{cases} \texttt{True}, & \text{si } |a-b| \leq \max(\max(|a|,|b|) \times \epsilon_{\text{rel}}, \epsilon_{\text{abs}}) \\ \texttt{False}, & \text{si non} \end{cases}$$
   >
   > Pour le calcule de la valeur absolu, vous pouvez utiliser la fonction suivant : `Math.abs(x)`. Pour calculer le maximum entre deux valeurs, vous pouvez utiliser la fonction suivant : `Math.max(a, b)`.

10. **Colinéarité**

    10.1. Ajoutez à la classe `VecteurEn2d` une fonction publique `estColineaire` à un paramètre qui retourne un `boolean` indiquant si l'instance courante du `VecteurEn2d` est colinéaire au `VecteurEn2d` 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. **Distance**

    11.1. Ajoutez à la classe `VecteurEn2d` une fonction publique `distanceA` à un paramètre qui retourne un `double` représentant la distance entre l'instance du `VecteurEn2d` et le `VecteurEn2d` passé en paramètre.

    > **Rappel**
    >
    > La formule de la distance entre deux points A et B, respectivement représenté par les vecteurs **a** et **b** de dimensions 2, est la suivante :
    >
    > $$\|\overrightarrow{AB}\| = \|\mathbf{b} - \mathbf{a}\| \tag{2}$$

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

12. **Vérification de l'orthogonalité entre deux segments**

    Vous avez précédemment écrit une fonction pour déterminer si deux vecteurs sont orthogonaux en prenant comme référence l'origine du repère (question 8). Pour des vecteurs **a** et **b** représentant les points A et B, vous vérifiez si le vecteur $\overrightarrow{OA}$ est orthogonal à $\overrightarrow{OB}$ au point O, qui est l'origine.

    Nous voulons maintenant déterminer si les segments $\overrightarrow{BA}$ et $\overrightarrow{BC}$ sont orthogonaux en B. Pour cela, les points doivent être déplacés afin que B soit à l'origine. Il faut donc soustraire le vecteur B des points A et C, ce qui donne $C' = C - B$ et $A' = A - B$. Une fois cela fait, utilisez votre fonction précédente pour voir si les points C' et A' sont orthogonaux avec la fonction de la 8.

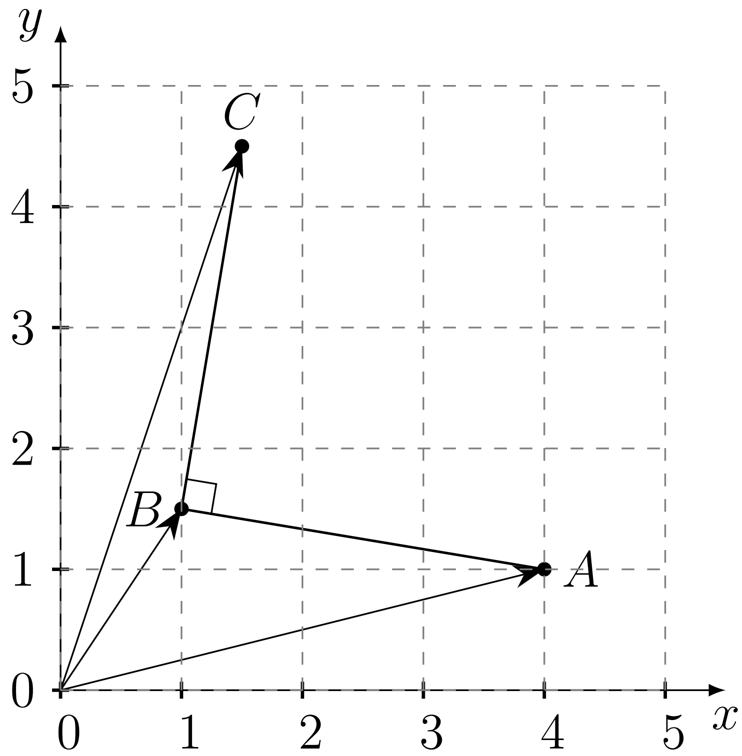

Figure 1: Illustration du problème avec les vecteurs représentés sur un graphique. Le segment BA est dessiné de B à A, et le segment BC est dessiné de B à C.

12.1. Ajoutez à la classe `VecteurEn2d` une fonction publique `segmentsSontOrthogonaux` à deux paramètres qui retourne une valeur `booleene` suivante :

```
public boolean segmentsSontOrthogonaux(final VecteurEn2d pVecteurA,
    final VecteurEn2d pVecteurC)
```

Si le segment défini par l'instance actuelle et `pVecteurA` est orthogonal au segment défini par l'instance actuelle et `pVecteurC`, la méthode renvoie `True`. Sinon, elle renvoie `False`.

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

13. **Application**

En vous aidant des fonctions des questions 11 et 12.

13.1. Créez les vecteurs $\mathbf{a} = (-1, 2)$, $\mathbf{b} = (2, 9)$, $\mathbf{c} = (9, 5.5)$ et $\mathbf{d} = (6, -1.5)$ représentant respectivement les cotés A, B, C et D d'un quadrilatère.

13.2. En utilisant les outils codés dans ce TP, déterminez la nature du quadrilatère ABCD : est-ce un carré, un rectangle, un losange, un parallélépipède ou un quadrilatère quelconque ?