Projet : fichier réponses

Raphaël Vock, Lomàn Vezin

16 mars 2019

1 La classe Vector3D

1.1 Conception

(P1.1) Les points et vecteurs de l'espace euclidien sont représentés au moyen d'instances de la classe `Vector3D` dont les attributs `x`, `y`, `z` (de type `double`) sont leurs coordonnées cartésiennes. Celles-ci sont des attributs privés, mais tout le reste est publique. Voici les opérations que nous avons implémentées en premier :

- `getCoords()` est une méthode qui retourne les coordonnées d'une instance via un `std::array<double,3>`.
- `norm()` et `norm2()` retournent la norme euclidienne (resp. norme euclidienne au carré).
- `distance(x,y)` et `distance2(x,y)` sont des méthodes statiques qui retournent la distance euclidienne (resp. distance euclidienne au carré).
- `unitary()` est une méthode qui retourne le vecteur directeur de même sens et de même direction (et lance une exception si l'instance est nulle ; voir point suivant).
- `is_zero()` est une méthode qui retourne un bool indiquant si l'instance est plus petite (en norme au carré) qu'une (petite) constante `EPSILON`.

1.2 Constructeur, constructeur de copie

(P4.1) L'unique constructeur de la classe prend en argument trois `double` (nuls par défaut) et initialise un `Vector3D` avec les coordonnées cartésiennes précisées. En l'absence d'attributs pointeurs, le constructeur de copie minimal par défaut réalise exactement ce que l'on attend, donc nous n'en avons pas defini.

1.3 Coordonnées sphériques

(P4.2) Nous n'avons pas implémenté de constructeur en coordonnées sphériques. Avant tout, l'ajout de ce dernier serait un peu technique au niveau du prototypage de car nous aurions *a priori* deux constructeurs aux protoypes identiques. Ce n'est pas un problème incontournable ; on pourrait, par exemple, écrire un seul et unique constructeur dont le

quatrième argument est un `bool` (`false` par défaut) qui précise si les arguments donnés doivent être compris comme des coordonnées sphériques ou sinon cartésiennes.

Mais le problème fondamental est le suivant : un repère sphérique est intextricablement lié à la donnée d'un point O considéré comme l'origine. Pour écrire un constructeur sphérique sans ambiguité il serait donc nécessaire :

- ou bien de faire un choix canonique de O qui ne changerait pas du début à la fin
- ou bien de préciser O en passant sa valeur (ou éventuellement son adresse) en argument.

Si l'on choisit la première option, l'unique avantage des sphériques serait perdue car on ne serait plus en mesure de choisir l'origine à notre gré pour simplifier, entre autres, des expressions de force (e.g. en choisissant O l'origine d'un champ de force central). Mais la deuxième option serait coûteuse à implémenter et lourde à utiliser pour des avantages qui finalement n'en vaudraient pas le coup.

1.4 Surcharge des opérateurs

(P4.3)

- Nous avons surchargé les opérateurs d'auto-affectation `+=`, `-=`, `*=` correspondant à l'addition (resp. soustraction) vectorielle et la multiplication par un scalaire. Celles-ci retournent le résultat de l'affectation pour optimiser certains calculs par la suite.
- À partir de ces derniers nous avons surchargé les opérateurs binaires `+`, `-`, `*`.
- Les opérateurs binaires `|` et $\wedge$ retournent respectivement le produit scalaire euclidien et le produit vectoriel.
- Accessoirement, la fonction `mixed_prod(u,v,w)` retourne le produit mixte des trois arguments, c'est-à-dire le scalaire `u|(v∧w)`.
- L'opérateur de comparaison `==` est défini à partir de la méthode `is_zero()` (cf. §1.1). L'opérateur `!=` retourne la négation logique de `==` ;