Laboratoire :
La loi de Coulomb

https://fr.wikipedia.org/wiki/Charles-Augustin_Coulomb

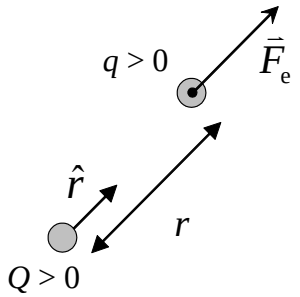
Table des matières

Introduction

Dans cette partie du laboratoire, vous allez implémenter la ***loi de Coulomb*** afin d'évaluer la superposition des forces électriques d'un très grand nombre de particules chargées. Pour ce faire, vous devrez implémenter des opérations mathématiques sur les ***vecteurs*** et intégrer ces opérations dans le calcul de la force électrique. Plus concrètement, vous devrez programmer la loi de Coulomb

$$\vec{F}_{\mathrm{e}} = k\,qQ\frac{\left(\vec{r}_q - \vec{r}_Q\right)}{\left\|\vec{r}_q - \vec{r}_Q\right\|^3}$$

où $\vec{F}_{\mathrm{e}}$ est la force électrique qu'applique la charge Q sur la charge q en newton, q est la charge qui subit la force en coulomb, Q est la charge qui applique la force en coulomb, $\vec{r}_q$ est le vecteur position de la charge q en mètre, $\vec{r}_Q$ est le vecteur position de la charge Q en mètre et k est la constante de la loi de Coulomb tel que $k \approx 9{,}00\times10^9\ \mathrm{N\cdot m^2/C^2}$.



La force de Coulomb qu'une charge Q positive applique sur une charge q positive.

Pour réaliser ce laboratoire, vous avez accès au ***projet Java* SIM**. Vous pouvez télécharger le projet avec lien suivant :

http://physique.cmaisonneuve.qc.ca/svezina/projet/coulomb/download/SIM-Coulomb.zip

Commencez par vous définir un répertoire (exemple « *java* ») qui vous permettra de définir votre ***workspace*** lors de l'ouverture du logiciel ***Eclipse***. Par la suite, décompressez le fichier « *SIM-Coulomb.zip* **»** dans le répertoire de votre ***workspace***. Vous devriez obtenir un répertoire au nom de « *SIM* ». Ouvrez pas la suite le logiciel ***Eclipse***.

Dans l'onglet ***File***, choisissez l'option ***Switch Workspace*** et identifiez la localisation de votre répertoire de projet (dans l'exemple : « *java* »). Votre ***workspace*** est maintenant configuré.

Dans l'onglet ***File***, choisissez l'option ***Open Projects from File System…*** . Dans la fenêtre ***Import Projects from File System or Archive***, modifiez le champ ***Import source*** à l'aide du bouton ***Directory…*** et sélectionnez le répertoire du projet contenu dans le fichier décompressé « *SIM-Coulomb.zip* » étant de nom *SIM*. Vous avez maintenant configuré votre environnement de développement.

Pour la remise de votre laboratoire, vous devez remettre vos sources (directives à venir) et compléter le document ***Rapport de laboratoire*** disponible au lien suivant :

http://physique.cmaisonneuve.qc.ca/svezina/projet/coulomb/download/Rapport-Coulomb.pdf

Description générale du programme

L'application permettant d'évaluer la superposition des forces électriques est disponible dans la ***classe*** **SIMCoulomb** disponible dans le ***package*** **sim.application**.

Pour y parvenir, vous allez compléter l'écriture de six classes :

- **SIMCoulomb**

Cette classe représente l'espace où seront effectuées l'application et la vérification de la loi de Coulomb

- **SVector3d**

Cette classe représente un vecteur à trois dimensions *x*, *y* et *z* pouvant réaliser des opérations mathématiques tel que l'addition, la soustraction, la multiplication par un scalaire et l'évaluation du module du vecteur. Les objets de type **SVector3d** sont dit « immuable », car on ne peut pas changer leur valeur (la méthode set(…) n'existe pas). Ainsi, effectuer un calcul mathématique avec un tel vecteur retournera nécessairement la construction d'un nouveau vecteur avec comme paramètre le résultat du calcul.

De plus, puisque la méthode toString( ) a été implémentée pour cet classe, vous pouver l'inclure dans l'instruction

System.out.print(…)

pour afficher son contenu dans la console.

- **SElectrostatics**

Cette classe permet d'appliquer des principes de physique en électrostatique comme la loi de Coulomb.

- **SGeometricDiscretization**

Cette classe utilitaire permettre de fractionner une ligne et une surface carré en plusieurs vecteurs positions. Cette classe sera utilisée pour construire des particules (classe **SParticle**) à inserrer dans un système de particules (classe **SParticleSystem**) afin de simuler la distribution de charges associée la géométrie de la TRIUC (tige rectiligne uniformément chargée) et la géométrie de la PPIUC (plaque plane infini uniformément chargée).

- **SParticleSystem**

Cette classe abstraite permettra de faire la gestion d'un système de particules et de calculer des forces entre celles-ci.

- **SElectricParticleSystem**

Cette classe permettra de définir le type d'interaction entre les particules étant dans le contexte de ce laboratoire de type Coulombienne.

1.1 – La première exécution du programme

Fichier à modifier : aucun

Prérequis : aucun

Dans la fenêtre ***Package Explorer*** du logiciel ***Eclipse***, ouvrez le répertoire **SIM** puis ouvrez le répertoire **src**. Constatez la présence de quelques ***packages*** nécessaires à l'exécution de ce programme. Ouvrez le ***package*** **sim.application** et lancez l'exécution de la classe **SIMCoulomb.java** à l'aide d'un « clic droit » sur le fichier. Sélectionnez dans le ***pop-pop menu*** l'option ***Run As*** et l'autre option ***Java Application***.

Présentement, l'application n'effectue par de calcul. Vous devrez compléter le programme afin de calculer informatiquement ce que vous avez calculés dans votre prélaboratoire d'où la référence à l'affichage « Résultat : Mise en situation - Question 1. (prélaboratoire », …

1.2 – L'exécution des JUnit Test

Fichier à modifier : aucun

Prérequis : aucun

Pour s'assurer de la qualité d'un programme, il est important de tester les fonctionnalités des différentes méthodes implémentées. L'environnement de développement Eclipse permet l'exécution de batterie de tests unitaires avec une gestion des succès (***succes***) et des échecs (***fail***).

Un test unitaire réalise l'exécution d'une méthode (ou quelques méthodes) d'une classe ou de plusieurs classes dans un scénario particulier. Le résultat de l'exécution (« ***calculated*** ») est alors comparé avec un résultat attendu (« ***expected*** »). Si le résultat calculé est identique au résultat attendu, le test est un succès. Dans le cas contraire, le test est alors un échec.

https://www.javacodegeeks.com/2013/12/parameterized-junit-tests-with-junitparams.html

Afin de vous familiariser avec l'exécution des ***JUnit Test***, vous allez réaliser l'exécution de l'ensemble des tests préalablement écrit pour vous afin de vous guider dans la qualité de vos implémentations.

Pour ce faire, vous allez :

- Dans la fenêtre ***Package Explorer***, ouvrez le répertoire **SIM** contenant l'ensemble des éléments du projet.
- Sur le répertoire **test/src**, effectuez un « clic droit » et sélectionnez dans le pop-pop menu l'option ***Run As*** et réalisez l'exécution de type ***JUnit Test***.
- (**Si les étapes précédentes ne fonctionnent pas**) Établir le lien avec la librairie ***JUnit 4***. Faire un clic droit sur le répertoire **SIM** dans la fenêtre ***Package Explorer*** et choisir l'option ***Properties***. Dans la propriété ***Java Build Path***, choisir l'onglet ***Libraries*** et activer le bouton ***Add Library***. Choisir ***JUnit*** et activer le bouton ***next***. Choisir ***JUnit 4*** et activer le bouton ***Finish***. Reprenez les étapes précédentes.

Dans la fenêtre ***JUnit***, vous pouvez visualiser l'ensemble des tests effectués pour valider certaines fonctionnalités du projet :

- Si toutes les implémentations sont adéquates, une **couleur verte** sera affichée (Failures : 0).
- S'il y a des implémentations inadéquates, une **couleur rouge** sera affichée (Failures : « nombre > 0 »).

Dans la fenêtre ***Console***, vous remarquerez la présente de messages indiquant que certains tests n'ont pas été effectués, mais qu'ils sont considérés comme des succès. Ce sont des tests reliés à des méthodes que vous devrez implémenter. Ces méthodes retournent présentement une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Lorsque l'implémentation sera réalisée, le test sera effectué « officiellement ».

Pour lancer l'exécution d'un nombre restreint de tests, vous n'avez qu'à effectuer le « clic droit » sur le répertoire ou le fichier désiré. Par exemple, vous pouvez lancer le test de la classe **SMathTest** du package ***sim.math*** situé dans le répertoire ***test/src/math***. Vous remarquerez qu'il est en succès (**couleur verte**), mais qu'il y a des tests non effectué.

Les opérations de base sur les vecteurs

Pour appliquer vectorielle la loi de Coulomb, vous devez compléter quelques opérations mathématiques en lien le concept de vecteur à trois dimension. Voici les quatre opérations à compléter :

L'addition de deux vecteurs : (écriture en java : SVector3d C = A.add(B);)

$$\vec{C} = \vec{A} + \vec{B} \quad \Rightarrow \quad \vec{C} = \left(A_x\vec{i} + A_y\vec{j} + A_z\vec{k}\right) + \left(B_x\vec{i} + B_y\vec{j} + B_z\vec{k}\right)$$

$$\Rightarrow \quad \vec{C} = \left(A_x + B_x\right)\vec{i} + \left(A_y + B_y\right)\vec{j} + \left(A_z + B_z\right)\vec{k}$$

La soustraction de deux vecteurs : (écriture en java : SVector3d C = A.substract(B);)

$$\vec{C} = \vec{A} - \vec{B} \quad \Rightarrow \quad \vec{C} = \left(A_x\vec{i} + A_y\vec{j} + A_z\vec{k}\right) - \left(B_x\vec{i} + B_y\vec{j} + B_z\vec{k}\right)$$

$$\Rightarrow \quad \vec{C} = \left(A_x - B_x\right)\vec{i} + \left(A_y - B_y\right)\vec{j} + \left(A_z - B_z\right)\vec{k}$$

La multiplication d'un vecteur par un scalaire : (écriture en java : SVector3d C = A.multiply(m);)

$$\vec{C} = m\vec{A} \quad \Rightarrow \quad \vec{C} = m\left(A_x\vec{i} + A_y\vec{j} + A_z\vec{k}\right) \quad \Rightarrow \quad \vec{C} = mA_x\vec{i} + mA_y\vec{j} + mA_z\vec{k}$$

Le module d'un vecteur : (écriture en java : SVector3d a = A.modulus( );)

$$A = \left\|\vec{A}\right\| \quad \Rightarrow \quad A = \left\|A_x\vec{i} + A_y\vec{j} + A_z\vec{k}\right\| \quad \Rightarrow \quad A = \sqrt{{A_x}^2 + {A_y}^2 + {A_z}^2}$$

2.1 La construction d'un vecteur avec le prélaboratoire (Q1)

Fichier à modifier : **SIMCoulomb.java**

Prérequis : aucun

Dans le but de vous familiariser avec la classe **SVector3d**, vous allez compléter la méthode

public static void miseEnSituationQ1()

située dans la classe **SIMCoulomb** disponible dans le ***package*** **sim.application**.

L'objectif de cette méthode sera de construire trois objets de type **SVector3d** à l'aide du constucteur

public SVector3d(**double** x, **double** y, **double** z)

déjà implémenté pour vous afin de représenter vos vecteurs positions $\vec{r}_1$, $\vec{r}_2$ et $\vec{r}_3$ associés à la position des trois sphères de charge Q_1, Q_2 et Q_3 de votre ***prélaboratoire (Q1.)***. Utilisez par exemple l'expression

SVector3d r1 = **new** SVector3d(0.0, 1.0, 0.0);

pour faire la construction de votre vecteur position $\vec{r}_1 = (0.0, 1.0, 0.0)$. Puisque le prélaboratoire fait référence à un positionnement à deux dimensions, utilisez la composante $z = 0.0$ pour la troisième dimension de vos vecteurs.

Affichez les trois vecteurs dans une console texte grâce à l'appel de la méthode

System.out.println(String s)

sachant que la méthode toString() est déjà redéfinie dans la classe **SVector3d**.

FAÎTES VALIDER VOTRE EXÉCUTION PAR VOTRE PROFESSEUR (CONTINUEZ SI CELUI-CI N'EST PAS DISPONIBLE DANS L'IMMÉDIAT) À L'AIDE D'UNE SIGNATURE DANS VOTRE DOCUMENT ***RAPPORT DE LABORATOIRE***.

2.2 L'addition de deux vecteurs

Fichier à modifier : **SVector3d.java**

Prérequis : aucun

Dans la classe **SVector3d** disponible dans le ***package*** **sim.math**, vous allez programmer la méthode

public SVector3d add(SVector3d v) .

Puisque la classe **SVector3d** est immutable (ses paramètres x, y et z sont ***final***), l'appel de la méthode add doit nécessairement construire un nouvel objet de type **SVector3d** et retourner le résultat de l'addition du vecteur courant et du vecteur passé en paramètre dans la méthode. Cette approche est très pratique, car elle vous permettra de réutiliser l'objet ayant lancé l'appel de la méthode add sans que celui-ci ait été modifié. C'est exactement le comportement que l'on recherche en mathématique, car on ne peut jamais utiliser deux vecteurs différents avec une même notation.

Présentement, cette méthode retourne une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation en utilisant les champs

x, y et z

de l'objet lançant l'appel de la méthode et en utilisant les champs

v.x, v.y et v.z

du vecteur passé en paramètre dans la méthode pour réaliser l'addition entre ces deux vecteurs ($\vec{C} = \vec{A} + \vec{B}$).

Pour vérifier votre implémentation, exécutez la batterie de tests unitaires de java ***JUnit Test*** disponible dans la classe **SVector3dTest** du ***package*** **sim.math** située dans le répertoire **test/src**. Corrigez au besoin votre implémentation afin de satisfaire les tests.

2.3 La soustraction de deux vecteurs

Fichier à modifier : **SVector3d.java** et **SIMCoulomb.java**

Prérequis : aucun

Dans la classe **SVector3d** disponible dans le ***package*** **sim.math**, vous allez programmer la méthode

public SVector3d substract(SVector3d v) .

Présentement, cette méthode retourne une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation afin de réaliser la soustraction entre deux vecteurs ($\vec{C} = \vec{A} - \vec{B}$).

Pour vérifier votre implémentation, exécutez la batterie de tests unitaires de java ***JUnit Test*** disponible dans la classe **SVector3dTest** du ***package*** **sim.math** située dans le répertoire **test/src**. Corrigez au besoin votre implémentation afin de satisfaire les tests.

2.4 La validation de la soustraction avec le prélaboratoire (Q2)

Fichier à modifier : **SIMCoulomb.java**

Prérequis : 2.3

Afin de valider l'opération de la soustraction avec vos données du prélaboratoire, vous allez compléter la méthode

```
public static void miseEnSituationQ2()
```

située dans la classe **SIMCoulomb** disponible dans le ***package*** **sim.application**.

L'objectif de cette méthode sera de calculer les vecteurs déplacements $\vec{r}_{13}$ et $\vec{r}_{23}$ en lien avec les trois sphères de votre ***prélaboratoire (Q2.)***. Utilisez la méthode substract que vous venez de programmer pour réaliser vos calculs et affichez vos résultats dans la console de texte.

FAÎTES VALIDER VOTRE EXÉCUTION PAR VOTRE PROFESSEUR (CONTINUEZ SI CELUI-CI N'EST PAS DISPONIBLE DANS L'IMMÉDIAT) À L'AIDE D'UNE SIGNATURE DANS VOTRE DOCUMENT ***RAPPORT DE LABORATOIRE***.

2.5 La multiplication d'un vecteur par un scalaire

Fichier à modifier : **SVector3d.java**

Prérequis : aucun

Dans la classe **SVector3d** disponible dans le ***package*** **sim.math**, vous allez programmer la méthode

```
public SVector3d multiply(double m) .
```

Présentement, cette méthode retourne une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation afin de réaliser la multiplication d'un vecteur par un scalaire ($\vec{C} = m\vec{A}$).

Pour vérifier votre implémentation, exécutez la batterie de tests unitaires de java ***JUnit Test*** disponible dans la classe **SVector3dTest** du ***package*** **sim.math** située dans le répertoire **test/src**. Corrigez au besoin votre implémentation afin de satisfaire les tests.

2.6 Le module d'un vecteur

Fichier à modifier : **SVector3d.java**

Prérequis : aucun

Dans la classe **SVector3d** disponible dans le ***package*** **sim.math**, vous allez programmer la méthode

```
public double modulus( ) .
```

Présentement, cette méthode retourne une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation afin d'évaluer le module d'un vecteur à trois dimensions ($A = \left\|\vec{A}\right\|$)

Pour vérifier votre implémentation, exécutez la batterie de tests unitaires de java ***JUnit Test*** disponible dans la classe **SVector3dTest** du ***package*** **sim.math** située dans le répertoire **test/src**. Corrigez au besoin votre implémentation afin de satisfaire les tests.

La loi de Coulomb

Pour effectuer le calcul de la loi de Coulomb

$$\vec{F}_e = k\,qQ\frac{\left(\vec{r}_q - \vec{r}_Q\right)}{\left\|\vec{r}_q - \vec{r}_Q\right\|^3} \quad ,$$

il nous faut des vecteurs pouvant réaliser **(1)** la soustraction (substract), **(2)** le calcul du module (modulus) ainsi que **(3)** la multiplication par un scalaire (multiply). Toutes ces opérations ont été précédemment implémentées. Il ne vous reste plus qu'à les assembler adéquatement.

3.1 La programmation de la loi de Coulomb

Fichier à modifier : **SElectrostatics.java** et **SIMCoulomb**

Prérequis : 2.3, 2.5 et 2.6

Dans la classe **SElectrostatics** disponible dans le ***package* sim.physics**, vous allez programmer la méthode

```
public static SVector3d coulombLaw(double Q, SVector3d r_Q, double q, SVector3d r_q) throws
IllegalArgumentException
```

Présentement, cette méthode retourne une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation afin d'évaluer la force de coulomb qu'une charge *Q* applique sur une charge *q*.

Utilisez la constante k disponible dans la classe pour représenter votre constante de Coulomb, car elle est plus précise que la constante $9{,}0 \times 10^9$ N m^2/C^2 traditionnellement utilisé en classe.

Dans votre implémentation, vous remarquerez qu'il y aura deux cas d'exception à gérer :

1) Deux particules sont aux mêmes endroit ($\vec{r}_Q = \vec{r}_q$).
2) L'une des particules est située à l'infini.

Pour identifier le 1er cas d'exception, n'utilisez pas l'instruction

```
if ( r_Q == r_q ) { ... }
```

car **SVector3d** est une classe et non un type prémitif. Utilisez à la place l'instruction

```
public boolean equals( … )
```

qui a déjà été implémentée pour vous dans la classe **SVector3d**. Si vous identifiez ce scénario, lancez l'exception avec l'instruction

```
throw new IllegalArgumentException(« écrivez votre message d'erreur »);
```

ce qui stoppera l'exécution de la méthode lorsque ce scénario se produira.

Pour identifiez le 2e cas d'exception, vous pouvez utiliser la méthode

```
public boolean isInfinity()
```

de la classe **SVector3d**. Si vous identifiez ce scénario, la méthode devra retourner une force nulle correspondant en ***JAVA*** à

```
return SVector3d.ZERO;
```

Pour vérifier votre implémentation, exécutez la batterie de tests unitaires de java ***JUnit Test*** disponible dans la classe **SElectrostaticsTest** du ***package* sim.physics** située dans le répertoire **test/src**. Corrigez au besoin votre implémentation afin de satisfaire les tests.

3.2 La validation de la loi de Coulomb avec le prélaboratoire (Q3) + (Q4)

Fichier à modifier : **SElectrostatics.java** et **SIMCoulomb**

Prérequis : 3.1

Afin de valider la loi de Coulomb avec les calculs de votre prélaboratoire, vous allez compléter la méthode

```
public static void miseEnSituationQ3Q4()
```

située dans la classe **SIMCoulomb** disponible dans le ***package*** **sim.application**.

L'objectif de cette méthode sera de calculer la force électrique $\vec{F}_{e13}$ et $\vec{F}_{e23}$ appliquée sur la sphère de charge Q_3 en lien avec le ***prélaboratoire (Q3. et Q4.)***. Utilisez la méthode statique coulombLaw de la classe **SElectrostatics** avec l'instruction

```
SElectrostatics.coulombLaw( ... );
```

que vous venez de programmer pour réaliser vos calculs et affichez vos résultats dans la console de texte.

FAÎTES VALIDER VOTRE EXÉCUTION PAR VOTRE PROFESSEUR (CONTINUEZ SI CELUI-CI N'EST PAS DISPONIBLE DANS L'IMMÉDIAT) À L'AIDE D'UNE SIGNATURE DANS VOTRE DOCUMENT ***RAPPORT DE LABORATOIRE***.

3.3 Le principe de superposition avec le prélaboratoire (Q5)

Fichier à modifier : **SIMCoulomb**

Prérequis : 3.1

Dans le but de valider votre calcul final de votre prélaboratoire, vous allez compléter la méthode

```
public static void miseEnSituationQ5()
```

située dans la classe **SIMCoulomb** disponible dans le ***package*** **sim.application**. L'objectif de cette méthode sera de calculer la force électrique totale $\vec{F}_{e3}$ appliquée sur la sphère de charge Q_3 en lien avec ***prélaboratoire (Q5.)***.

FAÎTES VALIDER VOTRE EXÉCUTION PAR VOTRE PROFESSEUR (CONTINUEZ SI CELUI-CI N'EST PAS DISPONIBLE DANS L'IMMÉDIAT) À L'AIDE D'UNE SIGNATURE DANS VOTRE DOCUMENT ***RAPPORT DE LABORATOIRE***.

3.4 L'architechture d'un système de particules à interaction électrique

Fichier à modifier : **SElectricParticleSystem**

Prérequis : 3.1

Dans le but d'évaluer la force électrique appliquées entre plusieurs particules, ce laboratoire vous propose un architecture où il y aura une classe représentant un système de particules au nom de **SElectricParticleSystem**.

Puisque les fonctionnalités de cette classe est comparable à un système de particules gravitationnelles (voir projet : Les orbites), une architecture avec interface et classe abstraite a été déployée pour définir la classe **SElectricParticleSystem** :

SApplyForce (Interface : Structure capable de calculer des forces sur une particule)

↦ **SParticleSystem** (Classe abstraite : Évaluer des opérations de système de particules)

↦ **SElectricParticleSystem** (Classe : Définir l'interaction des particules)

Dans la classe **SElectricParticleSystem** disponible dans le ***package* sim.physics**, vous allez programmer la méthode

```
public SVector3d interactionForce(SParticle p_source, SParticle p_target) throws IllegalArgumentException .
```

Présentement, cette méthode retourne une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Conservez le code déjà existant qui vérifie que les deux particules n'ont pas le même numéro d'identification et complétez l'implémentation en utilisant tout simplement la méthode

```
SElectrostatics.coulombLaw( … );
```

que vous avez déjà programmée avec les bons paramètres dans le but de calculer la loi de Coulomb qu'une particule source applique sur une particule cible.

Utilisez les fonctionnalités de la classe **SParticle** tel que

```
double getElectricCharge()    et    SVector3d getPosition()
```

pour obtenir vos paramètres. Après coup, cette classe aura spécifié comment l'interaction entre deux particules est calculée.

Pour vérifier votre implémentation, exécutez la batterie de tests unitaires de java ***JUnit Test*** disponible dans la classe **SElectricParticleSystemTest** du ***package* sim.physics** située dans le répertoire **test/src**. Corrigez au besoin votre implémentation afin de satisfaire les tests.

3.5 Le principe de superposition dans un système de particules

Fichier à modifier : **SParticleSystem.java**

Prérequis : 3.4

Dans la classe **SParticlesSystem** disponible dans le ***package* sim.physics**, vous allez programmer la méthode

```
public SVector3d force(SParticle particle) throws IllegalArgumentException
```

Présentement, cette méthode retourne une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation afin que cette méthode retourne la force électrique résultante appliquée par le système de particules sur la particule passée en paramètre (particle).

Utilisez la liste des particules

```
particles_list
```

disponible dans la classe **SParticleSystem** pour calculer vos forces sur particle en prenant soit d'utiliser la méthode

```
abstract public SVector3d interactionForce(SParticle p_source, SParticle p_target) throws IllegalArgumentException
```

que vous venez tout juste de définir pour la classe **SElectricParticleSystem**.

Remarque : Assurez-vous que la particule qui subit la force résultante (particle) ne s'appliquera pas de force sur elle-même, car il est possible que cette particule soit intégrée au système de particules. Utilisez l'instruction

particle.asSameID(p) où p sera une particule du système

pour comparer l'identité de vos particules.

Remarque : N'oubliez pas que la classe **SVector3d** est immuable ! Le résultat de la méthode add(…) retourne un nouveau vecteur (comme pour la classe ***String*** de JAVA). Vous devrez donc superposer adéquatement vos forces.

Pour vérifier votre implémentation, exécutez la batterie de tests unitaires de java ***JUnit Test*** disponible dans la classe **SParticleSystemTest** du ***package*** **sim.physics** située dans le répertoire **test/src**. Corrigez au besoin votre implémentation afin de satisfaire les tests.

3.6 Le principe d'action-réaction à un système de particules

Fichier à modifier : **SIMCoulomb.java**

Prérequis : 2.3

À l'aide de la classe **SIMCoulomb** disponible dans le ***package*** **sim.application**, vous allez vérifier la 3ième loi de Newton

$$\vec{F}_{\mathrm{AB}} = -\vec{F}_{\mathrm{BA}}$$

en l'appliquant à un système de particules. Pour ce faire, exécutez tout simplement la classe **SIMCoulomb**.

La méthode déjà implémentée pour vous

public static void newtonThirdLaw(String file)

effectue le calcul

$$\sum_{i=1}^{N}\sum_{\substack{j=1\\ j\neq i}}^{N}\vec{F}_{\mathrm{e}(i\,j)} = \sum_{i=1}^{N}\sum_{\substack{j=1\\ j\neq i}}^{N} kq_i q_j \frac{\vec{r}_j - \vec{r}_i}{\left\|\vec{r}_j - \vec{r}_i\right\|^3} = 0$$

sur un système de particules dont la description est inscrite dans un fichier texte dont le nom correspond à la variable file de l'appel de la méthode. À la fin de la méthode, le programme affiche le résultat du calcul : le vecteur somme des forces du système.

Présentement, l'application effectue la lecture du fichier texte

systeme_particule.txt

qui est disponible dans le répertoire de votre projet

SIM\data\systeme_particule.txt .

À l'exécution, vous devriez avoir un affichage contenant la liste des particules du système ainsi que le résultat de la 3ième loi de Newton. Vérifiez à l'exécution que le vecteur somme des forces du système est bel et bien égal à zéro (ou numériquement très très près de zéro).

FAÎTES VALIDER VOTRE EXÉCUTION PAR VOTRE PROFESSEUR (CONTINUEZ SI CELUI-CI N'EST PAS DISPONIBLE DANS L'IMMÉDIAT) À L'AIDE D'UNE SIGNATURE DANS VOTRE DOCUMENT ***RAPPORT DE LABORATOIRE***.

Répondez à la question conceptuelle suivante :

Question 3.6 :

Lors de l'appel de la méthode

public static void newtonThirdLaw(String file) ,

si deux particules du système occupent la même position, quel sera le vecteur somme des forces du système calculé par le programme ? Justifiez votre réponse.

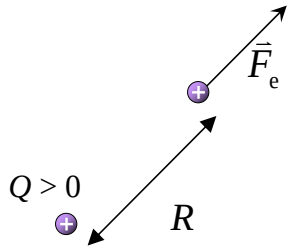
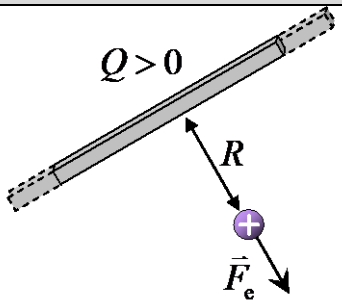
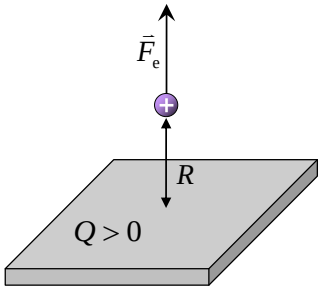
La discrétisation de la charge

En physique, il est possible d'utiliser l'intégrale pour évaluer la force électrique $\vec{F}_e$ qu'une distribution de particules chargées applique sur une charge q . L'expression à résoudre correspond à

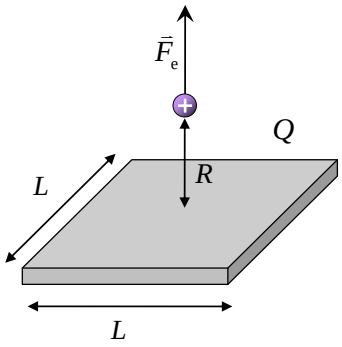
$$\vec{F}_e = \int k\, q \frac{\left(\vec{r}_q - \vec{r}_Q\right)}{\left\|\vec{r}_q - \vec{r}_Q\right\|^3} \mathrm{d}Q$$

et le résultat dépend principalement de la géométrie sur laquelle la distribution des charges $\mathrm{d}Q$ est réalisée.

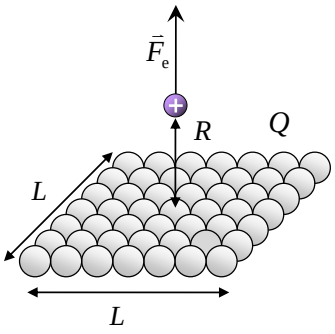
Dans le cadre de ce laboratoire, vous pourrez comparer trois géométries :

La charge ponctuelle	La tige infinie uniformément chargée (TRIUC)	La plaque plane infinie uniformément chargée (PPIUC)
		

Puisque nous n'allons pas utiliser une représentation analytique de nos géométries, nous allons les fractionner en plusieurs éléments ce qui correspond à discrétiser nos charges. Visuellement, une plaque uniformément chargée prendra la forme d'un très grand nombre de particules ponctuelles de charge *Q* / *N* où *N* sera le nombre de particules utilisées pour représenter la plaque.



Plaque en distribution de charge continue



Plaque en distribution de charge discrète

Le but des prochaines méthodes à implémenter sera de trouver la position des charges sur la géométrie d'une tige et la géométrie d'une plaque afin qu'elles soient uniformément réparties.

4.1 La discrétisation de la tige

Fichier à modifier : **SGeometricDiscretization.java**

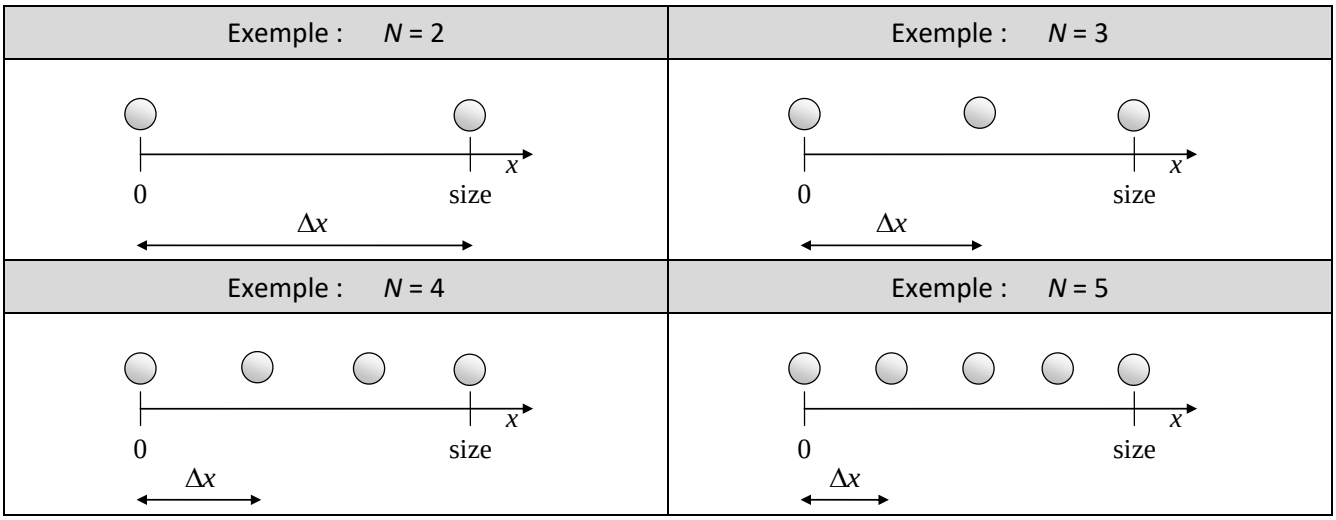
Prérequis : aucune

Dans la classe **SGeometricDiscretization** disponible dans le ***package* sim.geometry**, vous allez programmer la méthode

public static List<SVector3d> positiveLineXDiscretization(**double** size, **int** N) **throws** IllegalArgumentException

Présentement, cette méthode retourne une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation de la méthode afin de retourner une liste contenant N (où $N > 1$) vecteurs positions alignés sous la forme une droite le long de l'axe x débutant à $x = 0$ jusqu'à la position x = size. Vous devrez déterminer la distance Δx entre deux positions consécutives.

Voici un exemple de positionnement pour $N = 2$ à $N = 5$:



Pour vérifier votre implémentation, exécutez la batterie de tests unitaires de java ***JUnit Test*** disponible dans la classe **SGeometricDiscretizationTest** du ***package* sim.geometry** située dans le répertoire **test/src**. Corrigez au besoin votre implémentation afin de satisfaire les tests.

4.2 La discrétisation de la plaque

Fichier à modifier : **SGeometricDiscretization.java**

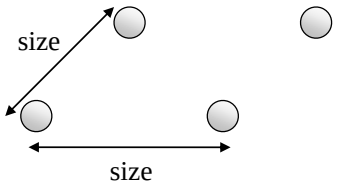
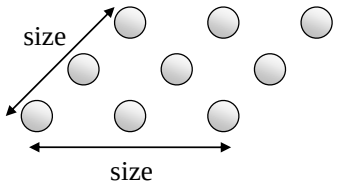
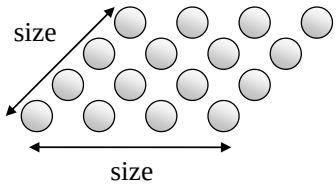
Prérequis : aucune

Dans la classe **SGeometricDiscretization** disponible dans le ***package* sim.geometry**, vous allez programmer la méthode

public static List<SVector3d> positiveSquareXYDiscretization(**double** size, **int** N) **throws** IllegalArgumentException

Présentement, cette méthode retourne une exception de type **SNoImplementationException** spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation de la méthode afin de retourner une liste contenant $N*N$ vecteurs positions alignés sous la forme d'un carré dans le plan xy positif débutant à $x = 0$ jusqu'à la position x = size et débutant à $y = 0$ jusqu'à la position y = size. Vous devrez déterminer la distance Δx et Δy entre les positions adjacentes.

Voici un exemple de positionnement pour $N = 2$ à $N = 4$:

Exemple : $N = 2$	Exemple : $N = 3$	Exemple : $N = 4$
		

Pour vérifier votre implémentation, exécutez la batterie de tests unitaires de java ***JUnit Test*** disponible dans la classe **SGeometricDiscretizationTest** du ***package*** **sim.geometry** située dans le répertoire **test/src**. Corrigez au besoin votre implémentation afin de satisfaire les tests.

4.3 La proportionnalité des forces

Fichier à modifier : **SIMCoulomb.java**

Prérequis : 3.5, 4.1 et 4.2

Dans la classe **SIMCoulomb** disponible dans le ***package*** **sim.application**, vous allez modifier l'exécution du programme dans la méthode main ( )

public static void main(String[ ] args)

afin de comparer le module de la force exercée par différentes géométries de charges sur une charge ponctuelle. Commencez par identifier dans le main ( ) la méthode suivante :

public static void comparaisonDistribution(**double** Q, **double** q, **double** L, **double** d, **int** N)

Cette méthode permet de calculer et d'afficher des forces électriques appliquées par une sphère, une TRIUC et une PPIUC de charge Q sur une particule ponctuelle de charge q située à une distance d du centre de la distribution de charges. Le paramètre N correspond au nombre d'éléments discrets de charges représentant la géométrie. Le paramètre L correspond à la longueur de la géométrie. Même si la méthode a déjà été implémentée pour vous, vous être invités à y jeter un coup d'œil.

Dans l'appel de la méthode comparaisonDistribution(…), calculez vos forces électriques de vos trois distributions de charges avec les paramètres suivants :

$$Q = 3\times10^{-6}\,\text{C} \qquad q = 1\times10^{-6}\,\text{C} \qquad L = 2\ \text{m} \qquad d = 1\ \text{cm} \qquad N = 1\,000\,000$$

Notez vos résultats dans votre document ***Rapport de laboratoire***. Recommencez l'exécution avec les distances

$$d = 2\ \text{cm} \quad \text{ainsi que} \quad d = 4\ \text{cm}$$

et notez vos résultats.

En comparant les modules des forces au fur et à mesure que l'on a augmenté la distance d où d correspond au paramètre R dans les représentations précédentes (voir La discrétisation de la charge), quelle est la relation de proportionnalité de la force électrique en fonction de la distance R pour les trois géométries ?

Choisissez l'une de ces réponses et inscrivez-la dans votre document ***Rapport de laboratoire***.

Choix des relations de proportionnalité				
$F_e \propto \frac{1}{R^2}$	$F_e \propto \frac{1}{R}$	$F_e \propto 1$ (constant)	$F_e \propto R$	$F_e \propto R^2$

FAÎTES VALIDER VOTRE EXÉCUTION PAR VOTRE PROFESSEUR À L'AIDE D'UNE SIGNATURE DANS VOTRE DOCUMENT ***RAPPORT DE LABORATOIRE***.

Conclusion

Félicitations ! Vous avez implémenté avec succès des opérations de base sur les vecteurs à trois dimensions et vous avez réussi à les utiliser dans un but scientifique visant à appliquer la loi de Coulomb.

Remise du programme

Pour effectuer la remise électronique de votre programme, envoyer le fichier ci-dessous dans l'espace de dépôt exigé par votre enseignant (exemple : OMNIVOX/LÉA) :

- Le **répertoire SIM** de votre projet dans un **format compressé** « zip » sous le nom *SIM-Coulomb.zip* comme vous l'avez initialement téléchargé.