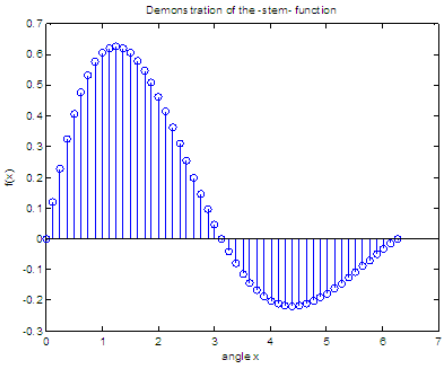
Laboratoire :
Les fonctions discrètes

$$f(x)$$



http://www.matrixlab-examples.com/matlab-plot-3.html

Table des matières

Introduction

Une fonction discrète correspond à un tableau de valeurs dont celles-ci sont reliées par une fonction continue.

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/fonctions/download/SIM-Fonctions.zip

Décompressez le fichier « *SIM.zip* » dans un répertoire « *java* » qui vous permettra de définir votre ***workspace*** lors de l'ouverture du logiciel ***Eclipse***.

À l'ouverture du logiciel ***Eclipse***, dans l'onglet ***File***, choisissez ***Switch Workspace*** et identifiez la localisation de votre répertoire de projet « *java* ». Dans l'onglet ***File***, choisissez ***New*** et prenez ***New Java Project***. Dans la boîte d'édition ***Project name***, entrez le nom de projet *SIM* (le nom du répertoire du projet contenu dans le fichier *SIM.zip*). Vous avez maintenant configuré votre environnement de développement.

Pour la remise de votre laboratoire, vous devez compléter le document ***Rapport de laboratoire*** disponible au lien suivant

http://physique.cmaisonneuve.qc.ca/svezina/projet/fonctions/download/Rapport-Fonctions.pdf

et obtenir des signatures de votre enseignant(e) ainsi que de rédiger des solutions écrites à quelques questions conceptuelles.

Description générale du programme

L'application permettant de construire des fonctions f, de faire des opérations mathématiques de base entre elles (addition, multiplication), d'appliquer des opérations mathématiques avancées sur elles (puissance, exponentiel, sinus), des opérations différentielles comme la dérivée et l'intégrale et de visualiser le résultat sous la forme d'un graphique. La classe à exécuter sera la ***classe*** SIMFunction qui est disponible dans le ***package*** sim.application.

Pour obtenir toutes les fonctionnalités du programme, vous allez compléter l'écriture des classes suivantes :

- SIMFunction (***package*** sim.application)

Cette classe représente l'application qui va réaliser les calculs sur les fonctions et les afficher.

- SRealFunction (***package*** sim.math)

Cette classe représente la fonction f. On y retrouve les données sous la forme d'un tableau effectuant la correspondance entre les abscisses x de la fonction et les ordonnées y de la fonction ainsi que des fonctionnalités permettant de réaliser des opérations mathématiques sur la fonction.

- SArrays (***package*** sim.util)

Cette classe utilitaire réalise des calculs sur des tableaux de données. Ces fonctionnalités seront intégrées dans la classe SRealFunction.

- SDiscreteFunction (***package*** sim.math)

Cette classe utilitaire permet d'analyser les données d'un tableau afin d'effectuer du calcul différentiel et intégral. Ces fonctionnalités seront intégrées dans la classe SRealFunction.

1.1 – La première exécution du programme

Fichier à modifier : SIMFunctions.java

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 SIMFunctions.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 exécute la méthode

```
private static void etape_1_1( ) .
```

Lisez le code qui se retrouve dans cette méthode afin de réaliser que les tâches effectuées sont les suivantes :

1) Définition du domaine $x \in [-1, 8]$ de la fonction $f(x)$ à l'aide des variables

```
double  x_min = -1.0;
double x_max = 8.0;
```

2) Définition du nombre d'éléments discrets

```
int nb = 2000;
```

de la fonction à 2000 éléments.

3) Construction de la fonction $f(x) = 1$ à l'aide de l'instruction

```
SRealFunction f = SRealFunction.one(nb, x_min, x_max);
```

de la classe SRealFunction.

4) Construction d'un objet SChart à l'aide des instructions

```
int height = 500;
double y_min = -1.0;
double y_max = 2.0;
String title = "Graphique : f(x) = 1";
SChart chart = new SChart(f, y_min, y_max, height, title);
```

permettant la représentation de la fonction dans un graphique ayant 500 pixels de hauteurs et 2000 pixels de largeurs (égal à nb de la fonction f). L'axe des x est affiché entre $[-1, 8]$ et l'axe des y est affiché entre $[-1, 2]$.

5) Écriture d'un fichier image *etape_1_1.png* correspondant au graphique chart à l'aide des instructions

```
String file_name = "etape_1_1.png";
SImageUtil.writeImagePNG(file_name, chart.getBufferedImage());
```

Exécutez le programme et constatez la présence de votre fichier *etape_1_1.png* dans le répertoire de votre projet. Vérifier que l'illustration de la fonction est bel et bien une ligne horizontale correspondant à $f(x) = 1$.

Question 1.1 :

Identifiez un avantage de définir une fonction numérique avec une valeur de nb très grande ?

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***.

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.

Question 1.2 :

Pourquoi est-il utile d'écrire en informatique des tests unitaires plutôt que de tout simplement créer une application validant l'intégralité d'un programme ?

La représentation d'une fonction discrète

Dans ce programme, la représentation informatique d'une fonction discrète est située dans la classe SRealFunction située dans le package ***sim.math***. Dans cette classe, on y retrouve les champs suivants :

1) private final double[ ] fx;

   Ce tableau correspond à toutes les valeurs de la fonction $f(x)$. Puisque la taille du tableau est limitée, la fonction $f(x)$ est bien défini seulement pour un nombre limité de valeurs de x.

2) private final double x_min;

   Cette valeur correspond au début du domaine où la fonction $f(x)$, car son domaine est limité dans l'intervalle $x \in [x_{\min}, x_{\max}]$.

2) private final double x_max;

   Cette valeur correspond à la fin du domaine où la fonction $f(x)$, car son domaine est limité dans l'intervalle $x \in [x_{\min}, x_{\max}]$.

Voici un exemple de représentation :

$f(x) = 5x + 3$ avec $x \in [0, 2]$ pour *N* = 6 éléments discrets

tel que la distance $\mathrm{d}x$ entre chaque éléments du domaine discret est $\mathrm{d}x = 0,4$:

x	0	0,4	0,8	1,2	1,6	2
	fx[0]	fx[1]	fx[2]	fx[3]	fx[4]	fx[5]
$f(x)$	3	5	7	9	11	13

Tout au long du laboratoire, vous allez ajouter des fonctionnalités à cette classe permettant à celle-ci de vous donner les valeurs de la fonction pour n'importe quelle valeur de x à l'intérieur du domaine et d'effectuer des opérations mathématiques sur la fonction. Cette fonction sera par la suite affichée dans un fichier image de type png.

1.3 – La distance entre deux éléments discrets d'une fonction

Fichier à modifier : SDiscreteFunction.java

Prérequis : aucun

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

public static double dx(double[ ] fx, double x_min, double x_max) throws IllegalArgumentException .

Cette méthode permet d'établir la distance $\mathrm{d}x$ entre chaque élément discret d'une fonction délimitée dans un domaine restreint $x \in [x_{\min}, x_{\max}]$ (avec l'usage des paramètres x_min et x_max).

Pour bien définir dx, il est important de respecter la correspondance suivante entre une valeur de x et sa position adéquate (son index) dans le tableau de la fonction :

	1er élément du tableau	2e élément du tableau	2e élément du tableau		Nième élément du tableau
Index	0	1	2	...	f.length - 1
Valeur de x	x_min	x_min + dx	x_min + 2*dx	...	x_max

Présentement, cette méthode retourne une exception de type ***IllegalArgumentException*** si le tableau possède une taille inférieure à 2, car il faut au moins 2 éléments dans un tableau pour définir dx. Puisque le cœur de la méthode n'a pas été écrit, une exception de type SNoImplementationExeception est lancée signifiant que la méthode est présentement non implémentée. Effacez la ligne

```
throw new SNoImplementationExcpetion("La méthode n'a pas été implémentée.");
```

et remplacez-là par votre implémentation.

Développez une équation mathématique simple permettant de réaliser le calcul de la distance dx et retournez le résultat dans l'implémentation de votre méthode.

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

Question 1.3 :

Selon-vous, quel problème informatique pouvez-vous anticiper si vous contruisez une fonction numérique sur le domaine [0, 1] avec une valeur de dx extrêmement petite ?

1.4 – La construction de la fonction *x*

Fichier à modifier : SRealFunction.java et SIMFunction.java

Prérequis : 1.3

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

```
public static SRealFunction x(int nb, double x_min, double x_max)
```

.

Cette méthode a pour but de générer la fonction

$$f(x) = x \quad .$$

Présentement, cette méthode retourne une exception de type SNoImplementationException spécifiant que la méthode n'a pas été implémentée. Dans votre implémentation, vous devrez :

(1) Allouer de l'espace mémoire (avec instruction new) pour un tableau de type ***double*** avec nb éléments. Vous pouvez utiliser le nom de variable tab pour votre tableau.

(2) Utiliser l'instruction

```
double dx = SDiscreteFunction.dx(tab, x_min, x_max);
```

pour évaluer la distance dx entre chaque élément discret de votre fonction ce qui vous permettra d'utiliser la méthode que vous venez tous juste de programmer.

(3) Remplir le tableau tab avec les valeurs de la fonction $f(x) = x$ en utilisant dx. Les valeurs devraient correspondre à l'exemple ci-dessous :

x	x_min	x_min + dx	x_min + 2dx	...	x_max
	fx[0]	fx[1]	fx[2]	...	fx[N-1]
$f(x)$	x_min	x_min + dx	x_min + 2dx	...	x_max

(4) Retourner la construction d'un objet de type SRealFunction avec l'instruction

```
return new SRealFunction(tab, x_min, x_max);
```

en utilisant le tableau tab que vous avez rempli ainsi que le début et la fin du domaine x_min et x_max.

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

Par la suite, dans la classe SIMFunctions.java du ***package*** sim.application, regardez rapidement le contenu de la méthode

```
private static void etape_1_4( )
```

et remarquez que le code est semblable à celui de la méthode etape_1_1( ) sauf pour les éléments suivants :

1) La fonction $f(x) = x$ est générée (et non pas $f(x) = 1$).
2) Le nom du graphique a été changé ainsi que le nom du fichier image.

Exécutez le programme et constatez la présence de votre fichier *etape_1_4.png* dans le répertoire de votre projet.

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***.

Question 1.4 :

Pouvez-vous formuler deux arguments en lien avec l'aspect visuel de votre fonction f qui justifient qu'elle est bel égale à la fonction $f(x) = x$.

Les opérations sur les fonctions

Voici les différentes opérations que l'on peut appliquer sur une fonction :

L'opération de base	Représentation mathématique de l'opération
Évaluer une fonction	$y = f(x)$
L'addition d'une fonction avec une constante	$g(x) = f(x) + A$
Additionner deux fonctions	$h(x) = f(x) + g(x)$
Soustraire deux fonctions	$h(x) = f(x) - g(x)$
multiplication d'une fonction par un scalaire	$g(x) = a\, f(x)$
Multiplication d'une fonction avec une fonction	$h(x) = f(x)g(x)$
Division d'une fonction avec une fonction	$h(x) = \frac{f(x)}{g(x)}$

Les composées de fonctions	Représentation mathématique de l'opération
Puissance d'une fonction	$g(x) = (f(x))^n$
L'exponentiel d'une fonction	$g(x) = e^{f(x)}$
Sinus d'une fonction	$g(x) = \sin(f(x))$
Cosinus d'une fonction	$g(x) = \cos(f(x))$

Les opérations différentielles	Représentation mathématique de l'opération
La dérivée d'une fonction	$f'(x) = \frac{\mathrm{d}\, f(x)}{\mathrm{d}x}$
L'intégrale d'une fonction	$F(x) = \int f(x)\mathrm{d}x + C$

L'objectif du laboratoire sera d'implémenter plusieurs méthodes permettant de construire n'importe quelle fonction $f(x)$ à partir des deux fonctions $f(x) = 1$ et $f(x) = x$ en utilisant différentes opérations mathématiques appliquées sur celles-ci.

Par exemple, vous pourrez construire les fonctions suivantes à l'aide d'une séquence d'instructions :

La fonction $f(x)$	La séquence d'instructions

Instruction préliminaire	double N = 50; // pour une fonction discrète à 50 éléments.
	double x_min = 1.0; // pour un domaine débutant à *x* = 1.0.
	double x_max = 8.0; // pour un domaine terminant à *x* = 8.0.
	
	// Construction de la fonction f(x) = 1.
	SRealFunction one = SRealFunction.one(nb, x_min, x_max);
	
	// Construction de la fonction f(x) = x.
	SRealFunction x = SRealFunction.x(nb, x_min, x_max);

$f(x)=7$	SRealFunction f = one.multiply(7);
$f(x)=4x+8$	SRealFunction f = x.add(x).add(x).add(x).add(8); SRealFunction f = x.multiply(4).add(8); SRealFunction f = x.multiply(4).add(one.multiply(8)); SRealFunction f = one.multiply(8).add(x.multiply(4));
$f(x)=2x^2+x^4+5$	SRealFunction f = x.multiply(x).multiply(2).add(x.pow(4.0)).add(5);
$f(x)=\cos(2x+5)+e^x$	SRealFunction f = RealFunction.cos(x.multiply(2).add(5)).add(SRealFunction.exp(x));

2.1 - L'addition de deux tableaux

Fichier à modifier : SArrays.java

Prérequis : aucun

Dans la classe SArrays disponible dans le ***package*** sim.util, vous allez programmer la méthode

public static double[ ] add(double[ ] tab1, double[ ] tab2) throws ArrayIndexOutOfBoundsException .

Présentement, cette méthode retourne une exception de type SNoImplementationException spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation afin de faire l'addition des deux tableaux index par index.

Pour ce faire, vous devrez :

(1) Allouer de l'espace mémoire pour le résultat de votre opération mathématique.
(2) Remplir le tableau avec le résultat de l'opération mathématique index par index.
(3) Retourner le nouveau tableau.

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

2.2 - L'addition de deux fonctions

Fichier à modifier : SRealFunction.java et SIMFonctions.java

Prérequis : 2.1

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

```
public SRealFunction add(SRealFunction f) throws SInvalidFunctionException .
```

Présentement, cette méthode retourne une exception de type SNoImplementationException spécifiant que la méthode n'a pas été implémentée. Votre tâche sera d'intégrer la méthode

```
public static double[ ] add(double[ ] tab1, double[ ] tab2)  throws ArrayIndexOutOfBoundsException
```

que vous avez programmé dans la classe SArrays à une fonctionnalité de la classe SRealFunction.

Pour ce faire, introduisez l'instruction suivante :

```
return new SRealFunction(SArrays.add(this.fx, f.fx), this.x_min, this.x_max);
```

Cette instruction retournera une nouvelle fonction contenant les informations de la fonction qui appel la méthode add (this) avec la fonction à additionner (f).

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

2.3 - La construction d'une fonction linéaire

Fichier à modifier : SIMFonctions.java

Prérequis : 2.2

Dans la classe SIMFunctions.java du ***package*** sim.application, modifiez la méthode

```
private static void etape_2_3( )
```

afin qu'elle puisse afficher le graphique de la fonction

$$f(x) = 3x + 1$$

Pour y arriver, vous devrez :

1) Construire la fonction $f(x) = x$ à l'aide de l'instruction

```
SRealFunction x = SRealFunction.x(nb, x_min, x_max);
```

2) Construire la fonction $f(x) = 3x + 1$ à l'aide de la variable x représentant la fonction $f(x) = x$ et des méthodes

```
public SRealFunction add(SRealFunction f)
public SRealFunction add(double a)
```

de la classe **SRealFunction**. Vous ne pourrez pas utiliser la méthode multiply puisqu'elle n'a pas encore été intégrée au programme.

3) Affecter le résultat de votre calcul à la variable f, car c'est elle qui sera utilisée en référence pour l'affichage du graphique.

Exécutez le programme et constatez la présence de votre fichier *etape_2_3.png* dans le répertoire de votre projet. Vérifiez que l'illustration de la fonction est bel et bien une droite correspondant à $f(x)=3x+1$ étant la simplification algébrique de la fonction exigée.

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***.

2.4 - Les opérations de base sur deux fonctions

Fichier à modifier : SArrays.java

Prérequis : aucun

Dans la classe SArrays disponible dans le ***package*** sim.util, vous allez programmer les 5 méthodes suivantes :

```
public static double[ ] substract(double[ ] tab1, double[ ] tab2) throws ArrayIndexOutOfBoundsException
public static double[ ] multiply(double[ ] tab, double a)
public static double[ ] multiply(double[ ] tab1, double[ ] tab2) throws ArrayIndexOutOfBoundsException
public static double[ ] divide(double[ ] tab1, double[ ] tab2) throws ArrayIndexOutOfBoundsException
public static double[ ] pow(double[ ] tab, double n) throws ArrayIndexOutOfBoundsException
```

Présentement, cette méthode retourne une exception de type SNoImplementationException spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation de chacune de ces méthodes afin de réaliser l'objectif précisé dans la documentation (voir ***javadoc***) de celle-ci.

N'hésitez pas à réutiliser le code de la méthode

```
public static double[ ] add(double[ ] tab1, double[ ] tab2) throws ArrayIndexOutOfBoundsException
```

de la classe SArrays et de le modifier légèrement afin qu'il réponde aux objectifs des nouveaux calculs. N'oubliez pas de retourner un nouveau tableau afin de préserver le contenu des tableau tab1 et tab2 (il ne faudrait pas que l'opération mathématique puisse altérer leur contenu).

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

Dans la classe SRealFunction.java disponible dans le ***package*** sim.math, vous n'aurez pas à intégrer ces nouvelles fonctionnalités (substract, multiply, divide, pow) dans la classe comme vous l'avez fait pour la méthode add, car ce travail a déjà été réalisé pour vous.

2.5 - La construction d'une fonction complexe

Fichier à modifier : SIMFunction.java

Prérequis : 2.2 et 2.4

Dans la classe SIMFunctions.java du ***package*** sim.application, modifiez la méthode

```
private static void etape_2_5( )
```

à l'endroit indiqué dans le code (voir commentaire dans le programme) afin qu'elle puisse afficher le graphique de la fonction

$$f(x)=4x^2-5x+\frac{1}{x-2}\ .$$

Pour y arriver, vous devrez utiliser les méthodes de base (addition, soustraction, multiplication, division, puissance) de la classe **SRealFunction** sur un objet représentant la fonction x.

Pour générer la partie $\frac{1}{x-2}$ de la fonction f, vous pourrez exécuter l'une des deux options suivantes :

Option 1) Utilisez l'opération pow afin de mettre l'expression $x-2$ à la puissance -1.

Option 2) Construire la fonction unitaire $g(x)=1$ dans le domaine x_min et x_max à l'aide de l'instruction

```
SRealFunction g = SRealFunction.one(nb, x_min, x_max);
```

et la diviser par l'expression $x-2$.

Exécutez le programme et constatez la présence de votre fichier *etape_2_5.png* dans le répertoire de votre projet.

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***.

Question 2.5 :

À l'aide d'un calcul de limite, démontrez la présence de l'asymptote verticale chez la fonction

$$f(x)=4x^2-5x+\frac{1}{x-2} \quad .$$

La dérivée d'une fonction discrète

La dérivée est une opération mathématique permettant de générer une fonction $f'(x)$ correspondant à la pente en tout point d'une fonction $f(x)$. En mathématique, la dérivée correspond au calcul suivant :

$$f'(x)=\lim_{\Delta x\to 0}\frac{f(x+\Delta x)-f(x)}{\Delta x}$$

Cependant, cette définition n'est pas unique. Voici d'autres définitions équivalentes de la dérivée :

La dérivée à droite	La dérivée à gauche	La dérivée au centre
$f'(x)=\lim_{\Delta x\to 0}\frac{f(x+\Delta x)-f(x)}{\Delta x}$	$f'(x)=\lim_{\Delta x\to 0}\frac{f(x)-f(x-\Delta x)}{\Delta x}$	$f'(x)=\lim_{\Delta x\to 0}\frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x}$

Lorsque l'on utilise une fonction discrète, on peut évaluer la dérivée de la fonction pour un élément i à l'aide de deux éléments pour effectuer le calcul de la pente de la fonction :

x	x_min	x_min + dx	...	x_min + i dx	...	x_max
$f(x)$	fx[0]	fx[1]	...	fx[i]	...	fx[N-1]
$f'(x)$	(fx[1] - fx[0]) / dx (à droite)	(fx[2] - fx[0]) / 2dx (au centre)	...	(fx[i+1] - fx[i-1]) / 2dx (au centre)	...	(fx[N-1] - fx[N-2]) / dx (à gauche)

3.1 - Le calcul de la dérivée discrète

Fichier à modifier : SDiscreteFunction.java

Prérequis : aucun

Dans la classe SDiscreteFunction disponible dans le ***package*** sim.math, vous allez programmer les méthodes suivantes :

public static double rightDerivate(int i, double[ ] fx, double dx) throws IndexOutOfBoundsException

L'objectif de cette méthode est d'effectuer le calcul de la dérivée à droite d'une fonction selon l'équation

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \qquad \text{ce qui donnera} \qquad f'_{[i]} = \frac{f_{[i+1]} - f_{[i]}}{\Delta x}$$

et d'obtenir le résultat pour une valeur x du domaine. Cette version de la dérivée porte le nom de « dérivée par la droite » puisque la pente de la fonction est évaluée avec la fonction f grâce aux valeurs x et $x + \Delta x$.

Présentement, cette méthode retourne une exception de type SNoImplementationException spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation de cette méthode afin qu'elle réalise le calcul de la dérivée discrète à droite pour l'élément en indice i d'un tableau fx représentant la fonction f.

Par la suite, vous allez programmer la méthode suivante :

public static double leftDerivate(int i, double[ ] fx, double dx) throws IndexOutOfBoundsException

L'objectif de cette méthode est d'effectuer le calcul de la dérivée à gauche d'une fonction selon l'équation

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x) - f(x-\Delta x)}{\Delta x} \qquad \text{ce qui donnera} \qquad f'_{[i]} = \frac{f_{[i]} - f_{[i-1]}}{\Delta x}$$

et d'obtenir le résultat pour une valeur x du domaine. Cette version de la dérivée porte le nom de « dérivée par la gauche » puisque la pente de la fonction est évaluée avec la fonction f grâce aux valeurs $x - \Delta x$ et x. Complétez l'implémentation de cette méthode afin qu'elle réalise le calcul de la dérivée discrète à gauche.

Finalement, vous allez programmer la méthode suivante :

public static double middleDerivate(int i, double[ ] fx, double dx) throws IndexOutOfBoundsException

L'objectif de cette méthode est d'effectuer le calcul de la dérivée au centre d'une fonction selon l'équation

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \qquad \text{ce qui donnera} \qquad f'_{[i]} = \frac{f_{[i+1]} - f_{[i-1]}}{2\Delta x}$$

et d'obtenir le résultat pour une valeur x du domaine. Cette version de la dérivée porte le nom de « dérivée au centre » puisque la pente de la fonction est évaluée avec la fonction f grâce aux valeurs $x - \Delta x$ et $x + \Delta x$. Complétez l'implémentation de cette méthode afin qu'elle réalise le calcul de la dérivée discrète au centre.

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

Question 3.1 :

Selon vous, est-il possible d'envisager de réaliser une dérivée numérique avec plus de précision, par exemple, en utilisant 5 points sur la fonction f au lieu de 3 points ? Si oui, présentez vos sources ou écrivez l'équation correspondante.

3.2 - Le choix du type de dérivée

Fichier à modifier : SDiscreteFunction.java

Prérequis : 3.1

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

public static double dfdx(int i, double[ ] fx, double dx)

L'objectif de cette méthode d'exécuter le bon type de dérivée (gauche, droite ou centre) sur un élément i d'un tableau fx à l'aide d'un certain dx. Le choix du type de dérivée dépend de l'élément i du tableau contenant N valeurs :

- Pour $i = 0$: Il faut prendre la dérivée à droite (rightDerivate), car l'élément $i = -1$ n'existe pas.
- Pour $i = N - 1$: Il faut prendre la dérivée à gauche (leftDerivate), car l'élément $i = N$ n'existe pas.
- Pour $i \in [1, N-2]$: Il faut prendre la dérivée au centre (middleDerivate), car elle correspond à la meilleure estimation de la dérivée étant une moyenne de celle à gauche et à droite.

Présentement, cette méthode retourne une exception de type SNoImplementationException spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation de cette méthode en retournant le calcul de la dérivée sur l'élément i du tableau fx.

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

3.3 - Appliquer la dérivée sur un tableau

Fichier à modifier : SDiscreteFunction.java

Prérequis : 3.2

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

public static double[ ] derivate(double[ ] fx, double x_min, double x_max)

L'objectif de cette méthode est de générer un nouveau tableau contenant pour chaque élément i du nouveau tableau la valeur de la dérivée du tableau fx en position i.

Présentement, cette méthode retourne une exception de type SNoImplementationException spécifiant que la méthode n'a pas été implémentée. Complétez l'implémentation de cette méthode en effectuant les tâches suivantes :

1) Allouer de l'espace mémoire pour enregistrer vos valeurs correspondant à la dérivée du tableau fx.
2) Définir la valeur de $\mathrm{d}x$ avec la méthode
   public static double dx(double[ ] fx, double x_min, double x_max) throws IllegalArgumentException
3) Remplir le nouveau tableau avec les valeurs de la dérivée en utilisant la méthode
   public static double dfdx(int i, double[ ] fx, double dx)
4) Retourner le nouveau tableau.

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

Dans la classe SRealFunction.java disponible dans le ***package*** sim.math, vous n'aurez pas à intégrer cette nouvelle fonctionnalité à la classe, car ce travail a déjà été réalisé pour vous.

3.4 - La dérivée d'un polynôme du 2e degré

Fichier à modifier : SIMFunction.java

Prérequis : 2.2, 2.4 et 3.3

Dans la classe SIMFunctions.java du ***package*** sim.application, modifiez la méthode

```
private static void etape_3_4( )
```

à l'endroit indiqué dans le code (voir commentaire dans le programme) afin qu'elle puisse afficher le graphique de la fonction

$$f(x) = 3x^2 + 5x - 6$$

et le graphique de la fonction

$$g(x) = f'(x) \ .$$

Ne construisez pas la fonction g en utilisant vos connaissances en mathématique vous permettant de calculer analytiquement le résultat de la dérivée de la fonction f.

Utilisez plutôt la méthode

```
public SRealFunction derivate( )
```

de la classe SRealFunction sur l'objet f. Ainsi, utiliser l'instruction

```
g = f.derivate( )
```

permettra d'obtenir $g(x) = f'(x)$.

Exécutez le programme et constatez la présence de votre fichier *etape_3_4a.png* correspondant à la fonction f et votre fichier *etape_3_4b.png* correspondant à la fonction g.

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***.

Question 3.4a :

Pouvez-vous formuler deux arguments en lien avec l'aspect visuel de vos deux fonctions f et g qui justifient que g est bel et bien la dérivée de f.

Question 3.4b :

Démontrez, pour la fonction $f(x) = 3x^2 + 5x - 6$ que la définition de la dérivée au centre

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}$$

donne bien la fonction dérivée calculée numériquement.

3.5 - La loi de Planck

Fichier à modifier : SIMFunction.java

Prérequis : aucun

Dans la classe SIMFunctions.java du ***package*** sim.application, modifiez la méthode

private static void etape_3_5( )

à l'endroit indiqué dans le code (voir commentaire dans le programme) afin qu'elle puisse afficher le graphique de la **loi de Planck**[1]

$$I(\lambda) = \frac{2\pi hc^2}{\lambda^5\left(e^{hc/\lambda kT} - 1\right)}$$

pour une température de $T = 3000\ \text{K}$ ainsi que sa dérivée dont les paramètres sont les suivants :

$I_\lambda(\lambda)$: Intensité de la lumière pour une bande de longueur d'onde autour de λ (W/m^3).

λ : Longueur d'onde de la lumière (m).

T : Température du corps noir (K).

h : Constante de Planck ($h = 6{,}63\times10^{-34}\ \text{J}\cdot\text{s}$).

c : Vitesse de la lumière ($c = 3\times10^{8}\ \text{m/s}$).

k : Constante de Boltzmann ($k = 1{,}38\times10^{-23}\ \text{J/K}$).

Puisque λ est la variable de la **loi de Planck**, remplacez cette variable dans votre code par votre fonction x.

Pour effectuer le calcul de « $e^{hc/\lambda kT}$ », utilisez la méthode statique

public static SRealFunction exp(RealFunction f)

pour mettre en exponentiel une fonction. Cette méthode a déjà intégrée pour vous.

Exécutez le programme et constatez la présence de votre fichier *etape_3_5a.png* correspondant à la **loi de Planck** et le fichier *etape_3_5b.png* correspondant à la dérivée de la **loi de Planck** par rapport à la longueur d'onde.

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***.

Question 3.5 :

Décrivez en mots, à l'aide d'argument basé sur le calcul différentiel, comment on peut identifier clairement la longueur d'onde (représentée par le paramètre *x* de vos graphiques) où il y a valeur maximale (et non minimale) chez la **loi de Planck** ?

[1] La loi de Planck permet de déterminer la « quantité d'énergie lumineuse » d'une couleur λ émise par un corps (ex : Soleil) selon la température de celui-ci.

Conclusion

Félicitations ! Vous avez implémenté avec succès les fonctionnalités d'une fonction mathématique. Vous pouvez maintenant utiliser ce programme pour effectuer tout type de calcul sur des fonctions à une variable et d'y afficher le résultat sous forme de graphique.

Remise du programme

Pour effectuer la remise de votre programme, envoyer le fichier ci-dessous à l'adresse suivante :

svezina@cmaisonneuve.qc.ca

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