Projet intégrateur : informatique

1ère thématique transversale

Propagation de la chaleur dans une plaque

CB1 2007-2008

Pour la première thématique transversale du projet intégrateur, l'objectif de la session informatique est que vous réalisiez un **outil de visualisation** de vos résultats expérimentaux – obtenus lors de la session de physique – et de la solution analytique correspondante – vue à la session de mathématiques. Cet outil vous aidera à comparer ces résultats et donc à confronter phénomène expérimental et prévisions théoriques.

La session comprend trois séances de TP. Lors de la première séance, vous définirez, par groupes de 5-6 élèves, le cahier des charges technique de l'application à réaliser. Les deux autres séances seront consacrées à la réalisation et la validation de l'application.

Ce document présente la session informatique pour la première thématique transversale du projet intégrateur. Il comprend :

- section 1 : une description du déroulement de la session
- section 2 : énoncé du TP1 – *cahier des charges et modélisation*
- section 3 : énoncé du TP2 – *calcul et visualisation d'une solution théorique*
- section 4 : énoncé du TP3 – *visualisation des résultats expérimentaux*

1 Objectifs et déroulement de la session informatique

1.1 Objectifs

Dans la session *physique*, vous avez mené une expérimentation sur le processus de refroidissement d'une plaque (ou d'un barreau) au cours de sa trempe et mesuré l'évolution temporelle de la température dans le solide. Dans la session *mathématiques*, vous avez modélisé ce phénomène de manière théorique et exprimé cette évolution par une fonction analytique. La phase suivante est, logiquement, la confrontation des résultats expérimentaux et théoriques avec, dans un premier temps, une représentation graphique de ces résultats. Durant la session informatique, vous allez développer une programme informatique en Java dans cette intention.

1.2 Organisation de la session et rendu des travaux

L'application sera réalisée en séances de TP et sur le temps de travail personnel :

- la première séance de TP (2h – vendredi 1er février) sera consacrée à l'**élaboration du cahier des charges technique** : par groupes de 5 à 6 élèves, il s'agira de spécifier les fonctionnalités et les méthodes de développement de l'application. Une série de questions posées en TP guidera votre réflexion. Le document rédigé (1 par groupe) devra être remis au début de la séance de TP suivante ;
- la seconde séance de TP (2h – vendredi 8 février) sera consacrée à l'implémentation d'une partie de l'application : **le calcul et la visualisation de la solution théorique**. Un squelette du programme en Java, à compléter, vous sera fourni avant le début de la séance. Votre programme (1 par élève) devra être déposé sur Campus à la fin de la séance. Il sera évalué et commenté par votre encadrant avant la séance de TP suivante ;
- la troisième séance de TP (1h – vendredi 15 février) sera consacrée à l'implémentation de la seconde partie de l'application : **la visualisation des résultats expérimentaux**. Votre programme complété devra de nouveau être déposé à la fin de la séance. Il sera de nouveau évalué et commenté par votre encadrant dans la semaine qui suit ;

– à l'issue de ces trois séances de TP, vous aurez une semaine supplémentaire pour prendre en compte les remarques de vos correcteurs afin de compléter et d'améliorer votre application, à déposer sur Campus avant le vendredi 22 février, 20h.

1.3 Évaluation

La note de la partie informatique du projet intégrateur (1ère thématique transversale) comprend :
– l'évaluation (collective) du cahier des charges (20%)
– l'évaluation des codes rendus à l'issue des TP 2 et 3 (40%)
– l'évaluation de l'application finale (40%)

L'avancement et la qualité de votre implémentation seront évalués selon les critères suivants :
– **conformité du programme** : tout programme rendu doit, *a minima*, compiler et s'exécuter. Les parties de code développées en TP, non terminées ou qui ne s'exécuteraient pas à l'issue de la séance, devront être placées en commentaire. Le programme rendu au final devra être validé par les tests.
– **justesse de la programmation et des algorithmes** : appel de méthodes, emploi des variables, passage de paramètres, parcours de tableaux, emploi des boucles, etc.
– **qualité (clarté) du code** : les règles de programmation énoncées dans les cours de programmation et d'algorithmique doivent être respectées.

Les documents à remettre (cahier des charges, fichiers de code source) seront à déposer sur Campus, dans l'espace intégrateur 1ère année. Le retour des évaluateurs se fera aussi par l'intermédiaire de Campus.

Notre objectif est que chaque élève pratique et développe sa propre application : **la fraude et la copie seront lourdement sanctionnées** (=0, sans préjuger d'éventuelles suites disciplinaires).

1.4 Groupes et enseignants

Philippe David, Sophie Demassey, Rémi Douence, Jérôme Fortin, Assia Hachichi, Nicolas Loriant. Resp : Sophie.Demassey@emn.fr, bureau B210.

2 TP1 : cahier des charges et modélisation

2.1 Objectif et déroulement de la séance TP1

Comme toujours, avant de débuter la réalisation d'une application, il est nécessaire de savoir précisément ce que l'on veut faire et comment on va le faire. Pour l'implémentation d'un logiciel, il s'agit d'identifier de manière **exhaustive** :
– la finalité du logiciel : pour qui ? pour quoi ? qu'est-il attendu en sortie ?
– les données d'entrée : quelles sont les valeurs à fournir en entrée ? lesquelles sont fixes pour chaque exécution du logiciel (constantes) ? lesquelles sont fixes à l'exécution mais peuvent être modifiées avant chaque exécution (paramètres) ?
– les technologies (langages, bibliothèques,...) de développement : lesquelles sont les plus adaptées au problème ? lesquelles sont les plus adaptées au projet (en fonction des coût, délai, compétences...) ?
– la modélisation du problème : comment transcrire le problème en langage informatique de manière à favoriser la maintenance du code, notamment en divisant la problématique – et donc le logiciel – en une série de sous-problèmes, résolus chacun par un programme informatique.
– les tests de qualité : que faudra-t-il tester pour s'assurer que le logiciel répond bien au besoin dans tous les cas d'utilisation possibles ?

Ceci entre – ainsi que bien d'autres aspects techniques (définition des tests de performance, du contrôle de versions, du mode de documentation, etc.), mais aussi juridiques, financiers, humains, temporels, etc. – dans la constitution du *cahier des charges* de toute application informatique. La première séance de TP est organisée pour répondre à cette attente.

Lors de cette première séance, vous allez, par groupes de 5 à 6 élèves, entamer la rédaction d'un mini cahier des charges technique. En séance, vous discuterez la série de questions de la section 2.2 qui vous servira de guide à la réflexion. Vous remettrez (en version manuscrite : à votre encadrant ou en version

électronique : sur Campus), au plus tard au début de la séance de TP suivante, le bilan de ces discussions (de 2 à 3 pages), rédigé suivant le plan proposé en section 2.3.

2.2 Questions à discuter

2.2.1 La finalité du logiciel

- quelles sont les données que vous souhaitez comparer ? quels sont les critères de comparaison ?
- comment représenter visuellement ces données ? comment comparer visuellement ces données ? quels types de graphiques sont les plus adaptés ?

2.2.2 Les spécifications et données d'entrée du logiciel

- quels sont les calculs ou algorithmes à effectuer ?
- quelles sont les données nécessaires à ces calculs ? différenciez paramètres, constantes, variables.

2.2.3 Les outils de développement

- quels sont les outils envisageables pour une telle représentation graphique ?
- quels sont les avantages d'une solution informatique de calcul et de tracé ?
- quelles technologies informatiques (hors hardware) sont nécessaires à la réalisation d'une telle solution ?

La technologie pour le développement de votre projet est imposée : Java.

- Pour l'affichage des courbes, vous utiliserez une bibliothèque spécifique JFreeChart (voir section 3.5). Une interface à cette bibliothèque vous sera également fournie de sorte que, pour le tracé d'une courbe, il vous suffira de fournir une liste de points appartenant à cette courbe.
- Pour quelles autres fonctionnalités de votre programme, serez-vous amenés à utiliser certaines bibliothèques (standards) du JDK ?
- Commencez la modélisation du projet : quelles sont les fonctionnalités du programme propres aux éléments expérimentaux et celles propres aux éléments théoriques ?

2.3 Plan du mini-cahier des charges de l'application

1. Description sommaire de l'application
   (a) Perspective
   exprimez en 4-5 lignes l'application à concevoir
   (b) Fonctionnalités
   listez les fonctionnalités proposées
   (c) Contraintes
   donnez au plus 3 contraintes relatives à l'utilisation de cette application : environnement, restrictions, prérequis
2. Spécifications
   pour chaque fonctionnalité, décrivez : l'algorithme ou le calcul effectué, les données en entrée (on se réfèrera au dictionnaire ci-dessous), les données en sortie
3. Dictionnaire des données
   listez l'ensemble des données d'entrée nécessaires pour chaque fonctionnalité de l'application. Précisez (si connu/applicable) : le nom, le symbole mathématique, l'élément représenté, l'unité de mesure, le type/format, la précision décimale, l'ordre de valeurs, la valeur par défaut,...

3 TP2 : calcul et visualisation de la solution théorique

3.1 Objectif de la séance

Vous allez réaliser, de manière individuelle, la première partie de votre application. Dans un programme Java, dont le squelette du code vous est fourni, vous allez implémenter les fonctionnalités suivantes :

1. lecture, dans un fichier texte, des valeurs numériques nécessaires au calcul et au tracé des courbes ;
2. calcul de la température théorique $T(x,t)$, à un point x de la plaque et un instant t, donnée par la formule analytique de développement en série de Fourier ;
3. tracé des courbes de température théorique :
   - évolution temporelle de la température en un ou plusieurs points x ;
   - profil de température dans la plaque à un ou plusieurs instants t.

3.2 Déroulement

Vendredi 8 février 2008, durée : 2h.

Récupérer le projet

Récupérez sur campus l'un ou l'autre des fichiers d'archives nominatifs. Enregistrez et décompressez-le dans votre répertoire de travail :

```
unzip th1_apa_XXX.zip                                    (ou bien)
gunzip th1_apa_XXX.tar.gz ; tar -xvf th1_apa_XXX.tar
```

Modifier le projet

L'archive contient une arborescence de répertoires et de fichiers composant votre projet :

`th1_apa_XXX/`	répertoire racine du projet
`th1_apa_XXX/setup.prop`	fichier de paramètres
`th1_apa_XXX/classes/`	répertoire des fichiers compilés (`.class`)
`th1_apa_XXX/lib/`	répertoire des bibliothèques Java pour le tracé de courbes
`th1_apa_XXX/src/`	répertoire des fichiers sources : `Temperature.java`, `TemperatureAnalytique.java`, `TraceSeries.java`

Il vous est demandé de compléter :

- le fichier texte `setup.prop` (voir section 3.3)
- le fichier source `TemperatureAnalytique.java` (voir section 3.4)
- le fichier source `Temperature.java` (voir section 3.5)

Compiler

L'instruction[1] pour compiler votre projet, depuis le répertoire `th1_apa_XXX/`, est :

```
javac -d classes -cp lib\jfreechart.jar;lib\jcommon.jar src\*.java   (Windows)
javac -d classes -cp lib/jfreechart.jar:lib/jcommon.jar src/*.java   (Linux/MacOS)
```

NB : *Pour compiler avec la bibliothèque* `JFreeChart`*, vous aurez besoin du JDK 1.5 ou plus. Pour vérifier que vous avez la bonne version sur votre machine, exécutez l'instruction :* `javac -version`
Si la version est inférieure à `1.5.x`*, installez un nouveau jdk (voir* `http://www.java.com/fr/`*).*

[1] cette instruction compile (`javac`) l'ensemble des fichiers sources du répertoire `src/*.java` à l'aide de la bibliothèque JFreeChart (`-cp lib/jfreechart.jar`). Le résultat de la compilation est placé dans le répertoire consacré (`-d classes`).

Exécuter

Votre projet se compose de 3 fichiers sources contenant chacun la définition d'une classe. Le point d'entrée du programme – c'est à dire la méthode `main` à exécuter au lancement – se trouve dans la classe `Temperature.java`. Pour exécuter, on pourra (ou non) spécifier en argument le nom du fichier de paramètres :

```
java -cp classes;lib\jfreechart.jar;lib\jcommon.jar Temperature nomFichierOuRien   (Windows)
java -cp classes:lib/jfreechart.jar:lib/jcommon.jar Temperature nomFichierOuRien   (Linux/MacOS)
```

Rendu

Vous déposerez sur Campus, à la fin de la séance (17h50 au plus tard), le fichier d'archives contenant le répertoire entier de votre projet :

`th1_apa_XXX.zip` ou bien `th1_apa_XXX.tar.gz`

Pour la création de l'archive, par exemple, en ligne de commande depuis la racine du projet :

```
cd ..                                    (va au répertoire parent de th1_apa_XXX)
tar cvf th1_apa_XXX.tar th1_apa_XXX/     (créé une archive du répertoire)
gzip th1_apa_XXX.tar                     (compresse le fichier d'archives)
```

Assurez-vous que votre programme ainsi rendu, même inachevé, **compile** et **s'exécute**. À défaut, pensez à mettre en commentaires les parties de code boguées ou incomplètes, accompagnées d'une explication.

Évaluation

Le code que vous aurez produit pendant la séance de TP sera évalué par votre encadrant par un système de bonus/malus sur une note de départ de 60/100. Les constatations suivantes donneront lieu à un malus :

- le code est plagié (-160, la valeur négative sera reportée sur la note globale du thème 1 de la session informatique) ;
- le programme n'a pas été modifié (-60) ;
- le programme ne compile pas ou génère une exception à l'exécution du code (-40) ;
- le code ne respecte pas les conventions Java ou les règles et consignes de lisibilité des cours d'algorithmique et de programmation : indentation, noms et identificateurs, commentaires, constantes et valeurs littérales, emploi des boucles et conditionnelles, etc. (environ -10 par type d'erreur).

Toute fonctionnalité proprement implémentée offre un bonus, avec un plafond global de 100/100.

En somme, il vous est demandé d'avoir produit, à l'issue de ce TP, un programme **individuel, exécutable et clair**, à défaut d'être complet. Vous aurez une semaine, d'ici le prochain TP, pour terminer votre programme... individuel, exécutable, clair, complet et juste.

3.3 La classe `Properties`

Le calcul de la température s'appuie sur différents paramètres (épaisseur de la plaque, diffusivité du matériau,...) qui sont propres à une étude donnée. Pour que votre programme soit utilisable dans différents contextes, sans re-compilation, il est préférable de maintenir ces valeurs numériques en dehors du code source, par exemple dans un fichier texte qui sera *chargé* et *lu* à l'exécution du programme.

La classe `Properties` est une classe standard de la librairie JDK, spécialement dédiée à la manipulation des fichiers de paramètres : elle en facilite le chargement et la lecture (ainsi que l'écriture).

Spécification partielle[2] de la classe `Properties` :

[2] Documentation complète disponible à http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html

`public Properties()`	construit une liste vide de paramètres
`public void load(InputStream is)`	charge une liste à partir d'une entrée (ex : un fichier texte où chaque ligne contient une description de la forme : « `nom` » = « `valeur` »)
`public String getProperty(String nom)`	retourne la chaîne « `valeur` » correspondant au parametre « `nom` »

Le programme actuel... s'exécute avec ou sans arguments. Quand le nom d'un fichier de paramètres est passé en argument du programme, celui-ci sert à initialiser l'instance `sa` et la variable d'instance `properties` de la classe `TemperatureAnalytique`. Autrement, le nom du fichier par défaut est `setup.prop`.

Le fichier texte `setup.prop` contient la définition d'un premier paramètre : le rang `M` de la somme partielle pour le calcul de la température.

L'accesseur en lecture `getRangSommePartielle()` de la classe `TemperatureAnalytique` retourne cette valeur. Il utilise la méthode de classe `double parseDouble(String)` de la classe standard `Double` qui convertit une chaîne de caractère en `double`. Une méthode équivalente existe (dans la classe `Integer`) pour convertir une `String` en `int` :

`public static double parseDouble(String s)`	classe `Double` : convertit `s` en `double`
`public static int parseInt(String s)`	classe `Integer` : convertit `s` en `int`

À partir de cet exemple, vous devez...

1. compléter le fichier `setup.prop` avec l'ensemble des valeurs des paramètres nécessaires ;
2. ajouter les accesseurs en lecture de ces paramètres à la classe `TemperatureAnalytique`.

3.4 Calcul de la température théorique

La solution analytique de propagation de la chaleur s'exprime sous la forme d'une série de Fourier :

$$T(x,t) - T_\infty = (T_0 - T_\infty)\frac{4}{\Pi} \sum_{\substack{m=1,\\ m \text{ impair}}}^{\infty} \frac{\sin(m\omega x)}{m} e^{-\frac{m^2 t}{\tau}}, \quad \forall x \in [0, L],\ \forall t \geq 0 \tag{1}$$

avec :

T_0	température initiale de la plaque	$w = \Pi/L$	pulsation spatiale
T_∞	température extérieure	κ	diffusivité thermique de la plaque
L	épaisseur de la plaque	$\tau = L^2/(\Pi^2\kappa)$	constante de temps

Pour évaluer les valeurs numériques de la température à partir de cette formule, il est nécessaire de recourir au calcul de sommes partielles $S_M(x,t)$ de la série (1). Autrement dit, il n'est possible de calculer qu'une valeur de la température théorique approchée par le calcul d'une somme partielle de rang M suffisamment grand. Dans la session *mathématiques*, vous avez estimé une grandeur acceptable minimale de M. Vous utiliserez cette valeur pour le calcul de la température théorique par votre programme.

1. Donner l'expression de la température normalisée $T'(x',t') = (T(x,t) - T_\infty)/(T_0 - T_\infty)$, en fonction des indices normalisés de position et de temps $x' = x/L$ et $t' = t/\tau$.
2. Implémenter la méthode `public double calculTemperatureNormalisee(double t, double x)` de la classe `TemperatureAnalytique`. Cette méthode prend en arguments un intant normalisé `t` et un point normalisé `x` et retourne la température théorique normalisée T'(`x`, `t`) approchée par la somme partielle de rang M = `getRangSommePartielle()`.

3.5 Tracé des courbes

3.5.1 La classe `TraceSeries` et la bibliothèque JFreeChart

Soit $f : [0, 1] \to \mathbb{R}$ une fonction analytique. Pour tracer la courbe d'une telle fonction, à la main, vous devez calculer un certains nombre de points suffisamment rapprochés puis les relier entre eux pour ainsi dessiner une forme approchée, linéaire par morceaux : il s'agit d'une *interpolation*. Avec l'ordinateur, c'est pareil, vous allez :

1. discrétiser l'espace de définition de la fonction : $0 = x_0 < x_1 < \cdots < x_n = 1$
2. faire calculer la valeur $y_k = f(x_k)$ de la fonction à ces différentes abscisses $x_0, x_1, \ldots, x_n$
3. faire afficher la courbe correspondant à la série de points $(x_k, y_k)_{k=0..n}$ ainsi obtenue

Pour cette troisième étape, vous utiliserez la bibliothèque JFreeChart[3]. Plus exactement, vous ferez appel aux méthodes d'interfaçage déjà définies dans la classe `TraceSeries` (nb : vous n'aurez pas besoin de les modifier) :

`TraceSeries()`	construit un objet vide pour le tracé d'une ou plusieurs courbes sur un même graphique ;
`void ajouteSerieAna(double[][] tab, String nom)`	ajoute une serie de points (x_k, y_k) definissant une courbe analytique `nom` : `tab[k][0]`$= x_k$ et `tab[k][1]`$= y_k$;
`void creeGraphique(String nom, String lX, String lY)`	crée un graphique `nom` dont les axes sont labellés `lX` (abscisse) et `lY` (ordonnée) ;
`void afficheGraphique(String nom)`	affiche le graphique dans une fenêtre ;
`void enregistreGraphique(String nom)`	enregistre le graphique dans le fichier d'image `nomFichier.png`

3.5.2 Tracé du profil de la température

L'objectif du programme est de comparer visuellement les résultats expérimentaux et théoriques. Pour cela, on souhaite obtenir quatre types de graphiques représentatifs de la propagation et de l'évolution de la température à travers la plaque et à travers le temps. La normalisation des données (température, temps et positions) permet la mise à l'échelle des courbes :

- position normalisée : $x' = x/L$ pour toute position $x \in [0, L]$
- temps normalisé : $t' = t/\tau$, pout tout temps $t \geq 0$
- température normalisée : $T'(x, t) = (T(x, t) - T_\infty)/(T_0 - T_\infty)$

NB : Dans la suite du sujet, on ne considère plus que des températures, temps et positions ***normalisés****.*

Le premier type de graphique à représenter est le profil de la température dans la plaque à un instant t donné :

$$T_t : x \mapsto T(x, t), \qquad \forall x \in [0, 1]. \tag{2}$$

Cette fonctionnalité est déjà (très) partiellement implémentée par la méthode `void traceParTemps(double)` de la classe `Temperature`. Cette méthode prend en paramètre l'instant t considéré et gère la création et l'affichage du graphique par l'intermédiaire d'un objet `traceur` de la classe `TraceSeries`.

1. Compilez et exécutez votre programme. Ouvrez le fichier `Temperature.java` et identifiez les méthodes `main` et `traceParTemps` et leurs fonctionnements respectifs.
2. Modifiez les noms et labels dans la méthode `traceParTemps`.

La méthode délègue à la variable d'instance `sa` de classe `TemperatureAnalytique` la création de la série de points définissant la courbe, avec l'appel à la méthode `void ajouteSerieParTemps(t, traceur)`.

[3] http ://www.jfree.org/jfreechart/

1. Ouvrez le fichier `TemperatureAnalytique.java` et identifiez la méthode `ajouteSerieParTemps`.
2. Modifiez cette méthode pour qu'elle réponde aux spécifications attendues : créer une série de points $(x_k, y_k = T(x_k, t))_{k=0..K}$ enregistrée sous la forme d'un tableau à 2 entrées x_k =`serie[k][0]`, y_k =`serie[k][1]` et l'ajouter à l'instance `traceur`.
   Remarque : le nombre de points K nécessaires à la discrétisation de l'espace sera précisé dans le fichier de paramètres du programme

3.5.3 Les 3 autres types de graphiques

À partir de cet exemple, `traceParTemps(t)/ajouteSerieParTemps(t)`, vous créerez trois autres types de graphiques :

1. évolution temporelle de la température à un point x donné.
   Remarque : le nombre de points N nécessaires à la discrétisation du temps et le pas de discrétisation du temps d seront précisés dans le fichier de paramètres du programme. Comme le temps est normalisé, vous penserez à normaliser ce pas de discrétisation $dtn = d/\tau$ dans le constructeur de la classe `TemperatureAnalytique`.
2. profil de la température dans la plaque à différents instants t : complétez la méthode `void ajouteSeriesParTemps(TraceSeries)` de la classe `TemperatureAnalytique` et créez l'appel correspondant dans la classe `Temperature`.
3. évolution temporelle de la température à différents points x.

4 TP3 : lecture et visualisation des résultats expérimentaux

4.1 Objectif et déroulement de la séance

Vendredi 15 février 2008, durée : 1h.

Vous allez réaliser, de manière individuelle, la seconde et dernière partie de votre application, en implémentant les fonctionnalités suivantes :

1. lecture, dans un fichier texte, des données expérimentales ;
2. conversion et normalisation des données expérimentales ;
3. tracé des courbes de températures expérimentales comparées aux courbes analytiques ;
4. calcul et tracé des marges d'incertitude des mesures.

Récupérez sur campus le fichier source `TemperatureExpe.java` et enregistrez-le dans votre projet. Pour implémenter ces nouvelles fonctionnalités, vous aurez à modifier ce fichier, ainsi que les fichiers `Temperature.java` et `setup.prop`.

L'évaluation de cette séance portera sur ces modifications uniquement, suivant les même règles qu'au TP2 : votre programme doit, a minima, compiler et s'exécuter sans lever d'exception et votre code doit être clair.

Vous déposerez l'archive complète de votre projet sur Campus à la fin de la séance (12h20 au plus tard).

4.2 La classe `TemperatureExe`

Une instance de la classe `TemperatureExpe` modélise l'ensemble des données expérimentales que vous avez recueillies et que vous souhaitez traiter : dates et positions de mesure, tensions mesurées. Les méthodes de la classe décrivent les algorithmes correspondant à chaque traitement attendu (conversion et normalisation, calcul d'incertitude, création des séries de points pour le tracé des courbes).

À la création d'une instance, le **constructeur** de la classe appelle la méthode `lectureFichier(String)` chargée de lire les données expérimentales depuis un fichier texte. Les valeurs numériques ainsi lues servent, après normalisation, à initialiser les **variables de l'instance** : `t` la liste des temps, `x` la liste des positions, `T` la table des températures.

Comme dans la classe `TemperatureAnalytique`, on spécifiera quatre méthodes de création de séries de points 2D pour le tracé des courbes de températures expérimentales normalisées :

- température en fonction du temps pour une ou toutes les positions de mesure
- température en fonction de la position pour un ou tous les temps de mesure

L'affichage des courbes n'est pas une fonctionnalité de la classe `TemperatureExpe` : elle est prise en charge par la classe `Temperature`, permettant ainsi de tracer sur un même graphique courbes expérimentales et analytiques. La classe `Temperature` doit donc être modifiée pour ordonner la création d'une instance `se` de `TemperatureExpe` (en plus de l'instance `sa` de classe `TemperatureAnalytique`). La classe `Temperature` gère la création des graphiques mais délègue à `se` (resp. `sa`) le calcul et la génération des séries de points 2D qui définissent les courbes expérimentales (resp. analytiques).

4.3 Le fichier de données expérimentales

Le nom du fichier texte contenant vos données expérimentales doit être spécifié dans le fichier de paramètres `setup.prop`, avec lequel l'instance `se` est initialisée via le constructeur `TemperatureExpe(Properties)`. Le fichier de données doit répondre à un format précis, compatible avec la méthode `lectureFichier(String)` de votre programme, chargée de le parcourir. Un tel format vous est proposé, mais vous êtes libre d'en choisir un autre : il vous faudra alors modifier la méthode de lecture en conséquence.

Créez un fichier texte de données répondant au format suivant (les valeurs sont séparées par des espaces ou des tabulations) :

I	J	Δ		
X	x_1	x_2	...	x_J
t_1	V_{11}	V_{12}	...	V_{1J}
t_2	V_{21}	V_{22}	...	V_{2J}
...				
t_I	V_{I1}	V_{I2}	...	V_{IJ}

avec :
– $x_1, x_2, \ldots x_J$: les positions de mesure (m) [la première chaîne de la ligne ("X") ne sera pas lue]
– $t_1, t_2, \ldots t_I$: les temps de mesure (min)
– V_{ij} : la tension mesurée au temps numéro i et à la position numéro j (mV)
– Δ : la valeur de la constante de calcul de l'incertitude $\Delta = \Delta V_{ij}$ (mV)
Remarque : si vos données expérimentales sont enregistrées dans un fichier tableur (type MSExcel), utilisez la fonction de votre tableur d'export (ou sauvegarder sous*...) au format CSV en précisant, comme séparateur, l'espace ou la tabulation.*
Ajoutez le nom de ce fichier dans les paramètres du programme (dans le fichier `setup.prop`, voir l'accesseur `String getNomFichier()` de la classe `TemperatureExpe`).

4.4 Lecture du fichier texte

La méthode `lectureFichier(String)` de la classe `TemperatureExpe` prend en paramètre le nom du fichier texte contenant les données expérimentales. Elle fait appel à des librairies standards de la JDK pour : ouvrir le fichier en lecture (`FileReader`), lire le fichier ligne à ligne (`BufferedReader`) puis mot à mot (`String`), enfin convertir les mots en valeurs numériques (`Double`, `Integer`). Voici une description courte et approximative[4] des méthodes employées (vous n'aurez à manipuler que les 4 dernières) :

`FileReader(String s)`	*classe `FileReader` : ouvre le fichier nommé `s` en lecture (converti en flux de caractères)*
`BufferedReader(FileReader)`	*classe `BufferedReader` : mise en tampon du flux*
`String readLine()`	classe `BufferedReader` : lit une nouvelle ligne de texte et la retourne en une chaîne de caractères
`String[] split(String r)`	classe `String` : sépare une chaîne de caractères, en un tableau de chaînes, suivant une expression régulière `r` (par exemple, `r="[ \t]+"` permet de séparer la chaîne mot à mot suivant les caractères espace (' ') ou tabulation ('\t') successifs ('+'))
`static double parseDouble(String s)`	classe `Double` : convertit `s` en `double`
`static int parseInt(String s)`	classe `Integer` : convertit `s` en `int`

[4] se référer à l'API Java pour plus de détails.

La méthode `lectureFichier(String)` est déjà partiellement implémentée :

1. ouverture du fichier :
   `BufferedReader f = new BufferedReader(new FileReader(nomFichier)) ;`
2. lecture de la première ligne du fichier : nombre de temps (variable locale `nbT`), nombre de positions (variable locale `nbX`) et constante d'incertitude (variable d'instance `delta`)
3. allocation du tableau des positions `x = new double[nbX] ;` (variable d'instance)
4. initialisation de `x` : lecture de la deuxième ligne du fichier et normalisation
5. allocation du tableau des temps normalisés `t` (variables d'instance) et du tableau des tensions `V` (variable locale)
6. **initialisation de `t` et `V` : à faire**
7. appel de méthode `this.calculeTemperature(V)` pour l'allocation et l'initialisation du tableau des températures `T` à partir de la table des tensions `V`.

Terminez l'implémentation de la méthode `lectureFichier(String)` (point 6).

4.5 Normalisation et calcul de la température

Les données expérimentales sont donc normalisées (temps, positions) ou calculées (température) avant d'être enregistrées dans les variables d'instance.

Implémentez les méthodes de normalisation et de calcul, répondant aux spécifications décrites en commentaires :
- `double normalisePosition(double x)` `// x` $\rightarrow$ `x`$/L$
- `double normaliseTemps(double t, double tau)` `// t` $\rightarrow$ `t`$*60/$`tau`
- `void calculeTemperature(double[][] V)` `// T`$_{tx}$`=(V`$_{tx}$`-V`$_{\infty x}$`)/(V`$_{0x}$`-V`$_{\infty x}$`)`

4.6 Création des séries de points 2D

La classe `TemperatureExpe` dispose de quatre méthodes de création de séries de points 2D (température/temps en une ou plusieurs positions et température/position en un ou plusieurs temps). Ces méthodes sont identiques à celles de `TemperatureAnalytique` à quelques différences près :
- les points 2D correspondent aux mesures effectuées : température relevée à un temps et une position de mesure (normalisés) ;
- on passe en paramètre un **numéro** de temps ou de position (au lieu de la **valeur** du temps ou de la position) ;
- on invoque la méthode `void ajouteSerieExpe(double[][], String)` de la classe `TraceSeries` (au lieu de la méthode `ajouteSerieAna`).

Implémentez les méthodes de création de série, répondant aux spécifications décrites en commentaires :
- `void ajouteSeriesParTemps (int, TraceSeries)`
- `void ajouteSeriesParTemps (TraceSeries)`
- `void ajouteSeriesParPosition (int, TraceSeries)`
- `void ajouteSeriesParPosition (TraceSeries)`

4.7 Tracé des courbes comparatives expérimental/analytique

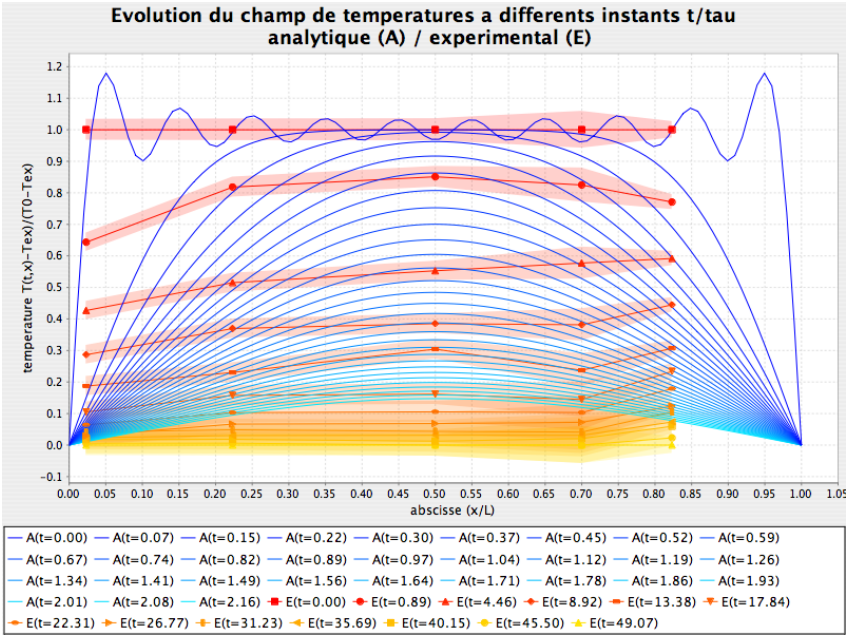
Complétez la classe `Temperature` pour permettre le tracé sur un même graphique des courbes analytiques et expérimentales.

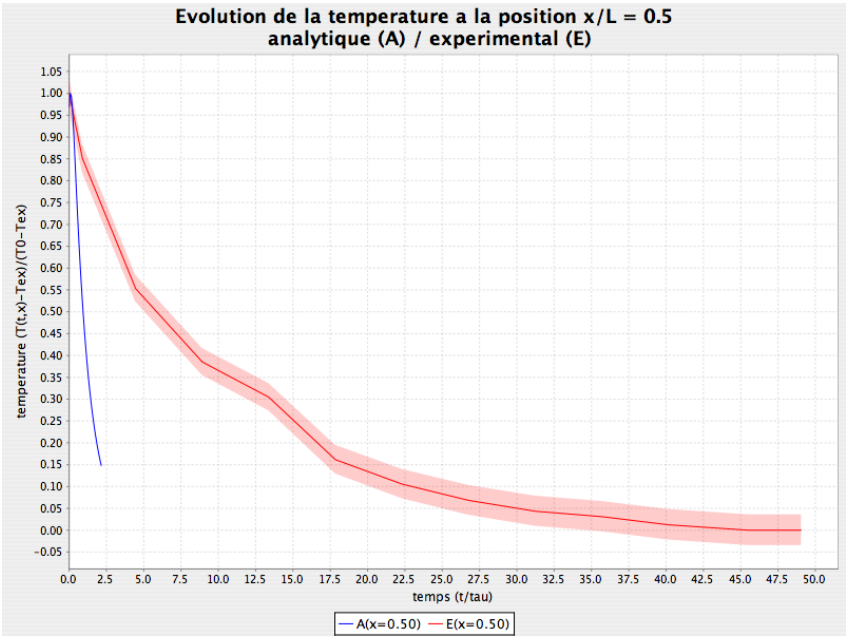
Indications : l'invocation de `TemperatureExpe` est identique à celle de `TemperatureAna` excepté pour le tracé des courbes à un temps donné ou à une position donnée (numéro/valeur). Notez la fonction des accesseurs `double getTemps(int)` et `double getPosition(int)` de la classe `TemperatureExpe`. Ces méthodes sont déclarées `public` et peuvent donc être invoquées en dehors de leur classe de définition. Comment faire afficher sur un même graphique les deux courbes analytique/expérimentale correspondant à un numéro de temps (resp. de mesure) donné ?

4.8 Question subsidiaire : incertitude

Complétez votre programme pour faire apparaître sur les courbes expérimentales, les marges liées à l'incertitude des mesures.

Indications : seule la classe `TemperatureExpe` devra être modifiée en implémentant la méthode `calculeTemperatureEtIncertitude(double[][] V)` et en l'invoquant en place de `calculeTemperature (double[][] V)` ; puis en invoquant la méthode `ajouteSerieErr(double[][], String)` dont vous trouverez la définition et l'usage dans la classe `TraceSeries`.





Version finale attendue le lundi 3 mars