Cours MOOC EPFL d'initiation à la programmation illustré en Java

Troisième devoir noté
Boucles et itérations

J. Sam & J.-C. Chappelier

du 24 septembre au 12 octobre 2015

1 Exercice 1 — démographie

1.1 Introduction

Le but de cet exercice est de simuler l'évolution de la population mondiale au cours du temps.

Télécharger le programme fourni sur le site du cours [1] et le compléter.

ATTENTION : vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernent spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :
   `Window > Preferences > Java > Editor > Save Actions`
   (et décocher l'option de reformatage si elle est cochée)
2. sauvegarder le fichier téléchargé sous le nom `Population.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement
   `[dossierDuProjetPourCetExercice]/src/` ;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;

1. `https://d396qusza40orc.cloudfront.net/initprogjava/assignements-data/Population.java`

4. écrire le code à fournir entre ces deux commentaires :

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier apres cette ligne.
 *****************************************/
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Population.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

1.2 Population d'une année donnée

La formule suivante :

$$p_{\text{fin}} = p_{\text{init}} \cdot \exp(\text{nb} \cdot \tau)$$

permet de calculer la population p_{fin} atteinte à partir d'une population initiale p_{init} au bout de l'écoulement de nb années lorsque le taux de croissance de la population est τ.

Il vous est demandé de compléter le programme en dessous de :

```
// ===== PARTIE 1 =====
```

de sorte à ce qu'il :

1. demande à l'utilisateur d'introduire une année, $\text{annee}_{\text{finale}}$, strictement supérieure à `anneeInitiale` (donnée dans le programme fourni) ; l'année sera redemandée à l'utilisateur tant qu'elle n'est pas strictement supérieure à `anneeInitiale` ;
2. calcule et affiche la population atteinte en $\text{annee}_{\text{finale}}$ selon l'exemple de déroulement ci-dessous et en appliquant la formule donnée.

L'instruction d'affichage sera **identique à celle fournie** pour afficher la population initiale (et qui permet notamment d'avoir un affichage avec 3 chiffres après la virgule).

Indications :

— la fonction expontentielle exp s'écrit `Math.exp` en Java, par exemple : `Math.exp(x)` ;

— le taux de croissance est donné en pourcentage dans le programme ; il conviendra de le diviser par 100 pour obtenir les valeurs adéquates.

Exemple de déroulement

```
======== PARTIE 1 ========
Population en 2011 : 7.000
Quelle année (> 2011) ? 2010
Quelle année (> 2011) ? 2008
Quelle année (> 2011) ? 2016
Population en 2016 : 7.433
```

Le programme affiche donc ici la population sur terre en 2016 sachant que l'on a atteint les 7 milliards en 2011 et que la croissance est estimée a 1.2% annuellement.

1.3 Evolution de la population

On souhaite maintenant faire en sorte que le programme affiche aussi l'évolution de la population chaque année, jusqu'à ce qu'elle atteigne un nombre de milliards donné, saisi par l'utilisateur.

Complétez le programme en dessous de :

```
System.out.println("\n======== PARTIE 2 ========");
```

de sorte à ce qu'il :

1. demande à l'utilisateur d'introduire une population cible (combien de milliards on souhaite atteindre) ; cette population cible sera strictement supérieure à `populationInitiale` (donnée dans le programme fourni) ; la population cible sera redemandée à l'utilisateur tant qu'elle n'est pas strictement supérieure à `populationInitiale` ;
2. calcule et affiche la population pour toutes les années devant s'écouler jusqu'à ce que la population cible soit atteinte, selon l'exemple de déroulement ci-dessous ; **le format des affichages pour chaque année sera identique à celui utilisé jusqu'ici** (fourni).

Exemple de déroulement

```
======== PARTIE 1 ========
Population en 2011 : 7.000
Quelle année (> 2011) ? 2021
Population en 2021 : 7.892
```

```
======== PARTIE 2 ========
Combien de milliards (> 7.0) ? 6.5
Combien de milliards (> 7.0) ? 7.0
Combien de milliards (> 7.0) ? 8.5
Population en 2012 : 7.085
Population en 2013 : 7.170
Population en 2014 : 7.257
Population en 2015 : 7.344
Population en 2016 : 7.433
Population en 2017 : 7.523
Population en 2018 : 7.613
Population en 2019 : 7.705
Population en 2020 : 7.798
Population en 2021 : 7.892
Population en 2022 : 7.988
Population en 2023 : 8.084
Population en 2024 : 8.182
Population en 2025 : 8.281
Population en 2026 : 8.381
Population en 2027 : 8.482
Population en 2028 : 8.584
```

1.4 On affine la simulation

Nous souhaitons maintenant être un peu plus réalistes en prenant en compte la surpopulation : à chaque fois que la population mondiale double, le taux de croissance est divisé par deux.

Complétez maintenant votre programme après :

```
System.out.println("\n======== PARTIE 3 ========");
```

de sorte à ce qu'il affiche la population chaque année jusqu'à atteindre la population cible voulue (qui reste la même que celle saisie dans la partie 2).

On affichera également le taux de croissance, en respectant un format strictement identique à celui produit dans l'exemple de déroulement ci-dessous.

Exemple de déroulement

```
======== PARTIE 1 ========
```

```
Population en 2011 : 7.000
Quelle année (> 2011) ? 2016
Population en 2016 : 7.433

======== PARTIE 2 ========
Combien de milliards (> 7.0) ? 29
Population en 2012 : 7.085
Population en 2013 : 7.170
Population en 2014 : 7.257
Population en 2015 : 7.344
Population en 2016 : 7.433
Population en 2017 : 7.523
Population en 2018 : 7.613
Population en 2019 : 7.705
Population en 2020 : 7.798
Population en 2021 : 7.892
...
Population en 2130 : 29.192

======== PARTIE 3 ========
Population en 2012 : 7.085 ; taux de croissance : 1.2%
Population en 2013 : 7.170 ; taux de croissance : 1.2%
Population en 2014 : 7.257 ; taux de croissance : 1.2%
Population en 2015 : 7.344 ; taux de croissance : 1.2%
Population en 2016 : 7.433 ; taux de croissance : 1.2%
Population en 2017 : 7.523 ; taux de croissance : 1.2%
Population en 2018 : 7.613 ; taux de croissance : 1.2%
Population en 2019 : 7.705 ; taux de croissance : 1.2%
Population en 2020 : 7.798 ; taux de croissance : 1.2%
Population en 2021 : 7.892 ; taux de croissance : 1.2%
...
Population en 2068 : 13.873 ; taux de croissance : 1.2%
Population en 2069 : 14.040 ; taux de croissance : 0.6%
Population en 2070 : 14.124 ; taux de croissance : 0.6%
...
Population en 2195 : 29.018 ; taux de croissance : 0.3%
```

Cet exemple est un extrait. A la place des `...` seront évidemment affichées toutes les années intermédiaires.

Vous noterez que dans cet exemple :

- la population double en 2069 où elle passe de 7 à 14 milliards ;
- qu'elle double aussi en 2185 où elle passe de 14 à 28 milliards.

2 Exercice 2 — Tour en vélo

2.1 Introduction

On s'intéresse ici à écrire un programme permettant de simuler le trajet d'un cycliste lors d'un tour en montagne [2].

Télécharger le programme fourni sur le site du cours [3] et le compléter.

ATTENTION : vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernent spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :
   `Window > Preferences > Java > Editor > Save Actions`
   (et décocher l'option de reformatage si elle est cochée)
2. sauvegarder le fichier téléchargé sous le nom `Cycliste.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement
   `[dossierDuProjetPourCetExercice]/src/` ;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;
4. écrire le code à fournir entre ces deux commentaires :

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier apres cette ligne.
 *****************************************/
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Cycliste.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

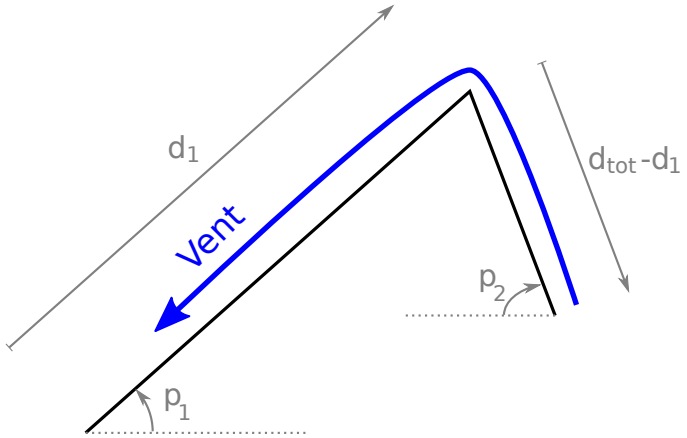

FIGURE 1 – Le trajet prévu pour le cycliste (de gauche à droite).

2.2 Modélisation du cycliste

Le programme demandé devra simuler le trajet (temps, distance, vitesse, accélération et puissance développée) d'un cycliste effectuant un trajet (cf figure 1) de d_{tot} kilomètres en montagne, constitué d'une montée de d_1 kilomètres à p_1 pourcents et d'une descente de $d_{\text{tot}} - d_1$ kilomètres à p_2 pourcents. Par exemple, un tour de 50 km ($d_{\text{tot}} = 50$) dont 30 km ($d_1 = 30$) sont en montée à 5 % ($p_1 = 5$) et le reste est une descente à 10% ($p_2 = 10$).

On supposera de plus qu'il y a un vent constant tout le long du trajet. Le sens du vent est pris dans le sens du cycliste : un vent négatif (vent contraire) signifie que le cycliste se prend le vent de face (comme dans l'exemple de la figure) ; un vent positif signifie que le vent pousse le cycliste dans le dos.

Le cycliste sera représenté dans le programme par sa masse (en kilogrammes), sa vitesse (en kilomètres par heure), son accélération (en kilomètres par heure par minute), la distance (en kilomètres) qu'il a parcouru depuis le début de la montée et la puissance (en watts) qu'il développe.

Nous vous fournissons déjà quelques variables dans le programme téléchargé :

- `t` représentera le temps (en minutes) depuis le début de la montée (le programme commence donc au temps `t`=0) ;
- `d`, la distance (en kilomètres) que le cycliste a parcouru depuis le début de la montée ;
- `v`, la vitesse du cycliste (en kilomètres par heure) au temps `t` ; au départ,

2. suite à la location de la semaine passée ; -)
3. `https://d396qusza40orc.cloudfront.net/initprogjava/assignements-data/Cycliste.java`

c'est donc la vitesse avec laquelle le cycliste aborde la montée (il ne part pas de zéro, mais attaque la montée à 30 km/h) ;
- `acc`, l'accélération du cycliste (en kilomètres par heure par minute) au temps `t` ; c'est donc le nombre de kilomètres par heure qu'il va gagner/perdre dans la prochaine minute ; elle sera calculée au fur et à mesure en fonction des autres paramètres ; sa valeur initiale n'a donc pas d'importance et nous avons choisi 0 ;
- et `p`, la puissance (en watts) que le cycliste développe au temps `t` en pédalant ; il faudra la recalculer à chaque fois en fonction de la fatigue du cycliste (voir plus loin) ; nous avons choisi une valeur initiale de 175 W.

Le reste du code fourni consiste en :
- plusieurs messages à afficher pour poser des questions à l'utilisateur (voir plus loin) ;
- la condition et la ligne permettant d'afficher les informations sur le cycliste toutes les 5 minutes ; la condition utilise des fonctions qui n'ont pas été présentées dans le cours (`Math.abs`, `Math.round`) dont le but est simplement d'éviter trop d'affichages en n'affichant qu'une ligne toutes les 5 minutes de simulation ; utilisez ces lignes fournies telles quelles ;
- deux messages utiles pour des conditions particulières (décrites plus bas) ;
- l'affichage des deux derniers messages, à ne pas modifier.

Pour commencer, votre programme devra demander à l'utilisateur les grandeurs suivantes (voir exemple de déroulement en fin de l'énoncé) :
- la masse du cycliste (en kilogrammes) ; cette masse doit être comprise entre 40 et 180 kg (inclus) ;
- la vitesse du vent (en kilomètres par heure) ; elle doit être comprise entre –20 et +20 km/h ;
- la distance totale du parcours (en kilomètres) ; cette distance doit être supérieure ou égale à 11 km et inférieure ou égale à 200 km ;
- la longueur (en kilomètres) de la montée ; cette distance doit être positive et telle qu'il reste au moins 10 km de descente ; il faudra afficher dans la question la longueur maximale possible ;
- la pente (en pourcents) de la montée ; elle doit être plus grande que zéro et inférieure ou égale à 20% ;
- la pente (en pourcents) de la descente ; elle doit être plus grande que zéro et inférieure ou égale à 20%.

En cas de valeur entrée incorrecte, votre programme devra (de suite) poser à nouveau la question (voir exemple de déroulement plus bas) ; et ce, jusqu'à obtention d'une réponse correcte.

Nous vous conseillons ensuite de définir deux constantes :

— le « pas de temps » `DT` avec lequel nous allons faire avancer les calculs de la simulation ; fixez-le à 1/60 ;
— la « puissance minimale » `P_MIN` que le cycliste fournit ; elle sera de 10 W.

2.3 Simulation du mouvement

Procédez de la façon suivante pour calculer l'évolution du cycliste :

— afficher les informations courantes (code fourni) ;
— augmenter le temps `t` d'un pas de temps (`DT` défini plus haut) ;
— calculer la vitesse `V` du vent par rapport au cycliste : c'est simplement la différence entre la vitesse du cycliste et celle du vent : `V=v`–vitesse du vent ;
— calculer la fatigue du cycliste : si le cycliste est encore en montée et que sa puissance est supérieure à la puissance minimale (définie plus haut : 10 W), alors il perd 0.5 fois `DT` de puissance (il pédale moins fort à cause de la fatigue) ;
— calculer l'accélération du cycliste ; elle vaut :
  — moins 2'118.96 fois le sinus de l'arctangente de la pente divisée par 100.[4] En Java, on écrit simplement :
  « `-2118.96 * Math.sin(Math.atan(pente/100))` » ;
  Notez que ce terme ne change pas tant que la pente ne change pas ;
  — moins 5 fois `V` fois `Math.abs(V)` divisé par la masse du cycliste ;[5]
  — si la puissance et la vitesse du cycliste sont toutes les deux plus grandes que zéro (i.e. s'il n'a pas posé le pied par terre), on ajoute sa poussée (il pédale) : 777.6[6] fois sa puissance divisé par le produit de sa vitesse et de sa masse :
  $$777.6 \times \frac{\mathtt{p}}{\mathtt{v} \times \mathtt{m}}$$
  — si après tous ces calculs la valeur absolue de l'accélération (`Math.abs(acc)`) est plus petite que 10^{-5} (qui s'écrit « `1e-5` » en Java), alors on la mettra à 0 ;
  — sinon, calculer la nouvelle vitesse du cycliste en ajoutant à l'ancienne valeur le produit de l'accélération par le « pas de temps » :
  $$\mathtt{v} = \mathtt{v} + \mathtt{acc} \times \mathtt{DT}$$

4. Pour ceux que la Physique intéresse, il s'agit là de l'accélération liée au poids : 2'118.96 étant $9.81 \times 60 \times 3.6$ en raison des unités (kilomètres par heure par minute).
5. Physique : c'est la force exercée par le vent sur le cycliste ; 5 provient de $0.3 \times 60/3.6$.
6. $60 \times 3.6 \times 3.6$

— puis mettre à jour la distance parcourue : lui ajouter le produit de la (nouvelle) vitesse et du « pas de temps » divisé par 60 : [7]

$$\mathtt{d} = \mathtt{d} + \mathtt{v} \times \frac{\mathtt{DT}}{60}$$

— terminer enfin par le traitement des cas particuliers :
  — dès que le cycliste aborde la descente :
    — afficher un message :
    `## Bernard a atteint le sommet en ... min.`
    — le cycliste arrête alors de pédaler fortement, il ne produit plus que la puissance minimale (définie plus haut) ; autrement dit : pendant toute la descente la puissance du cycliste est égale à la puissance minimale ;
    — penser également à changer la pente pour celle de la descente, qui doit de plus être négative ;
  — si la vitesse du cycliste devient inférieure à 3 km/h, le cycliste se décourage et abandonne : quitter votre programme en affichant « `## Bernard abandonne, il n'en peut plus` » et les informations (temps, distance, vitesse, accélération et puissance) ; les deux instructions nécessaires pour réaliser ces affichages sont fournies ; pour quitter le programme, simplement utiliser l'instruction « `return` ».

2.4 Exemples de déroulement

Voici plusieurs exemples de déroulement dans différentes situations :

1) 2 entrées erronées puis vent contraire

```
masse du cycliste (entre 40 et 180 ) ? 80
vent (entre -20 et +20 km/h) ? 40
vent (entre -20 et +20 km/h) ? -10
distance du parcours (<= 200 km) ? 50
distance au sommet du col (<= 40.0 km) ? 45
distance au sommet du col (<= 40.0 km) ? 30
pente moyenne jusqu'au sommet (<= 20 %) ? 5
pente moyenne après le sommet (<= 20 %) ? 10
0, 0.00, 30.00, 0.0000,  175.00
5, 1.05, 12.26, -0.0284,  172.50
10, 2.07, 12.12, -0.0286,  170.00
```

7. Cette division par 60 vient des unités : la vitesse est en kilomètres par heure alors que le temps est en minutes.

```
15, 3.07, 11.97, -0.0288,  167.50
20, 4.06, 11.83, -0.0290,  165.00
25, 5.04, 11.68, -0.0291,  162.50
30, 6.01, 11.54, -0.0293,  160.00
35, 6.97, 11.39, -0.0295,  157.50
40, 7.91, 11.24, -0.0297,  155.00
45, 8.84, 11.09, -0.0298,  152.50
50, 9.76, 10.95, -0.0300,  150.00
55, 10.66, 10.79, -0.0302,  147.50
60, 11.56, 10.64, -0.0304,  145.00
65, 12.44, 10.49, -0.0306,  142.50
70, 13.31, 10.34, -0.0308,  140.00
75, 14.16, 10.18, -0.0310,  137.50
80, 15.00, 10.03, -0.0312,  135.00
85, 15.83, 9.87, -0.0314,  132.50
90, 16.65, 9.71, -0.0316,  130.00
95, 17.45, 9.56, -0.0318,  127.50
100, 18.24, 9.40, -0.0320,  125.00
105, 19.02, 9.24, -0.0322,  122.50
110, 19.78, 9.07, -0.0324,  120.00
115, 20.53, 8.91, -0.0326,  117.50
120, 21.27, 8.75, -0.0328,  115.00
125, 21.99, 8.58, -0.0330,  112.50
130, 22.70, 8.42, -0.0332,  110.00
135, 23.39, 8.25, -0.0334,  107.50
140, 24.07, 8.08, -0.0336,  105.00
145, 24.74, 7.92, -0.0338,  102.50
150, 25.39, 7.75, -0.0341,  100.00
155, 26.03, 7.58, -0.0343,  97.50
160, 26.65, 7.40, -0.0345,  95.00
165, 27.26, 7.23, -0.0347,  92.50
170, 27.86, 7.06, -0.0350,  90.00
175, 28.44, 6.88, -0.0352,  87.50
180, 29.00, 6.70, -0.0354,  85.00
185, 29.56, 6.53, -0.0356,  82.50
## Bernard a atteint le sommet en 189 min.
190, 30.59, 48.28, 0.6407,  10.00
195, 34.62, 48.36, 0.0000,  10.00
200, 38.65, 48.36, 0.0000,  10.00
205, 42.68, 48.36, 0.0000,  10.00
210, 46.71, 48.36, 0.0000,  10.00
## Bernard est arrivé
214, 50.00, 48.36, 0.0000, 10.00
```

2) Vent dans le dos

```
masse du cycliste (entre 40 et 180 ) ? 80
vent (entre -20 et +20 km/h) ? 10
distance du parcours (<= 200 km) ? 35
distance au sommet du col (<= 25.0 km) ? 20
pente moyenne jusqu'au sommet (<= 20 %) ? 8
pente moyenne après le sommet (<= 20 %) ? 12
0, 0.00, 30.00, 0.0000,  175.00
5, 0.86, 9.92, -0.0288,  172.50
10, 1.69, 9.78, -0.0287,  170.00
15, 2.49, 9.64, -0.0287,  167.50
20, 3.29, 9.49, -0.0287,  165.00
25, 4.08, 9.35, -0.0286,  162.50
30, 4.85, 9.21, -0.0286,  160.00
35, 5.61, 9.06, -0.0286,  157.50
40, 6.36, 8.92, -0.0286,  155.00
45, 7.10, 8.78, -0.0286,  152.50
50, 7.82, 8.64, -0.0285,  150.00
55, 8.54, 8.49, -0.0285,  147.50
60, 9.24, 8.35, -0.0285,  145.00
65, 9.93, 8.21, -0.0285,  142.50
70, 10.61, 8.07, -0.0285,  140.00
75, 11.27, 7.92, -0.0285,  137.50
80, 11.93, 7.78, -0.0285,  135.00
85, 12.57, 7.64, -0.0284,  132.50
90, 13.20, 7.50, -0.0284,  130.00
95, 13.82, 7.35, -0.0284,  127.50
100, 14.43, 7.21, -0.0284,  125.00
105, 15.02, 7.07, -0.0284,  122.50
110, 15.60, 6.93, -0.0284,  120.00
115, 16.18, 6.79, -0.0284,  117.50
120, 16.73, 6.64, -0.0284,  115.00
125, 17.28, 6.50, -0.0284,  112.50
130, 17.82, 6.36, -0.0284,  110.00
135, 18.34, 6.22, -0.0284,  107.50
140, 18.85, 6.08, -0.0284,  105.00
145, 19.35, 5.93, -0.0284,  102.50
150, 19.84, 5.79, -0.0284,  100.00
## Bernard a atteint le sommet en 152 min.
155, 23.93, 73.72, 0.0000,  10.00
160, 30.08, 73.72, 0.0000,  10.00
## Bernard est arrivé
```

```
164, 35.01, 73.72, 0.0000, 10.00
```

3) Abandon par fort vent contraire

```
masse du cycliste (entre 40 et 180 ) ? 80
vent (entre -20 et +20 km/h) ? -20
distance du parcours (<= 200 km) ? 100
distance au sommet du col (<= 90.0 km) ? 90
pente moyenne jusqu'au sommet (<= 20 %) ? 15
pente moyenne après le sommet (<= 20 %) ? 10
0, 0.00, 30.00, 0.0000,  175.00
5, 0.41, 4.75, -0.0132,  172.50
10, 0.80, 4.69, -0.0132,  170.00
15, 1.19, 4.62, -0.0133,  167.50
20, 1.57, 4.56, -0.0133,  165.00
25, 1.95, 4.49, -0.0133,  162.50
30, 2.32, 4.42, -0.0133,  160.00
35, 2.69, 4.36, -0.0133,  157.50
40, 3.05, 4.29, -0.0133,  155.00
45, 3.40, 4.22, -0.0134,  152.50
50, 3.75, 4.16, -0.0134,  150.00
55, 4.10, 4.09, -0.0134,  147.50
60, 4.43, 4.02, -0.0134,  145.00
65, 4.77, 3.96, -0.0134,  142.50
70, 5.09, 3.89, -0.0134,  140.00
75, 5.41, 3.82, -0.0135,  137.50
80, 5.73, 3.75, -0.0135,  135.00
85, 6.04, 3.69, -0.0135,  132.50
90, 6.34, 3.62, -0.0135,  130.00
95, 6.64, 3.55, -0.0135,  127.50
100, 6.94, 3.48, -0.0135,  125.00
105, 7.22, 3.42, -0.0136,  122.50
110, 7.51, 3.35, -0.0136,  120.00
115, 7.78, 3.28, -0.0136,  117.50
120, 8.05, 3.21, -0.0136,  115.00
125, 8.32, 3.14, -0.0136,  112.50
130, 8.58, 3.08, -0.0136,  110.00
135, 8.83, 3.01, -0.0136,  107.50
## Bernard abandonne, il n'en peut plus
136, 8.86, 3.00, -0.0137,  107.22
```