Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Aix*Marseille université
Socialement engagée

Année 2023-24

Programmation avancée : TP 3
Code UE : SBICU05

Simulation de gènes

Simulation de réseaux de régulation

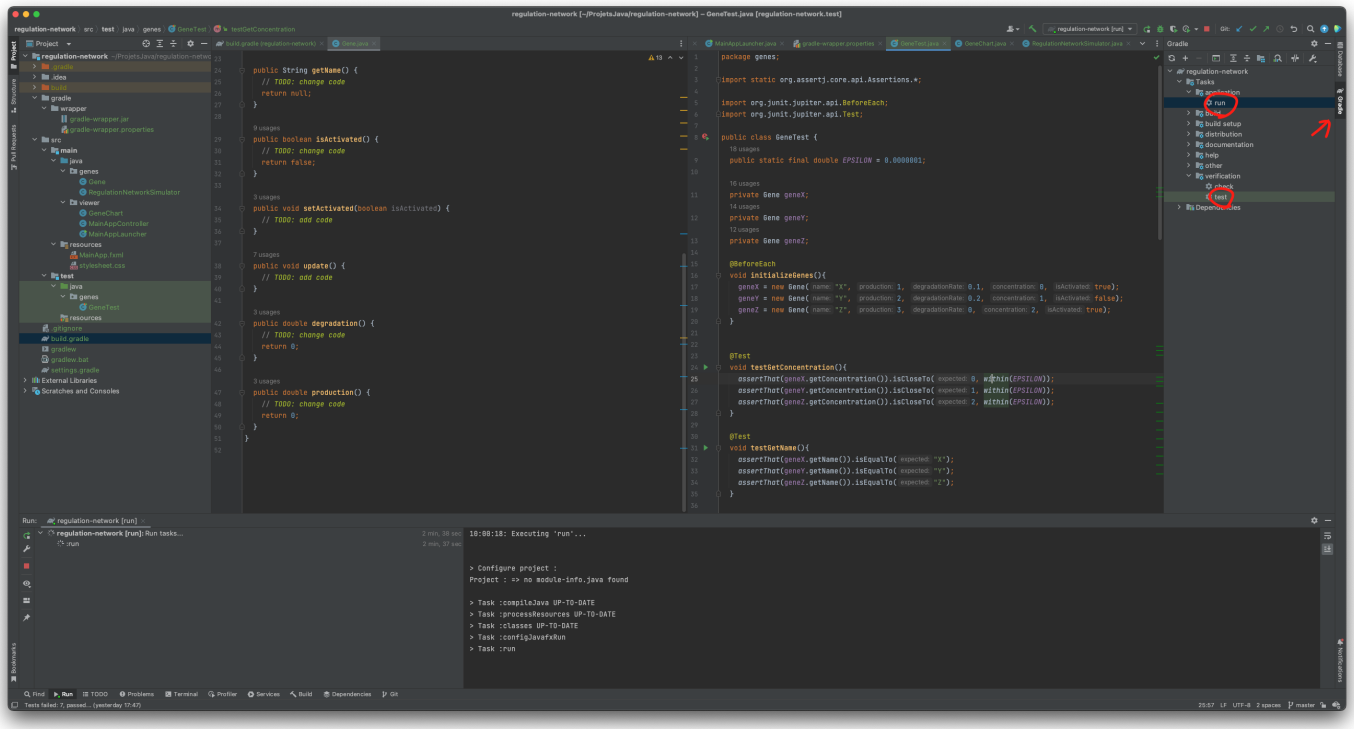
On va travailler sur ce TP sur les réseaux de régulation, c'est-à-dire la simulation de l'expressivité des gènes et la concentration des protéines dans une cellule.

Récupérer le dépôt

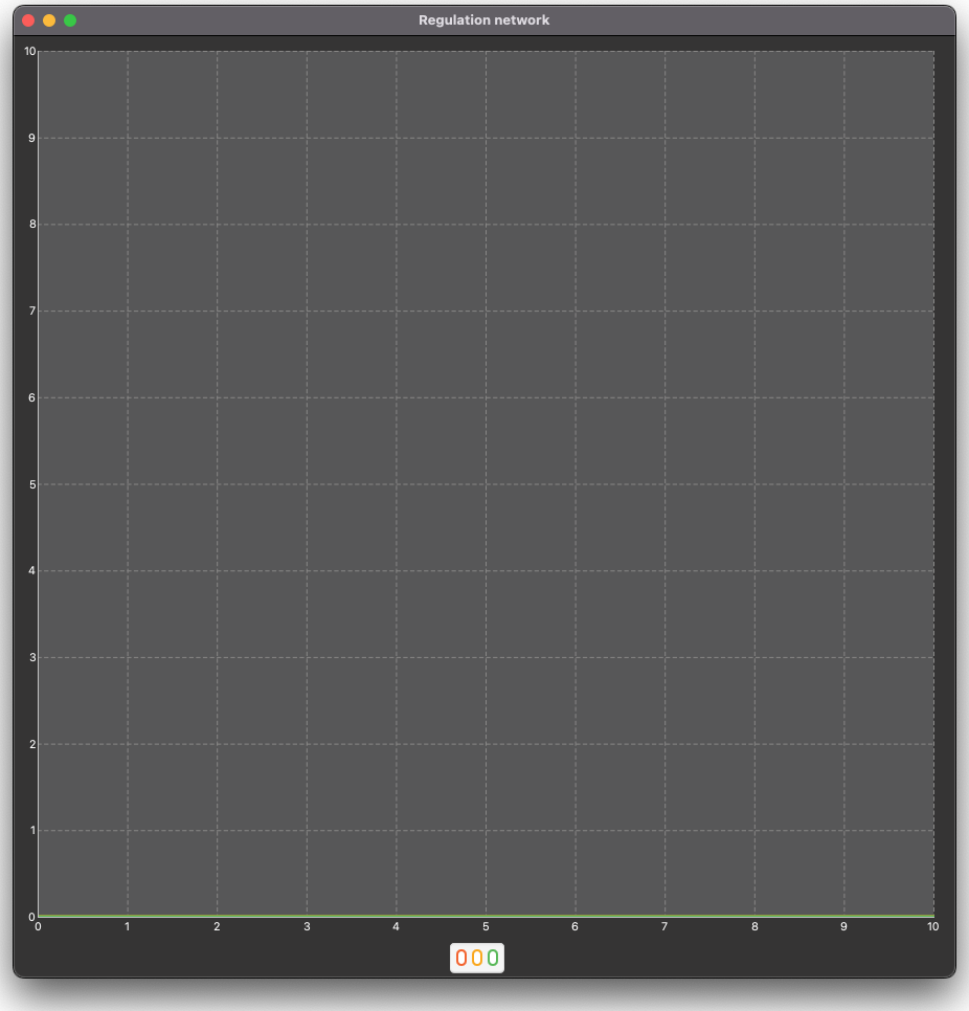
Comme pour le TP 2, on va utiliser git pour la gestion de versions. Il vous faut donc vous reporter aux consignes du précédent TP. Le lien vers le projet à forker est le suivant : lien.

Exécuter le projet du dépôt

Pour compiler et exécuter votre programme, il faut passer par l'onglet gradle à droite.



— pour les tests il faut cliquer deux fois sur `regulation-network` -> `Tasks` -> `verification` -> `test`. Pour le moment, les tests ne passeront pas car certaines classes sont incomplètes.
— pour l'affichage, il faut cliquer deux fois sur `regulation-network` -> `Tasks` -> `application` -> `run`. Vous devriez obtenir l'affichage suivant.



Simulation d'un gène

Explication de la simulation

La simulation est l'une des techniques fondamentales du calcul scientifique. Le programme contient un modèle de l'état du système et la manière de le mettre à jour après une étape de temps.

Pour ce TP, on va considérer la simulation d'un gène encodant une protéine X. Notre programme va simuler la concentration de la protéine au sein d'une cellule au fil du temps. Le gène peut être actif ou non. Quand le gène est actif (on dit aussi qu'il s'exprime), la cellule produit la protéine X et la concentration C de X dans la cellule augmente. Nous supposerons que la production augmente la concentration C à un taux constant β correspondant à une augmentation par seconde.

Il existe également des processus qui réduisent la concentration de X. Par exemple, la concentration de X peut diminuer à cause de la dilution quand la cellule grossit ou se divise ou bien parce que d'autres processus dégrade la protéine X. S'il s'agit d'une cellule bactérienne en croissance, elle pourrait se diviser toutes les 30 minutes

environ, et en l'absence de toute nouvelle production, la concentration de X serait alors réduite de moitié toutes les 30 minutes. Nous pouvons les regrouper dans un processus qui réduit la concentration C à un taux $\alpha.C$. Notez que le changement de concentration est proportionnel à la concentration elle-même : c'est-à-dire que nous avons une règle telle que "C baisse de 1 % par seconde".

Nous pouvons modéliser ce processus dans un programme de simulation. À chaque pas de temps, nous augmentons C d'une petite quantité constante si le gène est exprimé, pour modéliser la production, et la diminuons d'une petite quantité proportionnelle à la concentration pour modéliser la dégradation.

Classe `Gene`

La classe `Gene` représentera les propriétés du gène et de la production/concentration de la protéine associée. La classe `Gene` contient les données suivantes :

- la production `production` de la protéine si le gène est actif (augmentation en une étape de temps de la concentration de la protéine lorsque le gène est actif)
- le taux de dégradation `degradationRate` de la protéine (proportion de la concentration de la protéine disparaissant en une étape de temps)
- un booléen `activated` indiquant si le gène est activé ou désactivé
- une chaîne de caractères `name` contenant le nom de gène
- la concentration `concentration` de la protéine dans la cellule

La classe `Gene` contient un constructeur `Gene(String name, double production, double degradationRate , double concentration, boolean isActivated)` déjà écrit qui initialise les attributs du gènes avec les arguments donnés.

La classe `Gene` contient les méthodes suivantes (dont le code est à compléter) :

- une méthode `double getConcentration()` renvoie la concentration actuelle de la protéine.
- une méthode `String getName()` qui renvoie le nom du gène;
- une méthode `boolean isActivated()` qui renvoie `true` si le gène est actif et `false` sinon;
- une méthode `void setActivated(boolean isActivated)` qui met à jour l'état d'activation du gène avec celui donné en argument;
- une méthode `void update()` qui met à jour la concentration de la protéine en prenant en compte la production (si le gène est actif) et la dégradation de celle-ci;
- une méthode `double degradation()` qui calcule la quantité de protéine dégradé en multipliant le taux de dégradation par la concentration actuelle de la protéine;
- une méthode `double production()` qui calcule la production actuelle de la protéine (O si le gène n'est pas actif et égale à la production du gène sinon).

1. Complétez le code des méthodes de `Gene`.

2. Vérifiez que votre code fonctionne en lançant les tests via `regulation-network` -> `Tasks` -> `verification` -> `test` dans le menu gradle.

3. Vérifiez que l'affichage fonctionne en lançant l'application via `regulation-network` -> `Tasks` -> `application` -> `run` dans le menu gradle. Vous devriez obtenir l'affichage suivant :