1 Structuration du code de calcul

De nombreux codes de calculs ont été développés durant la thèse. Outre le début de la thèse qui s'est déroulée sous Scilab dans le but de maquetter rapidement le code résolvants les différents modèles EDP, plusieurs codes ont été développées dans différents langages. Sans tomber dans la présentation détaillée de l'ensemble des codes développés pendant ma thèse, faisons en un tour d'horizon. Ma thèse étant divisée en deux grandes parties, deux codes ont été développé. Le premier pour résoudre le modèle EDP, et fournir des simulations numériques reproduisant des croissances tumorales et des résistances aux traitements. Le second, pour calculer un quantificateur de l'hétérogénéité à partir d'une image, que ce soit une image provenant directement de l'appareil d'imagerie médicale ou une image provenant de la simulation numérique.

1.1 Code de calcul pour la simulation numérique du modèle EDP

Le code permettant la simulation numérique via le modèle EDP a été développé en C++, avec la librairie eLYSe. Cette librairie, commune a toute l'équipe de recherche et développée par Olivier Saut, est une interface qui fournit un certain nombre de solveurs pour les problèmes classiques (transport et diffusion) traités par une méthode de volumes finis sur maillages cartésiens. Elle est multi-plateforme (Linux et MacOS). Elle s'appuie sur la librairie d'algèbre linéaire PETSc, bien qu'aucune connaissance de PETSc ne soit nécessaire pour utiliser eLYSe. Ceci est principalement du au fait de la présence des structures de types Field (tableaux 2D et 3D) que cette librairie fournit.

Après avoir appris le C++, j'ai pu prendre en main cet outil. Divers modules ont pu être construit :

- Un module pour les schémas à stencil croisé (de type twin-WENO5)

- Un module fournissant diverses géométries en fonction de paramètres permettant de construire la condition initiale du modèle EDP.
- Une classe Modele qui est à même de calculer, en fonction de paramètres (ceux fournis dans le chapitre **??**, dans la Table **??** de la page **??**) l'évolution spatio-temporelle des différentes quantités du modèle EDP (notamment les diverses populations de cellules). Cette classe contient entre autre :
  - la définition des paramètres et leurs affectations,
  - les allocations mémoire des variables intervenant dans le modèle EDP,
  - les schémas en temps et le calcul de CFL,
  - les interactions entre les populations et la vascularisation,
  - les points de contrôle (les populations sont-elles toujours comprises entre 0 et 1, moyennant une certaine tolérance ?),
  - les sorties fichiers.
- Une classe de méta-modèle, permettant d'explorer, de différentes manières, l'espace des paramètres lié à un modèle. Cette classe m'a notamment servie pour réaliser le fit (présenté sur la Figure **??**, page **??**) des données cliniques par le modèle EDP. Plusieurs méthodes d'exploration ont été abordées : Monte-Carlo, méthode de sensitivité ou de descente, algorithmes génétiques... Le modèle EDP ayant beaucoup de paramètres et admettant beaucoup de minima locaux, il n'a pas été facile d'obtenir un fit acceptable.

Le code de calcul est séquentiel. La partie recherche d'optimum dans l'espace des paramètres a elle été (pseudo-)parallélisée. A un jeu de paramètres donné, il y a un modèle qui est calculé sur un unique processeur. Les méthodes d'exploration de l'espace des paramètres nécessitant l'évaluation de plusieurs modèles, on partage l'ensemble des évaluations à réaliser entre les différents processeurs. Par exemple, sur une méthode de gradient, à chaque pas de l'algorithme, une approximation du gradient (dérivée du modèle par rapport à ses n paramètres ici) est requise. Ainsi pour chaque paramètre, une évaluation est nécessaire. Chaque évaluation est réalisée sur une seul et unique processeur, mais on partage les n évaluations requises sur le nombre de processeurs disponibles.

Pour donner un ordre d'idée du code manipulé, présentons le en quelques chiffres. Ce code est subdivisé en une dizaine de modules conduisant à un total d'environ 10 000 lignes. Il est exécuté sur la plateforme de calcul INRIA, nommée PlaFRIM, sans laquelle de nombreuses simulations numériques n'auraient pas pu être réalisées. Le temps moyen d'exécution d'une simulation est d'environ 1h30.

1.2 Code de calcul aboutissant à la quantification de l'hétérogénéité

Le code de calcul générant le pourcentage d'hétérogénéité aussi bien clinique que numérique est très hétéroclite. Il y a plusieurs raisons à cela :

- On veut pouvoir manipuler des données de natures différentes. On souhaite générer les histogrammes des niveaux de gris depuis des sources diverses :
  - Les données cliniques sont dans un format particulier : le format DICOM. C'est un format de méta-image, qui nécessite l'utilisation d'un logiciel (ou d'une librairie) spécifique pour être visualisé et traité.
  - La simulation numérique fournit les différentes populations P, N et S. A partir de cela de niveaux de gris associés τ_P, τ_N et τ_S, une image scanner est synthétisée.
- Le code s'appuie sur des langages différents :
  - C++, avec la librairie ITK. Passage quasiment obligatoire (il y a très peu d'alternative) pour pouvoir traiter des données au format DICOM.
  - Python, avec la librairie Scikit-learn, qui contient un module dédié au mélange gaussien.

Noter qu'au niveau des données, il n'y a pas que la nature des images qui diffèrent : le moyen d'acquérir le contour diffère également. Pour les données cliniques, grâce au logiciel OsiriX, nous pouvons contourer manuellement les tumeurs, et en exporter le contour au format xml. Pour l'aspect images provenant de la simulation numérique, il n'y a pas de contour défini. Le choix (assez naturel) a été fait de le définir en prenant un seuil sur la population saine. Moyennant un parseur xml, on peut donc se ramener à un contour semblable dans les deux cas.

Le code s'articule donc en plusieurs briques dont l'arrangement est schématisé sur la la Figure 1.1. Outre un parseur xml écrit en Objective-C, deux briques principales se distinguent.

Une première brique a été développée en C++ avec la librairie ITK. Elle a pour but, à partir d'une image quelconque et d'un contour, d'extraire les valeurs des niveaux de gris des pixels présents à l'intérieur du contour. On obtient ainsi l'histogramme des niveaux de gris. La librairie ITK a l'avantage de traiter des images aux formats standards (.jpg, .png, etc) mais aussi et surtout le format DICOM, qui est le format des images médicales que nous possédons. Puisque ce code manipule différents format d'images, j'en ai profité pour qu'il soit capable d'exporter, depuis les données DICOM, des images dans des formats standards sur lesquelles sont mis en évidence les différents contours. Ce code a notamment permis de générer les images présentées dans les Figures **??** et **??** de l'annexe **??** (pages **??** et **??**).

Figure 1.1 – Schéma du code de calcul permettant la quantification de l'hétérogénéité. Le code est très hétéroclite et fait intervenir plusieurs langages de programmation

La seconde brique est chargée de trouver les paramètres de deux gaussiennes dont la somme décrit au mieux un histogramme donné. Cette brique est basée sur la librairie Python Scikit-learn, qui contient un module dédié au mélange gaussien. Une fois les paramètres du mélange bi-gaussien identifiés, on peut aisément construire et examiner les critères quantifiant l'hétérogénéité.

Afin de faciliter le chainage entre les différentes étapes du calcul, deux scripts Bash ont été développés (représenté par les cadres rouges et vert sur la Figure 1.1). Le premier permet de traiter de A à Z, les données cliniques. L'utilisateur fournit un dossier contenant un ou plusieurs scanners avec leur contourage provenant d'OsiriX. Le script fournit en sortie, des images post-traitées ainsi que les valeurs des histogrammes des niveaux de gris correspondant aux zones contourées et leur description en mélanges de gaussiennes. Le second script requiert en entrée les résultats de simulations numériques du modèle EDP, et les valeurs des 3 niveaux de gris τ_N, τ_P et τ_S. Il fournit les mêmes sorties que le premier script. Dans chacun de ces scripts une dernière option (non présentée sur le schéma) est disponible, pour réaliser après coup, le tracé de différents critères quantifiant l'hétérogénéité. L'ensemble de la chaine de calcul a donc été automatisée et s'exécute ainsi de manière très simple bien que sa structure interne soit hétéroclite.

Terminons en donnant brièvement quelques statistiques. Ce deuxième code fait environ 5 000 lignes. Le temps d'exécution est de quelques secondes.