Modélisation de performance de noyaux d'algèbre linéaire : approche par maximisation de vraisemblance vs. échantillonnage Bayésien

Auteur :
Hoël JALMIN

Encadrants :
Arnaud LEGRAND
Tom CORNEBIZE

Tome Principal ET Annexe

30/04/2019 - 19/07/2019

Jury :
— Olivier RICHARD, Maître de Conférence UGA
— Bernard TOURANCHEAU, Professeur UGA

Remerciements

Je tiens tout d'abord à remercier toute l'équipe du LIG pour son accueil, et sa sympathie, et particulièrement les équipes DATAMOVE et POLARIS avec qui j'étais le plus en contact, et pour m'avoir donné la chance d'intégrer le laboratoire le temps d'un stage.

Je remercie aussi mes encadrants professionnels, Arnaud Legrand et Tom Cornebize pour leur aide et leur soutien tout au long de mon stage, ainsi que mon tuteur enseignant, Olivier Richard, pour son suivi et son accompagnement. Je remercie également Vincent Arnone pour son aide avec Grid5000, et tous les membres du forum d'aide de Stan.

J'adresse également mes remerciements à Annie Simon pour son aide sur les démarches administratives et à Nadine Chiatti pour toutes les offres de stage qu'elles a transmis.

Enfin, je remercie toutes les personnes qui m'ont relu lors de la rédaction de ce rapport de stage, notamment ma soeur Aélys Jalmin.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Environnement de travail

Mon stage a été effectué dans le Laboratoire Informatique de Grenoble (LIG), qui accueille 24 équipes de recherche, dont l'équipe Polaris (Performance analysis and Optimization of LARge Infrastructures and Systems) dont je dépendais. L'équipe Polaris étudie les performances et l'optimisation de systèmes de grande envergure, générant beaucoup de données. Le LIG Grenoble est présent au sein du bâtiment IMAG de Saint-Martin d'Hères, avec 5 autres laboratoires :

— l'Agence pour les Mathématiques en Interaction avec les Entreprises et la Société (AMIES)
— Grenoble Alpes Recherche Infrastructure de CAlcul Intensif et de Données (GRICAD)
— Laboratoire Jean Kuntzmann (LJK)
— MI2S
— VERIMAG

Grâce aux compétences diverses des chercheurs travaillant dans le bâtiment, l'IMAG permet de profiter d'un grand nombre de séminaires et de conférences sur des sujets très variés.

1.2 Développement durable

Le bâtiment IMAG est considéré comme un bâtiment ńintelligentż au vu de sa faible consommation d'énergie. En effet, le bâtiment régule sa température grâce à l'ensoleillement : les bureaux sont pourvus d'un grand nombre de fenêtre, ce qui augmente la chaleur en hiver, et de volets avec capteurs de lumières se fermant quand le soleil les illumine, pour préserver la fraîcheur en été. De plus, le bâtiment possède des pompes à chaleur s'adaptant à la température extérieure et utilise la géothermie pour limiter l'utilisation de la climatisation.

En outre, comme mentionné auparavant, de nombreux séminaires ont lieu au sein du LIG, dont certains sur la thématique de l'écologie et du développement durable. De plus l'entreprise INRIA, partenaire du LIG, s'implique beaucoup dans le développement durable au travers du projet E2S (Energy Environment Solutions), réalisé en collaboration avec l'INRA et l'université de Pau et du Pays de l'Adour qui a pour but d'associer tous les acteurs de la recherche et de l'enseignement supérieur pour développer l'excellence académique et le développement socio-économique. Ce projet a été labellisé par le jury international des initiatives d'excellence.

Enfin, INRIA possède une équipe appelée STEEP (Soutenabilité, Territoires, Environnement, Economie et Politique) dont l'axe de recherche est basée sur le développement d'outils d'aide aux décisions pour instaurer la soutenabilité à l'échelle régionale, avec une gouvernance adaptée. On peut donc dire que le LIG et INRIA s'impliquent beaucoup dans la problématique de développement durable.

Chapitre 2

Contexte

2.1 Le calcul de haute performance

De nos jours, l'informatique de haute performance (High Performance Computing) est de plus en plus nécessaire dans de nombreux domaines : simulation scientifique, prédictions météorologiques, etc. Les superordinateurs sont utilisés pour des opérations traitant et générant des milliers de données, le plus rapidement possible ; mais les besoins grandissants pour ce type de machine augmente la complexité de leur architecture d'années en années.

En effet, si pendant des décennies les performances des ordinateurs doublaient tous les 18 mois selon la loi de Moore, en augmentant la fréquence d'horloge des processeurs, cela n'est plus le cas à cause des problématiques de consommation d'électricité et de thermorégulation. Pour répondre à ces problématiques, les fabricants de processeurs ont commencé à augmenter le nombre de curs par processeur. Ainsi, les superordinateurs contiennent des centaines de noeuds, chacun contenant plusieurs processeurs multi-curs ; avec certains de ces processeurs possédant plusieurs niveau de cache, ou mettant en uvre du multi-threading Ces architectures permettent une grande performance de calcul, mais créent une grande variabilité au niveau des calculs et une incertitude sur la prédiction et de l'évaluation de performance. Il devient difficile d'estimer les performances ńanormalesż, et d'en tirer des conclusions.

En outre, cette complexité d'architecture pousse les applications parallèles à utiliser des technologies comme MPI (Message Passing Interface) pour communiquer entre les différents curs d'un processeur, point à point ou de manière collective, afin que chaque processus puisse effectuer une partie d'un grand calcul. MPI a donc aidé aux développement de superordinateurs de plus en plus puissants, ce qui a conduit à l'établissement du TOP500 des machines les plus performantes. Ce TOP500 est basé sur les résultats du benchmark HPL, calculant la décomposition LU d'une matrice de grande taille et utilisant MPI. Il devient alors intéressant de simuler l'exécution d'applications parallèles sur des systèmes de haute performance, notamment avec SimGrid, un simulateur doté de plusieurs outils comme SMPI, une ré-implémentation de MPI sur SimGrid.

2.2 TODO Travaux de Tom : prédictions d'applications MPI

2.3 Type de mesures et de modèles

Les mesures récupérées pour ces expériences sont assez expérimentales, et peuvent être biaisées en fonction de la température interne des machines, de divers effets de cache, de la rapidité d'un cur par rapport à un autre, etc. De plus, les systèmes analysés ne sont pas toujours ergodiques ou

stationnaires ; c'est à dire qu'une collection d'échantillons aléatoires du système ne représentent pas forcément ses propriétés statistiques, et que le système peut changer dans le temps.

Ces contraintes ont poussé Tom à définir plusieurs types de modèles, selon les mesures. On définit M-x N-y comme un modèle de complexité x, avec un bruit de complexité y. Par exemple :

- M-0 indique un modèle où la durée d'exécution est constante et indépendante des paramètres du modèle. De même N-0 indique l'absence de bruit.
- M-1 indique un modèle linéaire, où la durée dépend d'une combinaison des paramètres donnés (souvent un paramètre dépendant de x et un paramètre constant). De même N-1 indique un bruit avec une distribution normale.
- M-2 indique un modèle polynomial, et de même pour N-2.
- M_H et N_H sont des notations spécifiques répondant à la problématique de variabilité spatiale, et indiquant donc que les mesures doivent être effectuées par hôte.
- M' indique un modèle linéaire pour certaines valeurs spécifiques, et N' un bruit dont la distribution serait une mixture de gaussiennes.

Ces notations ont ensuite été utilisées pour déterminer quel type de modèle utiliser. Il a été choisi que le noyau dgemm utiliserait un modèle M_H-2 N_H-2, tandis que pour les autres noyaux de calcul un modèle M-1 N-2 suffirait. Les communications MPI, étant linéaires en fonction de la taille du message mais dépendant du protocole utilisé, ont été modélisées par un modèle M'-1 N'-1.

2.4 Limitations des travaux précédents, objectifs du stage

Il existe quelques limitations à ce travail de simulationă: la prise en compte des variabilités spatiales et temporelles, ainsi que la spécificité du système, ont forcé Tom à utiliser des modèles et des solutions ad hoc pour ses estimations. En effet, les modèles choisis l'ont été en connaissance de cause, après avoir déjà remarqué les spécificités des différents noyaux de calcul à simuler : par exemple dgemm est plus long à s'exécuter sur certains nuds, et possède des valeurs pour la taille des matrices pour lesquelles la durée est systématiquement plus longue que pour d'autres, ce qui indique un comportement non linéaire. De même pour les communications réseaux discontinues. Il a également du générer du code, notamment ajouté un appel à la fonction random pour prendre en compte la variabilité temporelle. Cette solution fonctionne, mais ne permet pas une vision à long terme et une utilisation de ce travail dans un autre contexte.

Considérant les limitations mentionnées, l'objectif principal de mon stage était d'estimer la possibilité d'avoir une solution plus générique avec un sampler Bayésien, soit des modèles généraux pouvant facilement s'appliquer à plusieurs noyaux de calcul, voire même aux communications réseau, sans avoir à être beaucoup changés. En effet, on aurait besoin de modèles génériques, souvent linéaire mais parfois avec des ruptures ou des mixtures, pouvant s'adapter à des besoins un peu particuliers. Pour cela il fallait donc élaborer des modèles correspondant à des noyaux de calculs, puis les évaluer en terme de résultats et de performance. La précision des modèles et leur proximité à la réalité, la rapidité des estimations ainsi que la variabilité entre les estimations sont d'autant de problématiques que j'ai du aborder.

Avant de commencer mon stage, certaines contraintes avaient déjà envisagées par Arnaud et Tom ; notamment la complexité de certains modèles (surtout les modèles hiérarchiques), ainsi que la prise en compte des spécificités des noyaux de calculs, telles que la présence d'un bruit non linéaire ou le besoin de séparer les estimations selon les CPUs utilisés.

Chapitre 3

État de l'Art

3.1 L'approche Bayésienne

L'approche Bayésienne des statistiques interprète les probabilités comme une mesure d'incertitude, et les résultats comme des estimations. L'analyse Bayésienne n'a pas pour but de trouver un point précis du résultat, mais de trouver sa distribution. L'idée est donc de reconnaître l'existence de plusieurs chemins possibles, avec différentes probabilités, et d'élaguer les chemins au fur et à mesure selon les informations que l'on possède pour ne garder que le plus probable, ce qui peut se faire avec des connaissances préalables qu'on appellera prior.

Le théorème de Bayes est le suivant :

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B)} \tag{3.1}$$

Autrement dit, on cherche la probabilité de A sachant B, en fonction de notre connaissance de la probabilité de B sachant A et des probabilités de A et de B. On a donc une hypothèse dont on essaye de déterminer la probabilité selon les données qu'on possède déjà et nos connaissances préalables qu'on appellera prior.

On peut aussi écrire le théorème de la façon suivante :

$$p(A|B) \propto p(B|A) * p(A) \tag{3.2}$$

Ceci indique que la distribution du postérieur (la probabilité de A sachant B) est proportionnelle à la combinaison de la fonction de vraisemblance (ou likelihood) de cette distribution (la probabilité de B sachant A) et de nos priors sur les paramètres (la probabilité de A). L'approche bayésienne permet d'actualiser nos connaissances sur la distribution des paramètres des modèles. Les modèles sont construits au fur et à mesure, et s'actualisent à chaque fois que l'on récupère des données qui confirment ou réfutent nos hypothèse initiales. On a donc un système d'apprentissage. En théorie, si l'on a une grosse quantité de données ou si les priors sont peu précis, les données importent beaucoup plus que les priors (à tel point qu'ils deviennent presque inutiles), mais il est possible que l'impact du prior demeure malgré tout. De plus, des mauvais priors ne devraient pas impacter négativement les résultats, ils n'auront juste aucune utilité.

Pour connaître la distribution du postérieur, on fait des tirages d'échantillons de données jusqu'à l'approximer. L'échantillonnage (sampling) permet de trouver des valeurs proches des paramètres ayant permis de générer les données ainsi que leur distribution de probabilité, et de mieux comprendre cette dernière pour pouvoir ensuite l'exploiter, avec par exemple la simulation de nouvelles prédictions pour le modèle. Pour cela, l'algorithme de sampling parcours des chaînes de Markov qui

ont pour lois stationnaires les distributions à échantillonner. On expliquera le procédé de simulation du sampler Stan qui a été utilisé dans la section suivante.

L'approche Bayésienne consiste donc à trouver une distribution correspondant aux paramètres en utilisant une méthode intuitive : on pars de nos connaissances préalables, et en fonction des données qu'on dispose on affine notre modèle. Cette approche est donc utile dans des situations où on veut pouvoir renseigner des priors et quantifier notre incertitude par rapport aux résultats.

Il existe plusieurs samplers Bayésiens, mais ce domaine est encore assez récent car l'approche Bayésienne requiert une grande puissance de calcul que les ordinateurs n'avaient pas jusqu'à assez récemment. La majorité des samplers utilisent un procédé de simulation appelé Markov Chain Monte Carlo (MCMC) qui suit une variante de l'algorithme de Metropolis-Hastings. Cet algorithme fonctionne de la manière suivante. A chaque itération :

— On pars d'un point initial, représenté par le tirage précédent
— On propose d'aller sur un autre point, et on évalue si la distribution avec ce nouveau point explique mieux les données que l'ancienne distribution, donc si la probabilité d'obtenir nos données avec ces nouveau paramètre est plus élevée.
— Si oui on fait un tirage sur ce nouveau point

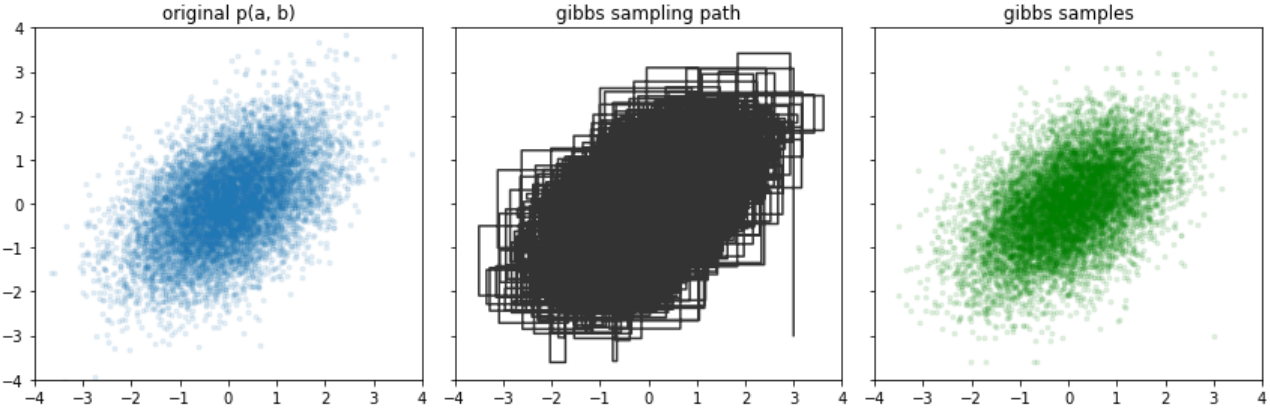

FIGURE 3.1 – source : https://jessicastringham.net/2018/05/09/gibbs-sampling/

On peut reconnaître cet algorithme dans l'image du milieu, où on comprend que la simulation a commencé à peu près au point (3,-3) et s'est ensuite rapprochée au fur et à mesure de la zone où il y avait les données.

3.2 L'approche Machine Learning

L'approche machine learning suit un principe d'auto-apprentissage : contrairement à l'approche bayésienne qui fait correspondre les données à un modèle avec des hypothèses à vérifier pour trouver les paramètres, l'objectif du machine learning est de trouver un modèle approximant les paramètres à l'origine des données, à l'aide duquel on va pouvoir effectuer des prédictions. La notion d'apprentissage est équivalente à construire le modèle qui se rapprochera le plus des données.

L'algorithme de machine learning récupère des estimations produites par une machine, dont la performance dépend des données rencontrées, et plus il rencontre d'observations, plus il s'améliore et récupère des estimations précises. La démarche consiste donc à faire une expérience plusieurs fois, et à calculer la probabilité empirique des résultats à chaque fois. Plus le nombre de fois qu'on fait l'expérience est élevé, meilleurs seront les résultats. Cependant, comme on fait une approximation de la réalité, on a une perte d'information qui correspond à un bruit non modélisé indépendant des

données.

On a trois algorithmes principaux : la descente de gradient, l'estimateur du maximum de vraisemblance et le clustering. La descente de gradient est un algorithme dont le but est de trouver le minimum d'une fonction dérivable et dont on connaît l'expression mais où le calcul du minimum est compliqué. Il suit une approche itérative qui à chaque pas calcule la pente de la fonction (sa dérivée) en fonction du point de départ et y avance plus ou moins selon la taille du pas d'apprentissage ; et ceci jusqu'à converger en un minimum.

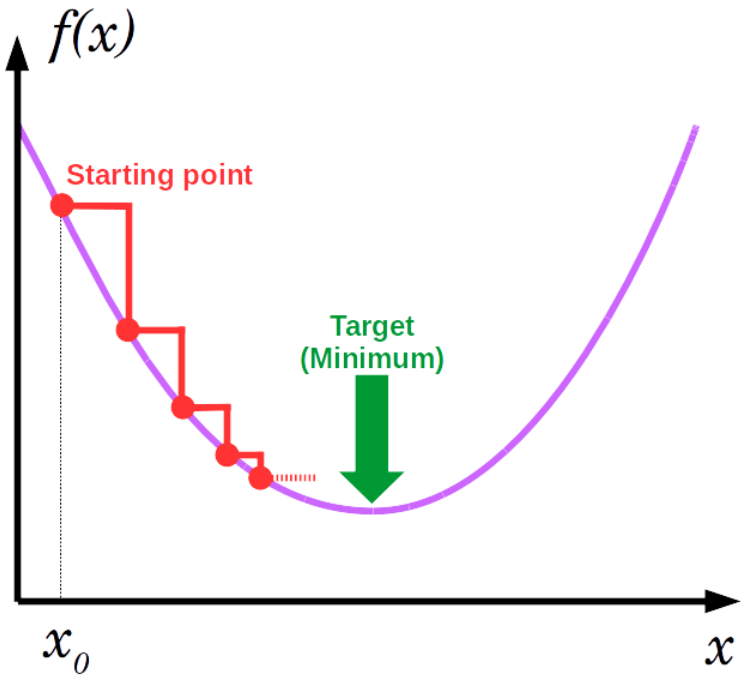

FIGURE 3.2 – source : https://www.neural-networks.io/fr/single-layer/gradient-descent.php

Attention à taille de : plus il est grand plus on avance loin à chaque pas donc plus on réduit théoriquement les itérations, mais si est trop grand on risque de manquer le minimum (surtout si le tracé de la fonction est un peu particulier) et d'avoir un comportement divergent. En revanche plus est petit plus on avance lentement, mais avec plus de chances de converger au final. Il existe deux limites à cet algorithme : les minimums locaux et le ńvanishing gradientż. En effet selon la valeur de départ choisie l'algorithme peut partir sur une mauvaise pente et s'arrêter sur un minimum local, mais pas global. Il faut donc que la valeur de départ soit plus proche du minimum recherché que d'un minimum local pour trouver un bon résultat. Le ńvanishing gradientż indique un tracé de fonction avec des valeurs ńplateauż où l'algorithme se bloquerait, l'empêchant de trouver le minimum.

L'estimateur du maximum de vraisemblance est un algorithme permettant d'estimer la valeur des paramètres maximisant la vraisemblance L d'un échantillon. L est la fonction de densité à paramètres correspondant à un échantillon de variables aléatoires discrètes. Soit : $L() = p(x1...xN|)$ représentant la probabilité d'avoir les observations x1…xN étant donné les paramètres . Les estimateurs du maximum de vraisemblance des paramètres sont les valeurs maximisant L, soit minimisant la fonction de perte. L'objectif de l'algorithme est donc d'inférer les paramètres de la loi de probabilité d'un échantillon en trouvant les valeurs des paramètres permettant d'atteindre le maximum de la vraisemblance L.

Enfin l'algorithme de clustering permet d'identifier et de former des petits groupes séparés par-

tageant des caractéristiques communes parmi les données. On peut indiquer en amont les différents groupes, ou juste leur nombre et laisser l'algorithme les trouver, par exemple avec l'algorithme kmeans qui affecte les données aux clusters selon leur proximité (au sens de la somme des carrés) aux points médians des clusters. Les emplacements des points médians sont affinés selon que l'on ajoute des points au cluster.

3.3 Le fonctionnement de Stan

Stan utilise l'algorithme MCMC présenté précédemment, ce qui permet à la simulation de parcourir un espace de valeurs possibles assez rapidement. Le procédé a également une période de ńwarm upż, où les tirages partent d'un point initial et peuvent donc être très éloignés des valeurs réelles et des autres tirages. Une fois le warm up terminé, le procédé a déterminé une zone réduite pour faire les tirages, et va alors continuer à l'affiner jusqu'à trouver des valeurs assez précises. Ce procédé de simulation fonctionne mieux lorsqu'on le lance plusieurs fois, soit avec plusieurs chaînes : en effet puisque les chaînes ne commencent pas au même point initial, on peut avoir une certaine confiance en notre résultat si on s'aperçoit qu'elles convergent (pour les itérations d'échantillonnage, puisque les résultats des itérations de ńwarm upż ne donnent pas des résultats significatifs).

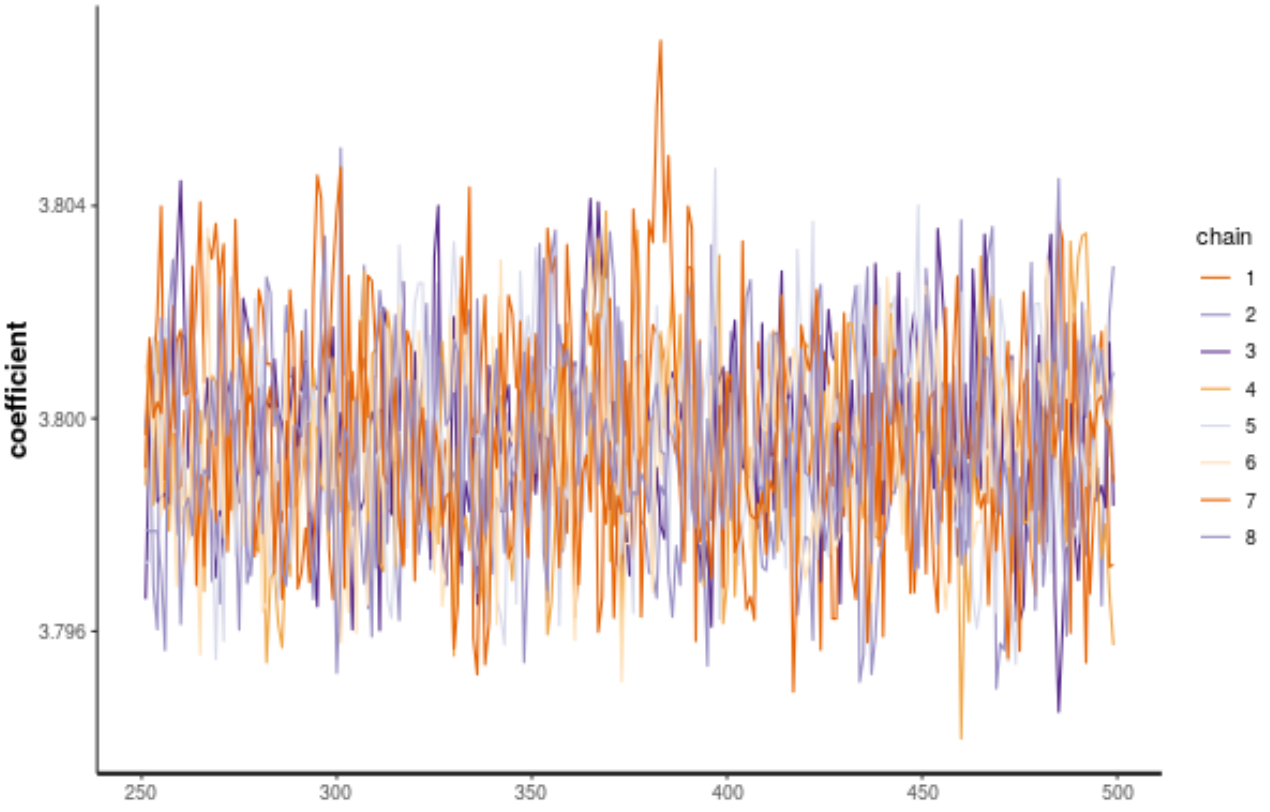

FIGURE 3.3 – trace du paramètre coefficient

En effet ici les 8 chaînes ont convergé autour de la même zone : environ la valeur 3,8.

Stan a une syntaxe sous forme de sections, ou bloc. Chacun des blocs a un but précis, et toute variable déclarée dans un bloc est accessible aux prochains, mais pas forcément aux précédents. Le bloc ńdataż permet de déclarer les données que l'on va fournir au sampler. On peut donner des limites

à ces données, comme préciser que certaines sont forcément positives, que d'autres sont sous forme de vecteur ordonné par valeur croissante, etc. Le bloc ńtransformed dataż permet de créer de nouvelles données, souvent à partir des données initiales. Le bloc ńparametersż indique les paramètres à estimer par le modèle. On peut seulement y déclarer des variables, et celles ci ne peuvent pas être des entiers. Le bloc ńtransformed parametersż permet de déclarer et assigner des valeurs à d'autres paramètres. Enfin le bloc ńmodelż permet d'indiquer les priors et la likelihood, et le bloc ńgenerated quantitiesż permet de créer de nouvelles données, de faire des prédictions sur les nouvelles données, etc. Cette syntaxe permet d'écrire des modèles précis, facilement compréhensibles.

Stan requiert obligatoirement l'utilisation de priors (si aucun n'est renseigné il utilise des priors non informatif par défaut), afin de faire mieux correspondre la distribution trouvée à nos données : les priors, surtout lorsqu'ils sont informatifs, permettent d'affiner les résultats. Cependant si on a assez peu d'informations, il est possible de donner un prior non informatif comme normal(0,10) ; ceci laisse un grand impact aux données dans le calcul du postérieur.

Une fois que la simulation a été faite, il faut vérifier les résultats trouvés. On peut commencer par une vérification graphique de la convergence des chaînes, comme mentionné précédemment : la convergence n'indique pas forcément un bon résultat, mais la non convergence est un signe que la simulation ne s'est pas bien déroulée, et qu'il faut sans doute changer le modèle c'est à dire ajouter des paramètres, modifier les priors, etc. De plus, si des chaînes démarrent à un point puis s'en éloignent beaucoup pour rester autour d'une autre zone, cela indique un problème au niveau des valeurs initiales à partir desquelles les tirages sont effectués.

A la fin de la simulation, il est aussi fréquent que Stan donne des avertissements indiquant les potentiels problèmes : les plus courants sont une simulation trop longue ou un manque d'information au niveau du postérieur. Il est également possible d'utiliser les outils de diagnostics du sampler afin de récupérer des informations sur les trajectoires divergentes, le temps de simulation, un résumé des valeurs trouvées, les valeurs initiales utilisées, etc. Il existe par ailleurs un package appelé shinystan offrant une interface graphique très détaillée aux outils de diagnostics. On en voit une partie dans l'image ci dessous.

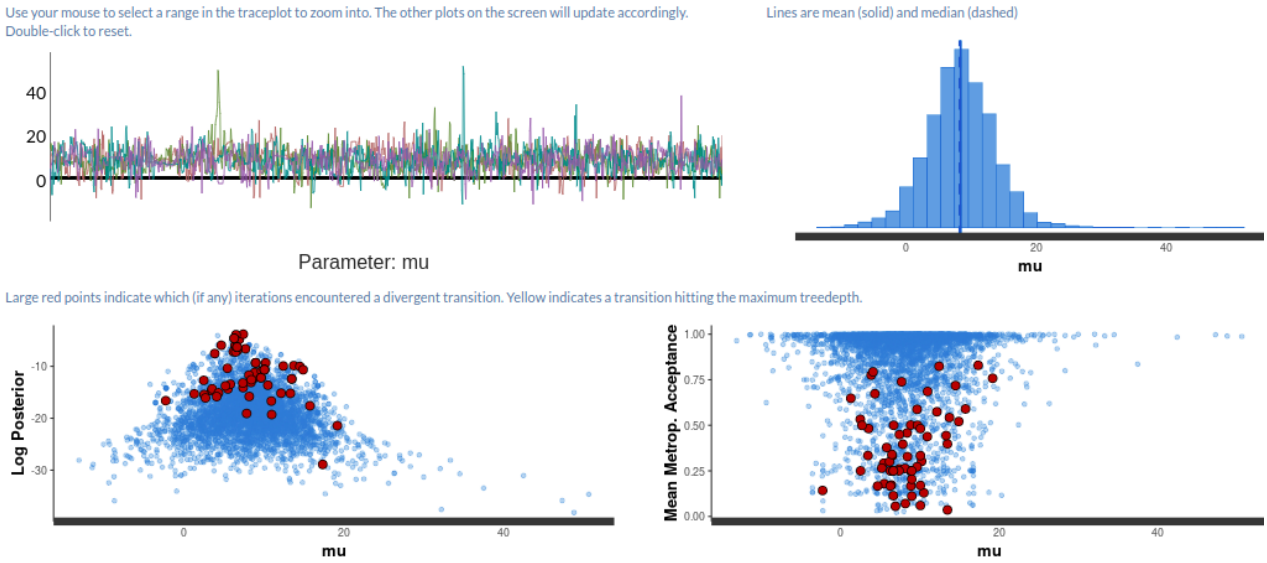

FIGURE 3.4 – L'interface de shinystan, avec l'affichage du log postérieur et des trajectoires divergentes en rouge

Enfin le plus important est de vérifier les valeurs trouvées pour les paramètres, et si elles ont du

sens par rapport au modèle : vérifier l'histogramme des paramètres pour voir si les priors donnés sont correct ou non, et essayer de régénérer de nouvelles données avec les paramètres pour comparer avec les données initiales.

Chapitre 4

Méthodologie

Une des problématiques auxquelles mon stage, comme tous les stages de recherche, devait répondre est la reproductibilité : en effet par soucis de transparence mes expériences doivent pouvoir être refaites de façon exacte, donc l'environnement de travail doit être contrôlé et les outils et données utilisées doivent être notés et disponibles. La problématique de reproductibilité m'a été présentée au travers du MOOC réalisé par Arnaud Legrand et d'autres.

Pour cela, mais également pour rendre le suivi de stage plus aisé, j'ai maintenu pendant ces trois mois un cahier de laboratoire, réalisé en Org-Mode sur l'éditeur de texte Emacs, que j'ai partagé sur GitHub. Ce cahier, complété quotidiennement, contenait non seulement les résultats majeurs de mes recherches mais aussi tous les détails de mon travail : les objectifs, le travail réalisé, les résultats et les conclusions tirées, les problèmes rencontrés, les corrections, etc. Ce journal a permis à mes encadrants de pouvoir suivre mon travail au jour le jour de façon très aisée, le document étant structuré de façon chronologique et thématique, avec des sections dépliables et une planification des tâches sous forme de Todo list. Mes encadrants pouvaient donc me faire des retours réguliers sous forme d'échanges par mail ou de réunion hebdomadaire pour définir les objectifs du stage au fur et à mesure.

De plus, la grosse majorité de mes expériences ont été réalisées sur ce cahier, à l'exception de celles réalisées sur Grid5000. En effet, Org-Mode inclus un langage de balisage similaire à Markdown, permettant d'exécuter du code sur le journal : celui ci contient donc des sections en langage naturel, suivi de sections de code avec différents langages de programmation. Org-Mode a donc permis de regrouper en un seul journal les notes de mes recherches et les expériences.

Cependant l'exécution de code sur le cahier de laboratoire n'était pas adapté à toutes mes expériences, qui pouvaient être très longues. J'utilisais alors Grid5000, qui est un testbed mis à la disposition des chercheurs pour la recherche reproductible, regroupant 12000 curs et 800 nuds en cluster dans toute la France. Il permet ainsi d'effectuer aisément des expériences à grande échelle liées au calcul de haute performance, et cela avec beaucoup de contrôle sur l'environnement (traçabilité, reconfiguration à chaque demande d'obtention d'un nud, possibilité d'exporter puis réimporter un environnement).

Enfin, j'utilisais à l'occasion l'environnement de développement Rstudio pour conduire certains tests, son interface graphique rendant les résultats plus facilement visibles et compréhensibles. Il a aussi été décidé dès le début de mon stage que le sampler Bayésien que j'utiliserai serait Stan, principalement en raison des connaissances préalables de mes encadrants de cet outil.

Chapitre 5

Contributions

5.1 Elaboration de modèles

Comme le but du stage était de comparer l'échantillonnage Bayésien à la maximisation de vraisemblance, j'ai commencé par faire un modèle simple des données avec une régression linéaire. Je me suis rapidement aperçu que celle ci avait deux inconvénients : on ne pouvait pas modéliser un bruit non linéaire, et le paramètre indépendant de x avait tendance à avoir des valeurs étranges car il n'était pas significatif dans la génération des données. Le problème est qu'il introduisait donc un biais dans l'estimation de nouvelles données à partir des paramètres. Ce modèle n'était donc pas idéal, et le but était de pouvoir l'écrire plus proprement, et d'avoir des résultats plus significatifs avec Stan.

Avant de réaliser des modèles sur les données des noyaux de calcul, j'ai travaillé avec des simples données générées, pour me familiariser avec l'outil Stan mais aussi pour résoudre des problèmes que je mentionnerais dans la section suivante, liés à la précision de la simulation. Ces premiers tests ont permis de remarquer que les modèles écrits en Stan sont très complets, et donc facilement compréhensibles, mais cela n'influe pas sur leur complexité : on peut très bien écrire des modèles très simples, qui s'exécuteront rapidement.

Ensuite j'ai travaillé sur les données de la fonction dgemm de OpenBlas fournies par Tom : plus précisément sur la durée d'exécution de cette fonction en fonction de la taille de la matrice (déterminée par le paramètre M*N*K). J'ai commencé par écrire un modèle linéaire avec du bruit polynomial (M-1 N-2) : celui ci contenait deux paramètres constants β et δ et deux paramètres dépendant de M*N*K : α et γ. La figure ci-dessous illustre ce modèle. J'ai ensuite écrit un modèle polynomial avec le même bruit (M-2 N-2), puis j'ai ajouté de la complexité à ces modèles par couche.

Le modèle polynomial est très similaire, la principale différence étant l'inclusion de plus de paramètres. En effet, cette fois ci on considère l'influence des coefficients M*N, M*K et N*K dans la vitesse d'exécution. La likelihood est donc légèrement modifiée :

$$duration \sim \mathcal{N}(\alpha_1 \cdot mnk + \alpha_2 \cdot mn + \alpha_3 \cdot mk + \alpha_4 \cdot nk + \beta, \gamma_1 \cdot mnk + \gamma_2 \cdot mn + \gamma_3 \cdot mk + \gamma_4 \cdot nk + \delta) \quad (5.1)$$

Par la suite, j'ai réécrit ces deux modèles en ajoutant une variable déterminante sur laquelle les estimations des paramètres devaient s'effectuer : le CPU utilisé. Dans les données fournies, dgemm avait été lancée sur 64 CPU différents. Les deux modèles suivants ont donc été conçus pour estimer les paramètres pour les 64 hôtes différents. La principale différence de ces modèles était que la likelihood devait donc être définie selon les hôtes. On avait donc la formule suivante pour le modèle linéaire :

$$duration_i \sim \mathcal{N}(\alpha_i * mnk_i + \beta_i, \gamma_i * mnk_i + \delta_i) \quad (5.2)$$

```
"data {
    int<lower=0> N; //nombre de données
    vector[N] mnk; //taille de la matrice
    vector[N] duration; //durée d'exécution
}
parameters {
    real alpha;
    real beta;
    real<lower=0> gamma; //contrainte positive sur le bruit
    real<lower=0> delta;
}
model {
    //priors sur nos paramètres
    alpha ~ normal(6e-11,6e-12);
    beta ~ normal(7e-07,7e-08);
    gamma ~ normal(1e-12,1e-13);
    delta ~ normal(7e-07,7e-08);

    //likelihood
    duration ~ normal(alpha*mnk+beta, gamma*mnk+delta);
}
"
```

FIGURE 5.1 – modèle linéaire, bruit polynomial

Et de même pour le modèle polynomial. Ces deux modèles permettent de simuler la performance de tous les noyaux de calculs utilisés dans HPL.

Cependant, on pourrait se demander si les estimations sont vraiment indépendantes selon les CPUs utilisés, s'il n'y aurait pas une distribution de probabilité des valeurs moyennes des paramètres. On estimerait alors la formule suivante (et de même pour les autres paramètres) :

$$\alpha_i \sim \mathcal{N}(\mu_\alpha, \sigma_a) \tag{5.3}$$

On chercherait alors à estimer principalement les valeurs des deux paramètres supplémentaires, qu'on appellera hyperparamètres, car une fois qu'on aura leur distribution de probabilité, on pourrait calculer des nouvelles valeurs α, β, γ et δ pour un nouveau CPU.

Dans ce modèle hiérarchique, on dira que $\mu_\alpha \sim \mathcal{N}(\alpha_moy, \alpha_sd)$ où α_moy et α_sd sont les priors et $\sigma_\alpha \sim \mathcal{N}(0, 1)$.

Le modèle hiérarchique a donné des bonnes estimations pour le modèle linéaire, mais des estimations assez moyennes avec le modèle polynomial, avec des valeurs un peu étranges et des chaînes qui ne convergeaient pas. On commence à observer une limite de Stan, qui permet d'écrire clairement des modèles assez complexes, mais a parfois du mal à les évaluer si on ne lui donne pas beaucoup d'indications.

Enfin, après avoir remarqué sur les histogrammes des paramètres que l'un d'entre eux (alpha précisément) ne ressemblait pas à une distribution normale mais plus à une mixture de distributions

normales, j'ai écrit un modèle incluant cette contrainte. Ce dernier modèle diffère un peu plus des précédents en raison de la syntaxe nécessaire pour indiquer qu'un paramètre est une mixture de gaussiennes. En effet, pour écrire une likelihood correspondant à une mixture de deux gaussiennes, la syntaxe est la suivante :

$$target = target + log_mix(\theta, normal_lpdf(y_n|mu_1, sigma_1), normal_lpdf(y_n|mu_2, sigma_2)) \tag{5.4}$$

Ici theta correspond à la proportion de données dans les courbes, et on exprime ensuite la présence de deux distributions normales, avec mu_1 et sigma_1 puis mu_2 et sigma_2.

Ce modèle n'est pas conclusif : l'expression d'une mixture de gaussiennes fonctionne relativement bien sur des données générées, lorsque cela concerne le résultat, mais lorsqu'on veut l'appliquer à un paramètre du modèle hiérarchique la simulation a besoin de priors extrêmement précis, et les résultats obtenus ne reflètent que les valeurs de ces priors.

J'ai comparé la performance et les résultats de ce modèle avec ceux d'une simple régression linéaire et d'un outil de clustering comme kmeans ou mclust. Pour cela, on a décidé de ne pas prendre en compte les paramètres beta et delta : en effet ils ont tous les deux une influence assez faible avec le postérieur, et ne sont pas corrélés à d'autres paramètres comme le sont alpha et gamma. Ainsi, on a effectué une simple régression type $lm(duration \sim mnk + 0)$, puis on a récupéré la moyenne et l'écart type du paramètre gamma ; ainsi que l'écart type de alpha (on assume qu'il est à peu près similaire pour les deux moyennes).

On a ensuite utilisé un outil de clustering pour regrouper les estimations de alpha en deux clusters ; puis récupéré les deux moyennes de alpha et la fréquence des points pour chaque cluster. Ces opérations ont été très rapides, et nous ont donc permis d'avoir les estimations des paramètres d'un modèle hiérarchique (mais pas leur distribution, on ignore l'incertitude qu'on a sur ces estimations). De plus ces estimations ne prenaient pas en compte le fait que le bruit n'est pas linéaire. Malgré tout, les estimations par régression linéaire du modèle hiérarchique étaient toutes aussi précises que celles de Stan, tout en étant beaucoup plus rapides à effectuer.

5.2 Amélioration de la précision de la simulations

Comme mentionné précédemment, Stan peut évaluer des modèles très complexes, mais a souvent besoin d'aide et d'indication pour avoir des résultats précis. Tout d'abord il faut optimiser l'écriture des modèles autant que possible, en écrivant les priors sous forme vectorielle et en évitant les boucles, pour limiter le temps d'exécution. Il y a également des techniques d'écriture, comme la décomposition QR de matrices, telle que la matrice réelle $A = Q * R$ avec Q une matrice orthogonale et R une matrice triangulaire supérieure. Cette décomposition permet de réduire la corrélation entre les paramètres utilisés pour calculer le postérieur et réduit le temps de simulation sans impacter négativement les résultats.

De plus, dès que l'on utilise des données de taille très petite (de l'ordre de 10^-5 à 10^-12), il faut écrire les modèles sous la forme de paramétrisation non centrée, car nos données ne sont pas assez informatives. Cette forme se caractérise par l'introduction de nouveaux paramètres, qui correspondent à des variables gaussiennes centrées en zéro. Ces variables permettent au sampler de trouver plus facilement les autres paramètres.

Ensuite, une autre façon d'offrir des indications à Stan est de lui donner des priors précis. En effet, les priors permettent d'améliorer la convergence des chaînes en leur indiquant plus précisément une direction à suivre, ce qui évite donc qu'elles fassent des tirages dans une zone trop large et finissent donc avec des résultats peu précis. Plus le modèle est complexe, plus il est préférable de donner des priors informatifs, soit assez proche des valeurs des paramètres, car sans le sampler arrivera

```
"data {
    int<lower=0> N;
    vector[N] mnk;
    vector[N] duration;
}
parameters {
    real alpha; //paramètres supplémentaires
    real beta;
    real<lower=0> gamma;
    real<lower=0> delta;
}
transformed parameters {
    real alpha = 6e-11 + alpha_raw * 6e-12;
    real beta = 7e-07 + beta_raw * 7e-08;
    real gamma = 1e-12 + gamma_raw * 1e-13;
    real delta = 7e-07 + delta_raw * 7e-08; //équivalent à delta ~ normal(7e-07,7e-08)
}
model {
    alpha ~ normal(6e-11,6e-12);
    beta ~ normal(7e-07,7e-08);
    gamma ~ normal(1e-12,1e-13);
    delta ~ normal(7e-07,7e-08);
    duration ~ normal(alpha*mnk+beta, gamma*mnk+delta);
}
"
```

FIGURE 5.2 – modèle linéaire avec paramétrisation non centrée

à converger mais aura des résultats erronés. De plus, l'utilisation de priors informatifs permet de réduire le temps de calcul de la simulation, puisque celle ci passe moins de temps à chercher la bonne zone où faire les tirages.

Cependant un compromis existe entre priors trop peu informatifs et trop informatifs, à savoir qu'un prior peu informatif serait par exemple $a \sim \mathcal{N}(0, 1)$ si la distribution du paramètre a est $a \sim \mathcal{N}(7.49e-07, 6.69e-08)$. Tout d'abord il faut considérer que les priors sont des connaissances ou hypothèses préalables, il n'est donc pas raisonnable de penser qu'elles puissent être extrêmement précises, et de plus il faut éviter de donner des priors erronés. En théorie, de telles indications devraient être plus ou moins ignorées par le sampler, qui basera uniquement son analyse sur les données comme expliqué précédemment ; cependant nos expériences prouvent le contraire. L'utilisation de priors erronés a donc tendance à biaiser le postérieur et floute donc nos résultats ; il faut donc être prudents quitte à donner des indications un peu moins précises.

Enfin, une dernière indication possible à donner est les valeurs initiales pour les chaînes. Cela permet en théorie d'améliorer leur convergence et de trouver des résultats plus précis. En pratique, lorsque l'on utilise des priors suffisamment informatifs la précision des valeurs initiales permet simplement d'accélérer un peu le temps d'exécution, et si on utilise des priors peu informatifs les valeurs initiales remplacent un peu leur rôle. Cependant le plus évident est de donner à peu près les mêmes valeurs entre la moyenne pour le prior et la valeur initiale du paramètre ; et donc dans ce cas les valeurs initiales impactent assez peu le postérieur.

5.3 Evaluation des modèles

Une fois que l'on a obtenu les résultats et vérifié que la simulation s'est bien déroulée (convergence des chaînes, pas de trajectoires divergentes ou un minimum), on peut vérifier les résultats graphiquement, en regardant leurs histogrammes ; mais cela ne nous permet pas de déterminer si le modèle est cohérent, et adapté à nos données.

Pour vérifier cela, on peut commencer par vérifier la sensibilité du modèle à des variations. Par exemple, on peut modifier un peu les priors avec d'autres valeurs plausibles, ou introduire plus de variables permettant de mieux expliquer le modèle. Nous avons effectué les deux par une étude des priors et de quelles valeurs permettaient à nos modèles de converger et d'avoir des résultats satisfaisants ; et en écrivant les modèles polynomiaux, qui permettent d'inclure un peu plus de données dans la distribution du postérieur. Nos modèles, surtout les plus complexes, ont ainsi tendance à être assez sensibles aux variations : les priors doivent être très précis et une variation sur ceux ci entraînera un problème de convergence ; et il y a de grandes différences entre les résultats du modèle linéaire hiérarchique et du modèle polynomial hiérarchique.

De plus, il est possible de visualiser graphiquement le postérieur, pour voir si les résultats trouvés ont du sens. L'outil ggpairs, fonctionnant en R avec ggplot, nous permet d'avoir sur un seul graphique l'histogramme des paramètres trouvés, mais également leur distribution par rapport aux autres paramètres, sous forme de nuage de points ou de densité. Cela nous permet d'observer d'éventuelles corrélations entre les paramètres qui pourraient poser problème au niveau de la simulation, et qui nous donnerait des indications qu'il faudrait réécrire notre modèle.

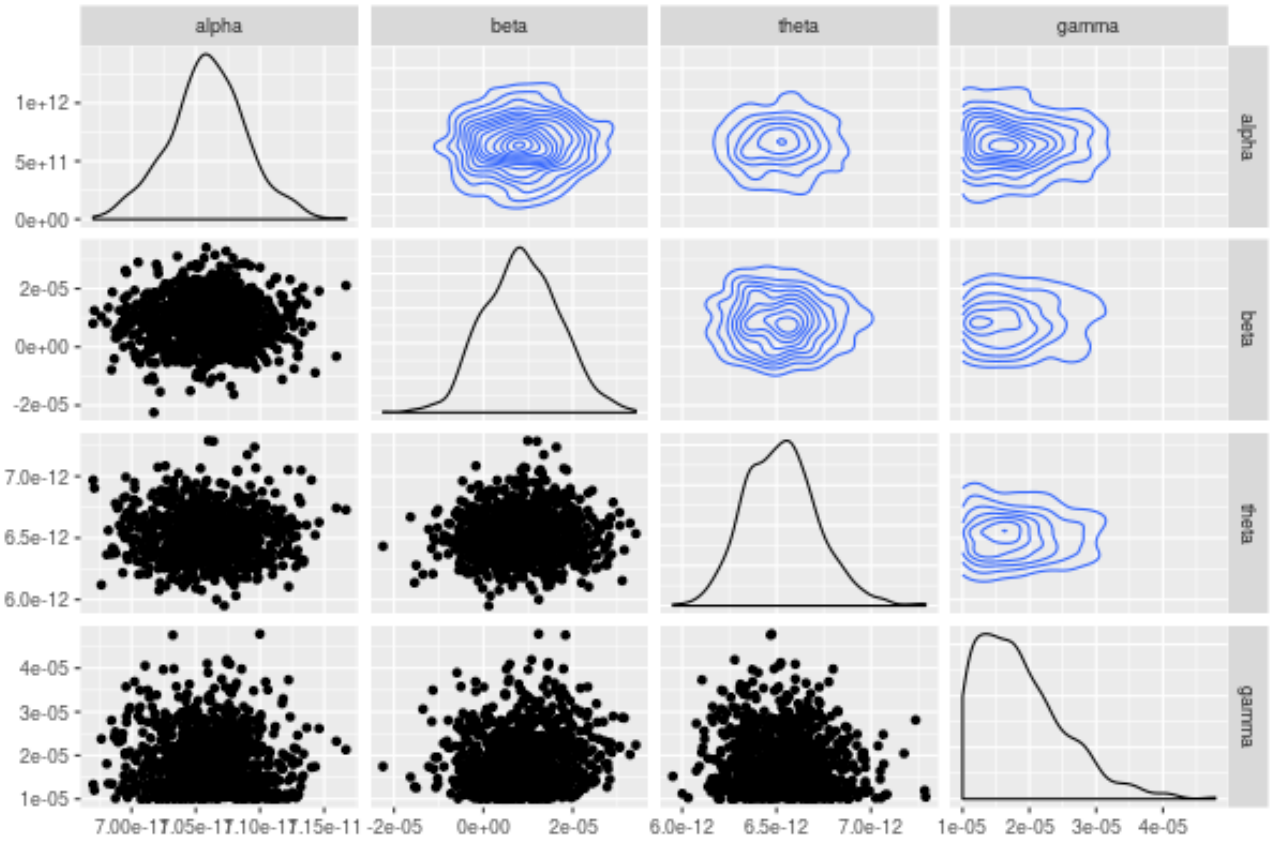

FIGURE 5.3 – Ggpairs avec modèle linéaire

On peut également dessiner les graphiques nous mêmes, à partir des distributions des paramètres trouvées par stan. Par exemple dans l'image ci dessous, nous avons dessiné la distribution d'alpha selon gamma, et ce pour chacun de nos 64 hôtes, avec une grande ellipse contenant 95% des distributions.

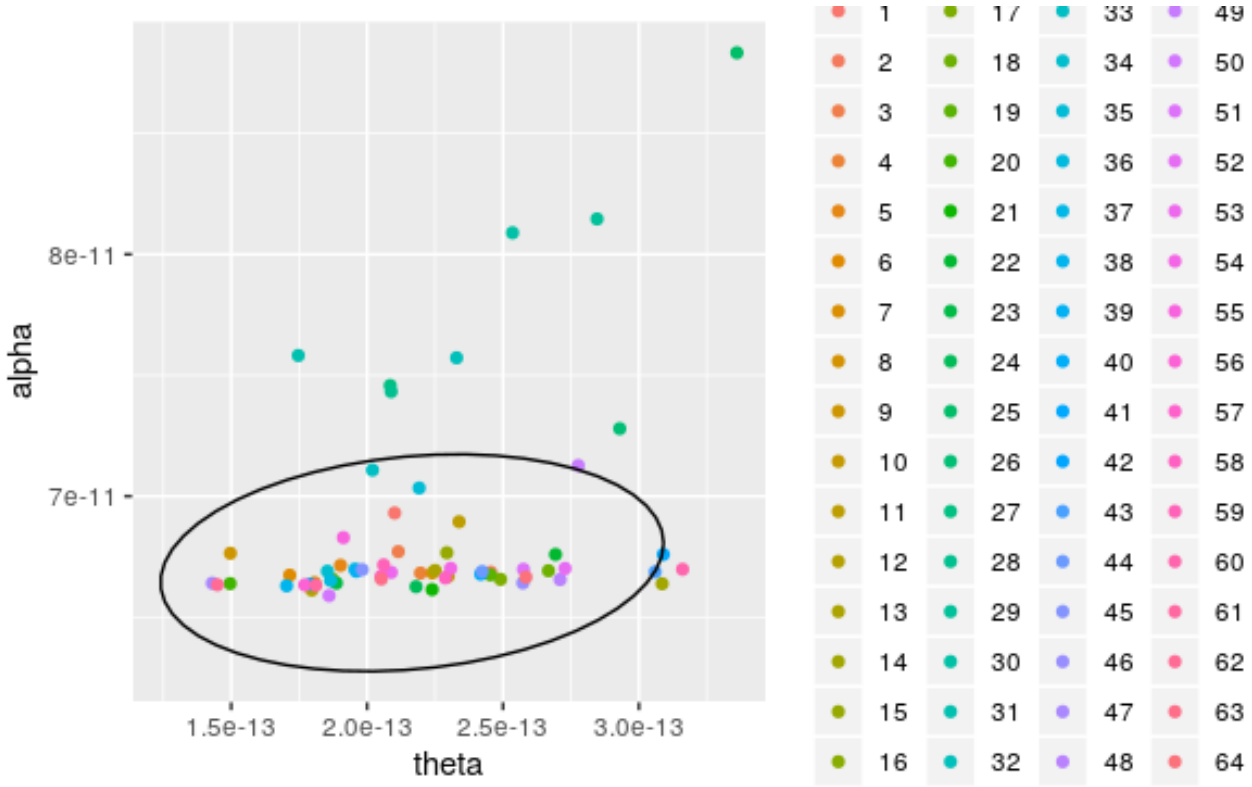

FIGURE 5.4 – distribution du paramètre alpha selon theta

Enfin le meilleur moyen de vérifier la précision du modèle après tous ces tests est de générer de nouvelles données à partir des prédictions des paramètres. Si notre modèle est précis, les données générées devraient à peu près couvrir les données initiales, et ne pas avoir trop de tirages où il n'y avait pas de données initiales. Stan permet la génération de nouvelles données à partir des paramètres estimés, mais on peut également le faire directement en R. L'image ci dessous montre une génération de données confirmant la précision du modèle.

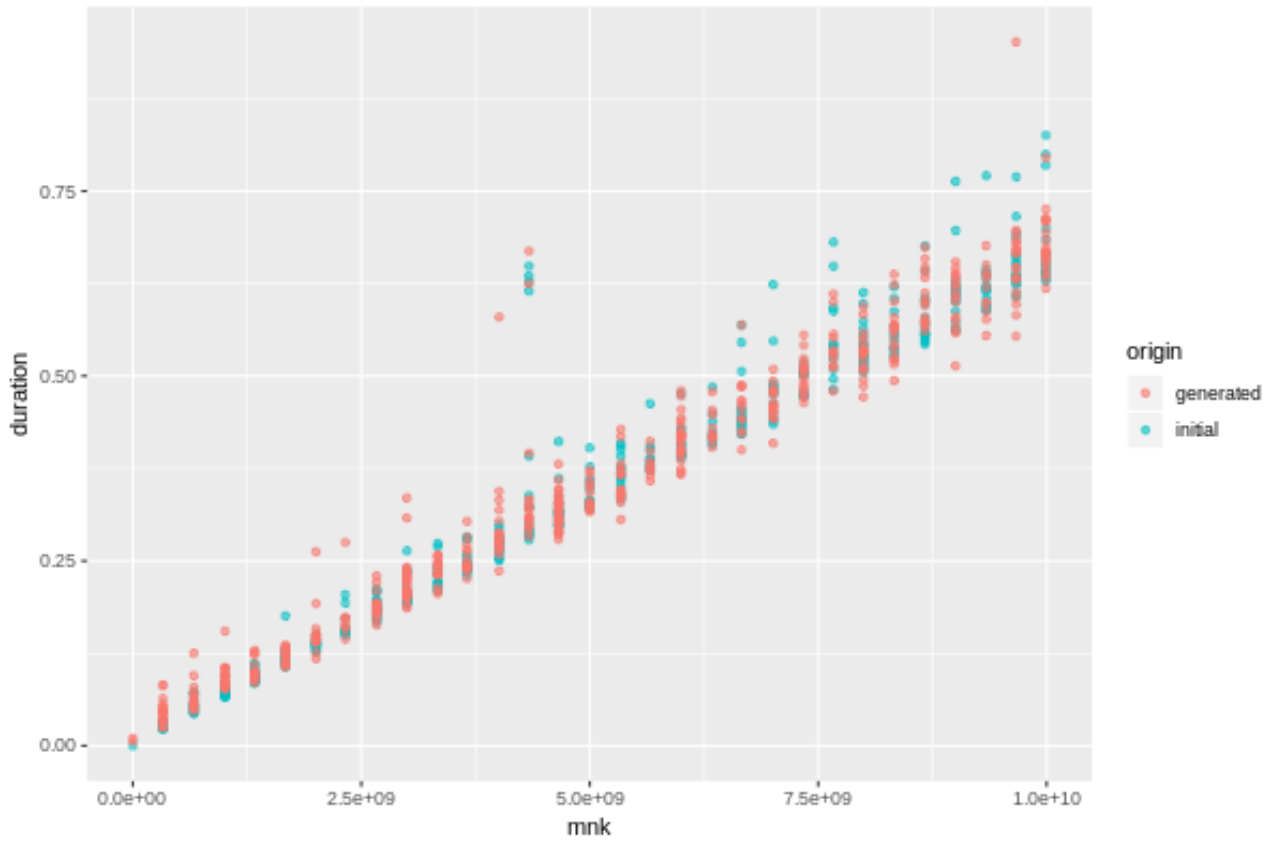

FIGURE 5.5 – Génération de nouvelles données, modèle polynomial

Chapitre 6

Conclusion

6.1 Sur Stan

Lors de ce stage, j'ai donc évaluer la viabilité de l'utilisation d'un sampler Bayésien tel que Stan pour la recherche sur Simgrid. J'ai créé des modèles permettant de représenter la performance d'un noyau de calcul, en y ajoutant de la complexité par couche afin de se rapprocher au plus possible de la réalité. Mes modèles sont peu adaptables aux changements tels qu'une variabilité dans les priors, mais adaptables au rajout ou à la suppression d'un hôte (surtout les modèles par hôte et les modèles hiérarchiques qui ont été conçus dans ce but).

Stan est un outil puissant permettant d'écrire des modèles précis et parfois très complexes ; rendant mieux compte de la réalité qu'une régression linéaire et un outil de clustering. Il permet de prendre en compte des hypothèses ou informations préalables, et permet d'avoir une mesure de l'incertitude de nos résultats. En principe, tout porterait à croire que ce serait un outil adapté pour les recherches de Simgrid.

Cependant certaines caractéristiques le rendent difficile à exploiter, notamment l'impact limité de la quantité de données au bout d'un certain seuil assez petit (peu de différence entre un échantillon de 2000 points et un de 5000 points à part le temps d'exécution). De plus, malgré toutes les indications que l'on peut lui donner, il semble que le sampling ne trouvera pas de résultats précis sans priors informatifs, ce qui implique donc d'avoir beaucoup d'informations sur nos données. De plus, malgré sa capacité à modéliser assez précisément l'exécution d'un noyau de calcul sur un cluster de plusieurs CPU, la simple durée des simulations le rend difficile à exploiter. En effet, même en utilisant Grid5000, la plupart des modèles ne s'exécutent pas en moins de 2 heures.

Ces limites, ainsi que les caractéristiques des données de recherche sur Simgrid (nombreuses, mais avec assez peu d'informations dessus), rendent mon travail assez improbable d'être implémenté dans la recherche de l'équipe Polaris. Surtout qu'il a été mis en évidence que l'utilisation d'outils plus simples (régression linéaire et mclust), bien que ignorant plusieurs paramètres, permettait également une modélisation assez proche de la réalité. Malgré tout, Stan serait peut être mieux adapté à d'autres usages, tels que la détection de nouveauté sur des données ; à savoir remarquer des gros changements dans une longue liste de données et les distinguer de la simple variabilité du modèle.

6.2 Bilan personnel

Ce stage a été pour moi une expérience extrêmement enrichissante dans un milieu qui ne m'était pas du tout familier jusqu'ici. Non seulement il m'a donné l'occasion de découvrir le secteur de la recherche, et m'a offert une autre perspective de travail que mon stage de DUT effectué dans une start-up, il m'a aussi permis de travailler sur des sujets que je ne maîtrisais pas vraiment, avec des

outils que je ne connaissais qu'assez peu.

J'ai pu apprendre énormément, à la fois sur l'aspect théorique de mon sujet de stage avec les nombreuses lectures que j'ai effectué pour comprendre les statistiques bayésiennes et la simulation de HPL, ainsi qu'avec les séminaires et soutenances de thèses que j'ai eu l'occasion d'assister, et sur l'aspect pratique de la prise de main de différents outils et de l'utilisation d'un testbed (Grid5000) pour effectuer des calculs. Enfin, j'ai eu la chance de pouvoir rencontrer des personnes passionnées par leur domaine, qui m'ont motivé à envisager le secteur de la recherche comme potentielle poursuite professionnelle.

Annexe A

Annexes

Le cahier de laboratoire, ainsi que les slides utilisées pour la pré-soutenance faite au laboratoire peuvent être trouvé à l'adresse suivante : https://github.com/hoellejal/automating-calculation-kernels-modelling

A.1 Bibliographie

— Fast and Faithful Performance Prediction of MPI Applications : the HPL Case Study. Tom Cornebize, Arnaud Legrand, Franz Heinrich.
— A Bayesian Course with examples in R and Stan. Richard McElreath.
— Bayesian Data Analysis, Third Edition. Aki Vehtari, Andrew Gelman, David B. Dunson, Donald Rubin, Hal S. Stern et John B. Carlin.
— Le MOOC sur la recherche reproductible
— Cours de Richard McElreath sur les statistiques bayésiennes
— Forum d'aide de stan
— https://mc-stan.org/users/documentation/case-studies/qr_regression.html
— https://betanalpha.github.io/assets/case_studies/identifying_mixture_models.html
— https://www.martinmodrak.cz/2018/02/19/taming-divergences-in-stan-models/
— https://www.grid5000.fr/w/Grid5000:Home
— http://modernstatisticalworkflow.blogspot.com/2017/04/an-easy-way-to-simulate-fake-data-from.html
— https://jessicastringham.net/2018/05/09/gibbs-sampling/