
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 Aélyl Jalmin.

Table des matières

1	Introduction	4
1.1	Environnement de travail	4
1.2	Développement durable	4
2	Contexte	5
2.1	Le calcul à haute performance	5
2.2	TODO Travaux de Tom : prédictions d'applications MPI	5
2.3	Type de mesures et de modèles	6
2.4	Limitations des travaux précédents, objectifs du stage	6
3	État de l'Art	8
3.1	La maximisation de la vraisemblance via l'approche Bayésienne	8
3.2	L'approche Machine Learning	9
3.3	Le fonctionnement de Stan	10
4	Méthodologie	14
5	Contributions	15
5.1	Elaboration de modèles	15
5.2	Amélioration de la précision de la simulations	17
5.3	Evaluation des modèles	19
5.4	Résultats des expériences	21
6	Conclusion	23
6.1	Sur Stan	23
6.2	Bilan personnel	23
7	Annexes	25
7.1	Bibliographie	25

Table des figures

3.1	Exemple de descente de gradient	9
3.2	trace du paramètre coefficient	11
3.3	L'interface de shinystan, avec l'affichage du log postérieur et des trajectoires divergentes en rouge	12
3.4	Exemple de l'algorithme de sampling Gibbs	13
5.1	Modèle linéaire, bruit polynomial.	16
5.2	modèle linéaire avec paramétrisation non centrée.	18
5.3	décomposition QR.	19
5.4	Ggpairs avec modèle linéaire	20
5.5	répartition du paramètre alpha selon theta	21
5.6	Génération de nouvelles données, modèle polynomial	22

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 taille, générant beaucoup de données. Le LIG Grenoble est présent au sein du bâtiment IMAG de Saint-Martin d'Hères, avec 4 autres laboratoires ou unités :

- 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)
- le laboratoire Jean Kuntzmann (LJK)
- le laboratoire 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 "intelligent" 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 fenêtres, 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'institut 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 à haute performance

De nos jours, le calcul à haute performance (High Performance Computing) est devenu indispensable 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, 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 cœurs par processeur. Ainsi, les superordinateurs contiennent des centaines de noeuds, chacun contenant plusieurs processeurs multi-cœurs ; avec certains de ces processeurs possédant plusieurs niveau de cache, ou mettant en œuvre du multi-threading. Ces architectures permettent une grande performance de calcul, mais induisent une certaine variabilité au niveau des durées de calculs et une incertitude sur la prédiction et de l'évaluation de performance. Il devient difficile d'estimer les performances "normales", et d'en tirer des conclusions.

En outre, cette complexité d'architecture impose aux applications parallèles d'utiliser des technologies Message Passing Interface (MPI) pour communiquer entre les différents noeuds d'un superordinateur, point à point ou de manière collective, afin que chaque processus puisse effectuer une partie d'un grand calcul. MPI a donc aidé au développement de superordinateurs de plus en plus puissants, ce qui a conduit à l'établissement du TOP500 des machines les plus performantes.

L'objectif de mon stage a été de contribuer aux techniques de modélisation et d'évaluation de performances d'applications parallèles sur des systèmes de haute performance, notamment à l'aide de simulations 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

abstract du papier 2.1 2.2 difficile prévoir perf optimiser perf avec techniques donc les gens exécutent du code pour savoir si ça tourne bien d'où simulation smpi pour exécuter l'appli sur ton ordi mais pour simuler mieux vaut remplacer les grosses fonctions par des modèles dans le papier : utiliser smpi pour perf hpl noyau calcul dgemv avec tel prototype, complexité $m \times n \times k$ / flop rate machine m-0

2.3 Type de mesures et de modèles

Les mesures récupérées pour modéliser ces systèmes sont issues d’expériences spécifiques et contrôlées mais peuvent néanmoins être biaisées en fonction de la température interne des machines, de divers effets de cache, de la rapidité d’un cœur 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 mes encadrants à définir plusieurs types de modèles, selon le degré de compatibilité du système observé. On définit M-\$ N-\$ comme un modèle de complexité \$, avec un bruit de complexité \$. 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 (le modèle de bruit le plus simple qui soit).
- 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 par morceaux, 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 selon les ressources (communications MPI, noyaux de calcul, ...). Il a été établi 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-0 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 cette approche et aux travaux présentés dans Cluster2019 : la prise en compte des variabilités spatiales et temporelles, ainsi que la spécificité du système, ont forcé mes encadrants 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 nœuds, 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 dû écrire du code spécifique, notamment la génération de nombres aléatoires pour rendre compte de la variabilité temporelle. Cette solution fonctionne, mais n’est pas générique et ne permet pas une vision à long terme et une réutilisation 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, permettant d’exprimer 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 été envisagées par mes encadrants; 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 La maximisation de la vraisemblance via 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, où A correspond à notre hypothèse et B à nos observations :

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

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, où la probabilité de A sachant B est le postérieur, la probabilité de B sachant A est le modèle et la probabilité de A est le prior :

$$p(A|B) \propto p(B|A) * p(A) \quad (3.2)$$

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 hypothèses 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.

— 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 θ correspondant à un échantillon de variables aléatoires discrètes. Soit : $L(\theta) = p(x_1 \dots x_N | \theta)$ représentant la probabilité d'avoir les observations $x_1 \dots x_N$ étant donné les paramètres θ . Les estimateurs $\hat{\theta}$ du maximum de vraisemblance des paramètres θ 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 θ permettant d'atteindre le maximum de la vraisemblance L .

L'approche d'inférence Bayésienne par échantillonnage 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.

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 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.

L'algorithme principal est la descente de gradient afin de calculer un estimateur du maximum de vraisemblance. Son 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 η ; et ceci jusqu'à converger en un minimum.

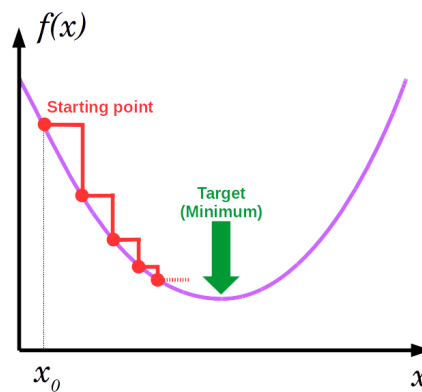


FIGURE 3.1 – Exemple de descente de gradient¹

Attention à taille de η : plus il est grand plus on avance loin à chaque pas donc plus on réduit théoriquement les itérations, mais si η 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 η est petit plus on avance lentement, mais avec plus de chances de converger au final. Il existe

1. source de l'image : <https://www.neural-networks.io/fr/single-layer/gradient-descent.php>

deux limites à cet algorithme : les minimums locaux et le "vanishing gradient". 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 "vanishing gradient" fait référence à un tracé de fonction avec des valeurs "plateau" où l'algorithme se bloquerait, l'empêchant de trouver le minimum.

L'utilisation de la descente de gradient requiert donc des conditions initiales, et plusieurs exécutions pour s'assurer que l'algorithme aura bien convergé sur le minimum global.

Enfin l'algorithme de clustering est un cas particulier utilisé dans un espace choisi permettant d'identifier et de former des petits groupes séparés de données partageant des caractéristiques communes. 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.

Ces algorithmes sont très riches et relativement rapide d'exécution mais ne donnent qu'une seule valeur comme résultat, et n'offrent aucune mesure d'incertitude.

3.3 Le fonctionnement de Stan

Stan utilise l'algorithme MCMC présenté précédemment, ce qui permet à la simulation de parcourir assez rapidement un espace de valeurs possibles, en suivant une certaine distribution. Le procédé a également une période de "warm up", 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 tirer dans cette zone 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, cela permet de s'affranchir des conditions initiales et d'augmenter la confiance en notre résultat si on s'aperçoit qu'elles convergent dans la même zone (pour les itérations d'échantillonnage, puisque les résultats des itérations de "warm up" ne donnent pas des résultats significatifs).

La figure 3.3 illustre la convergence de 8 chaînes indépendantes 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 "data" 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 "transformed data" permet de créer de nouvelles données, souvent à partir des données initiales. Le bloc "parameters" 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 "transformed parameters" permet de déclarer et assigner des valeurs à d'autres paramètres. Enfin le bloc "model" permet d'indiquer les priors et le modèle, et le bloc "generated quantities" permet de créer de nouvelles données, de faire des prédictions sur les nouvelles données, etc. Cette syntaxe générique permet d'écrire des modèles précis et aussi 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 $\text{normal}(0,10)$; 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

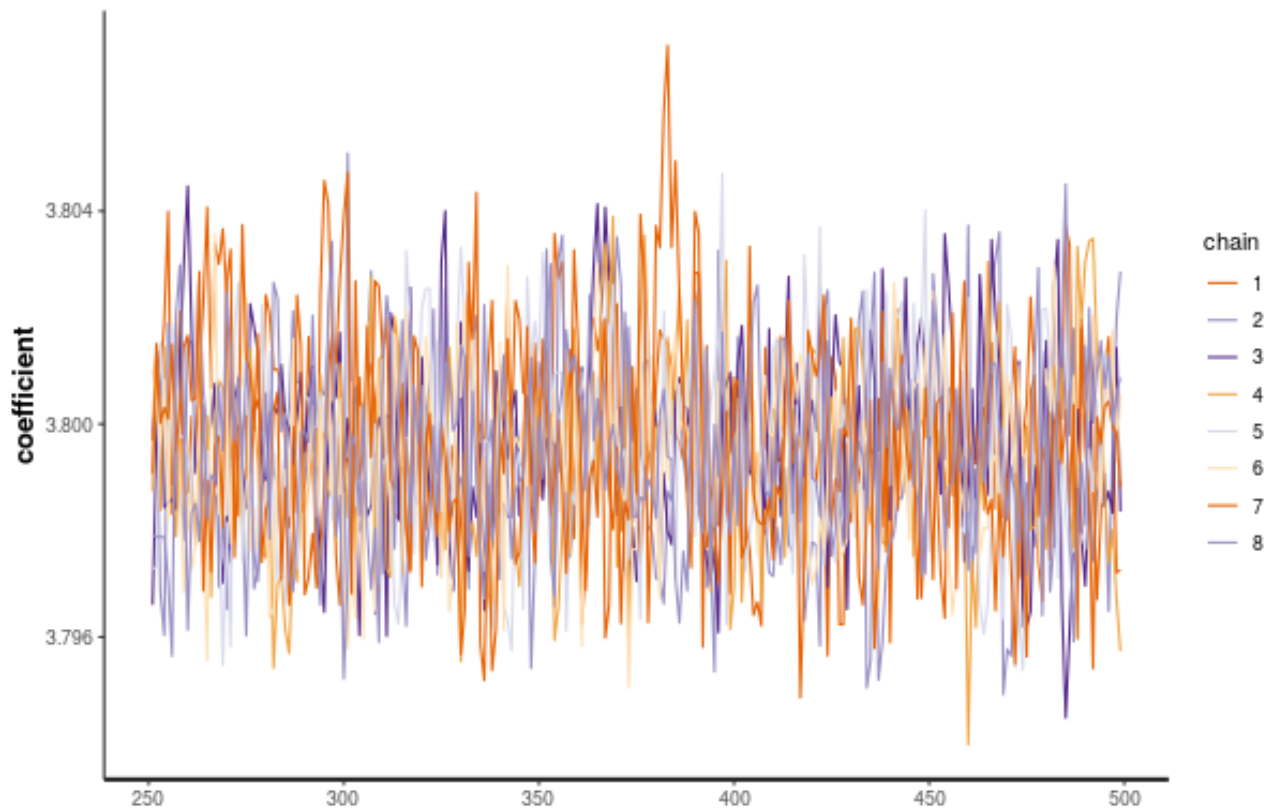


FIGURE 3.2 – trace du paramètre coefficient

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 3.3.

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.

- 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

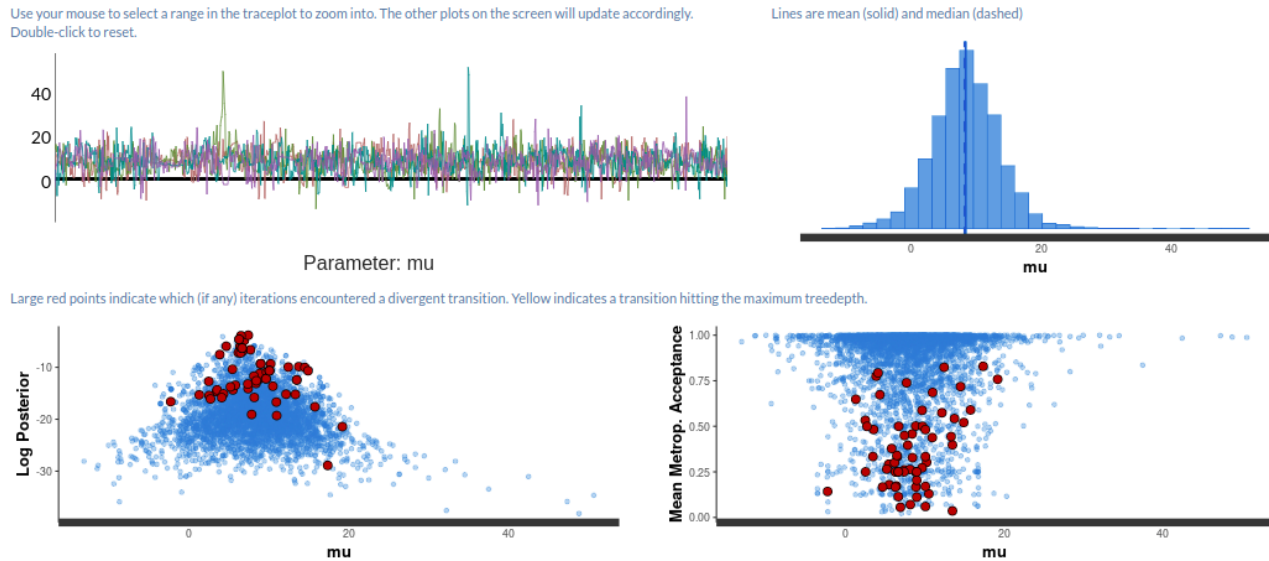


FIGURE 3.3 – L’interface de shinyStan, avec l’affichage du log postérieur et des trajectoires divergentes en rouge

la simulation de nouvelles prédictions pour le modèle. Pour cela, l’algorithme de sampling parcourt 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.

- 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

On peut reconnaître cet algorithme dans l’image du milieu de la figure 3.3, 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.

2. source de l’image : <https://jessicstringham.net/2018/05/09/gibbs-sampling/>

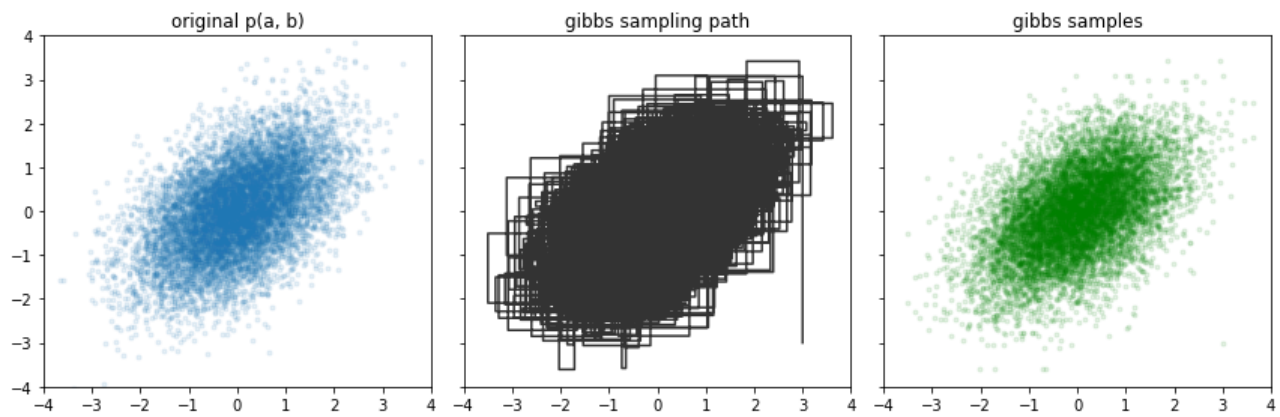


FIGURE 3.4 – Exemple de l'algorithme de sampling Gibbs²

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, Christophe Pouzat et Konrad Hinsén¹.

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, etc².

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 dépliantes 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 directement dans ce cahier, à l’exception de celles réalisées sur Grid5000. En effet, Org-Mode inclut 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 cœurs et 800 nœuds 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 nœud, 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.

1. <https://www.fun-mooc.fr/courses/course-v1:inria+41016+session02/info>

2. <https://github.com/hoellejal/automating-calculation-kernels-modelling/blob/master/journal.org>

Chapitre 5

Contributions

5.1 Elaboration de modèles

Comme le but du stage était de comparer l'échantillonnage Bayésien à l'approche machine learning, j'ai commencé par faire un modèle linéaire simple des données, suivant la forme $y = a * x + b$, avec une régression linéaire. Je me suis rapidement aperçu que la régression linéaire avait deux inconvénients dans notre cas : comme elle ne permet pas de modéliser le bruit, il était difficile de lui indiquer une distribution du bruit qui ne serait pas linéaire. De plus, le paramètre du modèle qui est indépendant de x , dans ce cas b , 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 trouvés. 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 données synthétiques, 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 simples, par exemple des modèles linéaires sans bruit, qui s'exécuteront rapidement.

Ensuite j'ai travaillé sur les données de la fonction `dgemv` de OpenBlas fournies par mes encadrants : plus précisément sur la durée d'exécution de cette fonction en fonction de la taille de la matrice (déterminée par les paramètres M, N et 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 γ et deux paramètres dépendant de MNK : α et θ .

$$duration \sim \mathcal{N}(\alpha \cdot mnk + \beta, \theta \cdot mnk + \gamma) \quad (5.1)$$

La figure 5.1 illustre ce modèle.

J'ai ensuite écrit un **modèle polynomial avec du bruit polynomial** (M-2 N-2), puis j'ai ajouté de la complexité à ces modèles par couche. Le modèle polynomial est très similaire au modèle linéaire, 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 $N*K$. Le modèle est donc légèrement modifié :

$$duration \sim \mathcal{N}(\alpha_1 \cdot mnk + \alpha_2 \cdot mn + \alpha_3 \cdot mk + \alpha_4 \cdot nk + \beta, \theta_1 \cdot mnk + \theta_2 \cdot mn + \theta_3 \cdot mk + \theta_4 \cdot nk + \gamma) \quad (5.2)$$

Par la suite, j'ai réécrit ces deux modèles en ajoutant une autre variable déterminante sur laquelle les estimations des paramètres devaient s'effectuer : le CPU utilisé. Dans les données fournies, `dgemv` avait été lancée sur 64 CPU différents. Les deux modèles précédents ont donc été adaptés pour estimer les paramètres pour les 64 hôtes différents. La principale différence de ces modèles était


```

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> theta; //contrainte positive sur le bruit
  real<lower=0> gamma;
}

model {
  //priors sur les paramètres
  alpha ~ normal(6e-11, 6e-12);
  beta ~ normal(7e-07, 7e-08);
  theta ~ normal(1e-12, 1e-13);
  gamma ~ normal(7e-07, 7e-08);

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

```

FIGURE 5.1 – Modèle linéaire, bruit polynomial.

que la likelihood devait donc être définie selon les hôtes. On avait donc la formule suivante pour le **modèle linéaire avec estimations par hôte** :

$$duration_i \sim \mathcal{N}(\alpha_i * mnk + \beta_i, \theta_i * mnk + \gamma_i) \quad (5.3)$$

Et de même pour le **modèle polynomial avec estimations par hôte**.

$$duration_i \sim \mathcal{N}(\alpha_{i_1} \cdot mnk + \alpha_{i_2} \cdot mn + \alpha_{i_3} \cdot mk + \alpha_{i_4} \cdot nk + \beta_i, \theta_{i_1} \cdot mnk + \theta_{i_2} \cdot mn + \theta_{i_3} \cdot mk + \theta_{i_4} \cdot nk + \gamma_i) \quad (5.4)$$

Ces deux derniers modèles permettent de simuler à peu près la performance des noyaux de calculs utilisés dans HPL.

Cependant, on pourrait estimer qu'il est possible d'avoir une distribution de probabilité de valeurs moyennes des paramètres, rendant compte de la variabilité spatiale et à partir de laquelle on tirerait des valeurs pour les paramètres à chaque fois. On estimerait alors les formules (5.5) pour utiliser un **modèle hiérarchique linéaire** :

$$\begin{cases} \alpha_i \sim \mathcal{N}(\mu_\alpha, \sigma_\alpha) \\ \beta_i \sim \mathcal{N}(\mu_\beta, \sigma_\beta) \\ \theta_i \sim \mathcal{N}(\mu_\theta, \sigma_\theta) \\ \gamma_i \sim \mathcal{N}(\mu_\gamma, \sigma_\gamma) \end{cases} \quad (5.5)$$

Et les formules suivantes 5.6 pour un **modèle hiérarchique polynomial**, où n serait compris entre 1 et 4 et α^n correspondrait alors aux coefficients de mnk, mn, mk et nk.

$$\begin{cases} \alpha_{i_n} \sim \mathcal{N}(\mu_{\alpha_n}, \sigma_{\alpha_n}) \\ \beta_i \sim \mathcal{N}(\mu_{\beta}, \sigma_{\beta}) \\ \theta_{i_n} \sim \mathcal{N}(\mu_{\theta_n}, \sigma_{\theta_n}) \\ \gamma_i \sim \mathcal{N}(\mu_{\gamma}, \sigma_{\gamma}) \end{cases} \quad (5.6)$$

Sachant que le cluster sur lequel on fait nos estimations, Dahu, n'est pas vraiment homogène donc cette hypothèse n'est pas très raisonnable pour un modèle voulant refléter la réalité.

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 , θ et γ pour un nouveau CPU.

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.

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.

De plus, dès que l'on travaille sur des valeurs 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. On donne ensuite les priors de nos paramètres dans le bloc "transformed parameters" en additionnant la moyenne avec le produit d'un des paramètres supplémentaire et de l'écart type (au lieu de l'écrire sous la forme d'une distribution normale). Ce type d'écriture permet au sampler de pouvoir trouver plus facilement les paramètres recherchés. Un exemple de modèle est donné dans la figure 5.2.

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 et les données précises et petites, plus il est préférable de donner des priors informatifs, soit assez proche des valeurs des paramètres, car sans cela le sampler arrivera à converger mais aura des résultats erronés autour de l'entier le plus proche. 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

```

data {
  int<lower = 0> N; //nombre de données
  vector[N] mnk; //taille de la matrice
  vector[N] duration; //durée d'exécution
}

parameters {
  //paramètres supplémentaires
  real alpha_raw;
  real beta_raw;
  real<lower=0> theta_raw; //contrainte positive sur le bruit
  real<lower=0> gamma_raw;
}

transformed parameters {
  //priors sur les paramètres
  real alpha = 6e-11 + alpha_raw * 6e-12;
  real beta = 7e-07 + beta_raw * 7e-08;
  real theta = 1e-12 + theta_raw * 1e-13;
  real gamma_raw = 7e-07 + gamma_raw * 7e-08;
}

model {
  //priors sur les paramètres supplémentaires
  alpha_raw ~ normal(0,1);
  beta_raw ~ normal(0,1);
  theta_raw ~ normal(0,1);
  gamma_raw ~ normal(0,1);

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

```

FIGURE 5.2 – modèle linéaire avec paramétrisation non centrée.

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.

Il est également possible de donner des 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 apportée par les 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.

Enfin, il existe des techniques d'écriture, comme la décomposition QR de matrices (voir figure 5.3), qui permet de réduire la corrélation entre les paramètres utilisés pour calculer le postérieur et réduit le temps de simulation sans impacter négativement les résultats.

```

data {
  int<lower=0> N;
  int<lower=1> M;
  matrix[M, N] mnk;
  vector[N] duration;
}

transformed data {
  matrix[N, M] Q = qr_Q(mnk')[, 1:M] * N;
  matrix[M, M] R = qr_R(mnk')[1:M, ] / N;
  matrix[M, M] R_inv = inverse(R);
}

parameters {
  vector[M] alpha_tilde;
  real beta;
  vector<lower=0>[M] theta_tilde;
  real<lower=0> gamma;
}

transformed parameters {
  vector[M] alpha = R_inv * alpha_tilde;
  vector[M] theta = R_inv * theta_tilde;
}

model {
  alpha ~ normal(6e-11, 6e-12);
  beta ~ normal(7e-07, 7e-08);
  theta ~ normal(1e-12, 1e-13);
  gamma ~ normal(7e-07, 7e-08);

  for(i in 1:N){
    duration[i] ~ normal(Q[i] * alpha_tilde[i] + beta,
      Q[i] * theta_tilde[i] + gamma);
  }
}

```

FIGURE 5.3 – décomposition QR.

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` (exemple dans la figure 5.4), 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.

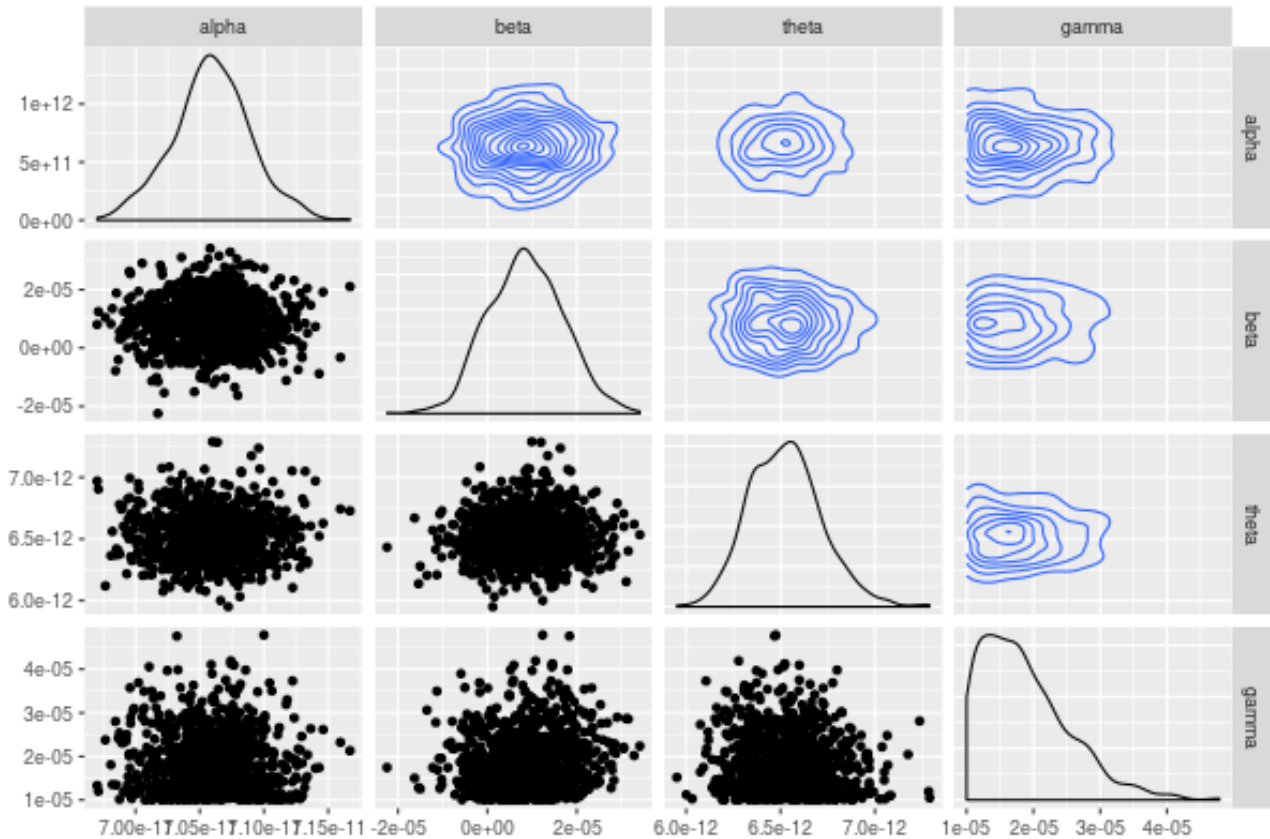


FIGURE 5.4 – Ggpairs avec modèle linéaire

On peut également dessiner les graphiques nous mêmes, à partir des distributions des paramètres $\bar{\alpha}_i$ et $\bar{\theta}_i$ trouvées par stan. Par exemple dans l’image 5.5, nous avons dessiné la répartition de α selon θ , et ce pour chacun de nos 64 hôtes, avec une grande ellipse contenant 95% des points.

Cette image nous fait remarquer que les paramètres α et θ sont relativement indépendants, et qu’on peut avoir une distribution différente pour chacun sans induire notre modèle en erreur. Par exemple nous avons observé dans les histogrammes des paramètres que modéliser θ_i par une loi normale semblait raisonnable, mais que α_i n’avait pas une forme typique de gaussienne, et qu’il serait probablement plus judicieux de le modéliser par une mixture de loi normales. Il serait donc possible d’avoir un modèle avec des priors différents sur les deux paramètres dépendant de m_{nk} .

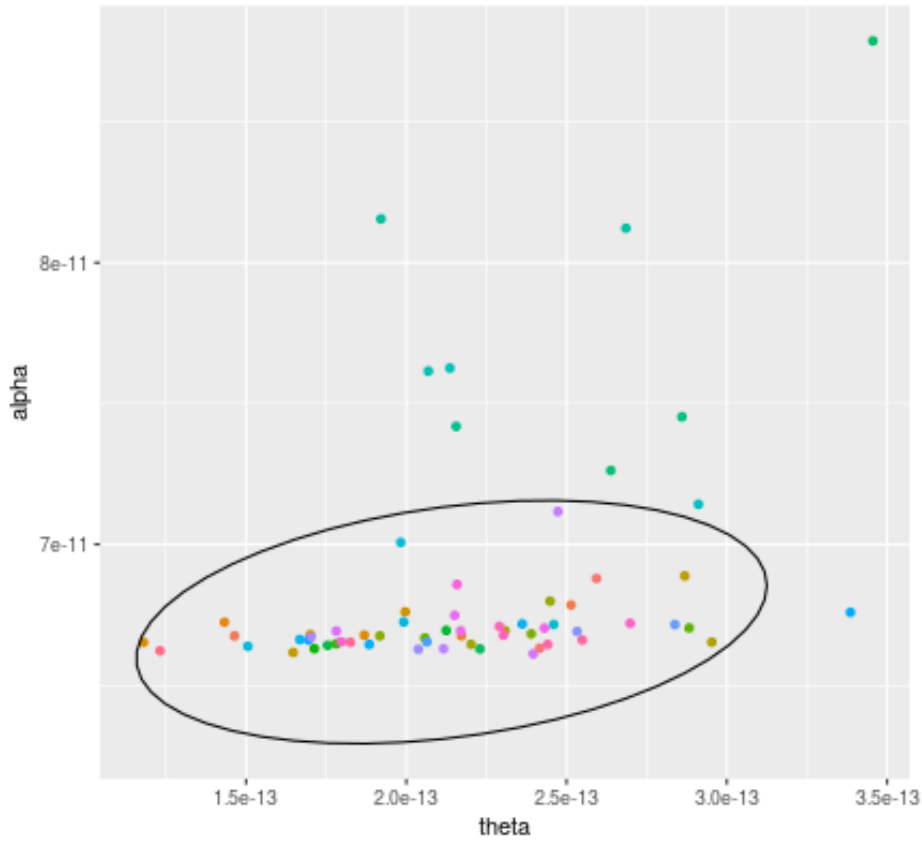


FIGURE 5.5 – répartition du paramètre alpha selon theta

J’ai donc écrit un {modèle linéaire hiérarchique avec α modélisé par une mixture de lois normales}. 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 :

$$target = target + \log_{mix}(\delta, normal_{pdf}(y_n|\mu_1, \sigma_1), normal_{pdf}(y_n|\mu_2, \sigma_2)) \quad (5.7)$$

Ici δ correspond à la proportion de données dans chacune des gaussiennes, et on exprime ensuite la présence de deux distributions normales, avec μ_1 et σ_1 puis μ_2 et σ_2 .

Enfin le meilleur moyen de vérifier la précision postérieure 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 5.6 montre une génération de données confirmant la précision du modèle.

5.4 Résultats des expériences

Mes expériences avec les différents modèles ont montré que les modèles avec estimations par hôte fonctionnaient bien avec nos données, mais que les modèles hiérarchiques commençaient à être

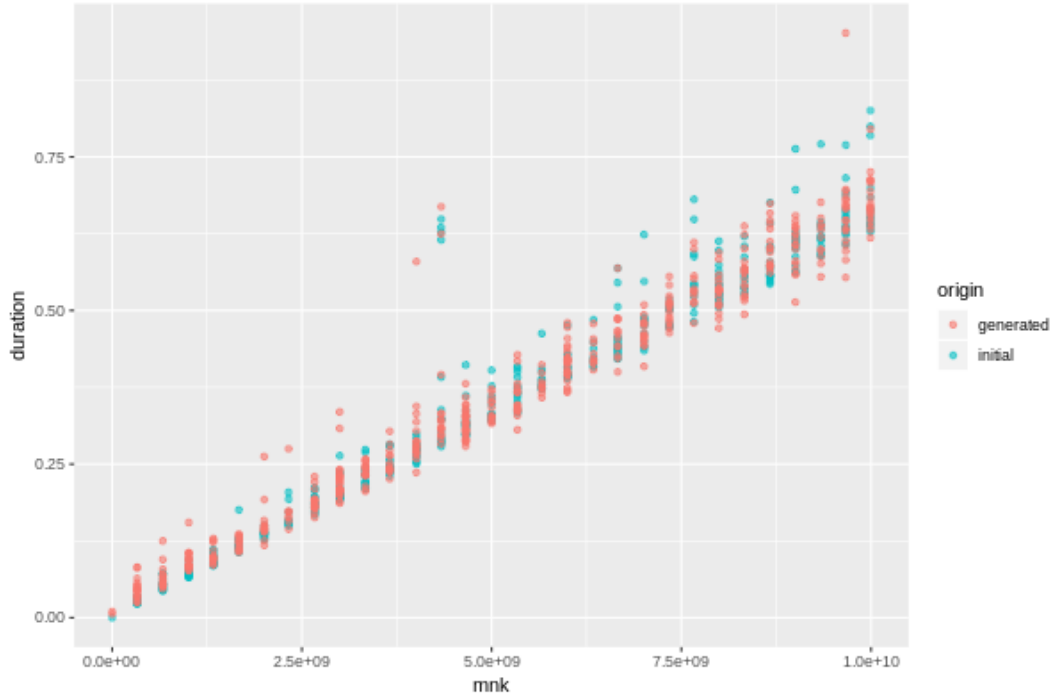


FIGURE 5.6 – Génération de nouvelles données, modèle polynomial

trop complexes pour la quantité d’informations dont on disposaient : il aurait fallu pouvoir donner plus de précision.

De plus, le dernier modèle où α était modélisé par une mixture de lois normales n’est pas conclusif : l’expression d’une mixture de gaussiennes fonctionne relativement bien sur des données générées, lorsqu’elle concerne la distribution du 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.

Chapitre 6

Conclusion

6.1 Sur Stan

Lors de ce stage, j'ai donc évalué 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.

Chapitre 7

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>

7.1 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://www.statmethods.net/advstats/cluster.html>



Etudiant : **Hoël JALMIN**

Année d'étude dans la spécialité : 4

Entreprise : **Laboratoire Informatique de Grenoble**

Adresse complète : **700 avenue Centrale, 38400 Saint-Martin-d'Hères**

Responsable administratif : **Annie SIMON**
Courriel : **annie.simon@inria.fr**

Tuteur de stage : **Arnaud LEGRAND**
Courriel : **arnaud.legrand@imag.fr**

Enseignant-référent : **Olivier RICHARD**
Courriel : **olivier.richard@inria.fr**

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

Résumé :