

1 Présentation du projet

L'objectif de ce projet consistait en l'utilisation de la programmation orientée objet afin de résoudre plusieurs problèmes via l'algorithme de Metropolis-Hastings. Afin de mener cette tâche à bien, j'ai créé les classes data, recuit, et minimisation, et plusieurs sous-classes pour chacune d'entre elles. Leur utilité sera discutée ci-dessous. Grâce à ces classes et sous-classes, l'algorithme de Metropolis-Hastings permet de trouver le minimum global d'une fonction quelconque à une dimension, puis de trouver les coefficients d'une fonction linéaire à partir d'un nuage de points (avec ajout d'un bruit gaussien), et enfin de résoudre le problème du voyageur de commerce.

2 Les classes et sous-classes

2.1 La classe data.h

La classe data.h contient 3 sous-classes qui créent des données utilisables par les autres classes. La sous-classe lin trace un nuage de n points suivant une fonction linéaire, la sous-classe ling fait de même en ajoutant un bruit gaussien sur chaque point, et la classe tab crée un tableau triangulaire rempli de valeurs aléatoires suivant une loi uniforme entre deux bornes, représentant les distances entre villes pour le problème du voyageur de commerce.

2.2 La classe minimisation.h

La classe minimisation contient 3 sous-classes contenant chacune une fonction d'optimisation. La sous-classe mcarres permet de calculer le khi2 d'une fonction linéaire par rapport à un nuage de points fourni dans la classe minimisation, la sous-classe fonction est simplement une classe modifiable à souhait afin d'avoir une fonction à minimiser de dimension 1, et la sous-classe dist permet, à partir d'un tableau triangulaire contenant les distances entre villes fourni dans minimisation, et d'un vecteur contenant l'ordre de parcours des villes de calculer la distance totale parcourue par le voyageur de commerce.

2.3 La classe recuit.h

La classe recuit contient 3 sous-classes contenant chacune un algorithme de Metropolis-Hastings. La sous-classe dm1 applique l'algorithme à une fonction d'optimisation de dimension 1, la sous-classe dm2 l'applique à un nuage de point pour retrouver une loi linéaire et la sous-classe voy l'applique à un tableau triangulaire afin de trouver la distance minimale. Chaque sous-classe contient également une fonction d'affichage pour visualiser l'évolution de l'algorithme.

3 Les différents problèmes abordés

3.1 Fonction à une dimension

La résolution de ce problème a peu d'intérêt pratique, mais elle est une bonne introduction à l'algorithme de Metropolis-Hastings. Le principe élémentaire est l'utilisation d'une boucle while dont le paramètre est une "température" abstraite d'évolution décroissante. A chaque étape, on modifie l'abscisse de la fonction d'un certain pas, et on observe les résultats sur la fonction. Si la fonction avec cette variable modifiée renvoie un résultat inférieur à celle avec la variable originale, on l'accepte et on continue les boucles avec elle. Si le résultat est supérieur, on a une certaine chance de l'accepter, cette chance devenant de plus en plus mince lorsque la température diminue. Ainsi, aux hautes températures (début de l'algorithme), on parcourt un très grand nombre de points, même éloignés du minimum de la fonction. Puis, on tend finalement vers ce minimum lorsque la température diminue. Ce mode de fonctionnement permet d'éviter de tomber dans un minimum local. Le principe sera foncièrement le même pour les autres problèmes en ajoutant d'autres paramètres.

3.2 Fonction linéaire avec et sans bruit gaussien

Contrairement au problème précédent, la résolution de ce problème possède un réel intérêt. Elle permet de retrouver les coefficients d'une fonction linéaire à partir d'un nuage de points, qui pourraient être par exemple les points obtenus lors d'une mesure dans une expérience de physique. Une fonction linéaire $ax + b$ dépendant de deux paramètres a et b , l'algorithme de Metropolis-Hastings doit être modifié pour fonctionner à deux dimensions. Ainsi, le pas élémentaire est effectué dans deux dimensions différentes, et la succession des pas en cas d'échec dans un tour de boucle est effectuée au hasard, afin de parcourir le plan dans toutes les directions. L'ajout d'un bruit gaussien, représentant les incertitudes de mesure dans une expérience, ne dérange pas le calcul des coefficients si l'on a assez de points par rapport à l'amplitude du bruit gaussien.

3.3 Problème du voyageur de commerce

Très succinctement, le problème du voyageur de commerce consiste à trouver le chemin le plus court pour traverser N villes, toutes séparées les unes des autres par une certaine distance. Ici, le nombre de dimensions du problème explose, et on s'y retrouve en prenant comme paramètre un vecteur contenant l'ordre de parcours des villes. La perturbation élémentaire que subit ce vecteur est l'échange de position dans l'ordre de parcours pour deux villes tirées au hasard, et on cherche à minimiser la distance du parcours total. L'intérêt ici étant que le nombre de données pour le problème du voyageur de commerce augmente de façon exponentielle avec le nombre de villes, il est ainsi impossible de trouver une réponse à ce problème dès que N devient grand. La complexité de l'algorithme de Metropolis-Hastings dépend de notre température, et la précision de la réponse également.

Ainsi, il est difficile d'avoir la certitude, pour un très grand nombre de villes, que l'algorithme trouve effectivement le chemin le plus court. On peut cependant être sûr que plus le nombre de passages dans cette boucle est grand, plus on a de chances de tomber dans ce minimum. L'arbitrage se situe donc entre le temps de calcul et les erreurs que l'on estime acceptables. Par exemple, en faisant tourner l'algorithme pour 40 villes avec $Tini=1$, $Tfinal=0.0001$ et $Lambda=0.99/0.999/0.9999$, on trouve des distances minimales d'environ 45/43,5/42,5. Si on fait maintenant tourner le programme pour $Tfinal=0.000001$ et $lambda=0.99999999$, le temps de calcul est d'environ 20 minutes et la distance minimale trouvée est 41.4, donc il y a encore une amélioration du résultat qui se paye par une importante augmentation du temps de calcul.

4 Conclusion

La réalisation de ce projet, au-delà de l'intérêt de l'algorithme du recuit simulé, m'a permis de me familiariser avec la syntaxe de la programmation orientée objet qui est, même si parfois fastidieuse, d'une certaine élégance. Merci aux enseignants pour leur investissement.