Optimisation TD 4
La recuit simulé

Carole Frindel

30 Novembre 2016

Le recuit simulé est une méthode d'optimisation empirique. Comme son nom l'indique, elle fait l'analogie entre la manière dont on manipule les métaux par différentes cuissons pour obtenir une structure cristalline d'énergie minimum et la recherche d'un minimum d'une fonction de coût.
Dans une version simple, on peut considérer l'algorithme de la manière suivante lorsqu'on cherche le minimum d'une fonction f :

1. **Initialisation** On part d'un point x_0 et d'une température T_0.
2. **Déplacement** On effectue un déplacement élémentaire aléatoire de la solution courante $x_{t+1} = x_t + D$. D est une variable aléatoire (de loi normale) de moyenne nulle et de variance $\kappa e^{-1/(1000*T)}$ où κ est un paramètre.
3. **Évaluation** On évalue x_{t+1}. Si la solution est meilleure on la garde SINON on la garde avec une probabilité $\kappa' e^{-1/(1000*T)}$
4. **Diminution de la température** On diminue T
5. **Critère d'arrêt** La solution est-elle acceptable ? Le nombre d'itérations dépassé ?

1 Fonction de $\mathbb{R}$ dans $\mathbb{R}$

On se propose d'appliquer les principes précédents pour la recherche d'un minimum de la fonction $f(x) = x^4 - x^3 - 20x^2 + x + 1$.

1. Tracer avec `matplotlib` la fonction et repérer le minimum global et (éventuellement) les minima locaux.
2. Implémenter le recuit simulé pour cette fonction, avec $x(0) = -1$. Faire varier les paramètres pour obtenir des résultats robustes. Indication : commencer par prendre $\kappa = 10$, $\kappa' = 0.5$ et $T = 1/t$ pour t variant de $t_0 = 1$ à $t_{max} = 10000$ par pas de 1. Jouer avec les paramètres $t_{\max}$, κ et κ' pour bien comprendre l'incidence de ces paramètres.
3. Tracer avec `matplotlib` la trajectoire x_t sur la surface de la fonction de coût. Que constatez-vous ?

2 Fonction de $\mathbb{R}^2$ dans $\mathbb{R}$

Refaire le même exercice avec la fonction $g(x, y) = f(x) + f(y)$. Note: commencer avec les mêmes paramètres qu'à l'exercice précédent.

3 Compréhension de l'algorithme

1. Contrairement aux méthodes de descente de gradient (ordres 1 et 2), le recuit simulé inclut une part d'aléatoire. Quelle est d'après vous son utilité (notamment en comparaison avec les méthodes préalablement testées) ?

2. Tout comme en métallurgie, la température évolue au cours de l'algorithme. Dans quels termes celle-ci intervient-elle ? Sauriez-vous expliquer son utilité ? Que se passerait-il si la température chutait trop vite ?

3. Donner maintenant votre interprétation des paramètres κ et κ'.

4 Modification de l'algorithme

- Maintenant que la température a été étudiée et que nous comprenons son influence dans la démarche exploratoire, on aimerait pouvoir guider d'avantage l'algorithme. On aimerait notamment lui indiquer s'il existe des régions énergétiques plus faible que celle où il se trouve.

- Pour répondre à cette question, on introduit un nouvel élément, ΔE, dans le calcul de la probabilité de choisir un pas qui augmente la fonction de coût tel que : $P = \kappa' e^{-\Delta \mathrm{E}/(1000*T)}$. ΔE va en quelque sorte analyser l'environnement énergétique de l'itération courante x_t.

- Pour ce faire, nous allons faire un palier pour chaque pas de température T (i.e. à chaque itération). Ce palier consiste à tester m déplacements aléatoires à partir de l'itération courante : $x_i = x_t + D_i$, afin d'évaluer $\Delta \mathrm{E} = \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - f(x_t))$ et de calculer la probabilité correspondante P.
  Attention durant le palier aucun changement n'est appliqué. C'est seulement à la fin du palier (une fois ΔE estimé) que l'on procède à la "vraie" itération en utilisant la probabilité P calculée.

Faites ces changements. Qu'observez-vous ? Quelle est l'influence de ΔE ? On prendra $m = 5$.

5 Le voyageur de commerce : cas de $\mathbb{R}^N$ dans $\mathbb{R}$

On se propose d'utiliser le recuit simulé (sans la modification de la section 4) pour résoudre le problème NP complet du voyageur de commerce introduit au début du cours. On se donne N villes dans un pays et on cherche comment parcourir toutes ces villes en minimisant la distance parcourue.

1. Faire un générateur aléatoire de villes (on peut écrire une fonction qui choisit aléatoirement des points dans un carré $[0, L]^2$ et qui prend en paramètre le nombre de villes à créer). Indication : utiliser le générateur aléatoire `rand` du package `numpy.random`.
   Points bonus[1] si vous générez une carte de lieux connus : villes de France, sites touristiques de Lyon, plan du campus...
   Points bonus[2] si vous faites une interface qui permet d'entrer le nom d'un lieu et qui la place au bon endroit sur la carte en faisant une requête pour les coordonnées GPS du lieu à Google Maps.
   Points bonus[3] si votre voyageur du commerce fait mieux que l'option "itinéraire" de Google Maps.

2. Faire une fonction qui calcule la distance euclidienne parcourue par le voyageur. La donnée d'entrée sera donc le trajet d'un voyageur sous la forme d'un vecteur de taille N. Celui-ci contiendra les entiers de 0 à $N - 1$ qui seront les indices des villes parcourues.

3. Le "déplacement aléatoire de la solution courante" sera ici la permutation de deux villes. Indication : utiliser le générateur aléatoire `randint` du package `numpy.random`.

4. Implémenter l'algorithme.

5. Visualiser avec `matplotlib` l'évolution de la solution. Faire des captures d'écran à toutes les k itérations en donnant la distance parcourue. Commenter. Note : Commencer par des problèmes simples.

[1] en fait, probablement pas, mais c'est sympa
[2] selon la qualité de l'application
[3] là, oui

6. Tester différents schémas de décroissance de la température (décroissance en $1/t$, $1/t^3$ et $1/log(t)$). Le choix du schéma de décroisance est crucial dans cet algorithme. Comment l'observez-vous ? Indication : Pour vous en convaincre, saisir une petite carte d'une dizaine de villes, et observer le résultat à la fin de l'algorithme. Répéter l'opération plusieurs fois avec chaque schéma de décroissance. Qui donne les meilleurs résultats ? En combien de temps ?

7. Enfin, plus subtil, comment appliquer la méthode de descente de gradient à ce problème ?