Annexe 3 : Minimisation/Recuit Simulé

Nous proposons une approche différente afin de trouver le minimum global d'une fonction. Cette approche est inspirée du recuit physique en métallurgie permettant aux atomes de se placer dans la configuration la plus solide. L'idée consiste à maîtriser le refroidissement du métal, et à éventuellement ralentir ce dernier en réchauffant le métal pour atteindre un état plus stable énergétiquement (les atomes étant alors plus libre de se déplacer). L'objectif étant d'atteindre un état d'énergie minimal.

L'extension de ce concept pour la minimisation de fonction s'appelle le recuit simulé. Il s'agit d'une méthode très générale, pouvant s'appliquer à une très large variété de problèmes. Le principal avantage de la méthode est sa capacité à ne pas rester bloquer dans un minimum local de la fonction à minimiser. Mais la méthode comporte également des inconvénients, comme le nombre de paramètres à ajuster et le nombre d'itérations souvent élevé pour obtenir une solution satisfaisante. Cela en fait souvent une solution de dernier recours.

L'idée clé de l'algorithme est d'autoriser une augmentation de la fonction à minimiser, suivant une certaine loi de probabilité, ce qui permet de «sortir» d'un éventuel minimum local. On explore ainsi une plus grande partie de l'espace. La présence d'aléatoire dans l'algorithme en fait un algorithme stochastique itératif.

En pratique, les grandes étapes de l'algorithme sont les suivantes :

- on choisit un état initial x_0, ainsi qu'un paramètre R, en général grand au début, mais qui sera amené à diminuer au cours de l'algorithme,
- En génère ensuite aléatoirement un voisin de $x_0 \to x_1$ (étape critique),
- On évalue notre fonction à minimiser :
  - Si $f(x_1) < f(x_0)$, alors on garde le nouvel état (x_0 devient x_1).
  - Sinon, on génère un nombre aléatoire p entre 0 et 1. Si

$$p < e^{-\frac{f(x_1)-f(x_0)}{R}},$$

    alors on accepte quand même l'état x_1 (x_0 devient x_1) et on diminue R (par exemple en prenant $R = 0.8R$).
  - on recommence la deuxième étape de génération d'un nouveau voisin.

Choisir R grand au début permet une plus grande liberté de mouvement, car alors $e^{-\frac{f(x_1)-f(x_0)}{R}} \approx 1$, et la probabilité de se déplacer est donc élevée. Plus on fait d'itérations, plus la probabilité d'accepter un déplacement va globalement diminuer, limitant ainsi les nouveaux états.

Une étape délicate et critique en pratique est la génération d'un nouveau voisin, il existe en effet de nombreuses façons de générer un voisin aléatoirement (en particulier à quel «distance» choisit-on ce voisin de notre état courant ?).

Nous proposons une implémentation du Recuit Simulé dans l'algorithme (1), d'autres façons de faire sont évidemment possibles, notamment pour la génération des voisins (ligne 5). On fera attention, s'agissant d'un algorithme stochastique, il faut s'attendre à ce que chaque exécution donne un résultat différent (parfois complètement faux si le nombre d'itérations n'est pas assez élevé ou les paramètres mal ajustés). On prendra donc garde de lancer plusieurs fois l'algorithme du recuit simulé pour garantir le résultat.

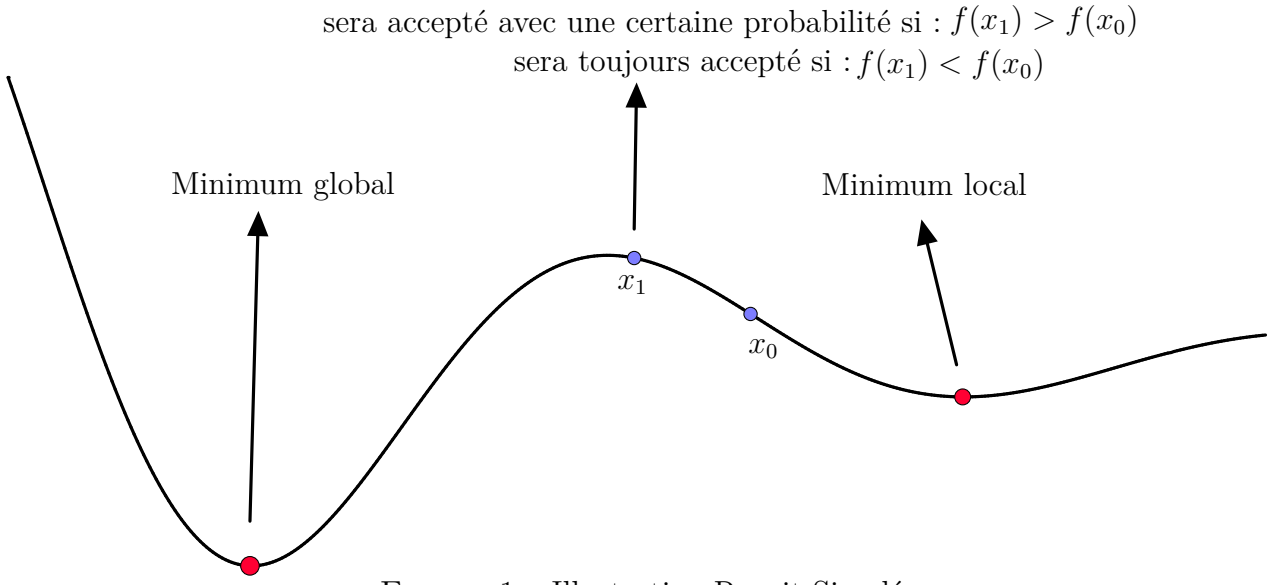

FIGURE 1 – Illustration Recuit Simulé

Algorithme 1 : Algorithme de Recuit-Simulé, pseudo-code

Données :

- Une fonction f, dont on cherche le minimum global, et un point de départ x_0.
- Un paramètre R «grand» (entre 100 et 500), $0 < \alpha < 1$, $v > 0$ (on le prendra entre 5 et 25)
- Un nombre max d'itérations n_{max}

Résultat : Une approximation x_{global} de la solution du problème : $\min_{x\in\mathbb{R}} f(x)$

```
k = 0;
y_0 = f(x_0) ;
x_global = x_0 ; y_global = y_0 ;
tant que R > 0.0001 et k ≤ n_max faire
    x_1=x_0+v(random(0,1)-0.7) ;
    Calcul de y_1 = f(x_1) ;
    si y_1 < y_0 alors
        x_0 = x_1 ; y_0 = y_1 ;
    sinon
        p=random(0,1) ;
        si p < e^{-(y_1 - y_0)/R} alors
            R = αR ;
            x_0 = x_1 ; y_0 = y_1 ;
        fin
    fin
    si y_1 < y_global alors
        y_global = y_1 ; x_global = x_global ;
    fin
    k = k + 1 ;
fin
```