I.A. TD 2 Recherche stochastique : la méthode du « recuit simulé »

14 octobre 2010

Table des matières

1 Le « recuit simulé »

Nous avons montré en cours sur l'exemple du taquin que la méthode du gradient n'était pas satisfaisante puisqu'elle pouvait conduire à un extremum local et qu'elle bloquait dessus. Pour éviter cela plusieurs méthodes ont été proposées, elles s'inspirent généralement de phénomène naturel. Citons, par exemple :
– Les algorithmes génétiques
– Les algorithmes tabous
– Le recuit simulé.
Nous nous intéressons dans ce TD au recuit simulé qui est une technique de recherche dans lequel on introduit de l'aléatoire. Les algorithmes génétiques seront introduit dans le TD suivant.

2 Descente aléatoire pure

Avant de nous attaquer au « recuit simulé » proprement dit, essayons une technique que l'on peut qualifier de « descente aléatoire pure ».

On la nomme descente aléatoire car elle effectue une recherche de la meilleure solution en « sautant » de solution en solution dans l'espace de recherche, mais en acceptant que les solutions qui sont meilleures que les précédentes.

On parle de « descente » car dans l'exemple général, on cherche le minimum d'une fonction. C'est ce que l'on se propose de faire ici pour mettre en œuvre cette méthode.

On cherche le minimum d'une fonction F :

$$F = \{ \begin{array}{l} \mathbb{R}^n \to \mathbb{R} \\ X \mapsto F(x) \end{array}$$

une fonction de n variables. Soit X_0 le vecteur initial et `normale()` une fonction qui détermine X_p en fonction de X selon une loi normale centrée sur X. Cela donne l'algorithme suivant :

```
float Xr = X0;
while( ! condition_de_fin ) {
  float Xp = normale( Xr );
  if( F( Xp ) < F( X ) ) {
    Xr = Xp;
  }
}
```

Le résultat X_r contiendra la plus basse valeur trouvée. L'inconvénient majeur de cette méthode est qu'elle admet que des diminutions de la fonction. Il est donc possible de bloquer sur un minimum local. Pour éviter cela on peut utiliser le recuit simulé.

3 Recuit simulé

Le *recuit* est un processus physique de chauffage. Ainsi, lorsqu'on chauffe un métal solide, il devient liquide à une certaine température dans ce cas les atomes qui le composent ont vu leur degré de liberté augmenter. Inversement lorsque l'on baisse la température le degré de liberté diminue jusqu'à obtenir un solide. Maintenant suivant la façon dont on diminue la température on obtient différent solides :
– Baisse brutale de la température (La trempe), cela produit une structure amorphe, un verre. On a alors un minimum local d'énergie.
– Baisse progressive de la température de façon à atteindre le minimum global d'énergie. On obtient dans ce cas un cristal.
Lorsque la baisse progressive est trop rapide on a alors des défauts au niveau du cristal. Le recuit permet de redonner de la liberté aux atomes pour tenter d'atteindre un nouvel état dynamique.

Le recuit simulé s'inspire donc de la thermodynamique. En effet, les systèmes physiques atteignent rapidement un état d'équilibre (minimum d'énergie) en dépit du nombre immense de configurations possibles. Le recuit simulé permet de sortir d'un minimum local en acceptant avec une certaine probabilité une augmentation de la fonction. Cette méthode a été proposée en 1982 par S. Kirkpatrick et al. à partir de la méthode de Métropolis (1953) qui était utilisée pour modéliser les processus physiques. En effet, il a constaté que la recherche des minima locaux était de même nature dans les deux cas (physique et informatique). Il s'est alors inspiré de la mécanique statistique en utilisant en particulier la distribution de Boltzmann. Ainsi, la probabilité p qu'un système physique passe d'un niveau d'énergie E_1 à un niveau E_2 est donnée par :

$$p = e^{-\frac{E_1 - E_2}{kT}} \tag{1}$$

Avec k la constante de Boltzmann qui vaut $k = 1.380510^{-23} J/K$ et T la température absolue.

Comme le montre l'équation 1 la probabilité d'observer une augmentation de l'énergie est d'autant plus grande que la température est élevée, donc au niveau du recuit simulé :
– Une diminution de la fonction sera toujours acceptée
– Une augmentation de la fonction sera acceptée avec une probabilité définie selon une formule du type 1.
k doit être adapté à l'application, on pourra prendre dans un premier temps $k = 1$ et T est un paramètre formel qui joue le rôle de la température.

3.1 Algorithme

X_0 représente le vecteur initial et `voisin(X)` est une fonction qui calcule un vecteur aléatoire Y voisin de X.

```
float recuit( float X0 ) {
  float X = X0;
  float T = T0;
  float Nt = /* iterations à chaque pas de temperature. */
  while( ! condition_d_arret ) {
    for( int m=0; m < Nt; m++ ) {
      float Y = voisin( X );
      float dF = F( Y ) - F( X );
      if( accepte( dF, T ) ) {
```

```
        X = Y;
      }
    }
    T = decroissance( T );
  }
  return X;
}
```

Il faut maintenant définir l'algorithme de la fonction `accepte()`. Cela nous donne :

```
boolean accepte( float dF, float T ) {
  if( dF >= 0 ) {
    float A = Math.exp( -dF/T );
    if( Math.random() >= A ) {
      return false;
    }
  }

  return true;
}
```

Il faut déterminer la température initiale T_0. Au début des itérations, on est éloigné du minimum il faut donc choisir une probabilité d'acceptation élevée, $p = 0.5$ par exemple. On génère un certain nombre de vecteurs X aléatoires et pour chacun de ces vecteurs on détermine $F(X)$, ensuite on détermine la valeur médiane M, c'est à dire la valeur qui partage la distribution en deux parties égales.

$$\begin{array}{rcl} p & = & e^{-\frac{M}{T_0}} = 0.5 \\ log_n p & = & \frac{-M}{T_0} \\ T_0 & = & \frac{-M}{log_n p} \\ T_0 & \simeq & 1.44M \end{array}$$

Au niveau de la décroissance du paramètre T on peut prendre une décroissance géométrique de la forme :

$$\begin{array}{rcl} T_{n+1} & = & R_T T_n \\ T_n & = & T_0 R_T^n \end{array}$$

Il reste à fixer le critère d'arrêt. On peut arrêter les itérations lorsque la température T_n est inférieure à une fraction donnée de la température initiale $T_n < T_{ratio} T_0$, avec $T_{ratio} = 10^{-6}$ par exemple. Maintenant on peut déterminer R_T en prenant $N_T = 100$ par exemple :

$$\begin{array}{rcl} R_T & = & T_{ratio}^{\frac{1}{N_T}} \\ N_T & = & 100 \\ R_T & = & (10^{-6})^{\frac{1}{100}} \simeq 0.87 \end{array}$$

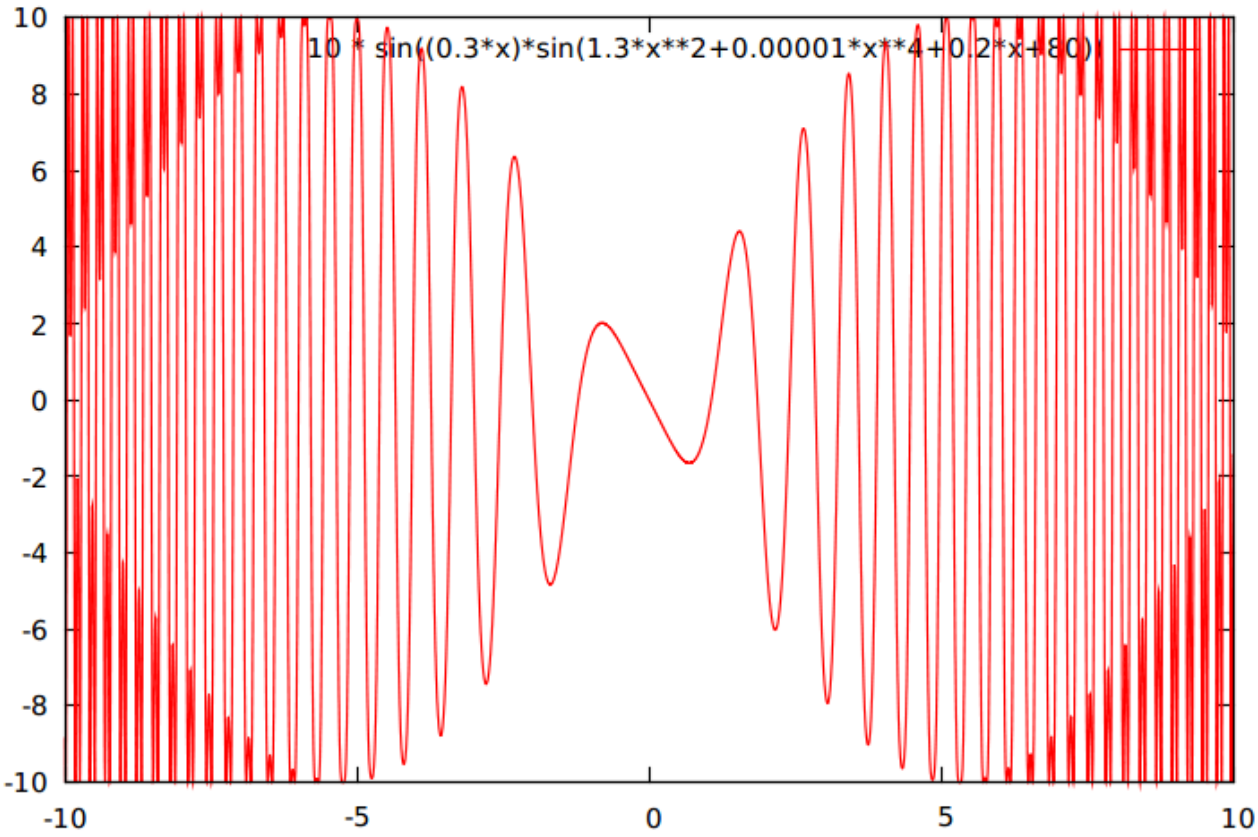
3.2 Mise en œuvre

3.2.1 Minimum d'une fonction

On se propose de rechercher le minimum à l'aide du recuit simulé de la fonction :

$$f(x) = 10 * sin((0.3 * x) * sin(1.3 * x^2 + 0.00001 * x^4 + 0.2 * x + 80)) \quad (2)$$

Dont la courbe représentative est la suivante :



3.2.2 Application au problème des n reines

On se propose donc d'appliquer le recuit simulé au problème des n reines. Supposons que l'on représente un état par un vecteur [1 .. N] avec par exemple $N = 8$, alors une solution possible est [4, 2, 7, 3, 6, 8, 5, 1] si l'on considère que la composante i représente la position de la reine en ligne i. L'état initial est alors une permutation aléatoire de l'ensemble [1 .. N]. Pour produire un nouvel état à partir de l'état courant, on échange deux positions (3, 6) : [1, 4, 3, 6, 2, 8, 5, 7] -> [1, 4, 6, 3, 2, 8, 5, 7] et l'énergie d'un état est donné par le nombre de conflits. [3, 1, 4, 7, 6, 8, 5, 2] Energie = 6, devient [4, 1, 3, 7, 6, 8, 5, 2] Energie = 4.