

TD IMSP 3 : Méthode du recuit simulé

ou comment une distribution statistique de physique classique (Boltzmann) a inspiré l'algorithme de Monte Carlo-Metropolis

Les métaheuristiques constituent un ensemble de méthodes permettant de concevoir des algorithmes dans le but de résoudre des problèmes d'optimisation : il s'agit généralement de méthodes stochastiques (aléatoires) et itératives dont le but est de converger vers un extremum préférentiellement global (absolu).

Le recuit simulé (-Simulated Annealing-) est la première méthode métaheuristique qui ait été proposée (à partir d'expériences réalisées par Metropolis et al. dans les années 50) et son utilisation pour la résolution de problèmes d'optimisation date des années 80. Elle imite le processus de recuit utilisé en métallurgie. Un cristal macroscopique présente forcément des défauts d'ordonnancement et, on s'aperçoit que l'on peut en améliorer la pureté en le chauffant brutalement puis en le laissant refroidir lentement. En effet, initialement le cristal est dans un état stable ou métastable (minimum local d'énergie) et le chauffage va permettre à la structure de sortir de ce puits de potentiel pour tomber éventuellement dans un puits plus profond d'énergie lors d'un lent refroidissement.

Nous traiterons tout d'abord une situation physico-chimique correspondant effectivement à une distribution statistique de Boltzmann : **les conformations de stabilités relatives de la molécule de butane**. Nous lui appliquerons le principe du recuit simulé (algorithme de Metropolis) pour repérer (dans le cadre réaliste d'une distribution de Boltzmann) le rôle de différents paramètres : température initiale, vitesse de chute de température, puits d'énergie potentielle, barrières d'énergie potentielle (maxima locaux séparant deux minima)...

Nous chercherons alors à adapter le programme Python à une situation pratique sans rapport avec ces considérations énergétiques : l'**optimisation du chemin d'un VRP** qui doit boucler une série de N visites dans des villes diverses et revenir à son point de départ en parcourant une distance minimale. (Problème du voyageur de commerce (en anglais : TSP pour Travelling Salesman Problem))

I. La statistique de Boltzmann...

I.1. ...observée en cours sur un exemple de statique des fluides

Lors de la modélisation de la basse atmosphère par un gaz parfait à l'équilibre thermodynamique à la température uniforme T, nous avons obtenu par la relation fondamentale de la statique des fluides et l'équation d'état des gaz parfaits, une évolution de la pression avec l'altitude de la forme :

$$p(z) = p(0) \cdot e^{\left(-\frac{Mgz}{RT}\right)}$$

avec : M la masse molaire moyenne de l'air

g l'accélération de la pesanteur supposée uniforme

z l'altitude

R la constante des gaz parfaits

T la température

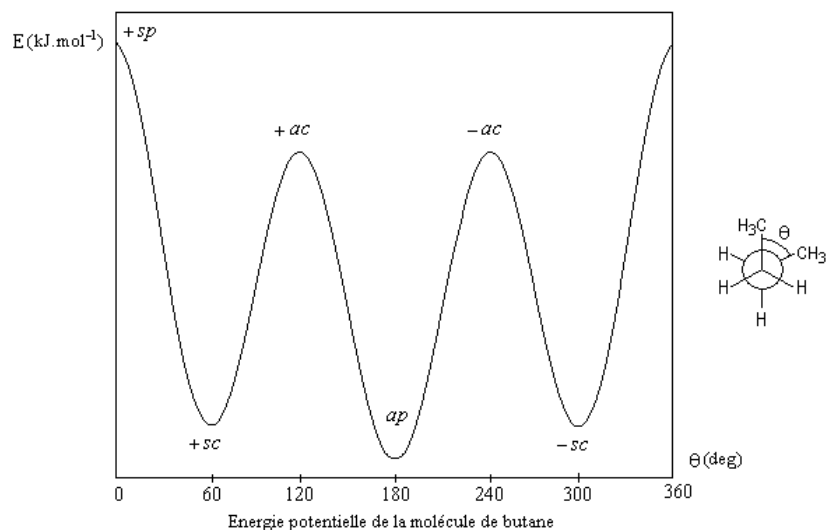
La masse volumique ou la densité particulière de molécules d'atmosphère suivent aussi cette même évolution exponentielle négative de l'altitude. La probabilité pour qu'une molécule choisie « au hasard » dans cette population ait une altitude z à Δz près (ou une énergie potentielle Mgz à $Mg\Delta z$ près) est donc proportionnelle à cette exponentielle.

On remarque que le quotient présent dans l'exposant est le rapport de deux énergies : l'une « ordonnante » au numérateur (énergie potentielle de pesanteur : sans agitation thermique, toutes les molécules d'air devraient se retrouver au sol !) et l'autre « désordonnante » au dénominateur (énergie d'agitation thermique).

Dans le cas où l'énergie de pesanteur est très supérieure à l'énergie d'agitation thermique, la variation de concentration en particules (ou de pression) est très importante de 0 à z (gradients élevés) alors qu'au contraire, si l'énergie d'agitation thermique l'emporte largement, la concentration (ou la pression) en z est peu différente de la pression au sol (z=0). Plus cette température sera faible plus les molécules seront confinées au voisinage de du sol soit au minimum d'énergie potentielle de pesanteur dans notre situation.

I.2....appliquée à des conformations moléculaires.

Prenons l'exemple de la molécule de butane. On parle parfois de libre rotation autour de la liaison reliant les atomes de carbone 2 et 3 du squelette de 4 carbones. Pourtant l'énergie d'une conformation dépend de la conformation par l'angle de torsion : la conformation éclipsant les groupements méthyles (CH₃) est la plus instable (conformation synpériplanaire) donc de plus forte énergie potentielle et la conformation les éloignant au maximum (toutes les longueurs de liaisons restants fixes par ailleurs) constitue un minimum d'énergie potentielle choisi comme référence $E_p=0$ kJ/mol (conformation antipériplanaire). Lorsque l'angle de torsion évolue, la conformation passe également par un minimum local (synclinale) puis un maximum local (anticlinale). Les valeurs approchées de l'énergie potentielle fonction de l'angle de torsion sont résumées ci-dessous :



q (deg)	0	60	120	180
Energie (kJ.mol ⁻¹)	18,8	3,8	15,9	0

On peut proposer une expression approchée assez simple pour la fonction $Ep(\theta)$ (respectant au moins les valeurs d'énergie potentielle au max absolu et aux deux minima) :

$$Ep(\theta) \approx 2,53 \cdot \cos\left(\frac{\alpha \pi}{180}\right) + 6,87 \cdot \cos\left(\frac{\alpha \pi}{60}\right) + 9,4$$

avec α en degrés et Ep en kJ/mol

(De manière plus complète, cette énergie est la somme d'un terme de torsion de liaison et de répulsion entre les groupements méthyles. Remarquons également que l'on va négliger les énergies intermoléculaires dans ce modèle).

A un instant t et une température T, la proportion de molécules correspondant à une énergie potentielle donnée à ΔEp près (et donc à une ou plusieurs conformations différentes) suit une loi statistique de Boltzmann.

La quantité relative de molécules dans une conformation d'énergie Ep à dEp près s'écrit donc :

$$dp = \frac{dN}{N} = K(T) e^{-\frac{E_p}{RT}} dE_p = \frac{e^{-\frac{E_p}{RT}} dE_p}{RT \left[1 - e^{-\frac{E_{p_{\max}}}{RT}} \right]} \text{ par normalisation}$$

- a- Exprimer le rapport entre les probabilités de conformation de :
- antipériplanaire/synpériplanaire
 - antipériplanaire/synclinaire
- b- Calculer ces rapports pour les températures suivantes :
- T=500 K
 - T=300 K (proche des températures ambiantes)
 - T=2K
- c- Interpréter RT comme une énergie cinétique pouvant être (au moins partiellement) convertie en énergie potentielle et expliquer ce qui va se passer à haute température puis lors d'un lent refroidissement.

Voici un programme Python utilisant un algorithme de type Metropolis basé sur la réelle distribution de Boltzmann des énergies entre $E_{\min}(\text{ap})=0$ kJ/mol et $E_{\max}(\text{sp})=18,8$ kJ/mol. Interpréter clairement le principe de l'algorithme, les sorties numériques et graphiques puis commenter complètement le programme.

```

8 import numpy as np
9 import matplotlib.pyplot as plt
10 import random as rd
11
12 """ Initialisations """
13 R=8.314
14 alphamax=360
15 alpha0=5
16 angles=[alpha0]
17 Ep0=2.53*np.cos(alpha0*np.pi/180)+6.87*np.cos(alpha0*np.pi/60)+9.4
18 energies=[Ep0]
19 essais=600000
20 T=0.5
21 T0=T
22 temperatures=[T0*1000]
23 coeff=0.99999
24
25 """ Recherche d'énergie minimale avec limite probabiliste de Boltzmann """
26 for k in range(1,essais):
27     dalphamodif=10*(rd.random()-0.5)
28     alpha=(alpha0+dalphamodif)
29     Ep=2.53*np.cos(alpha*np.pi/180)+6.87*np.cos(alpha*np.pi/60)+9.4
30     Ep0=2.53*np.cos(alpha0*np.pi/180)+6.87*np.cos(alpha0*np.pi/60)+9.4
31     ecart=Ep-Ep0
32
33     if (ecart<0)or(rd.random()<np.exp(-ecart/(R*T))):
34         alpha0=alpha
35         angles.append(alpha0)
36         energies.append(Ep)
37         temperatures.append(T*1000)
38
39     T=T*coeff
40
41 """ Valeurs finales obtenues après chute d'agitation thermique """
42 print ('température initiale',round(T0*1000,0),'Kelvin')
43 print ('température finale',round(T*1000,1),'Kelvin')
44 print ('angle final',round(alpha0,1),'degrés')
45 print ('Energie potentielle finale',round(Ep,1),'kJ/mol')
46

```

```

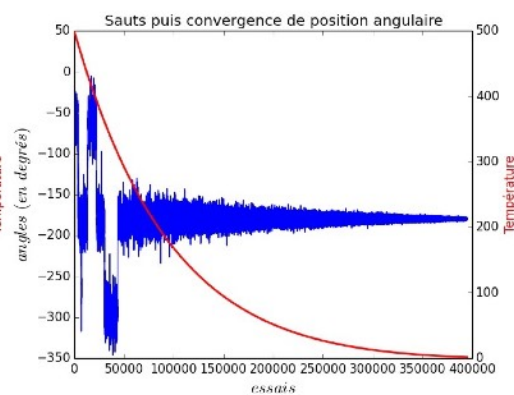
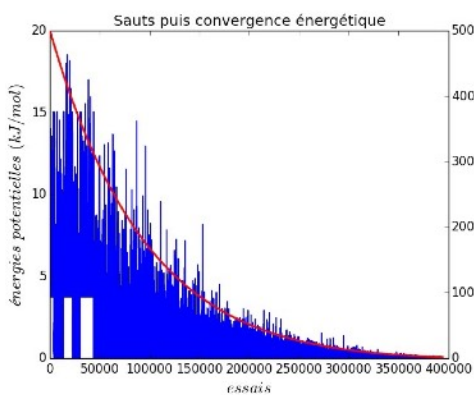
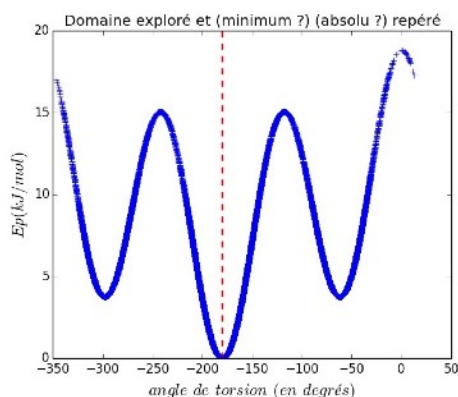
47 """ Représentations graphiques """
48 plt.figure()
49 plt.subplot(1,3,1)
50 plt.plot(angles,energies,'b+')
51 plt.plot([alpha0,alpha0],[0.0,20],'r--',lw=2)
52 plt.xlabel(u'$angle$,de$,torsion$, (en$,degrés)$', fontsize=16)
53 plt.ylabel(u'$Ep$(kJ/mol)$', fontsize=16, rotation=90)
54 plt.title(u'Domaine exploré et (minimum ?) (absolu ?) repéré')
55 plt.plot([alpha0,alpha0],[0.0,Ep],'b--',lw=2)
56 plt.subplot(1,3,2)
57 plt.plot(energies)
58 plt.xlabel(u'$essais$', fontsize=16)
59 plt.ylabel(u'$énergies$,potentielles$, (kJ/mol)$', fontsize=16, rotation=90)
60 plt.title(u'Sauts puis convergence énergétique')
61 plt.twinx()
62 plt.ylabel("Température", color="r")
63 plt.tick_params(axis="r", labelcolor="r")
64 plt.plot(temperatures, "r-", linewidth=2)
65 plt.subplot(1,3,3)
66 plt.plot(angles)
67 plt.xlabel(u'$essais$', fontsize=16)
68 plt.ylabel(u'$angles$, (en$,degrés)$', fontsize=16, rotation=90)
69 plt.title(u'Sauts puis convergence de position angulaire')
70 plt.twinx()
71 plt.ylabel("Température", color="r")
72 plt.tick_params(axis="r", labelcolor="r")
73 plt.plot(temperatures, "r-", linewidth=2)
74 plt.show()

```

Exemple de sorties numériques :

température initiale 500.0 Kelvin
température finale 1.2 Kelvin
angle final -180.0 degrés
Energie potentielle finale 0.1 kJ/mol

Sortie graphique :



I.3....et pour aller plus loin sur des calculs de chimie physique.

De façon générale, l'énergie potentielle d'une molécule est la somme de nombreuses contributions :

- l'énergie d'élongation des liaisons (assimilable à une énergie élastique) :
$$\sum_{\text{liaisons}} K_{\text{elong}} \left(\Delta r_{ij} - \Delta r_{ijeq} \right)^2$$
- l'énergie de répulsion angulaire plane entre liaisons (élastique) :
$$\sum_{\text{liaisons}} K_{\text{rep}} \left(\theta - \theta_{eq} \right)^2$$
- l'énergie de torsion angulaire diédrique (sinusoïdale comme celle traitée) :
$$\sum_{\text{angles diédriques}} \left(\sum_{n=1}^3 \left(\frac{K_n}{2} \right) \cdot (1 + \cos(n\varphi - \varphi_n)) \right)$$
- l'énergie électrostatique entre excédents de charge ponctuelle sur la molécule :
$$\sum_{\substack{\text{atomes} \\ \text{chargés}}} \frac{q_i q_j}{4\pi\epsilon_0 \epsilon_r r_{ij}}$$
- l'énergie potentielle de Lennard-Jones (interaction dipolaire) :
$$\sum_{\substack{\text{atomes} \\ \text{nonliés}}} \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^6}$$
- l'énergie potentielle des liaisons « Hydrogène » intermoléculaires :
$$\sum_{\substack{\text{liaisons} \\ \text{Hydrogène}}} \frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}}$$

Et il s'agit donc de minimiser la somme de ces termes sur une collection de molécules !

Un exemple assez simple traité par l'algorithme de Métropolis : un **agrégat condensé d'atomes d'Argon** à basse température. Le programme qui donne les résultats suivants (pour une petite collection de 25 atomes) a utilisé pour seule énergie potentielle d'interaction une expression de type Lennard-Jones et posait seulement le problème de manière bidimensionnelle. On constate, partant d'un maillage 2D carré de distance relative 2, qu'après chauffage et refroidissement lent jusqu'à des températures simulées de l'ordre du Kelvin, on converge vers un **maillage hexagonal régulier** de distance relative d'environ 1,25.

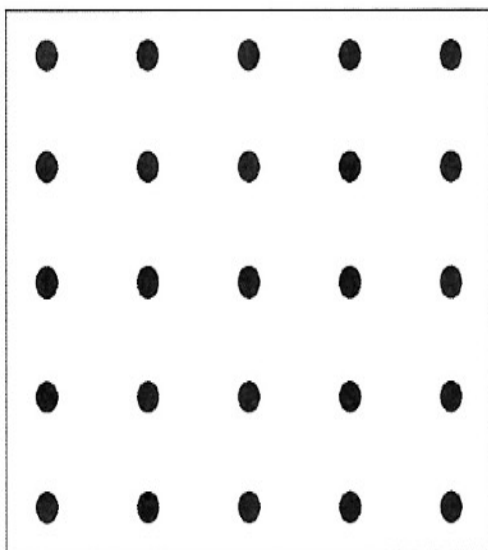


Figure 2 : Configuration de départ des centres des 25 atomes de l'agrégat selon un maillage régulier.

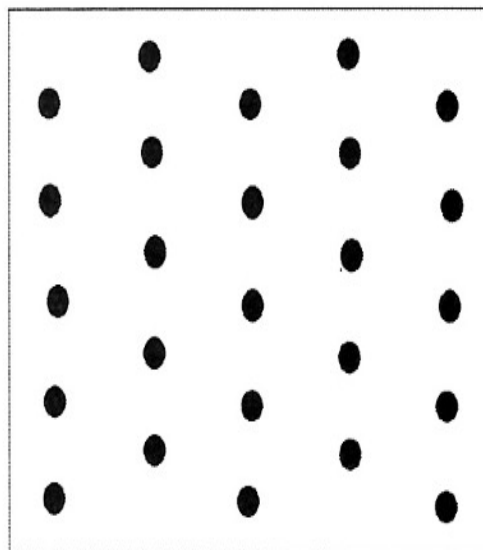


Figure 3 : Configuration des centres des 25 atomes d'argon de l'agrégat après 8000 générations.

II. Metropolis à la recherche du chemin minimum

Cette technique du « recuit simulé » peut être mise à profit dans la minimisation de fonctions (non énergétiques !) présentées comme une somme finie de contributions. Une probabilité Boltzmannienne sera associée à chaque configuration proposée et un paramètre « température » (sans signification physique !) permettra d'élaborer une procédure de type Metropolis.

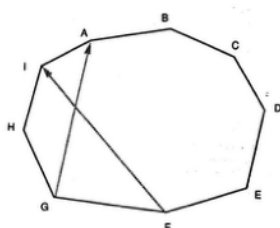
Le problème du voyageur de commerce

L'application historique de cet algorithme est le « problème du voyageur de commerce » (Kirkpatrick *et al.*)

Un VRP doit visiter N villes en passant une seule fois par chacune et revenir à sa ville de départ en minimisant la longueur de ce parcours fermé.

Le nombre de séquences possibles est de $N!$ (soit $\approx 2.10^8$ pour $N=20$ villes !) et le calcul de toutes les longueurs de trajets possibles est déraisonnable.

Il s'agit donc de tester des séquences de visites choisies aléatoirement en calculant à chaque fois la longueur totale de trajet. On choisit alors d'appliquer une perturbation locale aléatoire. Il peut s'agir d'une permutation assez simple autour de deux villes i et j comme illustrée ci-dessous :



La modification de séquence correspondant à deux villes permutées i et j serait (avec une numérotation des villes) :

$[1, 2, 3, \dots, i, i+1, \dots, j-1, j, \dots, N]$ devenant $[1, 2, 3, \dots, j, j-1, \dots, i+1, i, \dots, N]$

Remarquons que cette permutation serait ici bien malheureuse dans l'objectif de minimiser le trajet !

Si le nouveau trajet s'avère plus court que le précédent, on l'adopte systématiquement et on teste une nouvelle séquence.

Mais si le nouveau trajet est plus long que le précédent, on lui attribue une probabilité boltzmanienne :

$$p = \exp\left(\frac{(\text{nouvelledistance} - \text{distanceprécédente})}{\text{"Température"}}$$

On peut alors éventuellement (aléatoirement) choisir de l'adopter si cette probabilité est supérieure à une probabilité tirée aléatoirement entre 0 et 1. On « choisit » donc parfois de tolérer une augmentation de cette distance. L'intérêt étant de « pouvoir sortir » d'un minimum local relatif pour explorer d'autres configurations qui pourraient s'avérer elles voisines d'un minimum absolu. Mais cette option de tolérance d'une augmentation de distance voit ses chances s'amenuiser au fur et à mesure de la diminution du paramètre figurant la « température »...

Ci-dessous une évolution de chemin entre 20 villes sur 5000 générations :

Les figures 1, 2 et 3 montrent les résultats d'optimisation après plusieurs générations. La figure 3 présente une courbe fermée sans intersection de trajectoires.

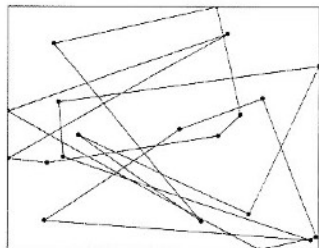


Figure 1 :
Après 10^2 générations

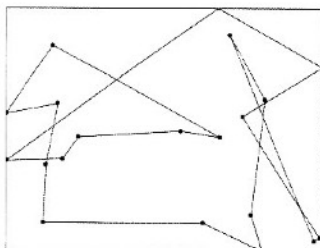


Figure 2 :
Après 10^3 générations

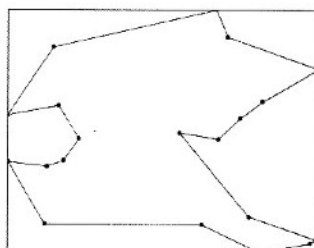


Figure 3 :
Après 5.10^3 générations

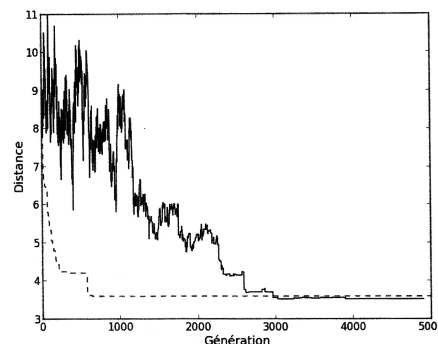


Figure 4 : Évolutions de la distance parcourue entre 20 villes, pour une température initiale de 0,15 et un taux de décroissance en température de 0,001 (trait plein) et 0,01 (trait pointillé).

On peut trouver des applets java interactifs sur internet comme le suivant :
<http://www-sop.inria.fr/mefisto/java/tutorial1/node18.html>

```
import random as rd
import matplotlib.pyplot as plt
import numpy as np
```

N= 40

Génération aléatoire de N couples de coordonnées cartésiennes
pour N villes dans un carré de côté unitaire

Calcul de la matrice symétrique NxN des distances intervilles

```
print(matricedistances)
```

```
def trajet(seqvoyage,matricedistances):
```

Définition d'une fonction trajet d'arguments une séquence de villes et la
matrice des distances qui retournant la longueur du trajet fermé

```
return longueur
```

```
def modifseq(seqvoyage):
```

Définition d'une fonction qui retournera une
nouvelle séquence de voyage (nouvseqvoyage)
à partir d'une séquence précédente (seqvoyage)
par substitution et renversement de deux
couples de villes successives

```
return nouvseqvoyage
```

```
essais=50000
```

```
T=0.2
```

```
T0=T
```

```
listelongueurs=[]
```

```
temperatures=[T0]
```

```
coeff=0.9999
```

```
seqvoyageinit=list(range(N))
```

```
seqvoyage=seqvoyageinit
```

```
for e in range(1,essais):
```

Procédure de recuit simulé itérée sur 5000 essais utilisant les deux
fonctions précédentes et la variable température décroissante
(On notera :
newseq la nouvelle séquence
newlong la nouvelle longueur
long l'ancienne longueur de la séquence seqvoyage
ecart l'écart)

liste des températures
et décroissance itérative

```
plt.figure(1)
```

```
plt.plot(listelongueurs)
```

```
plt.twinx()
```

```
plt.ylabel("Température", color="r")
```

```
plt.plot(temperatures, "r-", linewidth=2)
```

```
plt.figure(2)
```

```
plt.subplot(1,2,1)
```

```
x=[absc[seqvoyageinit[i]] for i in range(N)]+[absc[seqvoyage[0]]]
```

```
y=[ordo[seqvoyageinit[i]] for i in range(N)]+[ordo[seqvoyage[0]]]
```

```
plt.plot(x,y,"r.")
```

```
plt.plot(x,y,"k-")
```

```
plt.subplot(1,2,2)
```

```
x=[absc[seqvoyage[i]] for i in range(N)]+[absc[seqvoyage[0]]]
```

```
y=[ordo[seqvoyage[i]] for i in range(N)]+[ordo[seqvoyage[0]]]
```

```
plt.plot(x,y,"r.")
```

```
plt.plot(x,y,"g-")
```

```
plt.show()
```

