TP d'algorithmes Stochastiques 1

Méthode de Monte Carlo : calcul de surface, d'intégrale et d'extremum

décembre 2020

CROGUENNEC Guillaume

DUPONT Ronan

Enseignant : M. Maire

Table des matières

Introduction

Les méthodes de Monte-Carlo désignent des méthodes numériques utilisant le hasard via des nombres aléatoires. La méthode de Monte-Carlo la plus classique est le calcul d'une quantité que l'on a écrit sous forme d'une espérance et que l'on approche par sa moyenne empirique. Ces méthodes s'utilisent pour une très grande diversité de problèmes comme le calcul d'intégrales en grande dimension ou historiquement le calcul de criticité pour les réacteurs nucléaires. Elles remplacent les méthodes classiques d'analyse numérique quand ces problèmes deviennent trop complexes pour différentes raisons (grande dimension, géométrie complexe ou irrégularité de la solution).

1 Calculs de surfaces et de volumes

La première utilisation que l'on fait de la méthode de Monte Carlo est le calcul d'aire et de volumes. Pour la surface, le principe est de placer la figure dans une figure plus grande dont on connaît la surface. Ensuite on place aléatoirement des points selon une loi uniforme dans la grande figure, et on voit quelle fraction de ces points est à l'intérieur de la figure dont on veut connaître la surface. Celle-ci vaudra donc cette fraction multipliée par la surface de la grande figure. Pour le calcul d'un volume, la méthode est analogue mais en trois dimensions. Bien sûr, grâce à la loi des grands nombres, plus on met de points plus il y a de chances d'être proche de l'aire exacte.

1.1 Cercle

On commence par appliquer cette méthode au calcul de l'aire d'un cercle de centre O=(0,0) et de rayon R=1 d'équation $x^2 + y^2 = 0$ où $x, y \in \mathbb{R}^2$.
Pour cela on place les points $(X_n)_{n\geq 1}$, une suite de variables aléatoires indépendantes et uniformes, sur le domaine $D = [-1, 1]^2$ dans lequel le cercle est situé. On note l'aire de ce domaine $S = 2^2 = 4$.
La probabilité que X_i tombe dans le disque est donc $\frac{A}{D} = \frac{\pi}{4}$.
On introduit ensuite $(Y_n)_{n\geq 1}$ une suite de variable aléatoire qui vaut 1 si X_i est dans le disque et 0 si X_i est hors du disque.
D'après la lois des grands nombres, on a donc :

$$\mathbb{E}(Y) = \lim_{n\to+\infty} \frac{1}{n} \sum_{i=1}^{n} Y_i = \frac{\pi}{4}$$

On effectue donc N tirages aléatoires dans D comme on peut le voir figure 1 puis on compte le nombre de points (c) dans le disque pour trouver la valeur de $\frac{\pi}{4}$. A la fin on divise donc c par le nombre de tirages N et l'on obtient $\frac{1}{4}$ de la surface du disque. On multiplie donc le résultat obtenu par 4 afin d'obtenir l'air du cercle.

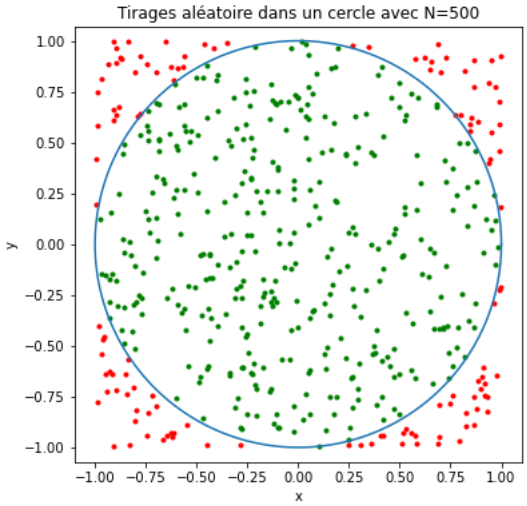

FIGURE 1 – Tirages dans un cercle

On trace ensuite l'erreur pour des simulations avec N allant de 10^3 à 10^6, plus précisément N =[1 000,5 000,7 000,10 000,50 000,70 000,100 000,300 000,700 000,1000000] et on a obtenu les résultats figure 2.

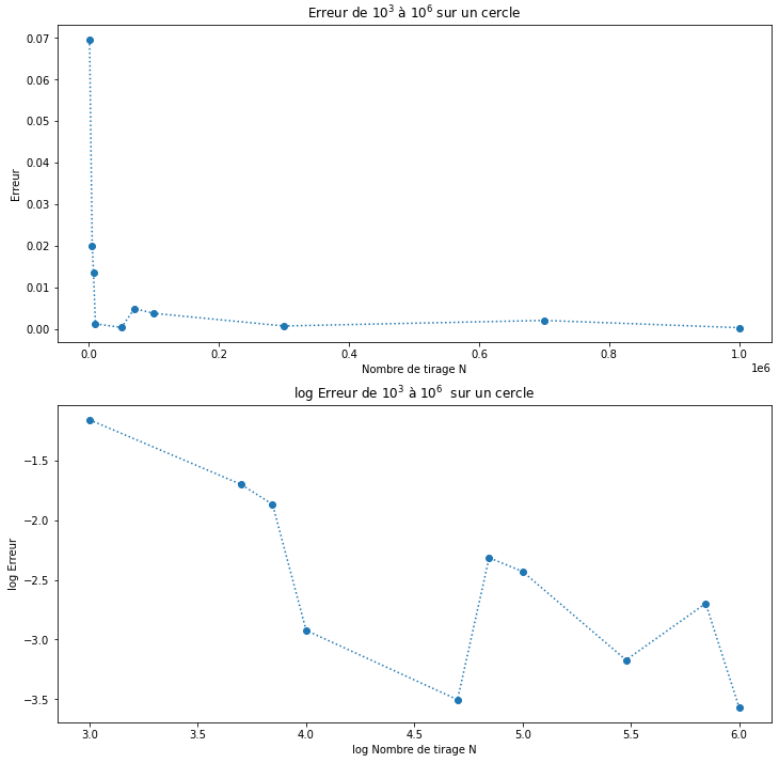

FIGURE 2 – Erreur de 10^3 à 10^6 cercle

On remarque que l'erreur n'est pas décroissante de manière linéaire comme pour des méthodes numériques habituelles. C'est dû au fait d'utiliser l'aléatoire, mais si on avait calculé l'erreur par rapport à la moyenne de plusieurs essaies pour chaque N on aurait eu quelque chose de bien plus proche d'être linéaire. Mais on devine déjà sur cette courbe que le logarithme de l'erreur est affine, mais que les points 4 et 5 sont trop bas et le point 9 trop haut. L'erreur décroît donc de manière exponentielle.

1.2 Ellipse

Ensuite, on utilise la même méthode pour calculer l'aire d'une ellipse de centre O=(0,0) et d'équation $(\frac{x}{2})^2 + y^2 = 0$ où $x, y \in \mathbb{R}^2$.
On peut aussi écrire cette équation sous forme paramétrique :

$$\begin{cases} x = 2\cos(t), \ t \in [0, 2\pi] \\ y = \sin(t), \ t \in [0, 2\pi] \end{cases}$$

On place donc les points $(X_n)_{n\geq 1}$, une suite de variables aléatoires indépendantes et uniformes, sur le domaine $D = [-2, 2]\text{x}[-1, 1]$ contenant l'ellipse. On note l'aire de ce domaine $S = 4.2 = 8$. La probabilité que X_i tombe dans l'ellipse est donc $\frac{A}{D} = \frac{1.2.\pi}{8} = \frac{2\pi}{4}$.
On introduit ensuite $(Y_n)_{n\geq 1}$ une suite de variable aléatoire qui vaut 1 si X_i est dans l'ellipse et 0 si X_i est hors de l'ellipse.
D'après la lois des grands nombres, on a donc :

$$\mathbb{E}(Y) = \lim_{n\to+\infty} \frac{1}{n} \sum_{i=1}^{n} Y_i = \frac{\pi}{4}$$

On effectue donc N tirages aléatoires dans D comme on peut voir figure 3 puis on compte le nombre de points (c) dans le disque pour trouver la valeur de $\frac{\pi}{4}$. A la fin on divise donc c par le nombre de tirages N et l'on obtient $\frac{1}{8}$ de la surface de l'ellipse. On multiplie donc le résultat obtenu par 8 afin d'obtenir l'air de l'ellipse.

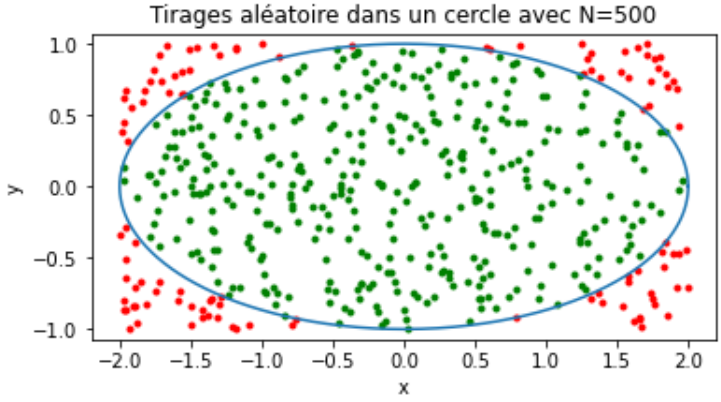

FIGURE 3 – Tirages dans une ellipse

On trace ensuite l'erreur pour des simulations avec N allant de 10^3 à 10^6, et plus précisément N =[1 000, 5 000, 7 000, 10 000, 50 000, 70 000, 100 000, 300 000, 700 000, 1 000 000] et on obtient les résultats figure 4.

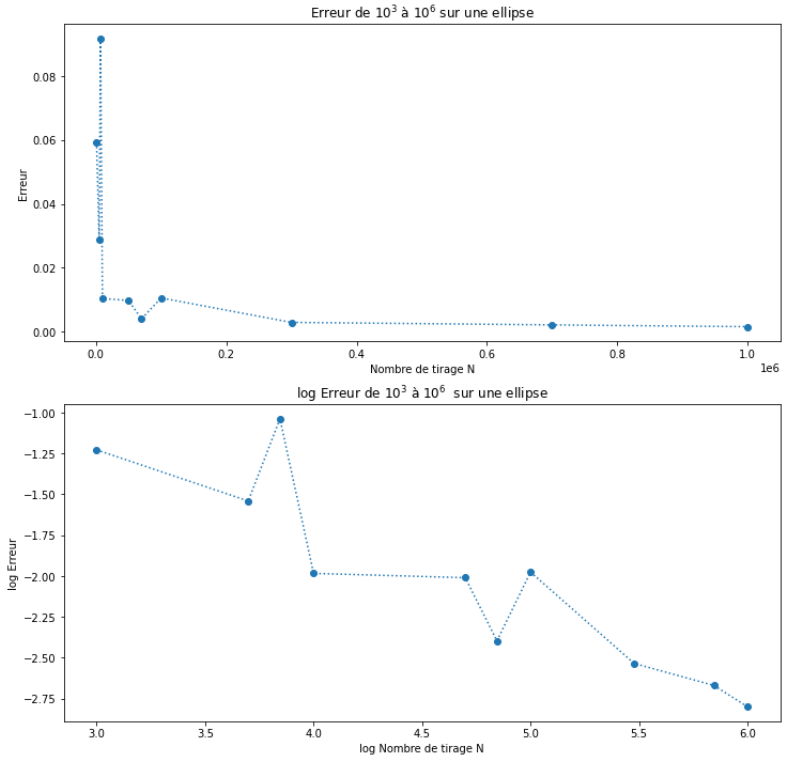

FIGURE 4 – Erreur de 10^3 à 10^6 ellipse

De même que pour le cercle, on remarque que le logarithme de l'erreur est affine, mais encore une fois que l'aléatoire donne une courbe irrégulière.

1.3 Sphère

Enfin, on utilise encore cette méthode pour calculer le volume d'une sphère de centre O=(0,0,0) et de rayon R=1 d'équation $x^2 + y^2 + z^2 = 0$ où $x, y, z \in \mathbb{R}^3$.
On place les points $(X_n)_{n\geq 1}$, une suite de variables aléatoires indépendantes et uniformes, sur le domaine $D = [-1, 1]^3$ contenant la sphère. On note le volume de ce domaine $S = 2^3 = 8$.
La probabilité que X_i tombe dans la sphère est donc $\frac{V}{D} = \frac{\frac{4}{3}\pi}{8} = \frac{\pi}{6}$.
On introduit ensuite $(Y_n)_{n\geq 1}$ une suite de variable aléatoire qui vaut 1 si X_i est dans la sphère et 0 si X_i est hors de la sphère.
D'après la lois des grands nombres, on a donc :

$$\mathbb{E}(Y) = \lim_{n\to+\infty} \frac{1}{n} \sum_{i=1}^{n} Y_i = \frac{\pi}{6}$$

On effectue donc N tirages aléatoires dans D puis on compte le nombre de points (c) dans la sphère pour trouver la valeur de $\frac{\pi}{6}$. A la fin on divise donc c par le nombre de tirages N et l'on obtient $\frac{1}{8}$ du volume de la sphère. On multiplie donc le résultat obtenu par 8 afin d'obtenir le volume de la sphère.
On trace ensuite l'erreur pour des simulations avec N allant de 10^3 à 10^6, et plus précisément N =[1 000, 5 000, 7 000, 10 000, 50 000, 70 000, 100 000, 300 000, 700 000, 1 000 000] et on obtient les résultats figure 5.
Encore une fois, on remarque que le logarithme de l'erreur est affine, donc que l'erreur décroît de manière exponentielle.

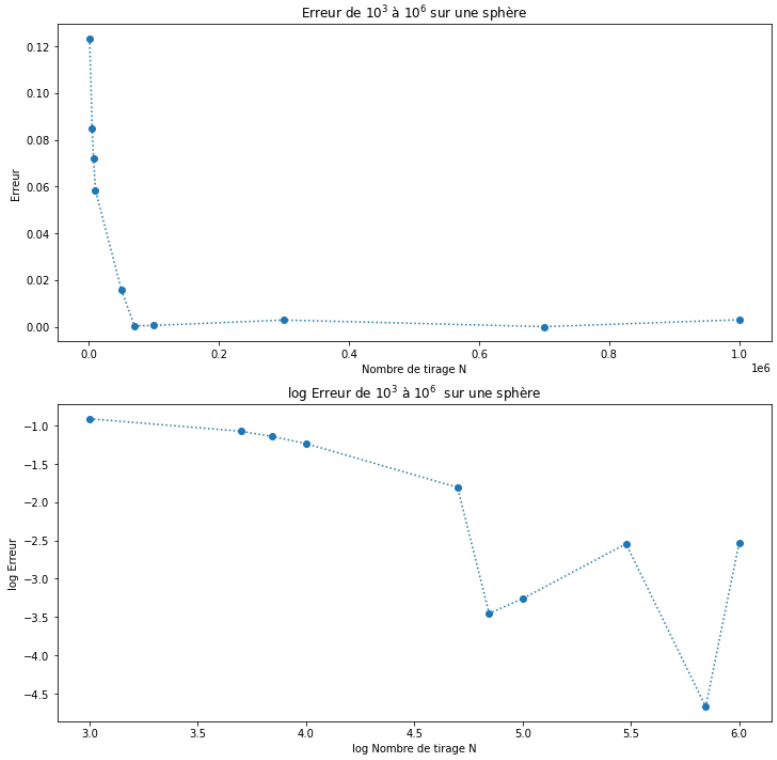

FIGURE 5 – Erreur de 10^3 à 10^6 sphère

1.4 Intervalles de confiances :

Dans cette partie,on étudie l'intervalle de confiance à 95 pour-cent afin de regarder la précision des calculs réalisés et de vérifier si la solution est bien comprise dans l'intervalle. Pour cela, on calcule la variance σ^2 qui a pour valeur : $\sigma = p(1-p)$ pour une loi de Bernoulli. On peut approcher cette variance par : $\boxed{s^2 = \frac{n_1}{N}(1-\frac{n_1}{N})}$ où on a donc n_1 le nombre de tirages dans la surface et N le nombre de tirages total. On arrive directement à la formule de l'intervalle de confiance :

$$[-\frac{A.B.a.s}{\sqrt{n}} + \frac{A.B.n_1}{n}, \frac{A.B.a.s}{\sqrt{n}} + \frac{A.B.n_1}{n}]$$

Avec ici A et B les longueurs des tirages respectivement selon x et y, et $a = 1.96$.

Cercle

Dans le cas du cercle on obtient :

N	σ	Borne inf	Borne Sup	π dans l'intervalle ?
1 000	0.40149221661197865	3.0924610184018344	3.291538981598166	Oui
5 000	0.42018924307982947	3.0374117936159806	3.1305882063840196	Non
7 000	0.4077144859344788	3.119509023500816	3.195919547927756	Oui
10 000	0.40730823708832603	3.1280670342122754	3.191932965787725	Oui
50 000	0.41072518841678074	3.1261593399592975	3.154960660040702	Oui
70 000	0.4094076198407566	3.135982549348406	3.160246022080165	Oui
100 000	0.4097357275122588	3.1360817266005507	3.1563982733994496	Oui
300 000	0.4100035554181885	3.138837951145582	3.150575382187751	Oui
700 000	0.4105624398956403	3.1376499296203444	3.145344356093941	Oui
1 000 000	0.4107244941794925	3.1373439199656326	3.1437840800343673	Oui

Ellipse

Dans le cas de l'ellipse on obtient :

N	σ	Borne inf	Borne Sup	2π dans l'intervalle ?
1 000	0.41559114523772034	6.017931151606072	6.430068848393929	Oui
5 000	0.41302392182535863	6.162812487809065	6.345987512190936	Oui
7 000	0.4023379633277152	6.299454242456694	6.450260043257591	Non
10 000	0.41144524544585515	6.20828538551409	6.33731461448591	Oui
50 000	0.4096937583122301	6.264231006986475	6.321688993013525	Oui
70 000	0.4101965260701265	6.2628897728751705	6.31151022712483	Oui
100 000	0.4096237805596741	6.273449004036307	6.314070995963693	Oui
300 000	0.4102917465928631	6.274360985393506	6.297852347939828	Oui
700 000	0.4103597198576301	6.2776350870257716	6.293016341545657	Oui
1 000 000	0.41068352779238665	6.275160482284216	6.288039517715784	Oui

Sphère

Dans le cas de la sphère on obtient :

N	σ	Borne inf	Borne Sup	$\frac{4}{3}\pi$ dans l'intervalle ?
1 000	0.498476679494638	4.06483276250765	4.559167237492351	Oui
5 000	0.49882898873261167	4.162985326920463	4.384214673079537	Oui
7 000	0.49978762836767754	4.0229053065592515	4.210237550583606	Oui
10 000	0.49904427659276884	4.168949857430254	4.325450142569746	Oui
50 000	0.4995314908992225	4.138091307419474	4.208148692580526	Oui
70 000	0.49944070249242045	4.159543589130287	4.218742125155428	Oui
100 000	0.4994460322196984	4.163475211367278	4.213004788632722	Oui
300 000	0.49942574835727305	4.177355956628762	4.205950710037904	Oui
700 000	0.4994426619436787	4.179451281449642	4.198171575693215	Oui
1 000 000	0.49946026160646656	4.177976463098011	4.193639536901989	Oui

On en déduit donc les intervalles de confiance ci-dessus. On remarque dans un premier temps que sur 30 simulations, on a que deux fois une solution qui n'est pas dans l'intervalle : la confiance à 95 pour-cent est donc bien présente.
Une autre remarque est que même pour N =1 000 000, l'intervalle de confiance reste assez large (≈ 0.02 d'étendu). On peut donc difficilement se fier à cette méthode pour obtenir une valeur "exacte". Il sera donc intéressant d'observer des méthodes plus optimisées que nous verrons par la suite.

2 Calculs d'intégrales

Une autre application de la méthode de Monte Carlo est le calcul d'intégrale.
Le principe est similaire à la méthode des rectangles, c'est-à-dire que l'on additionne plusieurs valeurs de la fonction dans l'intégrale puis on disive par le nombre de valeurs additionnées avant de multiplier par la taille de l'espace du domaine de l'intégrale. L'espérance de ce calcul est donc bien la valeur de l'intégrale, car en faisant tendre le nombre de points vers l'infini on n'a plus des rectangles mais bien l'aire sous la courbe.
On commence donc par calculer la valeur de cette intégrale :

$$I = \int_0^1 \int_0^1 \int_0^1 \int_0^1 e^{x+y+z+t} dxdydzdt$$

Pour calculer I, on peut donc utiliser une espérance :

$$I = \int_{u\in[0,1]^4} f(u)dX = \mathbb{E}(f(X))$$

où $X \sim U[0,1]^4$ de densité de probabilité $1_{[0,1]^4}(u)$. On pose $Y = f(X)$ et on a donc :

$$I = \mathbb{E}(Y) \approx \frac{1}{N}\sum_{i-1}^{N} Y_i = \frac{1}{N}\sum_{i-1}^{N} f(X_i) = \bar{Y}$$

Pour calculer I, on effectue donc N tirages de X dans un hypercube de dimension 4 et à chaque tirage on ajoute à un compteur la valeur de $f(X)$. A la fin en divisant la valeur du compteur par le nombre de tirages, nous obtenons la valeur de I. Il est inutile de multiplié par la taille de l'hypercube car il a une surface au carré qui vaut 1.
Afin de regarder la précision de la méthode, on calcule la solution exacte en utilisant les propriétés de l'intégrale et de l'exponentielle :

$$I = \int_0^1 \int_0^1 \int_0^1 \int_0^1 e^x e^y e^z e^t dxdydzdt = \left(\int_0^1 e^x dx\right)^4 = (1-e)^4 \approx 8.717211620141285$$

Ensuite on trace donc l'erreur en fonction du nombre de tirages N sur la figure 7.
On remarque que globalement, plus le nombre de tirage est grand, meilleure est la précision. C'est bien sûr dû à la loi des grands nombres. Encore une fois, on remarque que l'erreur décroît de manière exponentielle, avec seulement le premier point qui de part l'aléatoire est loin de sa valeur d'espérance.

Ensuite, on cherche à calculer la valeur de cette intégrale :

$$J = \int_0^1 \int_0^1 \int_0^1 \int_0^1 e^{xyzt} dxdydzdt$$

Mais cette fois-ci, on utilise une méthode différente, celle de la réduction **réduction de variance**. Le principe, comme le nom l'indique, est de réduire la variance. Pour cela, il faut trouver une fonction f qui suit à peu près la fonction dans l'intégrale à cet endroit. Mais il faut surtout être capable de calculer facilement la valeurs de l'intégrale de cette fonction f sur le même espace. Car le principe est de calculer avec la méthode de Monte Carlo l'intégrale de la fonction initiale moins la fonction f, et d'y ajouter la valeur exacte de l'intégrale de f. Ainsi on utilise la méthode de Monte Carlo sur des plus petites valeurs, et les erreurs dues au hasard sont donc plus petite comparées à la valeur exacte de l'intégrale de f. Ce qui réduit bien la variance.

Pour calculer J, on pose donc $\boxed{f(x,y,z,t) = xyzt}$ un Développement Limité à l'ordre 1 de e^{xyzt} (sans le +1 car il n'intervient pas dans la variance) et on a donc :

$$J = \int_0^1 \int_0^1 \int_0^1 \int_0^1 (e^{xyzt} - f(x,y,z,t))dxdydzdt + \int_0^1 \int_0^1 \int_0^1 \int_0^1 f(x,y,z,t)dxdydzdt$$

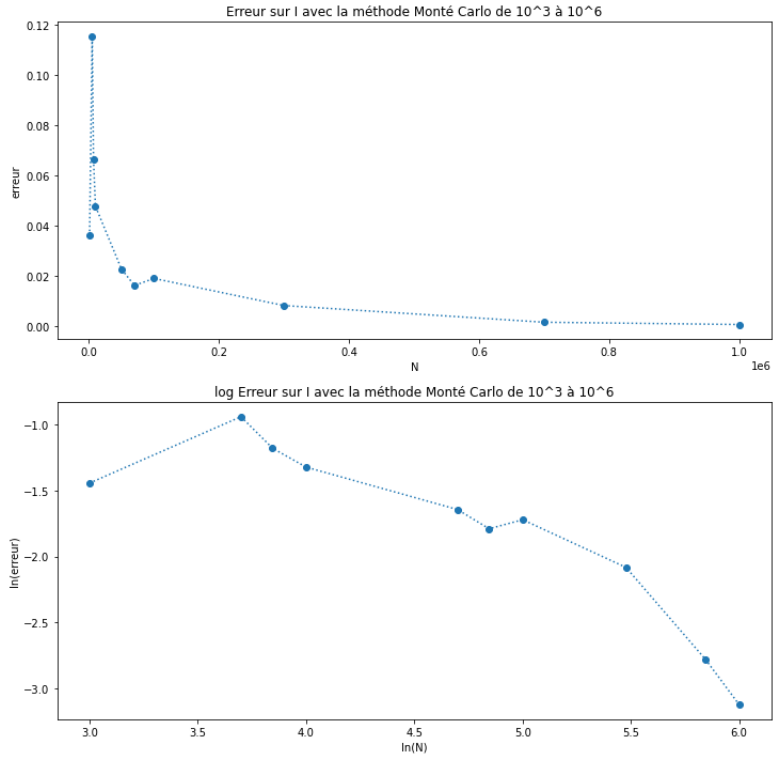

FIGURE 6 – Erreur de 10^3 à 10^6 sur I

Or on sait calculer la deuxième intégrale :

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 f(x,y,z,t)dxdydzdt = \frac{1}{16}$$

d'où

$$J = \int_0^1 \int_0^1 \int_0^1 \int_0^1 (e^{xyzt} - f(x,y,z,t))dxdydzdt + \frac{1}{16}$$

Avec cette méthode, on a donc forcément une variance réduite étant donné que sa formule est :

$$\sigma^2 = \mathbb{E}(Y^2) - \mathbb{E}(Y)^2 = \int_{u\in[0,1]^4} f^2(u)dX - \left(\int_{u\in[0,1]^4} f(u)dX\right)^2$$

f (la fonction dans la première intégrale cette fois) étant minimisé, il en va de même pour la variance.
On pourrait encore plus minimiser f en utilisant un Développement Limité d'ordre 2 avec $\boxed{g(x,y,z,t) = xyzt + \frac{(xyzt)^2}{2}}$ où on aurait donc

$$J = \int_0^1 \int_0^1 \int_0^1 \int_0^1 (e^{xyzt} - g(x,y,z,t))dxdydzdt + \frac{1}{162}$$

car

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 g(x,y,z,t)dydzdt = \frac{1}{162}$$

On prend donc N =100 000 pour calculer les différentes variances avec les différentes méthodes :

$N = 100000$	Monté Carlo simple	Réduction variance ordre 1	Réduction variance ordre 2
σ	0.012480010283972116	0.0005080948948434375	1.476515387758956e-05

On remarque que la variance de la méthode Monté Carlo simple est **25 fois** supérieure à celle avec réduction de variance d'ordre 1 et **850 fois** supérieure à celle avec réduction de variance d'ordre 2.
On compare donc la précision de ces méthodes de $N = 10^3$ à 10^6 en prenant comme solution de référence la valeur de J par la méthode réduction de variance d'ordre 2 avec 50 000 000 tirages (très longue simulation). On obtient les résultats ci-dessous figure 7 :

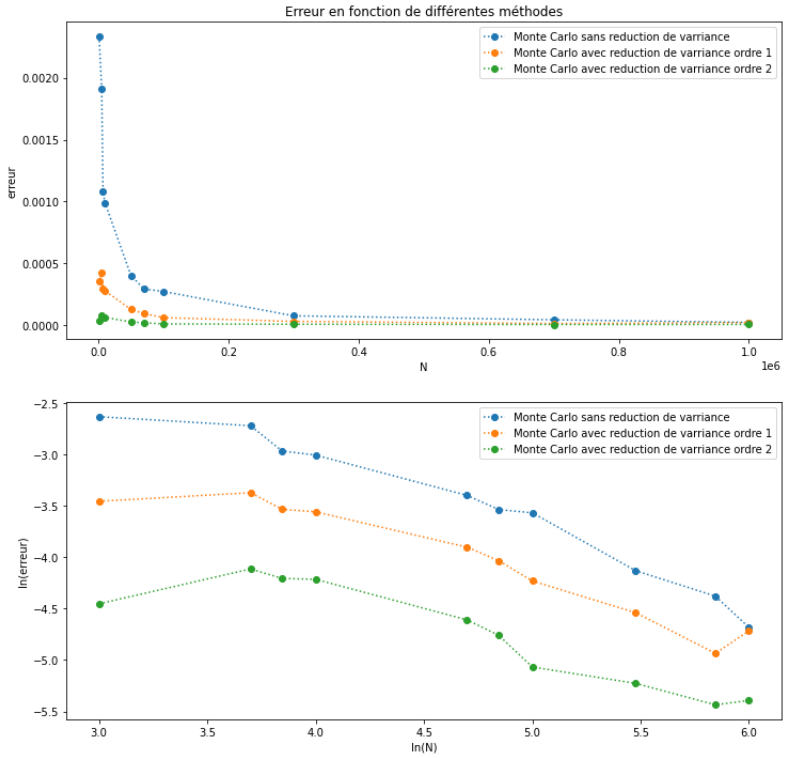

FIGURE 7 – Erreur avec la méthode de réduction de variance de 10^3 à 10^6 sur J

On remarque sans surprise que la méthode Monté Carlo avec réduction de variance d'ordre 2 est la plus précise puis celle d'ordre 1 puis la méthode simple.

3 Méthodes quasi-Monte Carlo et quantification

3.1 Cas uniforme

Le plus gros défaut de la méthode de Monte Carlo simple est qu'en tirant des points au hasard, ils ne sont pas forcément bien répartis. En effet, pour le calcul d'intégrales cela reviendrait à faire la méthode des rectangles mais avec des rectangles des endroits vides et d'autres où des rectangles se superposent.
Pour éviter cela, des méthodes dérivant de Monte carlo consistent à faire en sorte que les point se répartissent bien.
Pour les tester, on choisit de calculer la valeur de l'intégrale :

$$I = \int_0^1 \int_0^1 e^{x+y} dxdy$$

qui a pour valeur exacte $I = (1-e)^2$.

3.1.1 Monte-Carlo / Quasi-Monte-Carlo

Pour calculer I, on utilise donc la méthode dite quasi-Monte Carlo ainsi que celle de Monte Carlo simple pour pouvoir les comparer.
Les points générés par Monte Carlo sont aléatoires tandis que ceux générés par quasi-Monte Carlo suivent la suite : $(n\sqrt{2} - |n\sqrt{2}|, n\sqrt{3} - |n\sqrt{3}|)$ avec $n \in \mathbb{N}$ l'itération de la simulation.

On a tracé la répartition des points pour$N$ =10, 100 avec ces deux méthodes et nous avons obtenu les figure 8 et 9 suivantes :

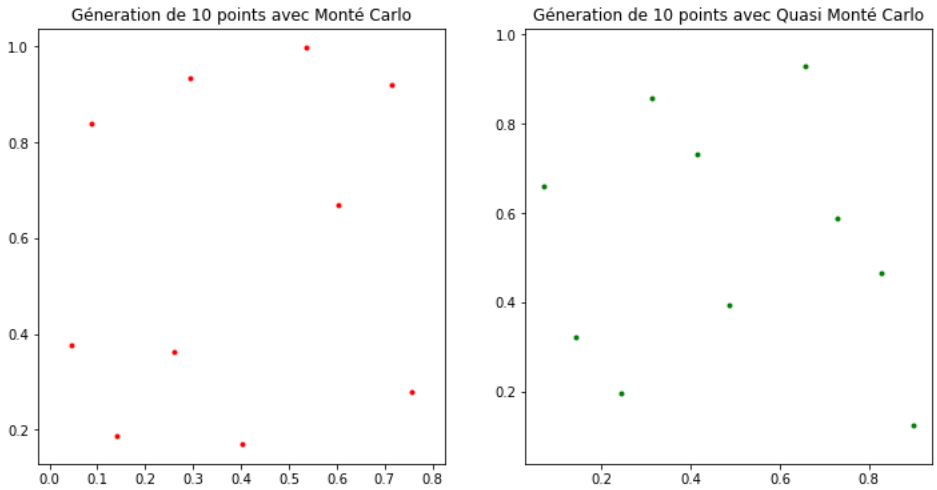

FIGURE 8 – Répartition des points pour $N = 10$ avec quasi et Monté-Carlo

On voit bien qu'avec la méthode quasi-Monte Carlo les points sont plus uniformément réparti, même s'ils restent par petits paquets. Mais en augmentant n les paquets se rejoignent et on a quelque chose d'encore plus uniforme.

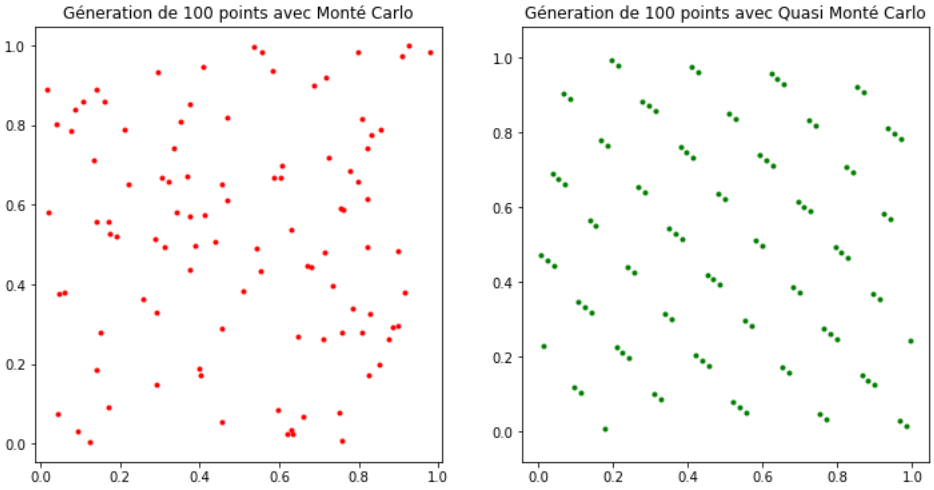

FIGURE 9 – Répartition des points pour $N = 100$ avec quasi et Monté-Carlo

On calcule donc I avec ces deux méthodes et on tracel'erreur pour N allant de 10^3 à 10^6. On obtient la figure 10 suivante :

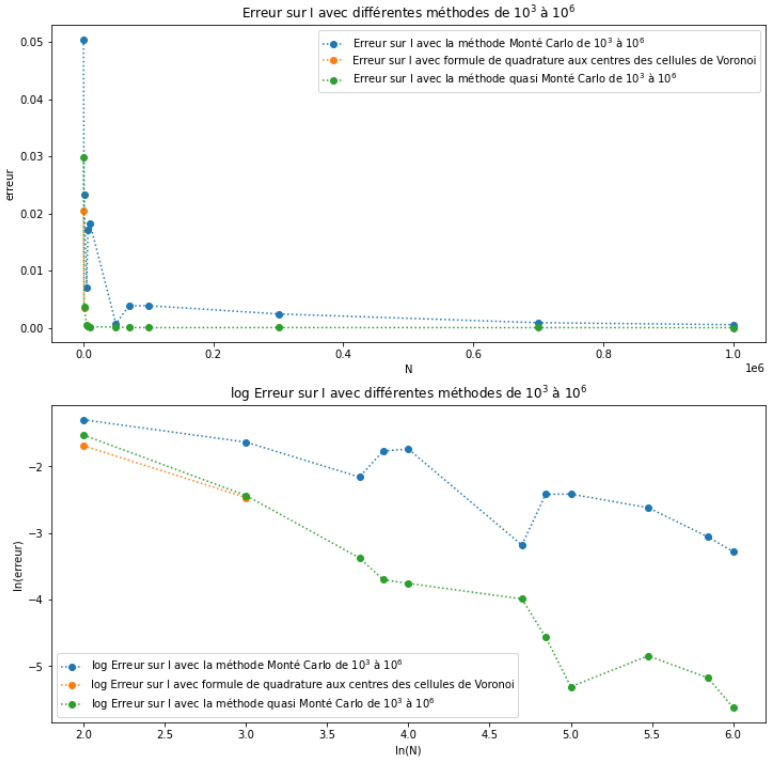

FIGURE 10 – Erreur avec avec quasi et Monté-Carlo

On remarque que dans les deux cas l'erreur décroît de manière exponentielle, mais qu'avec la méthode quasi-Monte Carlo le coefficient affine du logarithme de l'erreur a une valeur absolue plus grande et donc décroit bien plus vite.

3.1.2 Algorithmes de quantification

Dans cette méthode quasi-Monte Carlo, les points sont plus uniformément répartis, mais comme expliqué précédemment ils restent en paquets. Pour avoir des résultats encore plus précis, il faudrait donc essayer d'obtenir une répartition qui se rapproche plus d'une vrai uniformité. C'est le principe des méthodes de quantification.

3.1.2.1 Kohonen

Le premier algorithme de quantification utilisé est celui de Kohonen. On trouve l'idée imagée de l'algorithme dans la référence [1]. Le principe est le suivant :
On tire n sources selon une densité $w(x)$. A l'étape i de l'algorithme, on dispose de ces n positions notées $M_j(x_j, y_j)$ et on tire un point $P_i(x_a, y_a)$ au hasard toujours selon $w(x)$. Ensuite on calcul l'indice j_0 de la source la plus proche de P_i selon :

$$j_0 = \text{indice}(\min(d^2(M_j, P_n))) = \text{indice}(\min((x_j - x_a)^2 + (y_j - y_a)^2)))$$

Puis on replace la source j_0 selon :

$$(1 - \varepsilon_i)M_{j0} + \varepsilon_i P_i \text{ soit les coordonnées : } [(1 - \epsilon_i)x_{j0} + \varepsilon_i x_a, (1 - \varepsilon_i)y_{j0} + \varepsilon_i y_a]$$

Avec ici $\varepsilon = 0,01 - \frac{i}{n}(0,01 - 0,0001)$
Ainsi, les sources placées au début aléatoirement vont petit à petit aller se placer à équidistance de leurs voisins, et cela va donc tendre vers leur répartition uniforme.

On trace donc dans un premier temps la répartition de 10 points sur un domaine $D = [0, 1]^2$ et on obtient la figure 11 avec un nombre N d'itérations différentes.

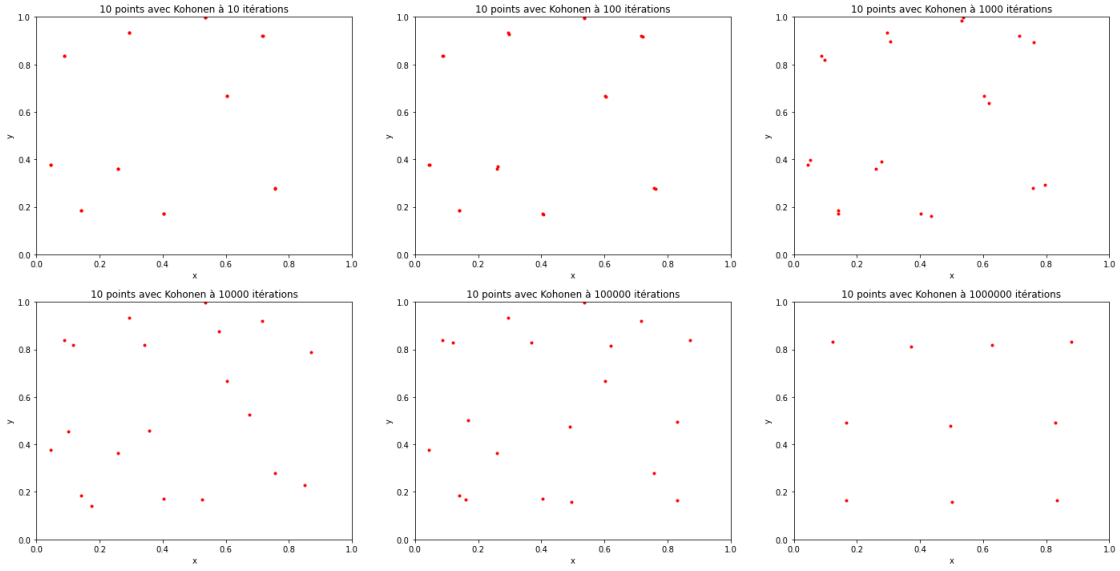

FIGURE 11 – Kohonen avec 10 points pour N itérations

On remarque qu'il y a convergence seulement à partir de 1 000 000 d'itérations dans notre cas.

De même, on trace la répartition de 100 points sur D et on obtient la figure 12 avec un nombre N d'itérations différentes.

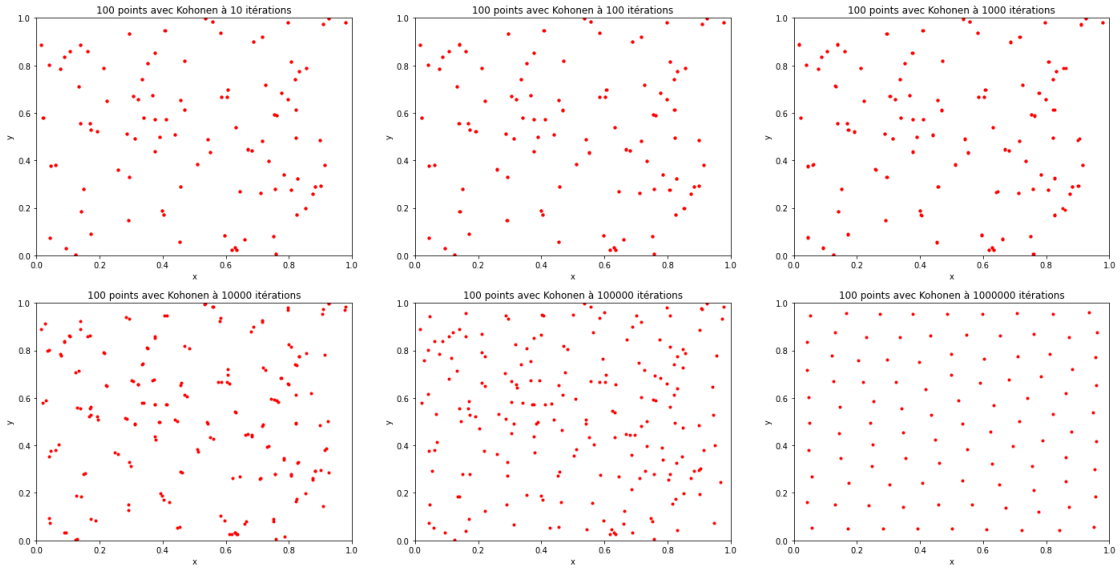

FIGURE 12 – Kohonen avec 100 points pour N itérations

Dans ce cas la répartition n'est à peu près uniforme que seulement à partir de 1 000 000 d'itérations.

3.1.2.2 K-means

Le principe de K-means est très similaire à celui de Kohonen sauf qu'à chaque itération, on génère un nombre Ns (avec $Ns > N$) de points aléatoires selon $w(x)$, on les regroupe sous forme de cluster (ou cellules de Voronoi C_j) comme on peut voir figure 13, un point tiré au hasard va dans le cluster de sa source la plus proche. Les sources initiales sont ensuite toutes modifiés simultanément en utilisant les coordonnées du barycentre du cluster associé.
Cela permet que chaque modification de la localisation de la source ait la même importance (chaque point de la cellule a la même importance). Alors qu'avec la méthode de Kohonen, chaque modification est plus importante que la suivante, ce qui fait tendre moins vite vers une répartition uniforme des sources.
Sur la figure 13 ci-dessous on a tracé différents cluster avec leur baricentre à l'itération 50 pour un nombre de points initial à 10 en simulant 500 nouveaux points par source.

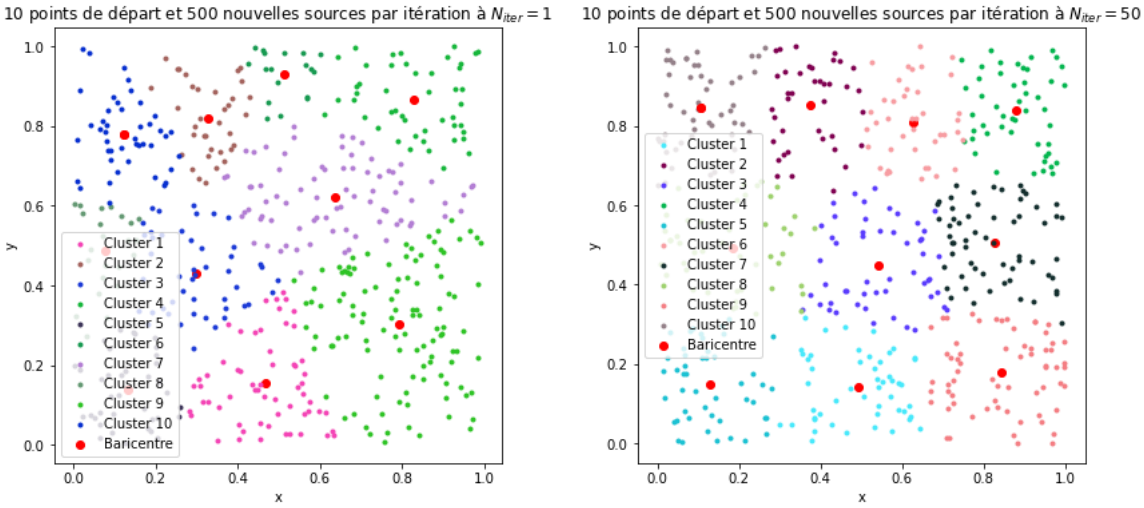

FIGURE 13 – Kmean avec 10 points pour 1 et 50 itérations et $Ns = 500$ nouvelles sources

On trace figure 14 la répartition de 10 points sur un domaine $D = [0, 1]^2$ à 50 itérations où à chaque itérations on simule Ns points.

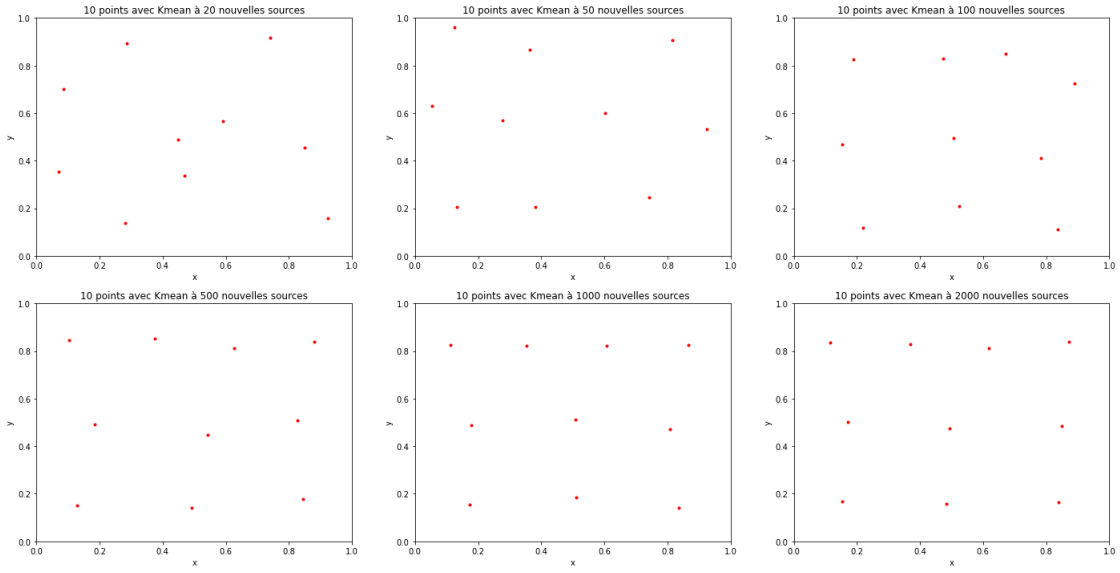

FIGURE 14 – Kmean avec 10 points pour 50 itérations et Ns sources

On remarque que contrairement à la méthode kohonen, pour 10 points, on obtient bien plus vite une répartition à peu près uniforme : à partir de 500 nouveaux points par itérations.
On trace ensuite figure 15 la répartition de 100 points sur un domaine D à 50 itérations où à chaque itération on simule Ns points.

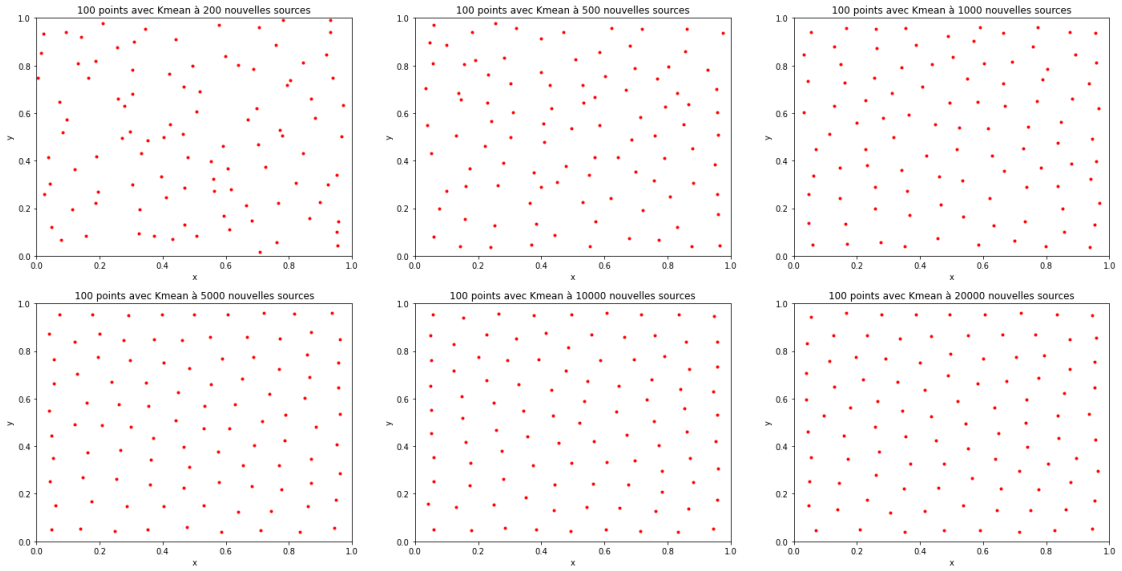

FIGURE 15 – Kmean avec 100 points pour 50 itérations et Ns sources

On remarque de même pour 100 points on obtient rapidement une répartition uniforme : à partir de 5000 nouveaux points par itérations.

On peut ensuite approximer I comme une formule de quadrature aux centres des cellules de Voronoi et comme poids la taille des cellules. Ainsi on prend plus en compte les sources qui sont plus près d'être à leur position pour la répartition uniforme. C'est donc la méthode qui répartie le plus uniformément depuis le début. On a donc tracé son erreur en orange sur la figure 10 pour la comparer aux méthodes de Monte Carlo simple et de quasi-Monte Carlo.
On remarque qu'elle n'est pas beaucoup plus précise que la méthode quasi-Monte Carlo. C'est dû au fait que même si elle répartit mieux les points, ça ne change pas grand chose par rapport à un cas où les points sont juste à peu près uniformément répartis. De plus, cette méthode est très coûteuse en temps de calcul par rapport à la méthode quasi-Monté Carlo qui est très rapide. On conclue donc que la plupart du temps la méthode quasi-Monte Carlo est préférable. Néanmoins, pour des fonctions extrêmement irrégulières dans l'intégrale les méthodes de quantifications peuvent être préférables car elles répartissent bien mieux les points.

3.2 Cas Gaussien

Toutes les méthodes utilisées jusqu'à présent étaient appliquées à des lois uniformes, mais on peeut en fait les appliquer à d'autres lois. Par exemple, les lois gaussiennes que l'on retrouve souvent ne posent aucun problème à ses méthodes. Pour les calculer, on utilise habituellement un changement de variables pour pouvoir tirer des points selon une loi uniforme qui vont étre transformés en points répartis selon une loi gaussienne.
On essaye donc les méthodes quasi-Monte Carlo et de quantification sur une loi gaussienne pour comparer avec une loi uniforme. L'intégrale à calculer choisie est :

$$J = \frac{1}{2\pi} \int_{\mathbb{R}^2} e^{-\frac{x^2+y^2}{2}} f(x,y)dxdy \text{ avec } f(x,y) = e^{x+y}$$

3.2.1 Calcul de J

Pour calculer J, on cherche donc à ramener cette gaussienne à un cas plus simple par changement de variables :

On pose donc $x = r\cos(\theta)$, $y = r\sin(\theta)$, et on trouve la Jacobienne :

$$J1 = \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial \theta} & \frac{\partial y}{\partial \theta} \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -r.\sin(\theta) & r.\cos(\theta) \end{pmatrix}$$

ce qui donne $|\det(J1)|=|r.\cos^2(\theta) + r.\sin^2(\theta)| = r$ et donc

$$J = \frac{1}{2\pi} \int_{r\in\mathbb{R}} \int_{\theta\in[0,2\pi]} e^{\frac{-r^2}{2}} f(r\cos(\theta), r\sin(\theta))rdrd\theta$$

De même, on pose $r = \sqrt{-2\ln(u)}$, $\theta = 2\pi v$, et on trouve la Jacobienne :

$$J2 = \begin{pmatrix} \frac{-1}{u\sqrt{-2ln(u)}} & 0 \\ -0 & 2\pi \end{pmatrix}$$

donc $|\det(J2)|=\frac{2\pi}{u\sqrt{-2ln(u)}}$ et :

$$J = \int_{(u,v)\in[0,1]^2} uf\left(\sqrt{(-2\ln(x))}\cos(2\pi y), \sqrt{(-2\ln(x))}\sin(2\pi y)\right) \frac{1}{u\sqrt{-2ln(u)}}\sqrt{-2\ln(u)}dudv$$

On a alors :

$$\frac{1}{2\pi} \int_{\mathbb{R}^2} e^{-\frac{x^2+y^2}{2}} f(x,y)dxdy = \int_0^1 \int_0^1 f\left(\sqrt{(-2\ln(x))}\cos(2\pi y), \sqrt{(-2\ln(x))}\sin(2\pi y)\right) dxdy \quad (1)$$

quelque soit f continue positive et bornée sur $\mathbb{R}^2$.
Pour calculer J, on utilise donc (1) en utilisant simplement la méthode expliquée partie 2.

3.2.2 Comparaison des méthodes pour calculer J

Comme dans le cas uniforme partie 3.1, on trace les erreurs en fonction des différentes méthodes avec un nombre d'itérations variable de $N = 10^3$ à 10^6.
Pour tracer les erreurs, il faut la valeur exacte de J :

On a :

$$\begin{aligned} J = \frac{1}{2\pi}\int_{\mathbb{R}}\int_{\mathbb{R}} e^{-\frac{x^2+y^2}{2}} e^{x+y} dxdy &= \frac{1}{2\pi}\int_{\mathbb{R}}\int_{\mathbb{R}} e^{-\frac{x^2-2x+y^2-2y}{2}} dxdy \\ &= \frac{e}{2\pi}\int_{\mathbb{R}}\int_{\mathbb{R}} e^{-\frac{(x^2-1)^2+(y^2-1)^2}{2}} dxdy \\ &= \frac{e}{2\pi}\int_{\mathbb{R}} e^{-\frac{(x^2-1)^2}{2}} dx \int_{\mathbb{R}} e^{-\frac{(y^2-1)^2}{2}} dy \\ &= \frac{e}{2\pi}\sqrt{2\pi}\sqrt{2\pi} = e \end{aligned}$$

On obtient donc $J = e$. On trace ci-dessous figure 16 la comparaison des méthodes :

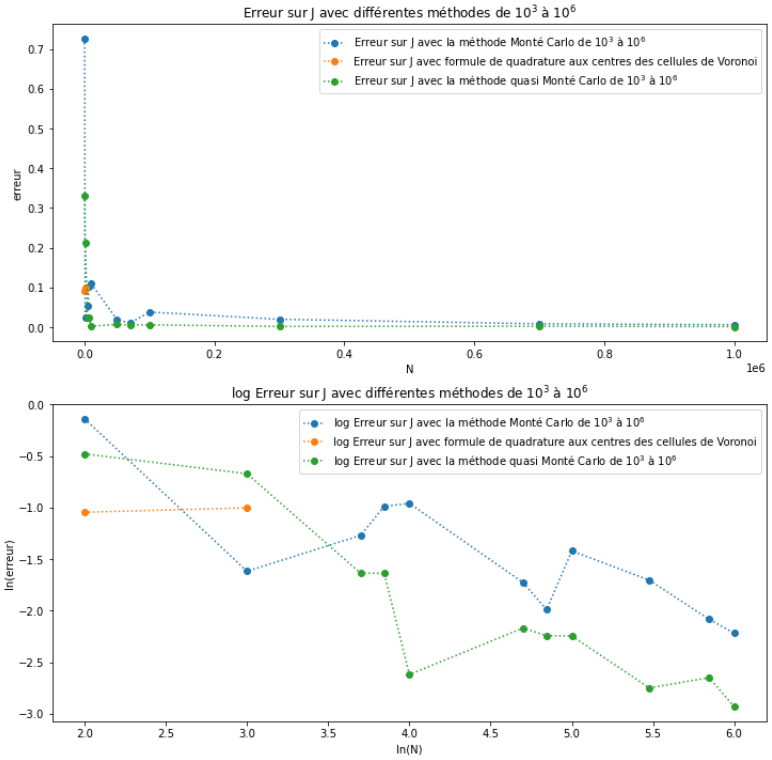

FIGURE 16 – Erreur avec avec quasi et Monté-Carlo

Comme précédemment, on trouve que la méthode quasi-monte Carlo est la plus optimisée car elle a un temps de calcul moindre et une précision quasiment identique au calcul de J en prenant en compte le poids des cellules. ET comme précédemment les erreurs décroissent encore de manière exponentielle.

4 Optimisation par un algorithme mimétique

Une autre utilisation de la méthode de Monte Carlo que celles du calcul d'aires et d'intégrales est de trouver le maximum ou le minimum d'une fonction.
Pour cela, le principe est de placer aléatoirement, de manière uniforme, un nombre points N. Puis aléatoirement, on choisit deux points, et on compare la valeur de la fonction à ces deux points. Quand on cherche le minimum de la fonction, on déplace le point où la fonction est la plus grande pour le placer près de l'autre. Et inversement si on cherche le maximum de la fonction. On répète ce procédé un grand nombre de fois. Ainsi, si on place assez de points au départ, on en a à proximité de chaque extremum, et cela revient à comparer les valeurs de ces extremum pour trouver le plus petit ou le plus grand.
Quand on déplace un point pour le placer près de celui avec lequel on vient de le comparer, on lui affecte cette nouvelle coordonnée $(x + \alpha_n X, y + \alpha_n Y)$ où X, Y sont des gaussiennes $N(0, 1)$ et $\alpha_n = p^n$. Cela permet de le placer proche de l'autre, mais un peu autour pour être sûr d'en avoir un vraiment proche de l'extremum. Et afin d'avoir convergence, on prend un p très proche de 1 comme par exemple $p = 0.999$.

A la fin des itérations, tous les points sont logiquement concentrés vers l'extremum recherché.

Pour appliquer cette méthode, on commence par chercher le maximum de cette fonction :

$$f(x, y) = a.e^{-b((x-1)^2+(y-2)^2)} + c.e^{-d((x+1)^2+(y+3)^2)}$$

sur le domaine $D = [-5, 5]^2$ avec dans un premier temps $a = 1, b = 1, c = 0.8, d = 1$.

En traçant sur matlab cette fonction on obtient la figure 17 à gauche :

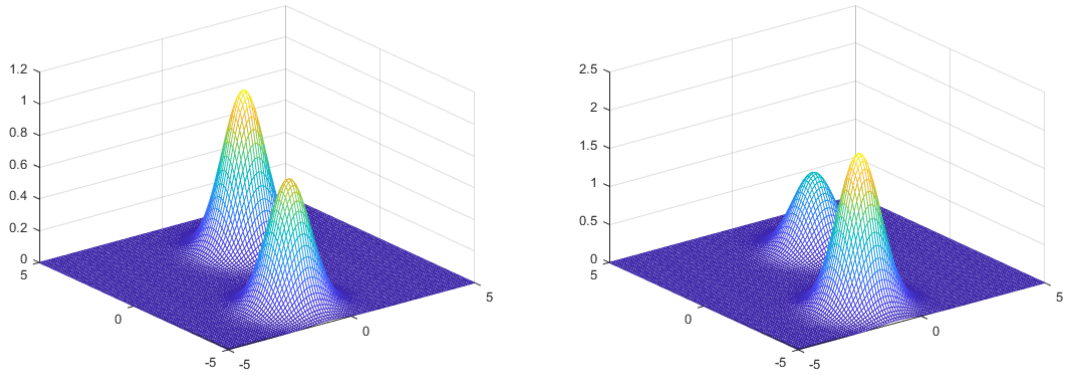

FIGURE 17 – Tracé de f avec les paramètres de l'énoncé et avec $c = 2$

On voit que le maximum est vers (1,2).

Et en prenant $p = 0.99$, $N = 100$ et N_{iter} =10 000, on trouve bien que le maximum est en (1,2).

En changeant les coefficients a et c, on remarque que cela modifie seulement la hauteur des Gaussiennes. On prend donc un $c = 0.99$ très proche de a et on remarque que l'algorithme peut se tromper et renvoyer les coordonnées de la Gaussienne en (-1,-3). Néanmoins avec les paramètres $p = 0.99$, $N = 100$ et N_{iter} =10 000, l'algorithme arrive toujours à trouver le bon maximum.

Ensuite, on cherche à calculer le minimum de la fonction :

$$f(x, y) = 20 + x^2 + y^2 - 10(\cos(2\pi x) + \cos(2\pi y))$$

sur le domaine $[-5.12, 5.12]^2$.

On trace sur matlab cette fonction et on obtient la figure 18 :

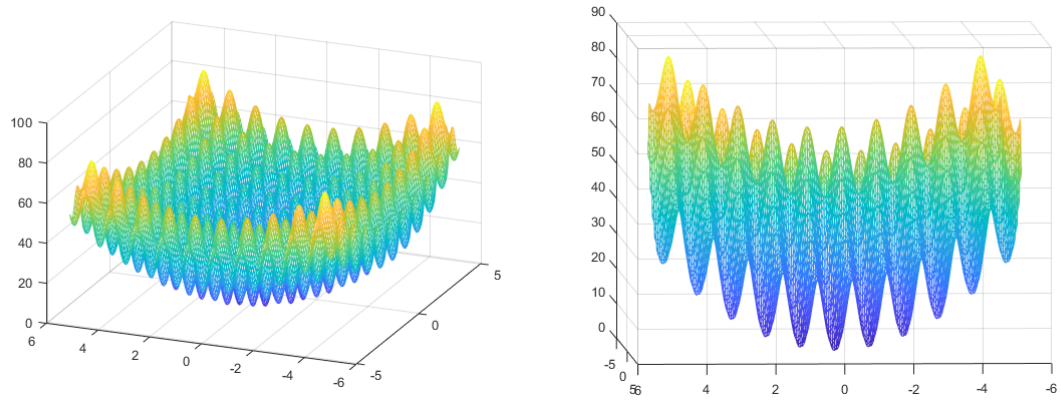

FIGURE 18 – Tracé de f

La fonction a beaucoup d'oscillations, donc l'algorithme peut "trouver" beaucoup de minimum locaux. Ainsi il est important de prendre un p très proche de 1 afin d'avoir le bon minimum.

Analytiquement on trouve le minimum en (0,0). Et algorithmiquement on trouve le même minimum en prenant comme paramètres $p = 0.999$, $N = 1000$ et N_{iter} =100 000.

Ainsi, pour trouver le bon extremum, il suffit juste de prendre un bon p, de placer assez de points de départ, et de faire assez d'itérations. Plus il y a d'extremums locaux, ou plus la dérivé autour est grande, et plus il faut avoir de points de départ et faire d'itérations.

5 Conclusion

Ainsi, la méthode de Monte Carlo peut être appliquée à de nombreux types de problèmes. Mais dans tous les cas l'avantage principal est le même : pouvoir résoudre un problème que l'on ne sait pas résoudre analytiquement en utilisant le hasard, notament des calculs d'aire, d'intégrale ou d'extremum.
L'inconvénient de cette méthode est que justement en utilisant le hasard, on n'obtient pas des points répartis uniformément, mais cela peut être amélioré grâce à certaines méthodes (quasi-Monte Carlo, Kohonen, k-means). Mais il faut savoir quelle méthode utiliser selon le problème, car certaines sont plus rapides, mais d'autres plus précises. Ainsi, en sachant une fonction "peu variante" on peut choisir la méthode rapide de quasi-Monte Carlo, mais dans le cas où on sait la fonction très variante la méthode k-means est préférable.
Durant ce travail ces améliorations de la méthode de Monte Carlo n'ont été utilisées que pour du calcul d'intégrale, mais le même principe peut être utilisé pour le calcul d'aire ou d'extremum.

Références

[1] Maire Sylvain. *Cours sur les méthodes Monte Carlo*, 2020.

6 Annexe

6.1 Partie 1

6.1.1 Calcul air d'un cercle

```
 program cercle
implicit none

integer                         :: N,i, j, k
real ( kind = 8)                :: ix,na,nmax,a,x,y,z,t,c
ix=51477.d0
na=16807.d0
nmax=2147483647.d0 ! nombre premier tres grand

N=100000
c=0.d0
DO i=1,N
  CALL rd(x) !Aléatoire dans [0,1]
  x=2*x-1 !Aléatoire dans [-1,1]
  CALL rd(y) !Aléatoire dans [0,1]
  y=2*y-1 !Aléatoire dans [-1,1]

  IF ((x**2+y**2)<=1) THEN !Test pour savoir si le point est dans C
    c=c+1
  ENDIF
ENDDO

print*,c*4/N

CONTAINS
SUBROUTINE rd(u)
real ( kind = 8)                :: u
ix=abs(ix*na)
ix=mod(ix,nmax)
u=dble(ix)/dble(nmax)
RETURN
END SUBROUTINE rd
! ============ FIN DU PROGRAMME =============
end program  cercle
```

6.1.2 Calcul air d'une ellipse

```
N=100000
c=0.d0
DO i=1,N
        CALL rd(x) !Aléatoire dans [0,1]
        x=4*x-2 !Aléatoire dans [-2,2]

        CALL rd(y) !Aléatoire dans [0,1]
        y=2*y-1 !Aléatoire dans [-1,1]

        IF (((x/2)**2+y**2)<=1) THEN !Test pour savoir si le point est dans l'éllipse
```

```
                c=c+1
        ENDIF
ENDDO
print*,c*8/N
```

6.1.3 Calcul air d'une sphère

```
N=100000
c=0.d0
DO i=1,N
        CALL rd(x) !Aléatoire dans [0,1]
        x=2*x-1 !Aléatoire dans [-1,1]

        CALL rd(y) !Aléatoire dans [0,1]
        y=2*y-1 !Aléatoire dans [-1,1]

        CALL rd(z) !Aléatoire dans [0,1]
        z=2*z-1 !Aléatoire dans [-1,1]

        IF ((x**2+y**2+z**2)<=1) THEN !Test pour savoir si le point est dans C
                c=c+1
        ENDIF
ENDDO
print*,c*8/N
```

6.2 Partie 2

6.3 Calcul intégrales

6.3.1 Calcul de I

```
N=1000000
c=0.d0
DO i=1,N
  CALL rd(x)
  CALL rd(y)
  CALL rd(z)
  CALL rd(t)
  c=c+dexp(x+y+z+t)
ENDDO
print*,c/N
```

6.3.2 Calcul de J par Monte Carlo

```
N=100000
c=0.d0
DO i=1,N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
```

```
        y=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        z=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        t=(dble(ix)/dble(nmax))

        c=c+dexp(x*y*z*t)
ENDDO

print*,c/N
```

6.3.3 Calcul de J avec des variables de contrôle d'ordre 1

```
ix=51477.d0
na=16807.d0
nmax=2147483647.d0

N=100000
c=0.d0
DO i=1,N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        z=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        t=(dble(ix)/dble(nmax))

        c=c+exp(x*y*z*t)-x*y*z*t
ENDDO

print*,c/N+1/16
```

6.3.4 Calcul de J avec des variables de contrôle d'ordre 2

```
ix=51477.d0
na=16807.d0
nmax=2147483647.d0

N=100000
```

```
c=0.d0
DO i=1,N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        z=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        t=(dble(ix)/dble(nmax))

        c=c+exp(x*y*z*t)-x*y*z*t-(x*y*z*t)**2/2
ENDDO

print*,c/N+1/16+1/162
```

6.4 Partie 3

6.4.1 quasi-Monte Carlo

```
c=0
N=10000

do i=1,N
        x=i*(2**(1/2.d0))-int(i*(2**(1/2.d0))) !obtenir des x et des y mieux
        y=i*(3**(1/2.d0))-int(i*(3**(1/2.d0))) ! répartis sur le domaine

        c=c+dexp(x+y)

enddo

print*,c/N
```

6.4.2 Kohonen

```
ix=51477.d0
na=16807.d0
nmax=2147483647.d0

N=100
c=0.d0
DO i=1,N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax)) ! on place des sources au hasard sur le domaine
```

```
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        Un(i,1)=x
        Un(i,2)=y
ENDDO

ix=51477.d0
na=16807.d0
nmax=2147483647.d0

DO i=1,N*N

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax)) ! on tire des points au hasard sur le domaine

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        k=1

        DO j=2,N

                if ((Un(j,1)-x)**2+(Un(j,2)-y)**2<(Un(k,1)-x)**2+(Un(k,2)-y)**2) then

                        k=j ! on cherche la source la plus proche du point tiré

                endif
        enddo

        epsilon=0.01-i*(0.01-0.0001)/N

        Un(k,1)=(1-epsilon)*Un(k,1)+epsilon*x ! on déplace un peu la source
        Un(k,2)=(1-epsilon)*Un(k,2)+epsilon*y ! vers le point tiré

ENDDO

c=0

do i=1,N

        c=c+dexp(Un(i,1)+Un(i,2))

enddo

OPEN ( UNIT =48 , FILE = 'S.dat')
DO i =1,N
        WRITE (48 ,*) Un(i,1), Un (i ,2)
```

```
ENDDO
CLOSE (48)

print*,c/N

print*,(1.d0-dexp(1.d0))**2
```

6.4.3 k-means

```
ix=51477.d0
na=16807.d0
nmax=2147483647.d0

N=100
c=0.d0
DO i=1,N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax)) ! on place des sources au hasard sur le domaine

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        Un(i,1)=x
        Un(i,2)=y
ENDDO

ix=51477.d0
na=16807.d0
nmax=2147483647.d0
do l=1,50
DO i=1,N

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax)) ! on tire des points au hasard dans le domaine

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        Wn(i,1)=x ! on enregistre ces points
        Wn(i,2)=y

        k=1

        DO j=2,N

                if ((Un(j,1)-x)**2+(Un(j,2)-y)**2<(Un(k,1)-x)**2+(Un(k,2)-y)**2) then

                        k=j ! on cherche la source la plus proche du point choisi
```

```
                endif
        enddo

        Wn(i,3)=k ! on enregistre chaque point dans la cellule de Voronoi
                          ! de sa source la plus proche

ENDDO

do i=1,N

        k=0
        x=0
        y=0

        do j=1,N

                if (Wn(j,3)==i) then

                        k=k+1
                        x=x+Wn(j,1)
                        y=y+Wn(j,2) ! on place chaque source au barycentre de sa cellule
                        ! de Voronoi
                endif
        enddo

        if (k>0) then

                Un(i,1)=x/k
                Un(i,2)=y/k
        endif

enddo
enddo
c=0

do i=1,N

        c=c+dexp(Un(i,1)+Un(i,2))

enddo

print*,c/N
```

6.4.4 k-means avec quadrature

```
ix=51477.d0
na=16807.d0
nmax=2147483647.d0

N=100
c=0.d0
```

```
DO i=1,N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        Un(i,1)=x
        Un(i,2)=y
ENDDO

ix=51477.d0
na=16807.d0
nmax=2147483647.d0

do i=1,N
        Wn(i,4)=0
enddo

do l=1,50
DO i=1,N

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        Wn(i,1)=x
        Wn(i,2)=y

        k=1

        DO j=2,N

                if ((Un(j,1)-x)**2+(Un(j,2)-y)**2<(Un(k,1)-x)**2+(Un(k,2)-y)**2) then

                        k=j

                endif
        enddo

        Wn(i,3)=k

ENDDO

do i=1,N
```

```
        k=0
        x=0
        y=0

        do j=1,N

                if (Wn(j,3)==i) then

                        k=k+1
                        x=x+Wn(j,1)
                        y=y+Wn(j,2)

                endif
        enddo

        Wn(i,4)=Wn(i,4)+k ! on enregistre la taille de chaque cellule de Voronoi

        if (k>0) then

                Un(i,1)=x/k
                Un(i,2)=y/k
        endif

enddo
enddo
c=0

do i=1,N

        c=c+Wn(i,4)*dexp(Un(i,1)+Un(i,2)) ! Plus une cellule de Voronoi est grande plus
                                                                    ! elle est lourde

enddo

k=0
do i=1,N
        k=k+Wn(i,4)
enddo

print*,c/k ! on ne divise pas par le nombre de sources mais par la somme des "poids"
                ! des cellules de Voronoi
```

6.4.5 Calcul Gaussienne bi-dimensionnel

```
N=1000000
c=0.d0
DO i=1,N
  CALL rd(u1)
  CALL rd(u2)
  G1=(-2.d0*dlog(U1))**0.5*dcos(3.14159265*2.d0*U2)
  G2=(-2.d0*dlog(U1))**0.5*dsin(3.14159265*2.d0*U2)
```

```
  c=c+dexp(G1+G2)
ENDDO
print*,c/N
```

6.4.6 Kohonen

6.4.7 Kmean

6.4.8 Quadrature

6.5 Partie 4

6.5.1 CEP pour le maximum

```
ix=51477.d0
na=16807.d0
nmax=2147483647.d0

N=100
c=0.d0

DO i=1,N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))
        x=x-0.5
        x=x*10

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax)) ! on place des sources aléatoirement
        y=y-0.5                 ! sur le domaine
        y=y*10

        Un(i,1)=x
        Un(i,2)=y
ENDDO

ix=51477.d0
na=16807.d0
nmax=2147483647.d0

DO i=1,N*N*N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))! on choisit aléatoirement deux des sources
        j=int(x*N)

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))
        k=int(y*N)

if (dexp(-((Un(j,1)-1)**2+(Un(j,2)-2)**2))+0.8*dexp(-((Un(j,1)+1)**2+&
```

```
&(Un(j,2)+3)**2))<dexp(-((Un(k,1)-1)**2+(Un(k,2)-2)**2))+0.8*&
&dexp(-((Un(k,1)+1)**2+(Un(k,2)+3)**2))) then ! on trouve le maximum entre les deux
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        z=(-2*dlog(x))**(1/2.d0)*dcos(2*3.1415*y)
        t=(-2*dlog(x))**(1/2.d0)*dsin(2*3.1415*y)

        Un(j,1)=Un(k,1)+0.999**i*z ! on déplace le minimum des deux vers le maximum
        Un(j,2)=Un(k,2)+0.999**i*t

endif

if (dexp(-((Un(j,1)-1)**2+(Un(j,2)-2)**2))+0.8*dexp(-((Un(j,1)+1)**2&
&+(Un(j,2)+3)**2))>dexp(-((Un(k,1)-1)**2+(Un(k,2)-2)**2))+0.8*dexp&
&(-((Un(k,1)+1)**2+(Un(k,2)+3)**2))) then
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        z=(-2*dlog(x))**(1/2.d0)*dcos(2*3.1415*y)
        t=(-2*dlog(x))**(1/2.d0)*dsin(2*3.1415*y)

        Un(k,1)=Un(j,1)+0.999**i*z
        Un(k,2)=Un(j,2)+0.999**i*t

endif


ENDDO


print*,Un(1,1),Un(1,2)
```

6.5.2 CEP pour le minimum

```
ix=51477.d0
na=16807.d0
nmax=2147483647.d0

N=100
c=0.d0
```

```
DO i=1,N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))
        x=x-0.5
        x=x*10

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax)) ! on place des sources aléatoirement sur le domaine
        y=y-0.5
        y=y*10

        Un(i,1)=x
        Un(i,2)=y
ENDDO

ix=51477.d0
na=16807.d0
nmax=2147483647.d0

DO i=1,N*N*N
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax)) ! on choisit aléatoirement deux sources
        j=int(x*N)

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))
        k=int(y*N)

if (20+Un(j,1)**2+Un(j,2)**2-10*(dcos(2*3.1415*Un(j,1))+dcos(2*3.1415*Un(j,2)))>&
&(20+Un(k,1)**2+Un(k,2)**2-10*(dcos(2*3.1415*Un(k,1))+dcos(2*3.1415*Un(k,2))))) then
        ! on compare ces deux sources
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        z=(-2*dlog(x))**(1/2.d0)*dcos(2*3.1415*y)
        t=(-2*dlog(x))**(1/2.d0)*dsin(2*3.1415*y)

        Un(j,1)=Un(k,1)+0.999**i*z
        Un(j,2)=Un(k,2)+0.999**i*t ! on déplace le maximum des deux vers le minimum

endif
```

```
if (20+Un(j,1)**2+Un(j,2)**2-10*(dcos(2*3.1415*Un(j,1))+dcos(2*3.1415*Un(j,2)))<&
&(20+Un(k,1)**2+Un(k,2)**2-10*(dcos(2*3.1415*Un(k,1))+dcos(2*3.1415*Un(k,2))))) then
        ix=abs(ix*na)
        ix=mod(ix,nmax)
        x=(dble(ix)/dble(nmax))

        ix=abs(ix*na)
        ix=mod(ix,nmax)
        y=(dble(ix)/dble(nmax))

        z=(-2*dlog(x))**(1/2.d0)*dcos(2*3.1415*y)
        t=(-2*dlog(x))**(1/2.d0)*dsin(2*3.1415*y)

        Un(k,1)=Un(j,1)+0.999**i*z
        Un(k,2)=Un(j,2)+0.999**i*t

endif


ENDDO
```