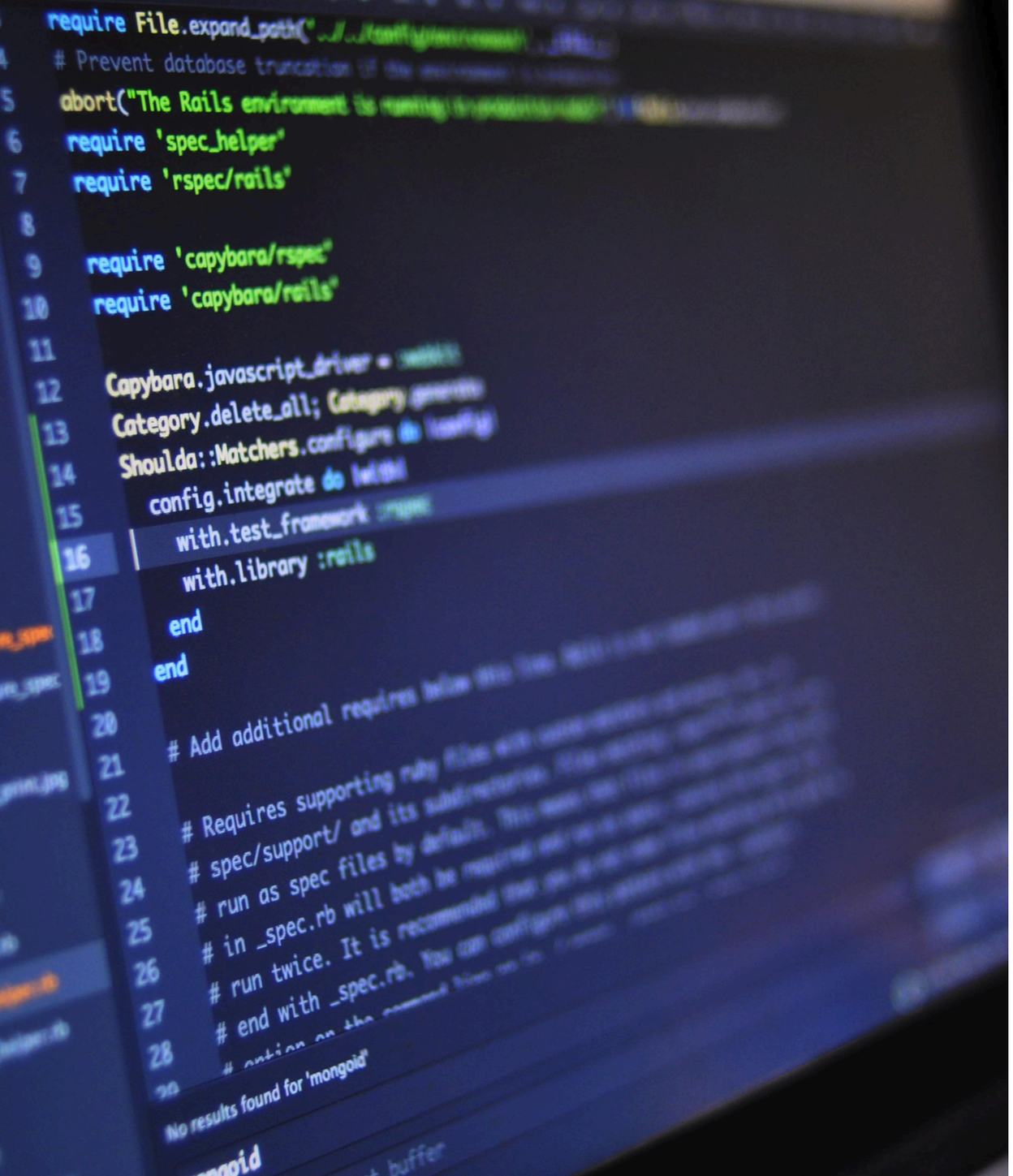


Monte Carlo et Simulation

Stratification

Présenté par : Diego Renaud, Raphaël Dalbarade et Marion chabrol

SOMMAIRE

- Monte Carlo et Quasi Monte Carlo
- Estimateurs d'Haber
- Comparaisons des estimateurs
- Importance sampling

Introduction

But : On cherche à calculer l'intégrale de la fonction suivante

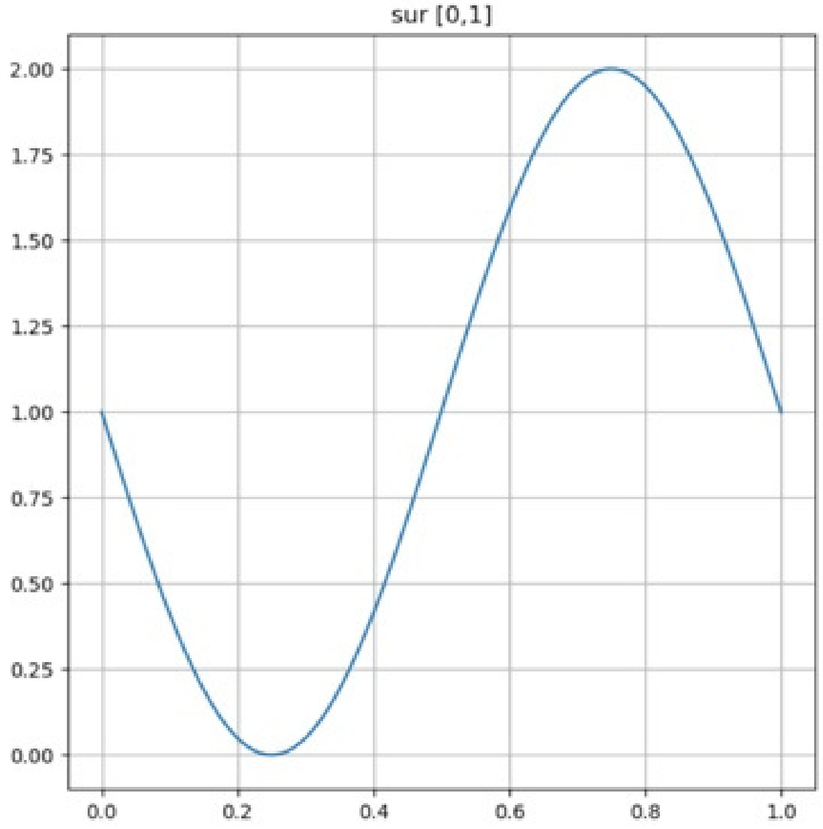
$$f(u) = 1 + \sin\left(2\pi\left(\frac{1}{d}\sum_{i=1}^{d} u_i - \frac{1}{2}\right)\right) \qquad u \in [0,1]^d \quad d \geq 1$$

Comment ?

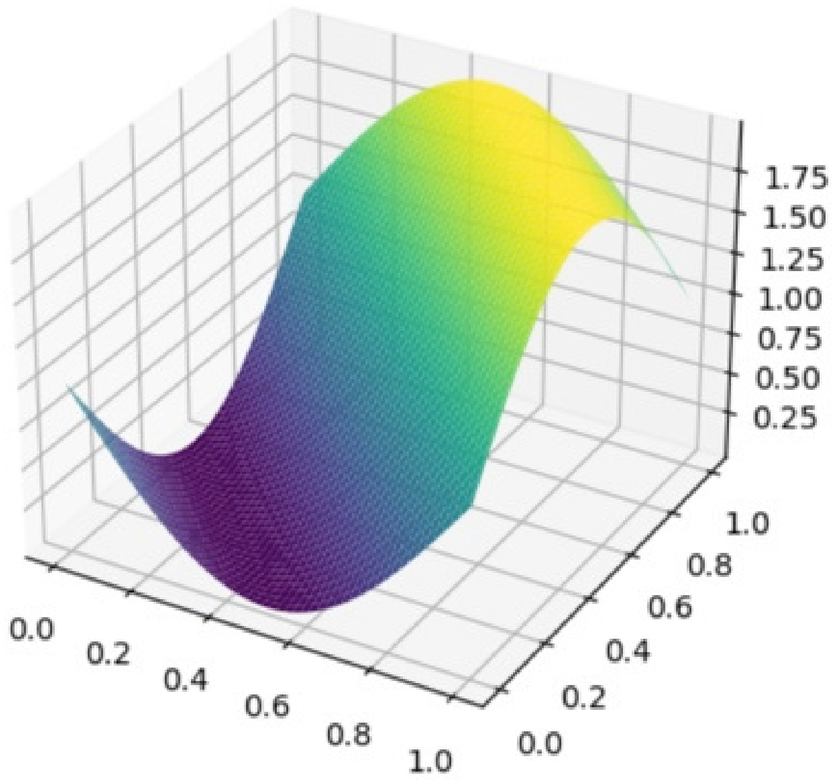
Nous allons approximer la valeur de l'intégrale de f à l'aide de différentes méthodes vu en cours et dans l'article associé à notre sujet.

Visualisation de la fonction cible

d = 1



d = 2

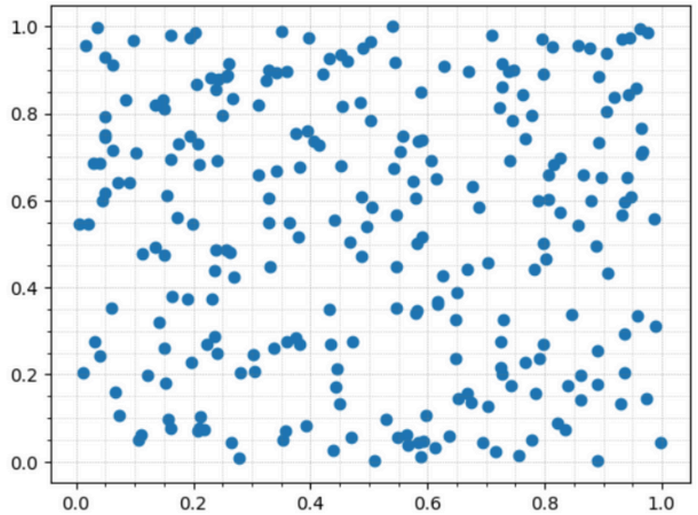


La méthode de Monte Carlo

I On génère un échantillon de N nombres aléatoires X_i distribués uniformément sur [0,1]^d

II On évalue chaque nombre X_i par la fonction f

III On calcule la moyenne des valeurs f(X_i) pour tous les X_i



→ On obtient l'approximation de l'intégrale de f sur [0 , 1]^d

La méthode de Quasi Monte Carlo

I On génère un échantillon de N nombres quasi-aléatoires X_i distribués uniformément sur [0,1]^d

II On évalue chaque nombre X_i par la fonction f

III On calcule la moyenne des valeurs f(X_i) pour tous les X_i

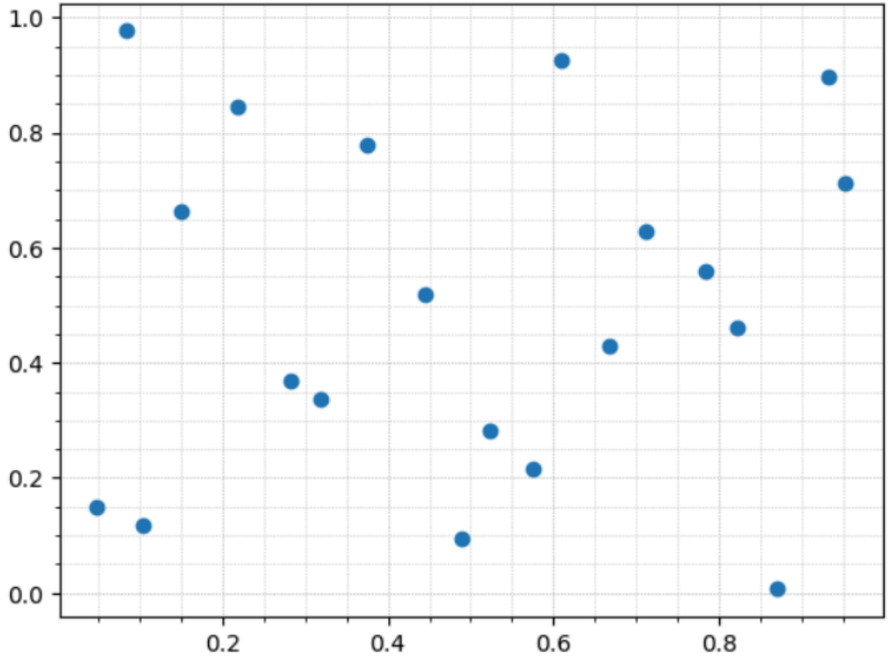
→ On obtient l'approximation de l'intégrale de f sur [0 , 1]^d

⚠ Dans cette méthode, l'échantillon n'est plus généré aléatoirement mais le choix des nombres suit une structure régulière : une suite à discrépance faible

Latin Hypercube Sampling

Méthode de Quasi Monte Carlo

Idée de la méthode de génération du Latin Hypercube Sampling: Il y a un point par ligne et un par colonne .



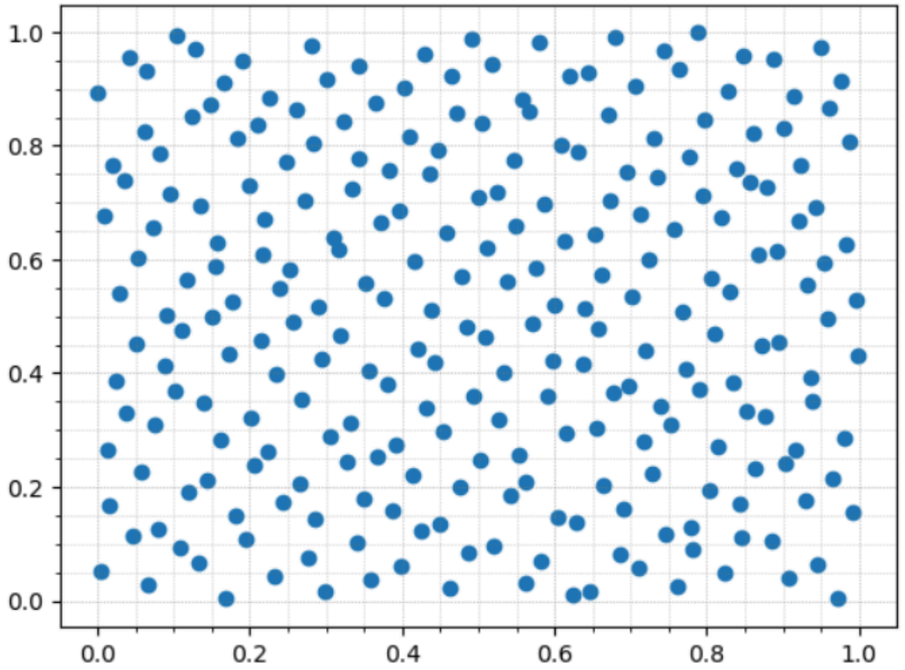
Dans cet exemple un échantillon de N = 20 points est généré dans un espace à deux dimensions:

La méthode de génération de points LHS consiste à diviser chaque dimension de l'espace de paramètres en N = 20 intervalles égaux, puis à sélectionner un point aléatoire dans chaque intervalle.
Les points ainsi sélectionnés sont ensuite permutés de manière à couvrir uniformément l'espace de paramètres.

Sobol

Méthode de Quasi Monte Carlo

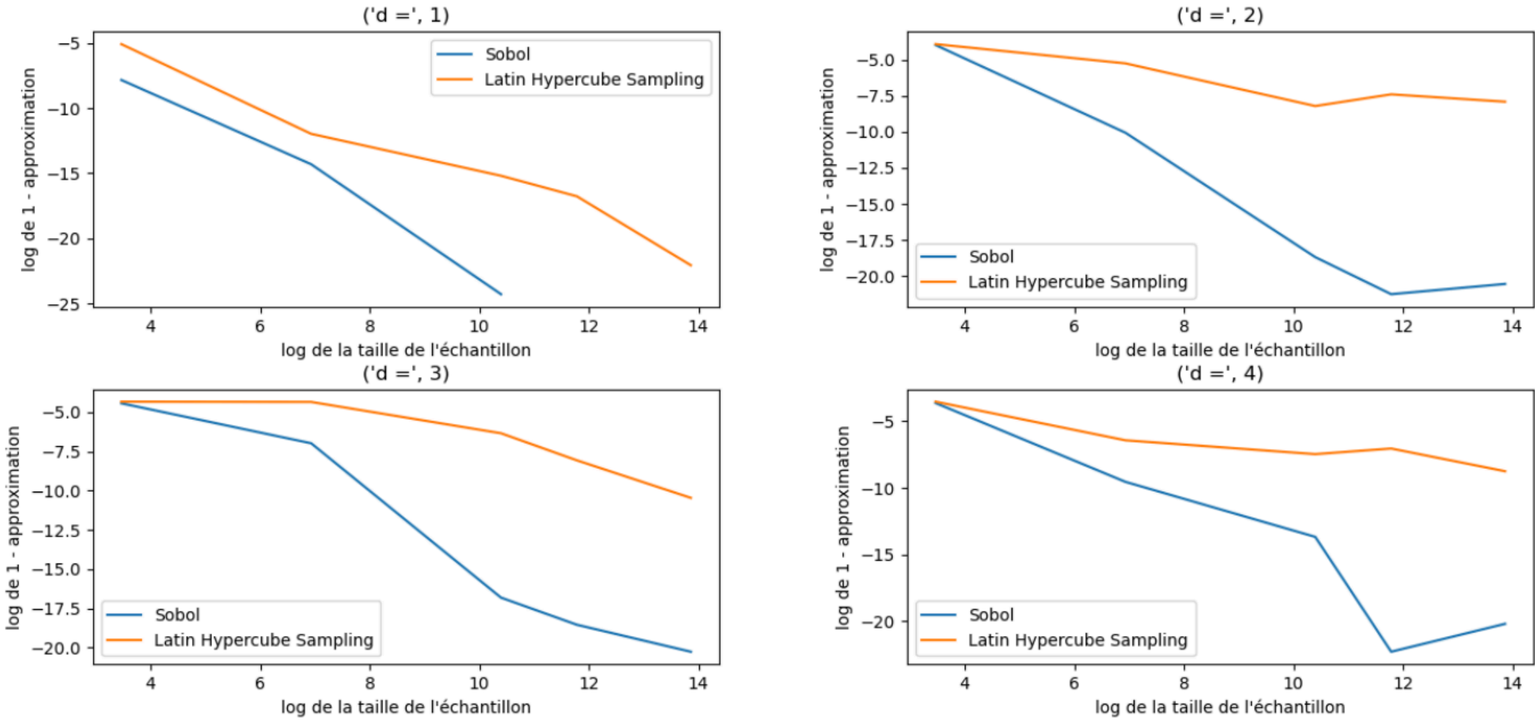
Dans cet exemple un échantillon de N = 256 points est généré dans un espace à de dimension 2:



La méthode de génération de points de Sobol est assez complexe, son idée est de combler le mieux possible l'espace en minimisant les trous entre les points générés. Cette méthode repose sur beaucoup de choix de paramétrages.

Comparaison

Méthode de Quasi Monte Carlo



Interprétation : On observe clairement que la méthode de génération de points de Sobol permet d'obtenir une estimation plus précise de l'intégrale de f que la méthode du Latin Hypercube Sampling quelque soit la dimension que l'on regarde

Les estimateurs d'Haber

Les estimateurs d'Haber d'ordre I et 2 sont les suivants :

$$\widehat{\mathcal{I}}_{1,k}(f) := \frac{1}{k^s} \sum_{c \in \mathfrak{C}_k} f(c + U_c), \qquad U_c \sim \mathcal{U}\left(\left[-\frac{1}{2k}, \frac{1}{2k}\right]^s\right)$$

$$\widehat{\mathcal{I}}_{2,k}(f) := \frac{1}{k^s} \sum_{c \in \mathfrak{C}_k} g_c(U_c), \qquad U_c \sim \mathcal{U}\left(\left[-\frac{1}{2k}, \frac{1}{2k}\right]^s\right)$$

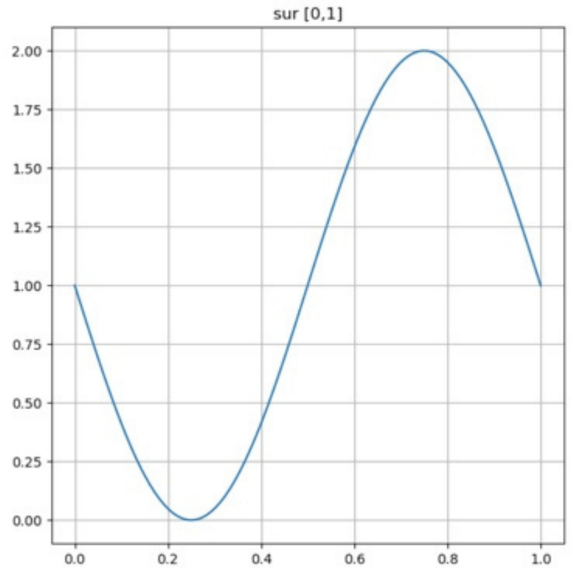
$$g_c(u) := \{f(c+u) + f(c-u)\}/2$$

Les estimateurs d'Haber

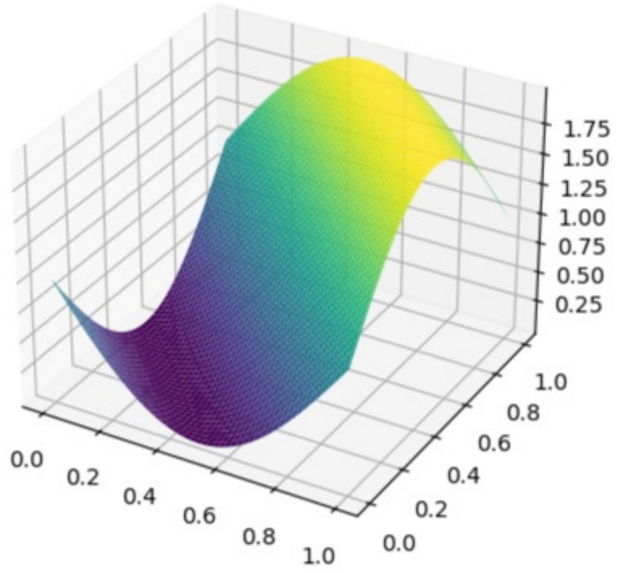
HYPOTHÈSES SUR LA FONCTION A INTÉGRER

- L'estimateur d'Haber d'ordre r requiert que la fonction intégrée appartienne à C^r([0,1]^d)
- C'est le cas pour notre fonction

d = 1



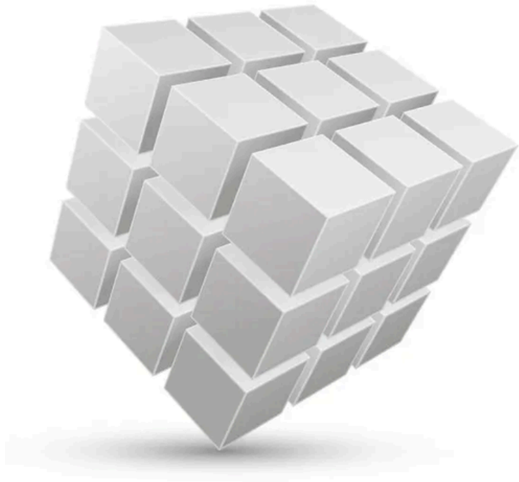
d = 2



Les estimateurs d'Haber

IDÉE

- L'approche d'Haber repose sur la stratification de [0, 1]^d en k^d hypercubes proches
- Puis d'évaluer f à l'intérieur de chaque hyper-cube
- Pour nos estimateurs, nous aurons besoin des coordonnées des centres des hypercubes



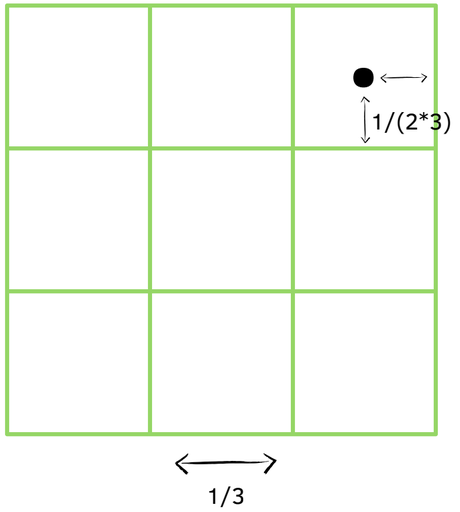
Estimateurs d'Haber

ESTIMATEUR 1

- Soient Ck l'ensemble des centres des hypercubes
- A chaque c ∈ Ck nous associons une variable aléatoire Uc telle que Uc ~U([-1/2k, 1/2k]^d)
- *Remarque* : le support de (c + Uc) est donc l'hyper-cube [c ± 1/2k]^d
- L'estimateur 1 évalue ensuite f aux points c + Uc
- Puis estime l'intégrale en faisant la moyenne des k^d évaluations de f

$$\widehat{\mathcal{I}}_{1,k}(f) := \frac{1}{k^s} \sum_{c \in \mathfrak{C}_k} f(c + U_c), \qquad U_c \sim \mathcal{U}\left(\left[-\frac{1}{2k}, \frac{1}{2k}\right]^s\right)$$

Pour k = 3 et d = 2:



Estimateurs d'Haber

RESULTATS

- Le RMSE converge vers 0 à vitesse n^(−1/2−1/d) si f ∈ C^1([0,1]^d)
- C'est la vitesse de convergence optimale pour une fonction de C^1([0,1]^d)
- A titre de comparaison, le RMSE pour la méthode de Monte Carlo converge vers 0 à vitesse n^-1/2

$$\widehat{\mathcal{I}}_{1,k}(f) := \frac{1}{k^s} \sum_{c \in \mathfrak{C}_k} f(c + U_c), \qquad U_c \sim \mathcal{U}\left(\left[-\frac{1}{2k}, \frac{1}{2k}\right]^s\right)$$

$\mathbb{E}[\hat{I}] = \frac{1}{k^s} \sum_{c \in C_k} \mathbb{E}[f(c + U_c)]$

or $\mathbb{E}[f(c + U_c)] = k^s \int_{[c \pm \frac{1}{2k}]^d} f(u)du$

d'où $\mathbb{E}[\hat{I}] = \frac{1}{k^s} \sum_{c \in C_k} k^s \int_{[c \pm \frac{1}{2k}]^d} f(u)du$

$= \int_{[0,1]^d} f(u)du = I$

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Estimateurs d'Haber

ESTIMATEUR 2

- L'idée du second estimateur est assez similaire au premier
- Cet estimateur s'appuie sur 2*k^d évaluations de la fonctions f
- Le RMSE converge vers 0 à vitesse n^(−1/2−2/d) si f ∈ C^2([0,1]^d)
- C'est la vitesse de convergence optimale pour une fonction de C^2([0,1]^d)

$$\widehat{\mathcal{I}}_{2,k}(f) := \frac{1}{k^s} \sum_{c \in \mathfrak{C}_k} g_c(U_c), \qquad U_c \sim \mathcal{U}\left(\left[-\frac{1}{2k}, \frac{1}{2k}\right]^s\right)$$

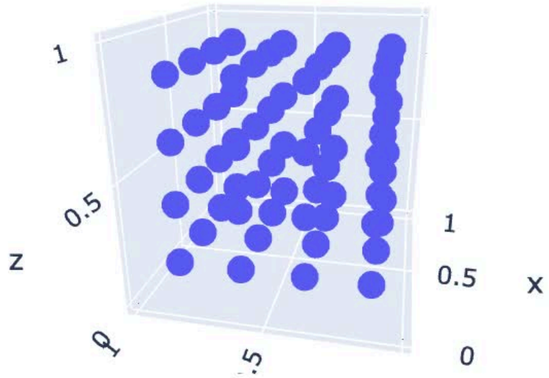
$$g_c(u) := \{f(c+u) + f(c-u)\}/2$$

Estimateurs d'Haber - code

Création de la fonction qui calcule les coordonnées des centres des hypercubes en fonction du nombre de cellules souhaitées n_cells

```python
#calcul des coordonnées des centres de chaque cellule

from itertools import product

def cells_centers(d = 2, n_cells = 5):
    vecteur = np.around(np.linspace(1/(2*n_cells), 1 - 1/(2*n_cells), n_cells), decimals=10) #découpe le segment [0,1] en n_cel
    cell_coord = product(vecteur, repeat = d) #product donne toutes les combinaisons possibles de groupes de taille d du vecteu
    cell_coord = np.array(list(map(list, cell_coord))) #changement de type
    return(cell_coord) #product a permis d'avoir toutes les combinaisons possibles du vecteur, ce qui correspond à toutes les c
```

```
array([[0.1, 0.1],
       [0.1, 0.3],
       [0.1, 0.5],
       [0.1, 0.7],
       [0.1, 0.9],
       [0.3, 0.1],
       [0.3, 0.3],
       [0.3, 0.5],
       [0.3, 0.7],
       [0.3, 0.9],
       [0.5, 0.1],
       [0.5, 0.3],
       [0.5, 0.5],
       [0.5, 0.7],
       [0.5, 0.9],
       [0.7, 0.1],
       [0.7, 0.3],
       [0.7, 0.5],
```



Estimateurs d'Haber - code

Création de la fonction gc pour l'estimateur 2

```
#Implémentation de la fonction g, utilisée dans l'estimateur d'Haber d'ordre 2
def g_c(x,c):
    return((f(c + x) + f(c - x))/2)
```

$$g_c(u) := \{f(c+u) + f(c-u)\}/2$$

Création de la fonction habers_estimation

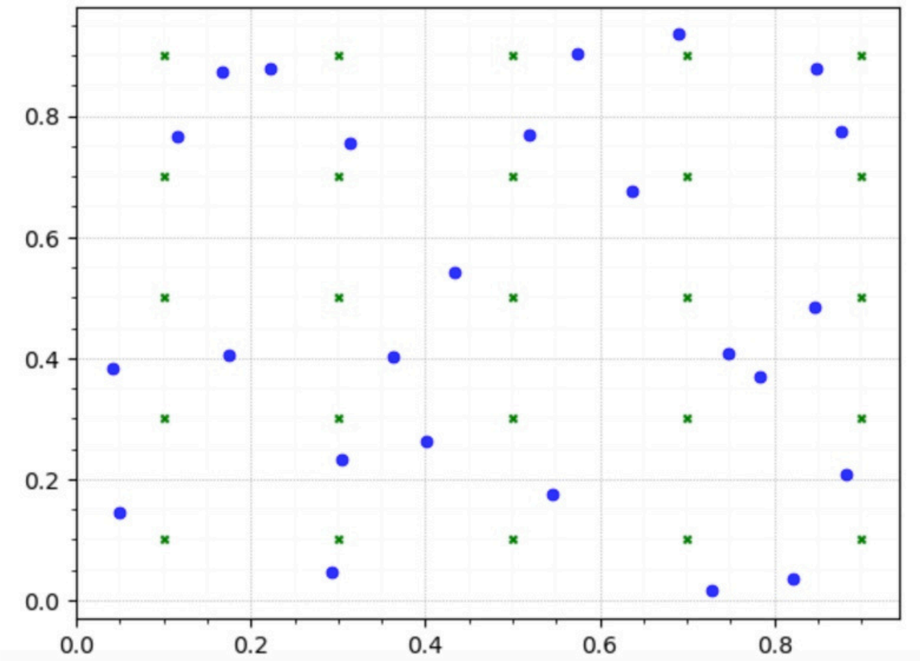
```
#Implémentation des estimateurs d'Haber

#génère les points uniformes tels qu'ils sont définis dans le papier (Uc ~U([-1/2k,1/2k]^d)
def generate_haber(n, k, d):
    return(np.random.uniform(low=-1/(2*k), high=1/(2*k), size=(n, d)))
```

$$U_c \sim \mathcal{U}\left(\left[-\frac{1}{2k}, \frac{1}{2k}\right]^s\right)$$

```
def habers_estimation(f, order, k, d):
    cell_coord = cells_centers(d, n_cells = k)
    U = generate_haber(len(cell_coord), k, d)
    if order == 1:
        f_values = f(cell_coord + U) #évalue f aux centres de chaque hypercube déplacé
    if order == 2:
        f_values = g_c(U, cell_coord) #évalue f aux centres des hypercubes déplacé par l'intermédiaire de la fonction g
    return np.sum(f_values)/k**d #estime ensuite l'intégrale
```

Estimateurs d'Haber - code

Visualisation des points c + Uc



- Points bleus : c + Uc
- Croix verts : c (centres des hypercubes)

Résultat

```
#ordre 1 (5^2 = 25 évaluations)
habers_estimation(f, order = 1, k = 5, d = 2)
```

```
0.9918627711796595
```

```
#ordre 2 (2*5^2 = 50 évaluations)
habers_estimation(f, order = 2, k = 5, d = 2)
```

```
0.9996462490709679
```

Comparaison

Comparaison de nos 4 estimateurs selon les critères suivants :

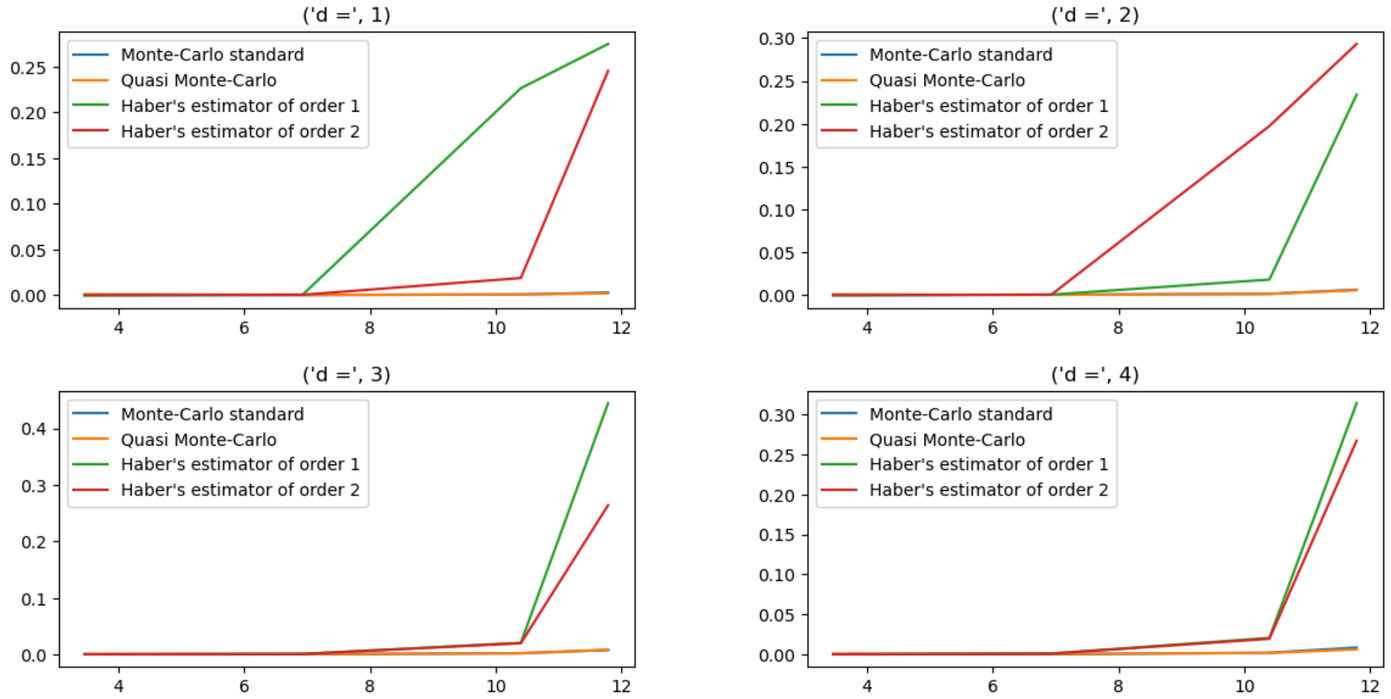
→ **Critère du temps de calcul**

→ **Critère de précision et de stabilité:**

- **qualité de la prédiction**
- **variance**

Temps de calcul

Comparaison du temps de calcul des 4 méthodes



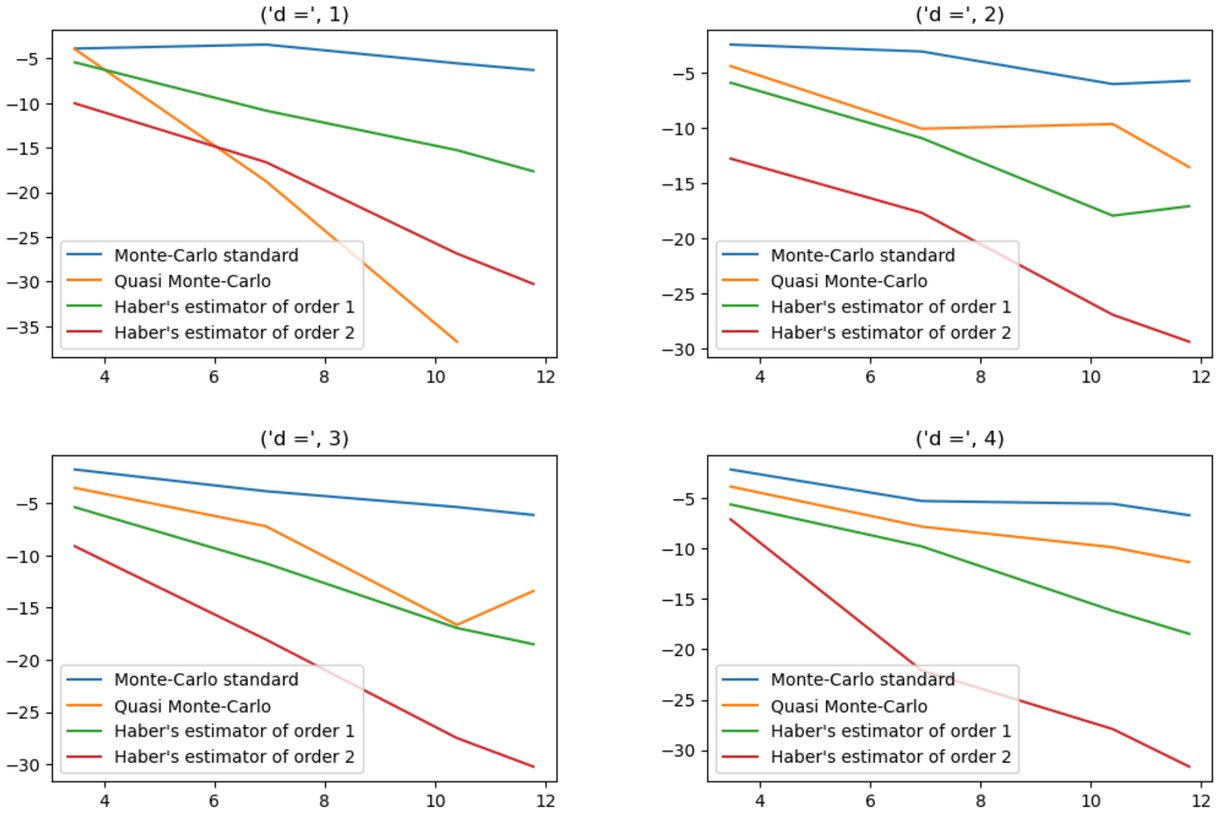
Résultat

Selon le critère du temps de calcul, voici le classement des nos 4 méthodes:

- Monte Carlo Standard
- Quasi-Monte Carlo
- Haber's Estimators

Précision de la prédiction

Comparaison de la précision des 4 méthodes



Résultat

Selon le critère de précision de la prédiction, voici le classement des nos 4 méthodes:

- Haber's Estimators
- Quasi-Monte Carlo
- Monte Carlo Standard

Précision de la prédiction

Comparaison de la précision et de la stabilité des 4 méthodes

Méthode	Moyenne des erreurs	Variance
MC	0.015349	3.861711e-04
QMC	0.000047	6.978027e-09
haber1	0.000724	8.084262e-07
haber2	0.000018	5.139204e-10

Table 1: Moyennes des erreurs et variances pour différentes méthodes

Résultat

Finalement, en regardant la moyenne des erreurs ainsi que la variance, il apparaît que Haber 2 et Quasi Monte-Carlo sont les 2 méthodes les plus précises et stables pour estimer notre intégrale.

Importance Sampling

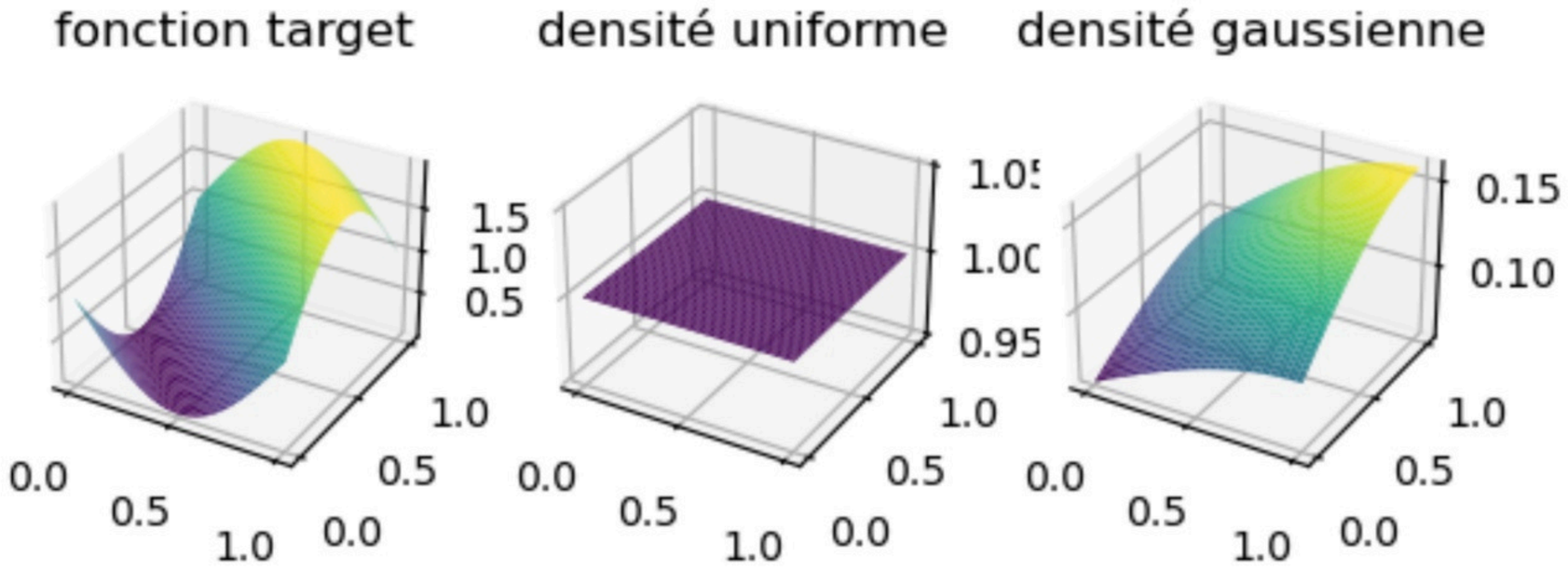
Principe de l'Importance Sampling (échantillonnage par importance) :

$$E[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx \approx \frac{1}{n}\sum_i f(x_i)\frac{p(x_i)}{q(x_i)}$$

But: trouver la meilleure distribution d'échantillonnage q

Analyse Importance Sampling

Analyse de notre fonction d'intérêt



Comparaison de notre fonction d'intérêt avec des potentielles distributions d'échantillonnage

Résultat

Nous avons estimé qu'une loi gaussienne multidimensionnelle d'espérance (1,...,1) était une loi proche de notre fonction d'intérêt. Nous allons donc l'utiliser comme distribution d'échantillonnage pour notre approche d'Importance Sampling

Importance Sampling code

Génération de points d'échantillonnage selon notre distribution gaussienne

```
def gen_normal(N, mu, cov):
    return(np.random.multivariate_normal(mu, cov, N))

def mu_cov_gaussian(d):
    return(np.array([1]*d), np.eye(d))
```

Création de la fonction Importance Sampling

```
def importance_sampling_gaussian(f, g, N, d):
    mu, cov = mu_cov_gaussian(d)
    u = gen_normal(N, mu, cov)
    weights = f(u)*g(u)/gaussienne(u, mu, cov)
    integral = np.sum(weights)/N
    return integral

importance_sampling_gaussian(f, g, N = 1000000, d = 2)
```

```
0.9991358927906124
```

Conclusion

Nous obtenons une approximation de notre intégrale satisfaisante mais moins précises qu'en utilisant les méthodes de Monte Carlo, Quasi MC ou les estimateurs d'Haber. Cela pourrait provenir du fait de notre distribution d'échantillonnage qui n'est pas suffisamment proche de notre fonction d'intérêt.