TP3 — Quasi-Monte-Carlo, wing weight (corrige)

Source : TP3/TP3-Student_corrige.ipynb — PDF compile (code execute).

MAP556, Ecole Polytechnique, 2023-24

TP3 - The wing weight function

La fonction suivante est un modèle pour le poids de l'aile d'un avion :

$0.036*S_w^{0.758}W_{fw}^{0.0035} \left(\frac{A}{\cos^2(\Lambda)}\right)^{0.6} q^{0.006}\lambda^{0.04}\left(\frac{100t_c}{\cos(\Lambda)}\right)^{-0.3}(N_xW_{dg})^{0.49} + S_wW_p$

Elle est extraite de la bibliothèque virtuelle de fonctions de test de simulations expérimentales mise au point par Surjanovic et Bingham (2013).

Les significations et les plages des variables sont indiquées dans le tableau ci-dessous.

| Variable | Range | Meaning | |:----------|:--------------|:-------------------------------------| | S_w | [150, 200] | Wing Arra (ft²) | | W_{fw} | [220, 300] | Weight of Fuel in the Wing (lb) | | A | [6, 10] | Aspect Ratio | | Λ | [-10, 10] | Quarter-Chord Sweep (degrees) | | q | [16, 45] | Dynamic Pressure at Cruise (lb/ft²) | | λ | [0.5, 1] | Taper Ratio | | t_c | [0.08, 0.18] | Aerofoil Thickness to Chord Ratio | | N_x | [2.5, 6] | Ultimate Load Factor | | W_{dg} | [1700, 2500] | Flight Design Gross Weight (lb) | | W_p | [0.025, 0.08] | Paint Weight (lb/ft²)

La bibliothèque virtuelle contient du code pour implémenter cette fonction ainsi que des références à son origine. Notez que Λ est donné en degrés, de −10 à 10. Il se situe alors entre $\pm 10\pi/180$ radians, donc $\cos(\Lambda)$ ne s'approche pas de zéro, et les poids des ailes sont bornés.

Nous étudierons la moyenne de cette fonction sur l’hypercube à 10 dimensions défini par les plages de ses variables d’entrée. Notre intégrande sur $[0,1]^{10}$ adapte d'abord chaque variable à sa plage, puis calcule le poids de l'aile. On ne chercherait normalement pas le poids moyen d’une aile d’avion conçue au hasard. Cet exemple est utile à titre d’illustration car il a une origine scientifique/ingénierie tout en ne nécessitant pas l’accès à un logiciel propriétaire spécialisé pour le calculer. Nous ignorerons également le fait que neuf des dix variables d’entrée peuvent être intégrées pour donner une forme élémentaire fermée. L’exception est Λ.

Nous pouvons appliquer un échantillonnage de Monte-Carlo simple ainsi qu'un échantillonnage quasi-Monte-Carlo à cet intégrande. Le but du TP est de montrer l'efficacité de la méthode quasi-Monte-Carlo en comparaison de l'échantillonnage Monte-Carlo.

1- Illustration de la répartition d'une suite dans [0,1]

Rappelons que la discrépance $D_N(x)$ et la discrépance étoile $D_N^*(x)$ de la suite $x=\{x_n\}_{n \geqslant 1}$ dans $[0,1]$ sont respectivement données par $$ D_N(x) = \sup_{0\leqslant a<b \leqslant 1} \left| \frac{1}{N} \sum_{k=1}^N 1_{[a,b)}(x_k) - (b-a)\right|\,,\qquad D_N^*(x) = \sup_{ 0\leqslant a \leqslant 1} \left| \frac{1}{N} \sum_{k=1}^N 1_{[0,a)}(x_k) - a\right|. $$

On se donne une suite de 100 points obtenue par un générateur aléatoire.

In [1] :

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

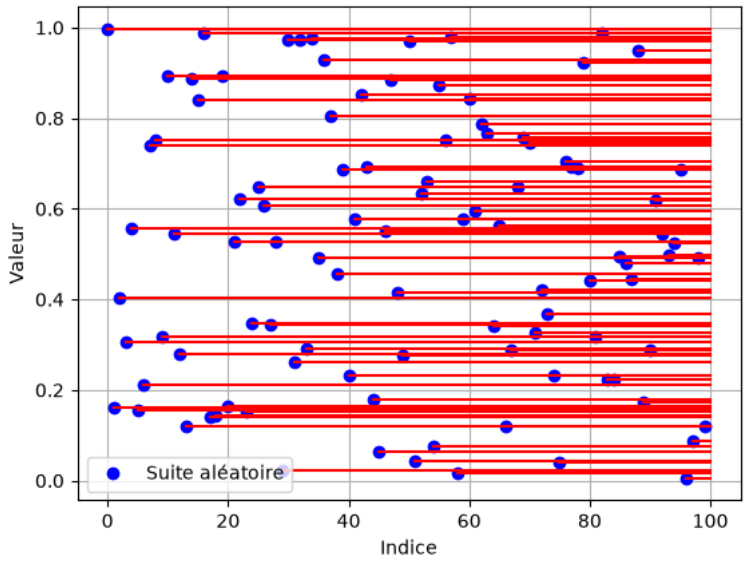
size_sequence = 100 # Nombre de points dans la suite
sequence = np.random.random(size_sequence)
```

On peut afficher la suite pour visualiser la manière dont elle remplit l'intervalle $[0,1]$.

In [2] :

```
        # Afficher la suite et son remplissage de [0,1]
        plt.plot(sequence, 'bo', label='Suite aléatoire')
        for i in range(size_sequence):
        plt.plot([i,size_sequence],[sequence[i],sequence[i]],'r-')

        plt.xlabel('Indice')
        plt.ylabel('Valeur')
        plt.legend()
        plt.grid(True)
        plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

2- Calcul de la discrépance

On cherche à calculer la discrépance de la suite précédente. Mais pour calculer un supremum sur a et b dans $[0,1]$, on est clairement face à un problème informatique. Une première idée est de se représenter une grille finie G sur $[0,1]\times [0,1]$ constituée de couples de variables $(a_i,b_j)_{i\in I, j\in J}$ dans laquelle on va faire varier les variables a et b . En calculant la discrépance locale entre a et b définie par $$ D_N(x,a,b) = \left| \frac{1}{N} \sum_{k=1}^N 1_{[a,b)}(x_k) - (b-a)\right|, $$ on obtient une borne inférieure de la discrépance par la formule $$ D_N(x) \geq \sup_{(a,b)\in G} D_N(x,a,b). $$

$\blacktriangleright$ Question 1 [code] : Ecrire un code python pour calculer la discrépance locale entre deux bornes a et b. On pourra utiliser la fonction `filter(lambda x: a <= x < b, sequence)` pour extraire une sous-suite.

In [3] :

```
        ######################################################
        # Réponse 1 [code] :
        ######################################################
        def calculate_local_discrepancy(sequence, a, b):
        N = len(sequence)
        count = np.sum((sequence >= a) & (sequence < b))
        # cas b==1 : inclure le point 1 eventuel
        if b == 1:
        count = np.sum((sequence >= a) & (sequence <= b))
        return abs(count / N - (b - a))
```

A l'aide de la discrépance locale, on peut alors calculer le supremum sur une grille uniforme de taille $\mathcal{O}(P^2)$ avec $G:=\left\{\left(\frac{i}{P},\frac{j}{P}\right)\right\}_{0\leq i<j\leq P}$.

In [4] :

```
        def calculate_discrepancy_on_mesh(sequence,size_mesh):
        max_discrepancy = 0
        for i in range(0, size_mesh):
        for j in range(i + 1, size_mesh+1):
        left = i/size_mesh
        right = j/size_mesh
        d_local = calculate_local_discrepancy(sequence,left,right)
        max_discrepancy = max(max_discrepancy, d_local)

        return max_discrepancy
```

In [5] :

```
        # Afficher la borne inférieure de la discrépance de la suite
        size_mesh = 100;
        discrepancy = calculate_discrepancy_on_mesh(sequence,size_mesh)
        print(f'Pour une taille de maillage {size_mesh}x{size_mesh} la borne inférieure de la
        discrépance est {discrepancy:.4f}.')
```

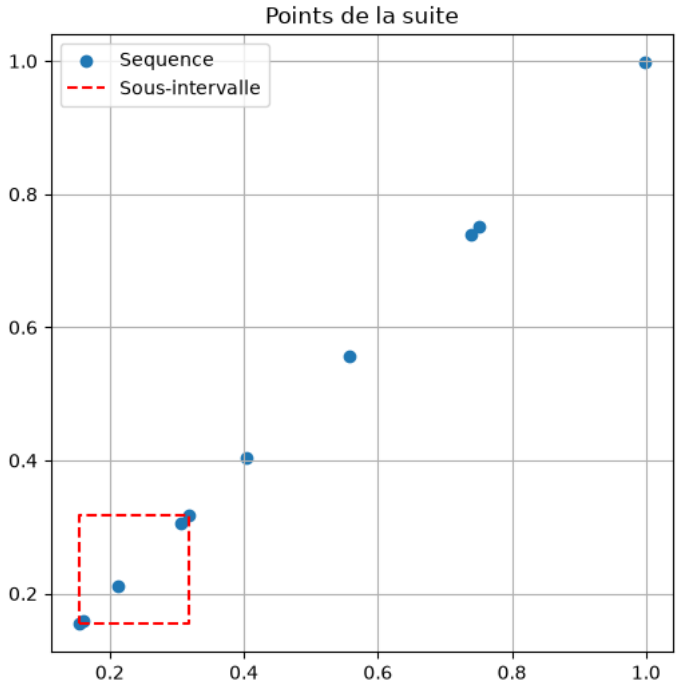
```
Pour une taille de maillage 100x100 la borne inférieure de la discrépance est 0.0700.
```

$\blacktriangleright$ Question 2 [code] : La méthode n'est clairement pas optimale. Proposer une amélioration simple en choisissant astucieusement la grille G. On pourra s'inspirer du dessin suivant réalisé en 2D pour une meilleure visualisation.

In [6] :

```
        plt.figure(figsize=(6, 6))
        plt.scatter(sequence[0:10],sequence[0:10], marker='o', label='Sequence')
        plt.plot([sequence[5], sequence[9],sequence[9],sequence[5],sequence[5]],
        [sequence[5],sequence[5],sequence[9],sequence[9],sequence[5]], 'r--',
        label='Sous-intervalle')

        # Afficher le graphique
        plt.title('Points de la suite')
        plt.legend()
        plt.grid(True)
        plt.show()
```




```
<Figure size 600x600 with 1 Axes>
```

In [7] :

```
        ####################################################
        # Réponse 2 [code] :
        ####################################################
        def calculate_discrepancy_sub_interval(sequence):
        """Discrepance exacte D_N : le sup est atteint pour a,b pris
        parmi {0,1} U {points de la suite}. O(N^2)."""
        x = np.sort(np.asarray(sequence, dtype=float))
        N = len(x)
        pts = np.concatenate(([0.0], x, [1.0]))
        best = 0.0
        # intervalles [pts[i], pts[j])
        for i in range(len(pts)):
        for j in range(i + 1, len(pts)):
        a, b = pts[i], pts[j]
        cnt = np.sum((x >= a) & (x < b))
        if b == 1.0:
        cnt = np.sum((x >= a) & (x <= b))
        best = max(best, abs(cnt / N - (b - a)))
        return best
```

In [8] :

```
        # Afficher la discrépance de la suite
        discrepancy = calculate_discrepancy_sub_interval(sequence)
        print(f'La discrépance est {discrepancy:.4f}.')
```

```
La discrépance est 0.0668.
```

$\blacktriangleright$ Question 3 [code] : En remarquant que la discrépance étoile a une expression simplifiée (voir TD), écrire un code python pour calculer la discrépance étoile d'une suite.

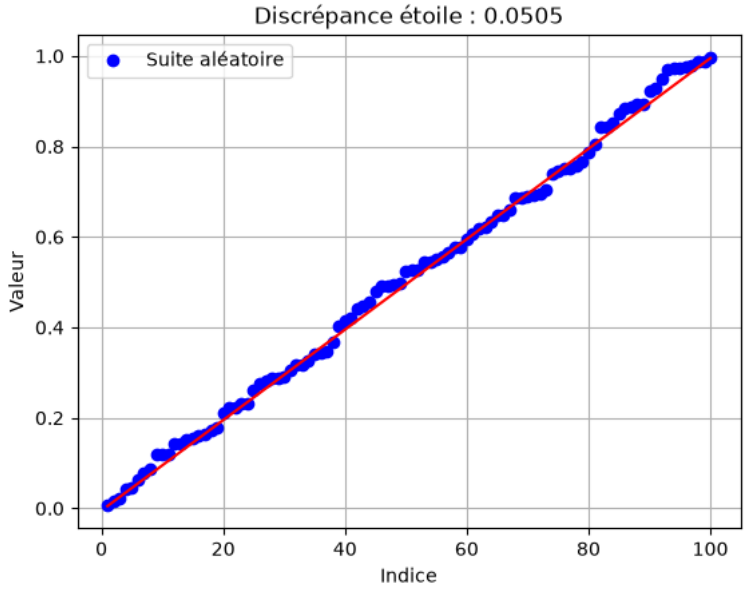
In [9] :

```
        ####################################################
        # Réponse 3 [code] :
        ####################################################
        def calculate_discrepancy_star(sequence):
        """D*_N = 1/(2N) + max_i |x_(i) - (2i-1)/(2N)| (suite triee)."""
        x = np.sort(np.asarray(sequence, dtype=float))
        N = len(x)
        i = np.arange(1, N + 1)
        return float(1.0 / (2 * N) + np.max(np.abs(x - (2 * i - 1) / (2 * N))))
```

In [10] :

```
        # Afficher la discrépance étoile de la suite
        discrepancy = calculate_discrepancy_star(sequence)
        size_seq = len(sequence)
        print(f'La discrépance étoile est {discrepancy:.4f}.')
        # Afficher la suite et la discrépance étoile
        sorted_sequence = np.sort(sequence)
        plt.plot(range(1,size_seq+1),sorted_sequence, 'bo', label='Suite aléatoire')
        plt.plot(range(1,size_seq+1),(2*np.linspace(1,size_seq,size_seq)-1)/(2*size_seq),'r-')
        plt.title(f'Discrépance étoile : {discrepancy:.4f}')
        plt.xlabel('Indice')
        plt.ylabel('Valeur')
        plt.legend()
        plt.grid(True)
        plt.show()
```

```
La discrépance étoile est 0.0505.
```




```
<Figure size 640x480 with 1 Axes>
```

3- Génération de suites déterministes à discrépance faible.

On peut générer une suite (non aléatoire) à discrépance faible. Un premier exemple consiste à utiliser une écriture en base b pour générer la suite de van der Corput.

$\blacktriangleright$ Question 4 [code] : Définir une fonction qui permet de générer les n premiers éléments de la suite de van der Corput en base b.

In [11] :

```
######################################################
# Réponse 4 [code] :
######################################################
def van_der_corput_sequence(n, base=2):
"""n premiers points de van der Corput en base b (inversion radicale de 1..n)."""
seq = np.zeros(n)
for i in range(1, n + 1):
x, f = 0.0, 1.0 / base
k = i
while k > 0:
x += (k % base) * f
k //= base
f /= base
seq[i - 1] = x
return seq
```

In [12] :

```
# Générer une séquence de Van der Corput avec 10 points (en base 2)
size_sequence = 15 # Nombre de points dans la suite
sequence = van_der_corput_sequence(size_sequence)

# Afficher les premiers points générés
print("Séquence de Van der Corput :")
for i, value in enumerate(sequence, start=1):
print(f"Point {i}: {value:.4f}")
```

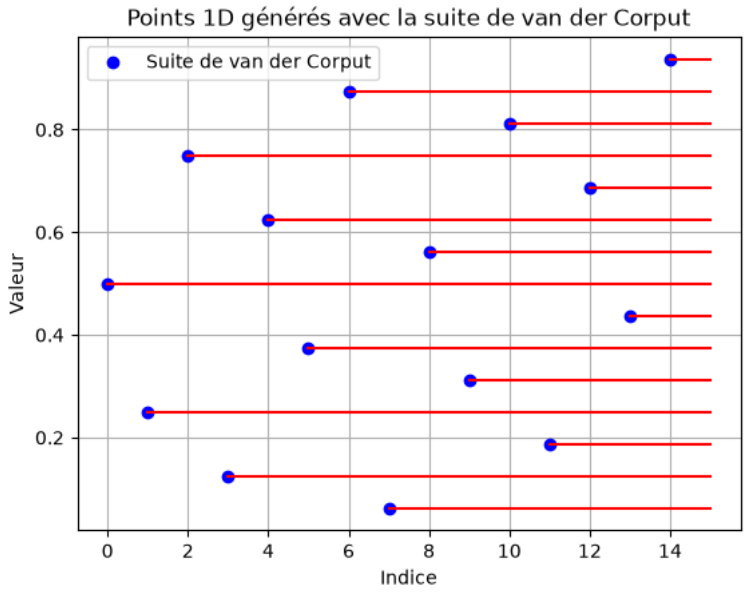
```
Séquence de Van der Corput :
Point 1: 0.5000
Point 2: 0.2500
Point 3: 0.7500
Point 4: 0.1250
Point 5: 0.6250
Point 6: 0.3750
Point 7: 0.8750
Point 8: 0.0625
Point 9: 0.5625
Point 10: 0.3125
Point 11: 0.8125
Point 12: 0.1875
Point 13: 0.6875
Point 14: 0.4375
Point 15: 0.9375
```

In [13] :

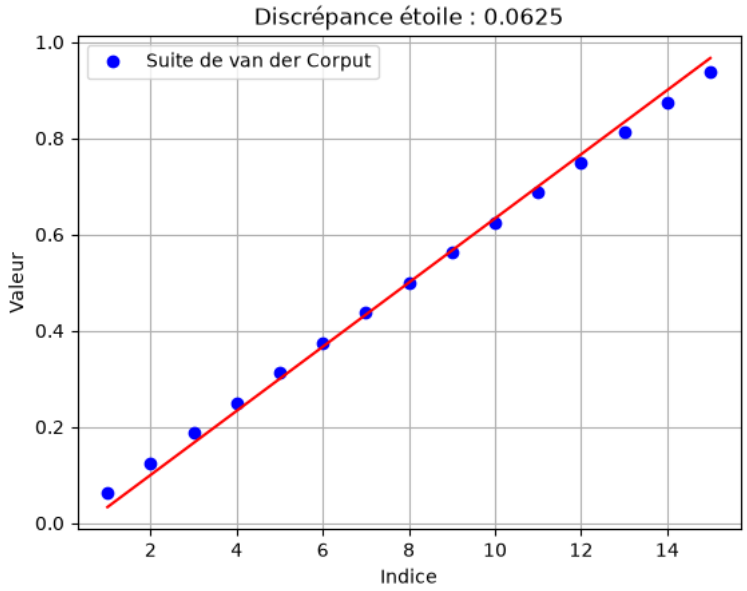
```
        # Afficher la discrépance étoile de la suite
        discrepancy = calculate_discrepancy_star(sequence)
        size_seq = len(sequence)
        print(f'La discrépance étoile est {discrepancy:.4f}.')
        # Afficher la suite et son remplissage de [0,1]
        plt.plot(sequence, 'bo', label='Suite de van der Corput')
        for i in range(size_sequence):
        plt.plot([i,size_sequence],[sequence[i],sequence[i]],'r-')

        plt.xlabel('Indice')
        plt.ylabel('Valeur')
        plt.title('Points 1D générés avec la suite de van der Corput')
        plt.legend()
        plt.grid(True)
        plt.show()
        # Afficher la suite et la discrépance étoile
        sorted_sequence = np.sort(sequence)
        plt.plot(range(1,size_seq+1),sorted_sequence, 'bo', label='Suite de van der Corput')
        plt.plot(range(1,size_seq+1),(2*np.linspace(1,size_seq,size_seq)-1)/(2*size_seq),'r-')
        plt.title(f'Discrépance étoile : {discrepancy:.4f}')
        plt.xlabel('Indice')
        plt.ylabel('Valeur')
        plt.legend()
        plt.grid(True)
        plt.show()
```

```
La discrépance étoile est 0.0625.
```



```
<Figure size 640x480 with 1 Axes>
```




```
<Figure size 640x480 with 1 Axes>
```

$\blacktriangleright$ Question 4bis [code] : Définir une fonction qui permet de générer les n premiers éléments de la suite de Halton en base $b=3$. Et définir une fonction qui permet de générer les suites de Halton en toutes dimensions.

Indication : Pour permettre son utilisation dans la suite du TP, on se donne le début de code suivant pour générer les suites de Halton en toutes dimensions. Il est basé sur le calcul des nombres premiers successifs. Et on fournit une fonction de visualisation en dimensions 1 et 2.

In [14] :

```
        ###################################################
        # Réponse 4bis [code] :
        ###################################################
        # Fonction simple pour calculer la suite de Halton en dimension 1 en base 3.
        def halton_sequence(n, base=3):
        sequence = np.zeros(n)
        for i in range(n):
        x, f = 0.0, 1.0 / base
        k = i + 1
        while k > 0:
        x += (k % base) * f
        k //= base
        f /= base
        sequence[i] = x
        return sequence

        # Fonction pour calculer le prochain nombre premier.
        def next_prime():
        prime = 2
        while True:
        yield prime
        prime += 1
        while not all(prime % np.arange(2, int(np.sqrt(prime)) + 1)):
        prime += 1

        # Fonction pour calculer la suite de Halton en dimension d.
        def halton_sequence_dimension(n, d):
        sequence = np.zeros((n, d))
        primes = next_prime()
        for j in range(d):
        base = next(primes)
        for i in range(n):
        x, f = 0.0, 1.0 / base
        k = i + 1
        while k > 0:
        x += (k % base) * f
        k //= base
        f /= base
        sequence[i, j] = x
        return sequence
```

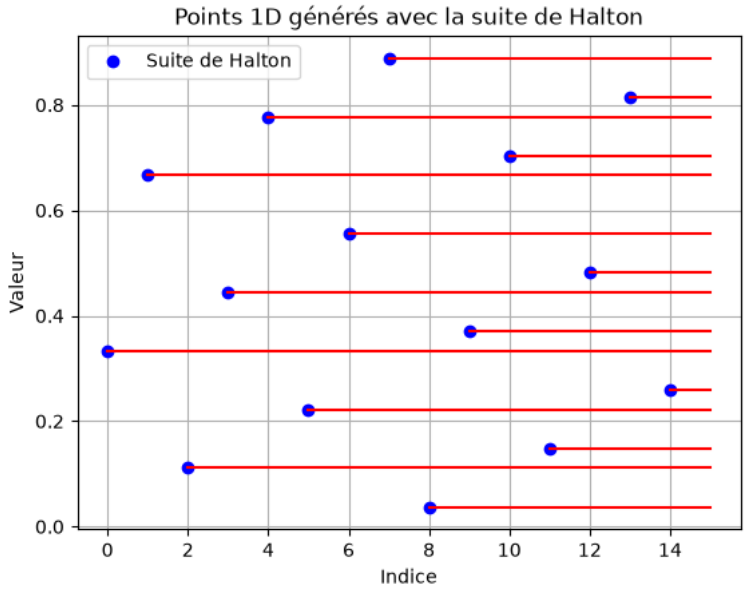
In [15] :

```
# Générer les premières valeurs de la suite de Halton 1D
dimension = 1
size_sequence = 15 # Nombre de points dans la suite
sequence = halton_sequence(size_sequence)

# Afficher la suite et son remplissage de [0,1]
plt.plot(sequence, 'bo', label='Suite de Halton')
for i in range(size_sequence):
plt.plot([i,size_sequence],[sequence[i],sequence[i]],'r-')

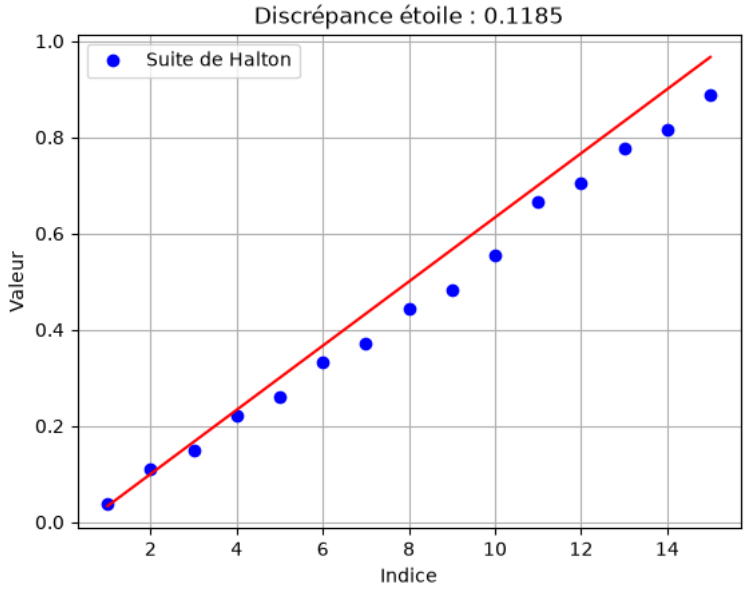
plt.xlabel('Indice')
plt.ylabel('Valeur')
plt.title('Points 1D générés avec la suite de Halton')
plt.legend()
plt.grid(True)
plt.show()

# Afficher la discrépance étoile de la suite
discrepancy = calculate_discrepancy_star(sequence)
size_seq = len(sequence)
print(f'La discrépance étoile est {discrepancy:.4f}.')
# Afficher la suite et la discrépance étoile
sorted_sequence = np.sort(sequence)
plt.plot(range(1,size_seq+1),sorted_sequence, 'bo', label='Suite de Halton')
plt.plot(range(1,size_seq+1),(2*np.linspace(1,size_seq,size_seq)-1)/(2*size_seq),'r-')
plt.title(f'Discrépance étoile : {discrepancy:.4f}')
plt.xlabel('Indice')
plt.ylabel('Valeur')
plt.legend()
plt.grid(True)
plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

```
La discrépance étoile est 0.1185.
```

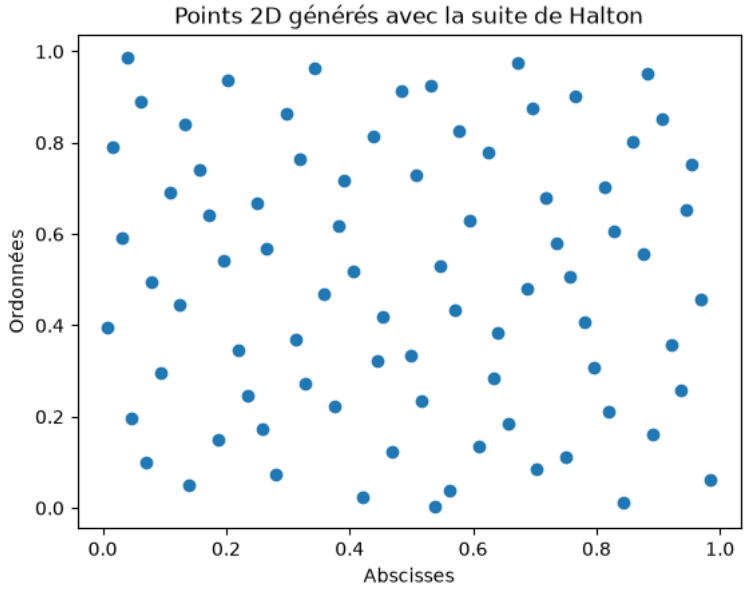



```
<Figure size 640x480 with 1 Axes>
```

In [16] :

```
# Générer les premières valeurs de la suite de Halton 2D
dimension = 2
size_sequence = 81 # Nombre de points dans la suite
sequence_2D = halton_sequence_dimension(size_sequence,dimension)

# Afficher les points générés
plt.scatter(sequence_2D[:, 0], sequence_2D[:, 1], marker='o')
plt.title('Points 2D générés avec la suite de Halton')
plt.xlabel('Abscisses')
plt.ylabel('Ordonnées')
plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

4- Calcul d'intégrales

$\blacktriangleright$ Question 5 [code] : Ecrire une fonction pour calculer l'intégrale d'une fonction quelconque sur l'intervalle $[0,1]$ à l'aide d'une suite

donnée.

In [17] :

```
        ####################################################
        # Réponse 5 [code] :
        ####################################################
        def compute_integral(func, sequence):
        sequence = np.asarray(sequence)
        try:
        vals = func(sequence)
        vals = np.asarray(vals, dtype=float)
        if vals.shape != sequence.shape:
        raise ValueError
        return float(np.mean(vals))
        except Exception:
        return float(np.mean([func(x) for x in sequence]))
```

On définit la fonction `square` suivante :

In [18] :

```
        def square(x):
        return x*x
```

$\blacktriangleright$ Question 6 [code] : On définit

$$ I_N(x,f) := \frac{1}{N}\sum_{n=0}^N f(x_n) $$

les intégrales approchées utilisant les N premiers points de la suite $x:=\{x_n\}_{n\geqslant 1}$. Tester votre code précédent en calculant l'intégrale approchée I_N de la fonction $square$ sur $[0,1]$ avec les $N=1000$ premiers points des différentes suites à discrépance faible construites précédemment.

In [19] :

```
        ####################################################
        # Réponse 6 [code] :
        ####################################################
        # Générer les 3 suites.
        N = 1000
        seq_random = np.random.random(N)
        seq_vdc = van_der_corput_sequence(N, base=2)
        seq_halton = halton_sequence(N, base=3)
        # Calculer les intégrales de square (exact = 1/3).
        result_random = compute_integral(square, seq_random)
        result_vandercorput = compute_integral(square, seq_vdc)
        result_halton = compute_integral(square, seq_halton)
        # Affichage des résultats
        print("Intégrales calculées:", result_random, result_vandercorput, result_halton)
        print("Valeur exacte = 1/3 =", 1/3)
        print(f"D* random={calculate_discrepancy_star(seq_random):.4f},
        VdC={calculate_discrepancy_star(seq_vdc):.4f},
        Halton={calculate_discrepancy_star(seq_halton):.4f}")
```

```
Intégrales calculées: 0.3141221618934767 0.332109169960022 0.33178639878284805
Valeur exacte = 1/3 = 0.3333333333333333
D* random=0.0361, VdC=0.0025, Halton=0.0031
```

$\blacktriangleright$ Question 7 [code] : Illustrer les vitesses de convergence de $I_N(x,f)$ vers $\int_0^1 f(u) du$ en fonction du nombre N de points de chaque suite x.

In [20] :

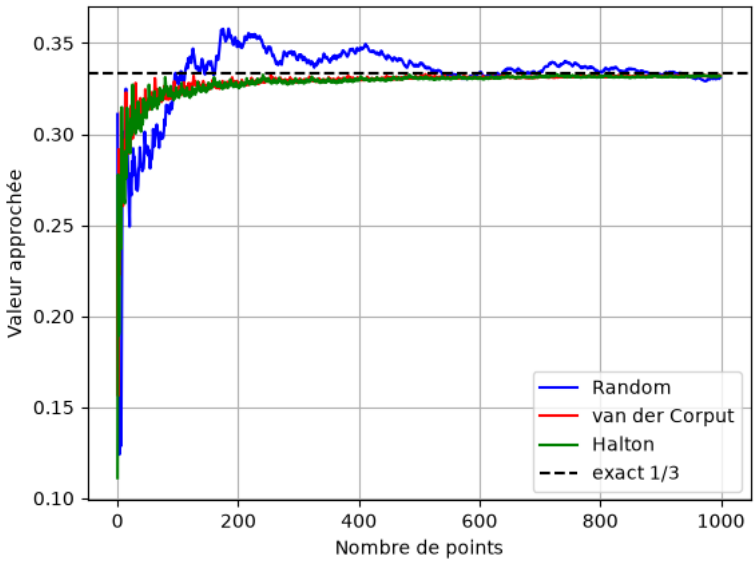
```
        ####################################################
        # Réponse 7 [code] :
        ####################################################
        N = 1000

        func = square

        results_random = np.zeros(N)
        results_van_der_corput = np.zeros(N)
        results_halton = np.zeros(N)
        sequence_random = np.random.random(N)
        sequence_van_der_corput = van_der_corput_sequence(N)
        sequence_halton = halton_sequence(N)

        # moyennes cumulatives I_n = (1/n) sum_{k<=n} f(x_k)
        cr = np.cumsum([func(x) for x in sequence_random])
        cv = np.cumsum([func(x) for x in sequence_van_der_corput])
        ch = np.cumsum([func(x) for x in sequence_halton])
        results_random = cr / np.arange(1, N + 1)
        results_van_der_corput = cv / np.arange(1, N + 1)
        results_halton = ch / np.arange(1, N + 1)

        plt.plot(results_random, 'b-', label='Random')
        plt.plot(results_van_der_corput, 'r-', label='van der Corput')
        plt.plot(results_halton, 'g-', label='Halton')
        plt.axhline(1/3, color='k', linestyle='--', label='exact 1/3')
        plt.xlabel('Nombre de points')
        plt.ylabel('Valeur approchée')
        plt.legend()
        plt.grid(True)
        plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

$\blacktrianglerightright$ Question 8 [code] : Calculer l'intégrale de la fonction Wing weight avec une suite aléatoire et la suite de Halton en dimension 10. En premier lieu, on généralisera le calcul de l'intégrale uni-dimensionnelle au cas d'une intégrale multiple sur $[0,1]^d$ en dimension d.

In [21] :

```
# Renormalisation des paramètres dans l'hypercube [0,1]^d
def wing_weight(x):
S_w = 150 + (200-150)*x[0]
W_fw = 220 + (300-220)*x[1]
A = 6 + (10-6)*x[2]
lambda_majus = (-10 + (10-(-10))*x[3])*np.pi/180
q = 16+(45-16)*x[4]
lambda_minus = 0.5 + (1-0.5)*x[5]
t_c = 0.08 + (0.18-0.08)*x[6]
N_x = 2.5 + (6-2.5)*x[7]
W_dg = 1700 + (2500-1700)*x[8]
W_p = 0.025 + (0.08-0.025)*x[9]
return 0.036 * S_w**0.758 * W_fw**0.0035 * (A / (np.cos(lambda_majus)**2))**0.6 * q**0.006
* lambda_minus**0.04 * (100 * t_c / np.cos(lambda_majus))**(-0.3) * (N_x * W_dg)**0.49 +
S_w * W_p
```

In [22] :

```
#####################################################
# Réponse 8 [code] :
#####################################################
# Fonction pour calculer une intégrale sur l'hypercube [0,1]^d
def compute_integral_dimension(func, sequence):
sequence = np.asarray(sequence)
vals = np.array([func(row) for row in sequence], dtype=float)
return float(np.mean(vals))
```

In [23] :

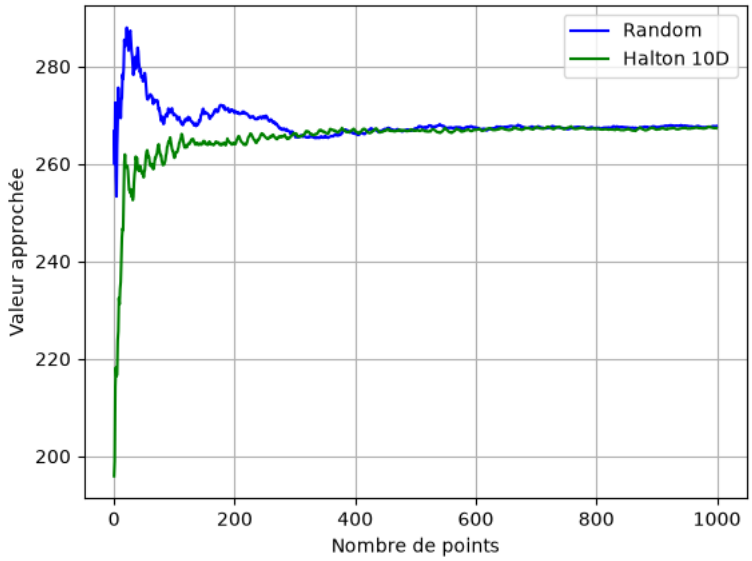
```
#####################################################
# Réponse 8 [code] : convergence wing weight MC vs Halton
#####################################################
N = 1000
d = 10

func = wing_weight

results_random = np.zeros(N)
results_halton = np.zeros(N)
sequence_random = np.random.random((N, d))
sequence_halton = halton_sequence_dimension(N, d)

vals_r = np.array([func(row) for row in sequence_random])
vals_h = np.array([func(row) for row in sequence_halton])
results_random = np.cumsum(vals_r) / np.arange(1, N + 1)
results_halton = np.cumsum(vals_h) / np.arange(1, N + 1)

plt.plot(results_random, 'b-', label='Random')
plt.plot(results_halton, 'g-', label='Halton 10D')
plt.xlabel('Nombre de points')
plt.ylabel('Valeur approchée')
plt.legend()
plt.grid(True)
plt.show()
print(f"MC final={results_random[-1]:.2f}, Halton final={results_halton[-1]:.2f}")
print("Halton converge plus vite/regulierement (QMC en O((log N)^d/N) vs MC en
O(1/sqrt(N))).")
```




```
<Figure size 640x480 with 1 Axes>
```

```
MC final=267.77, Halton final=267.36
Halton converge plus vite/regulierement (QMC en O((log N)^d/N) vs MC en O(1/sqrt(N))).
```

Les moyennes cumulatives de Halton semblent suivre une tendance monotone à mesure que n augmente. Une explication possible est que les moyennes cumulées des entrées ont tendance à s'approcher de 0,5 par le bas et que la fonction de poids de l'aile augmente de manière monotone dans la plupart de ses entrées. Peut-être qu'un échantillonnage antithétique avec la séquence de Halton améliorerait l'estimation du poids moyen des ailes.

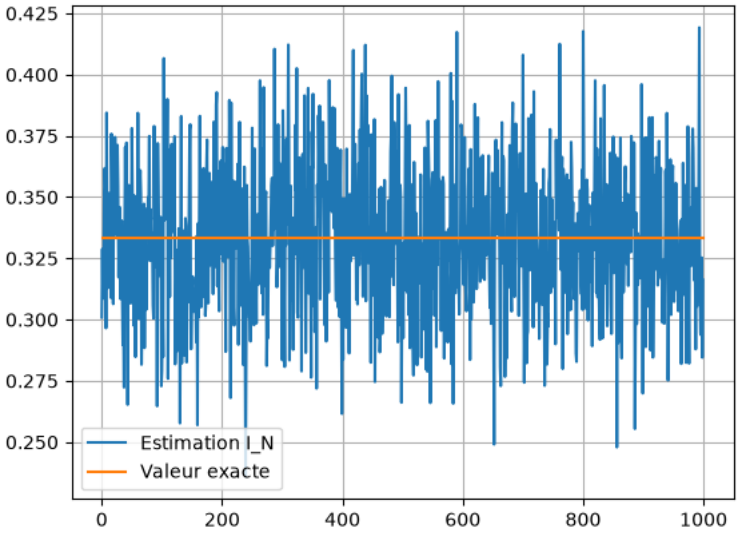
$\blacktriangleright$ Question 9 [math+code] : Dans le cas d'une suite aléatoire, pouvez-vous donner un ordre de grandeur du comportement de I_N lorsque N tend vers $+\infty$? Simulez un échantillon de taille M de variables aléatoires I_N avec N et M grands pour vérifier votre conjecture. On pourra s'appuyer sur les résultats fournis par la loi forte des grand nombres et le théorème central limite.

In [24] :

```
        ######################################################
        # Réponse 9 [code] : LFGN pour I_N (suite aleatoire)
        ######################################################
        N = 100 # Taille de la suite
        M = 1000 # echantillon Monte-Carlo
        I_N = np.zeros(M)

        # M realisations independantes de I_N = moyenne de N Uniformes passes dans square
        U = np.random.random((M, N))
        I_N = np.mean(U**2, axis=1)

        exact = 1/3
        ######################################################
        # Affichage des valeurs aléatoires
        plt.plot(I_N, label="Estimation I_N")
        plt.plot([1,M],[exact,exact],label="Valeur exacte")
        plt.legend(loc="best"); plt.grid(True); plt.show()
        print(f"LFGN : I_N -> E[f(X)]=1/3. Moyenne observee={I_N.mean():.4f}")
```



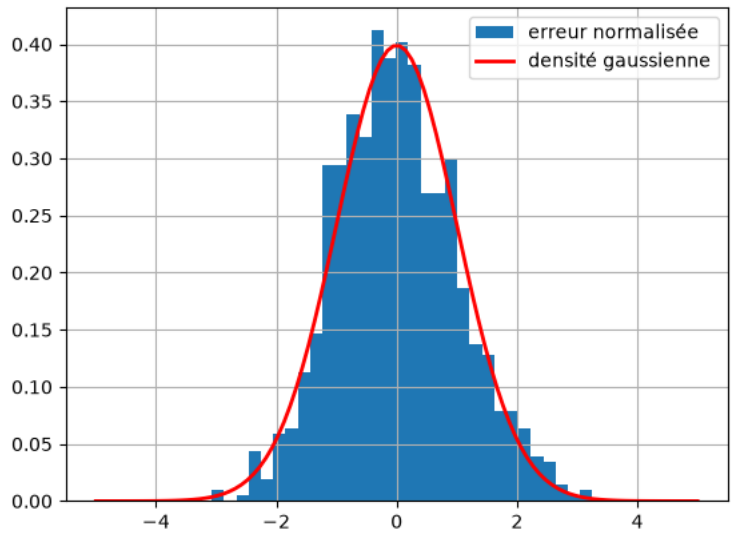

```
<Figure size 640x480 with 1 Axes>
```

```
LFGN : I_N -> E[f(X)]=1/3. Moyenne observee=0.3335
```

In [25] :

```
        ######################################################
        # Réponse 9 [code] : TCL
        ######################################################
        N = 100
        M = 1000
        U = np.random.random((M, N))
        I_N = np.mean(U**2, axis=1)
        # Var(f(X)) = 1/5-1/9 = 4/45
        sigma2 = 1/5 - 1/9
        erreurNormalisee_N = (I_N - 1/3) / np.sqrt(sigma2 / N)

        ######################################################
        # Affichage
        plt.hist(erreurNormalisee_N, density="True", bins=int(np.sqrt(M)), label="erreur
        normalisée")
        x = np.linspace(-5,5,100)
        densiteGaussienne = np.exp(-x*x/2.) / np.sqrt(2. * np.pi)
        plt.plot(x,densiteGaussienne, color="red", label="densité gaussienne", linewidth=2.0)
        plt.legend(loc="best"); plt.grid(True); plt.show()
```




```
<Figure size 640x480 with 1 Axes>
```

$\blacktrianglerightQuestion 10 [math+code] ($\textcolor{red}{optional}$) : Dans le cas de la suite de Halton ou de van der Corput, pouvez-vous donner un ordre de l'ordre de convergence de I_N lorsque N tend vers $+\infty$

In [26] :

```
########################################################
# Réponse 10 [code] : Koksma-Hlawka pour f(x)=x^2
########################################################
# Module de continuite borne par V(f)*D* ; ici V(f)=1 (variation totale de x^2 sur [0,1]).
def continuity_modulus(delta):
return delta # |x^2-y^2|<=2|x-y| ; borne simple

########################################################
N = 100 # Taille maximale

I_N = np.zeros(N)
D_N = np.zeros(N)
MD_N = np.zeros(N)

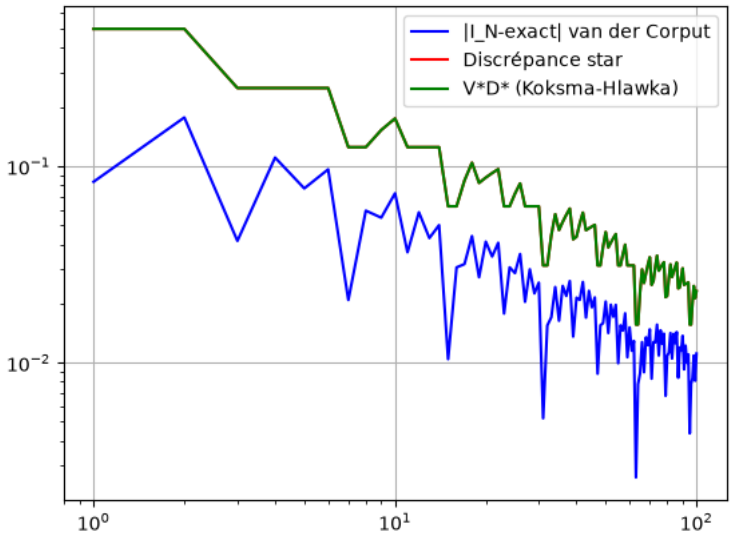
sequence_van_der_corput = van_der_corput_sequence(N)

for n in range(1, N+1):
sub = sequence_van_der_corput[:n]
I_N[n-1] = np.mean(sub**2)
d = calculate_discrepancy_star(sub)
D_N[n-1] = d
MD_N[n-1] = 1.0 * d # V(f)=1 -> borne de Koksma-Hlawka

exact = 1/3

########################################################
plt.loglog(np.arange(1,N+1), abs(I_N-exact)+1e-16, color="blue", label="|I_N-exact| van
der Corput")
plt.loglog(np.arange(1,N+1), D_N+1e-16, color="red", label="Discrépance star")
plt.loglog(np.arange(1,N+1), MD_N+1e-16, color="green", label="V*D* (Koksma-Hlawka)")

plt.legend(loc="best"); plt.grid(True); plt.show()
print("QMC : erreur en O((log N)/N) majoree par V(f).D*_N (Koksma-Hlawka), vs O(1/sqrt(N))
en MC.")
```




```
<Figure size 640x480 with 1 Axes>
```

```
QMC : erreur en O((log N)/N) majoree par V(f).D*_N (Koksma-Hlawka), vs O(1/sqrt(N)) en MC.
```

La dernière étape consiste à ajouter un comportement aléatoire à la suite de discrépance faible pour bénéficier de nouveau des intervalles de confiance. Dans le temps restant, vous pouvez développer une version "randomisée" des suites à discrépance faible et étudier leur comportement asymptotique.