CC3 Optimisation numérique
2023-24 : corrigé

Ex1) 6 pts

1) Soit $v \in \mathbb{R}^n \setminus \{0\}$, Comme $\mathcal{D}$ engendre positivement $\mathbb{R}^n$, on a

$$v = \sum_{i=1}^{N} \lambda_i d_i \quad \text{avec}$$

$\lambda_i \geq 0$ et $d_i \in \mathcal{D}$. En particulier

$$\langle v, v \rangle = \sum_{i=1}^{N} \lambda_i \underbrace{\langle v, d_i \rangle}_{(\equiv v^T d_i)} > 0$$

On a donc nécessairement l'existence de $d \in \mathcal{D}$ tel que $\langle v, d \rangle > 0$ (raisonner par l'absurde). En appliquant ce resultat à $-v$, on a aussi l'existence de $d \in \mathcal{D}$ tel que $\langle v, d \rangle < 0$

2 pts

2) On a déjà avec la question précedente (avec $v \to -v$)

$$\max_d (v^T d) > 0, \quad \forall v \in S^1$$

De plus, la fonction

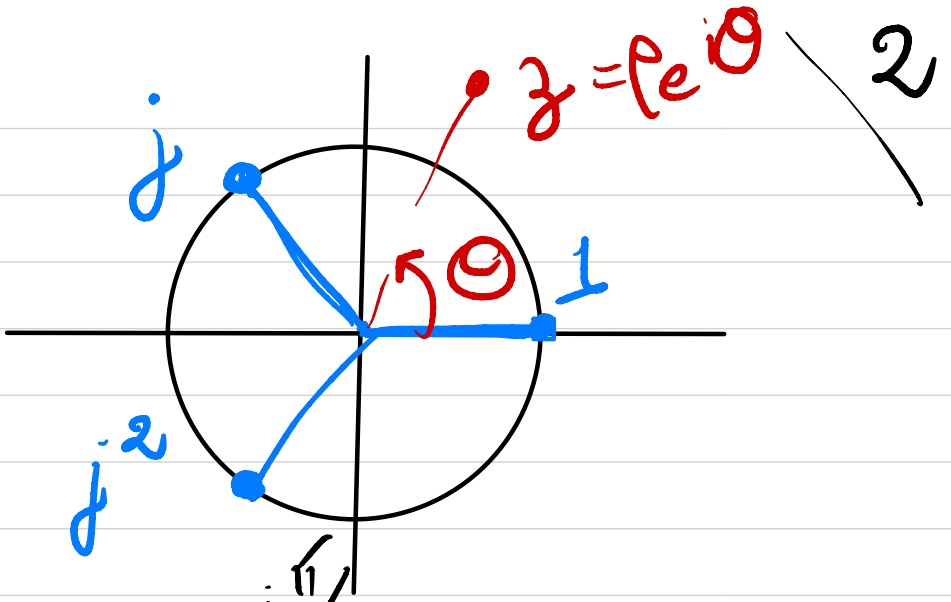
$\varphi\left(\begin{array}{l} S^1 \to \mathbb{R} \\ v \mapsto \max(v^T d) \end{array}\right)$ est C^∞

Comme $S^1 = \{v \in \mathbb{R}^n, \|v\| = 1\}$ est compact, φ atteint ses bornes.

On a donc $K = \min_{v \in S^1}(\varphi) > 0$.

2 pts

3) On a $z = \rho e^{i\theta}$. En fonction de θ ($\theta \in [0, \frac{2\pi}{3}]$ ou $[\frac{2\pi}{3}, \frac{4\pi}{3}]$ ou $[\frac{4\pi}{3}, 2\pi]$), on peut décomposer z (par rapport à $(1, j)$, (j, j^2) ou $(1, j)$ respectivement) avec des coefficients positifs. 1pt



Pour $z = e^{i\frac{\pi}{3}}$, on a

$\begin{cases} \langle e^{i\frac{\pi}{3}}, 1 \rangle = \cos(\frac{\pi}{3}) = \frac{1}{2} \\ \langle e^{i\frac{\pi}{3}}, j \rangle = \frac{1}{2} \\ \langle e^{i\frac{\pi}{3}}, j^2 \rangle = -1 \end{cases}$

Il s'agit de la valeur minimale de $\max \langle z, d \rangle$ (voir dessin) $d \in D$

et $K = \frac{1}{2}$ 1pt

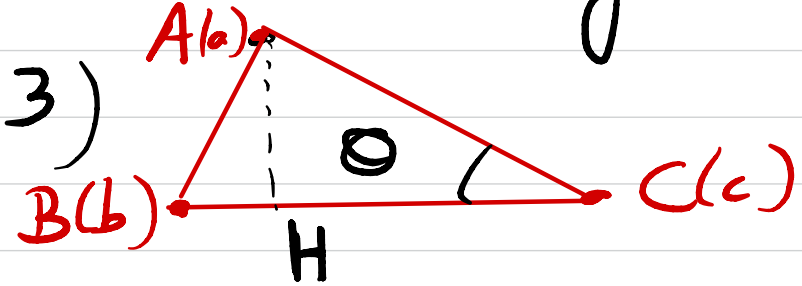
2

Ex 2 | 4 pts

1) La suite $(f_k)_k$ est décroissante (ceci est vrai pour tous les cas, y compris pour un rétrécissement). Comme elle est minorée, elle est convergente. 1 pt

2) Pour k assez grand, aucun rétrécissement ne se produit. A partir de cette valeur, toutes les suites $(f_k^i)_k$ sont décroissantes. En effet, le simplexe est toujours amélioré, étant donné que le nouveau point qui entre dans le simplexe est meilleur que le plus mauvais point du simplexe en cours, qui en sort. Avec le même raisonnement qu'en 1), toutes les suites sont convergentes. 1 pt

3)



On a $|\det(a-c, b-c)|$

$= \|\overrightarrow{CA} \wedge \overrightarrow{CB}\|$

$= \|\overrightarrow{CA}\| . \|\overrightarrow{CB}\| \sin(\Theta)$

et $\frac{1}{2}|\det(a-c, b-c)|$ 1pt

$= \frac{1}{2} CB.AH = Vol(ABC)$

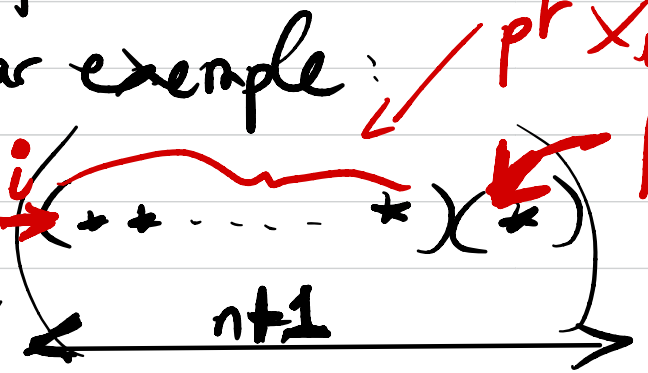
4) En cas de rétrécissement, on a un simplexe dont les arêtes sont réduites de moitié. Ainsi

$$Vol(S_{k+1}) = \left(\frac{1}{2}\right)^n Vol(S_k)$$ 1pt

Ex3 7pts (toutes les questions à 1 pt)

1) Le script utilise une stratégie d'évolution avec pas adaptatif et sélection sur les "enfants" pour rechercher le minimum de la fonction de Rosenbrock.

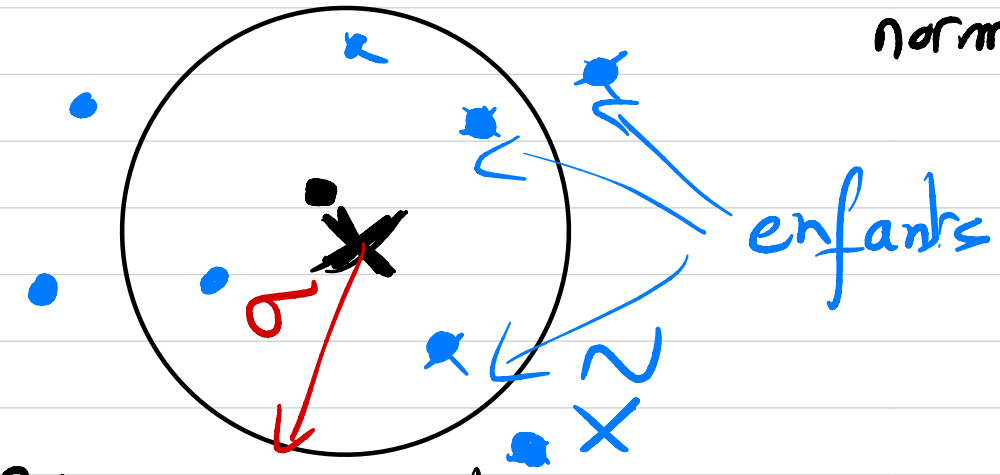
2) Les matrices Ap et Ae contiennent à chaque génération les informations relatives aux "parents" (Ap en nombre mu) et "enfants" (Ae en nombre lmbda) sous la forme d'une matrice. On a par exemple :

$A_p =$ 

3) On a

$Ae[i,n] = f(Ae[i:n])$

4) Sigma représente un pas (ou une intensité) dans la génération des nouveaux éléments suivant une loi normale.



Plus précisément

$\tilde{X} = X + \mathcal{N}(0, \sigma^2)$

(σ : variance de la loi normale)

σ va évoluer en fonction des taux de succès des mutations ($\sigma < \frac{1}{5} \rightarrow \sigma$ décroit)

5) On classe les termes de [3.3, 4, 2, 6, 1] du plus petit au plus grand et on donne les positions dans a.

Ici, a = np.array [4, 2, 0, 1, 3]

6) La méthode n'est pas élitiste puisque on ne garde pas le meilleur des parents à la génération suivante.

7) On peut proposer la sélection "parents + enfants"

Dans ce cas, il faut par exemple remplacer A_e par un tableau contenant "parents + enfants" à chaque génération, par exemple A_{pe} :

$A_{pe} = np.zeros((lmbda+mu, n+1))$

$A_{pe}(:mu, :) = A_p$

et

$A_{pe}(mu+i, :n) = \ldots$

$A_{pe}(mu+i, :n) = \ldots$

Ex 4 — 5 pts — /6

```
n = 2
Nstep= 50
Ngen = 200
sigma = 0.1
it=1
# Rosenbrock function
def rosen(x):
    return 100 * (x[0,1] - x[0,0] ** 2) ** 2 + (x[0,0] - 1) **
# Initialization
x= np.random.rand(1,n)
BestX = x
for k in range(Ngen):
    tau = 0
    for i in range(Nstep):
        it=it+1
        y = x + sigma * np.random.normal(0, 1, size=n)
        fx=rosen(x)
        fy=rosen(y)
        p=np.random.rand()
        pr=np.exp(-(fy-fx)*np.log(it))
        if fy< fx:
            tau = tau+ 1/Nstep
            BestX=y
        if (p<pr):
            x=y
    sigma = np.exp(1 / 3 * (tau - 1 / 5) / (1 - 1 / 5)) * sigma
print(BestX)
```