

TP1 – Computational Statistics

Coërchon Colin

1^{er} novembre 2024

Table des matières

1	Exercice 1	1
2	Exercice 2	5

1 Exercice 1

1. On considère les variables aléatoires $X = R \cos(\Theta)$ et $Y = R \sin(\Theta)$, où R est une variable aléatoire suivant une loi de Rayleigh avec densité

$$f_R(r) = r \exp\left(-\frac{r^2}{2}\right) \mathbb{1}_{\mathbb{R}^+}(r),$$

et Θ est uniformément distribuée sur $[0, 2\pi]$. On suppose que R et Θ sont indépendantes. Nous souhaitons montrer que X et Y sont indépendantes et que chacune suit une loi normale $\mathcal{N}(0, 1)$.

Pour ce faire, considérons une fonction $g : \mathbb{R}^2 \rightarrow \mathbb{R}$ bornée et mesurable. Nous calculons alors l'espérance $\mathbb{E}[g(X, Y)]$:

$$\mathbb{E}[g(X, Y)] = \int_{\mathbb{R}^2} g(x, y) f_{X,Y}(x, y) \, dx \, dy$$

On applique alors le changement de variable $(x, y) \mapsto (r, \theta)$ défini par l'inverse de la fonction F :

$$F : \begin{cases} \mathbb{R}_+ \times [0, 2\pi] & \longrightarrow \mathbb{R}^2 \\ (r, \theta) & \longmapsto (r \cos \theta, r \sin \theta) \end{cases}$$

La matrice jacobienne associée est alors :

$$J_F(r, \theta) = \begin{pmatrix} \cos \theta & -r \sin \theta \\ \sin \theta & r \cos \theta \end{pmatrix}$$

avec $|J_F(r, \theta)| = r$.

On a donc :

$$f_{X,Y}(x, y) \, dx \, dy = f_{R,\Theta}(r, \theta) \times |J_F(r, \theta)|^{-1} \, dr \, d\theta = f_{R,\Theta}(r, \theta) \times \frac{1}{r} \, dr \, d\theta$$

Or, R et Θ sont indépendantes, donc la densité conjointe de (R, Θ) est donnée par :

$$f_{R,\Theta}(r, \theta) = f_R(r) \cdot f_\Theta(\theta) = r \exp\left(-\frac{r^2}{2}\right) \cdot \frac{1}{2\pi}$$

Ainsi, l'espérance $\mathbb{E}[g(X, Y)]$ devient :

$$\begin{aligned} \mathbb{E}[g(X, Y)] &= \int_0^{2\pi} \int_0^{+\infty} g(r \cos(\theta), r \sin(\theta)) \cdot \frac{1}{2\pi} r \exp\left(-\frac{r^2}{2}\right) \cdot \frac{1}{r} dr d\theta \\ &= \int_0^{2\pi} \int_0^{+\infty} g(r \cos(\theta), r \sin(\theta)) \cdot \frac{1}{2\pi} \exp\left(-\frac{r^2}{2}\right) dr d\theta \end{aligned}$$

Et donc, pour conclure (en sachant que $r = \sqrt{x^2 + y^2}$), on a :

$$\begin{aligned} \mathbb{E}[g(X, Y)] &= \int_{\mathbb{R}^2} g(x, y) \frac{1}{2\pi} \exp\left(-\frac{x^2 + y^2}{2}\right) dx dy \\ &= \int_{\mathbb{R}^2} g(x, y) \underbrace{\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)\right)}_{f_X(x)} \underbrace{\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right)\right)}_{f_Y(y)} dx dy \end{aligned}$$

Cela nous permet de conclure que $(X, Y) \sim \mathcal{N}(0_2, I_2)$, et donc :

$$\boxed{X \sim \mathcal{N}(0, 1), \quad Y \sim \mathcal{N}(0, 1) \quad \text{et} \quad X \perp\!\!\!\perp Y}$$

2. On a :

$$\begin{aligned} \forall r \in \mathbb{R}_+, \quad F_R(r) &= \mathbb{P}(R \leq r) \\ &= \int_0^r x \exp\left(-\frac{x^2}{2}\right) dx = \left[-\exp\left(-\frac{x^2}{2}\right)\right]_0^r = 1 - \exp\left(-\frac{r^2}{2}\right) \end{aligned}$$

Or, F_R est continue et strictement croissante sur $\mathbb{R}_+$, la méthode de la transformée inverse nous donne alors :

$$\text{Si } U \sim \mathcal{U}([0, 1]), \quad \text{alors } F_R^{-1}(U) \sim R$$

Or,

$$\forall u \in [0, 1], \quad F_R^{-1}(u) = \sqrt{-2 \ln(1 - u)}.$$

Et on sait que $1 - U \sim \mathcal{U}([0, 1]) \iff U \sim \mathcal{U}([0, 1])$, donc on peut prendre $\sqrt{-2 \ln(u)}$.

De plus, si $U \sim \mathcal{U}([0, 1])$, alors $2\pi U \sim \mathcal{U}([0, 2\pi])$.

Ainsi, on peut en déduire un algorithme pour simuler indépendamment des variables $X \sim \mathcal{N}(0, 1)$. Cet algorithme s'appelle l'**algorithme de Box-Muller**.

3. (a) À la fin de la boucle "while", comme $U_1, U_2 \sim \mathcal{U}([0, 1])$ et que $V_1^2 + V_2^2 \leq 1$, on en déduit que :

$$\boxed{(V_1, V_2) \sim \mathcal{U}(D)}$$

avec D le disque unité sur $\mathbb{R}^2$ défini par $D \triangleq \{(x, y) \in [-1, 1]^2 \mid x^2 + y^2 \leq 1\}$.

Algorithm 1 Algorithme de Box-Muller

1: Générer deux variables aléatoires indépendantes U_1 et U_2 uniformément distribuées sur $[0, 1]$.

2: Calculer :

$$Z_1 = \sqrt{-2 \ln(U_1)} \cos(2\pi U_2)$$

$$Z_2 = \sqrt{-2 \ln(U_1)} \sin(2\pi U_2)$$

3: Retourner Z_1 et Z_2 , qui sont des variables indépendantes suivant une loi $\mathcal{N}(0, 1)$.

(b) On note Y la variable aléatoire qui donne le nombre de tour dans la boucle "while" avant d'en sortir. On peut apparenter les tours dans la boucle "while" comme des **épreuves de Bernoulli** indépendantes et le fait d'en sortir comme le succès de probabilité $p \in]0, 1[$.

Ainsi, Y suit une loi géométrique de paramètre p , où $p = \mathbb{P}(V_1^2 + V_2^2 \leq 1)$. Comme $V_1, V_2 \sim \mathcal{U}([-1, 1])$, la probabilité p peut se calculer très simplement :

$$p = \mathbb{P}(V_1^2 + V_2^2 \leq 1) = \frac{\text{aire}(D)}{\text{aire}([0, 1]^2)} = \frac{\pi}{4}$$

Donc, $Y \sim \mathcal{G}\left(\frac{\pi}{4}\right)$.

Ainsi, on a facilement :

$$\mathbb{E}[Y] = \frac{1}{p} = \frac{4}{\pi}$$

C'est-à-dire qu'il y a en moyenne de $\frac{\pi}{4} \approx 1.27$ tours dans la boucle "while".

(c) On pose :

$$T_1 = \frac{V_1}{\sqrt{V_1^2 + V_2^2}} \quad \text{et} \quad V = V_1^2 + V_2^2.$$

1. Distribution de V en utilisant la méthode de la fonction muette

Pour obtenir la distribution de $V = V_1^2 + V_2^2$, considérons une fonction g bornée et mesurable, et calculons $\mathbb{E}[g(V)]$.

L'espérance de $g(V)$ est donnée par :

$$\mathbb{E}[g(V)] = \iint_D g(V_1^2 + V_2^2) f_{V_1, V_2}(v_1, v_2) dv_1 dv_2,$$

où $f_{V_1, V_2}(v_1, v_2) = \frac{1}{\pi}$ car $(V_1, V_2) \sim \mathcal{U}(D)$, avec une densité uniforme de $\frac{1}{\pi}$ sur le disque unité.

Passons en coordonnées polaires (r, θ) avec $r = \sqrt{v_1^2 + v_2^2}$ et $\theta = \tan^{-1}\left(\frac{v_2}{v_1}\right)$, où $r \in [0, 1]$ et $\theta \in [0, 2\pi]$. Le jacobien de cette transformation est r , donc :

$$dv_1 dv_2 = r dr d\theta$$

L'espérance devient alors :

$$\mathbb{E}[g(V)] = \int_0^{2\pi} \int_0^1 g(r^2) \cdot \frac{1}{\pi} \cdot r dr d\theta$$

Intégrons par rapport à θ , en notant que $\int_0^{2\pi} \frac{1}{\pi} d\theta = 1$, et simplifions :

$$\mathbb{E}[g(V)] = \int_0^1 g(r^2) \cdot 2r dr$$

En posant $u = r^2$, de sorte que $du = 2r dr$, on obtient :

$$\mathbb{E}[g(V)] = \int_0^1 g(u) du$$

Cela montre que $V = V_1^2 + V_2^2$ suit une distribution uniforme sur $[0, 1]$, car l'espérance de $g(V)$ est identique à celle d'une variable uniforme.

Donc :

$$\boxed{V \sim \mathcal{U}([0, 1])}$$

2. Distribution de T_1 en utilisant la méthode de la fonction muette

Pour la distribution de $T_1 = \frac{V_1}{\sqrt{V_1^2 + V_2^2}}$, nous appliquons aussi la méthode de la fonction muette. Remarquons que T_1 est en fait le cosinus de l'angle Θ que fait le vecteur (V_1, V_2) avec l'axe V_1 .

Soit une fonction h bornée et mesurable, nous voulons calculer $\mathbb{E}[h(T_1)]$.

Nous avons :

$$\mathbb{E}[h(T_1)] = \iint_D h\left(\frac{v_1}{\sqrt{v_1^2 + v_2^2}}\right) f_{V_1, V_2}(v_1, v_2) dv_1 dv_2$$

En passant en coordonnées polaires (r, θ) avec $v_1 = r \cos(\theta)$ et $v_2 = r \sin(\theta)$, et en utilisant le jacobien r , on obtient :

$$\mathbb{E}[h(T_1)] = \int_0^{2\pi} \int_0^1 h(\cos \theta) \cdot \frac{1}{\pi} \cdot r dr d\theta$$

En intégrant par rapport à r , on trouve :

$$\mathbb{E}[h(T_1)] = \int_0^{2\pi} h(\cos \theta) \cdot \frac{1}{\pi} \cdot \frac{1}{2} d\theta = \int_0^{2\pi} h(\cos \theta) \cdot \frac{1}{2\pi} d\theta$$

Cette intégrale est l'espérance de $h(\cos(\Theta))$ où Θ est uniformément distribué sur $[0, 2\pi]$. Cela montre que $T_1 = \cos(\Theta)$ avec $\Theta \sim \mathcal{U}([0, 2\pi])$, et donc :

$$\boxed{T_1 \sim \cos(\Theta) \text{ avec } \Theta \sim \mathcal{U}([0, 2\pi])}$$

3. Indépendance de T_1 et V

Pour démontrer que V et T_1 sont indépendantes, nous allons analyser leurs expressions en coordonnées polaires.

Nous avons que (V_1, V_2) est uniformément distribué dans le disque unité $D \subset \mathbb{R}^2$. En passant en coordonnées polaires, posons :

$$r = \sqrt{V_1^2 + V_2^2} \quad \text{et} \quad \theta = \tan^{-1}\left(\frac{V_2}{V_1}\right),$$

de sorte que $V_1 = r \cos(\theta)$ et $V_2 = r \sin(\theta)$.

Nous pouvons alors exprimer T_1 et V en fonction de r et θ :

$$T_1 = \frac{V_1}{\sqrt{V_1^2 + V_2^2}} = \cos(\theta)$$

$$V = V_1^2 + V_2^2 = r^2$$

Ainsi :

- $T_1 = \cos(\theta)$ dépend uniquement de l'angle θ .
- $V = r^2$ dépend uniquement de la norme r .

Comme r et θ sont indépendants dans une distribution uniforme sur le disque, il en résulte que :

$$\boxed{T_1 \text{ et } V \text{ sont également indépendantes.}}$$

(d) On avait vu dans la question 2 que :

$$\text{Si } U \sim \mathcal{U}([0, 1]), \quad \text{alors } \sqrt{-2 \ln(U)} \sim \mathcal{R}(1)$$

où $\mathcal{R}(1)$ symbolise la loi de Rayleigh de paramètre 1.

Comme $V = V_1^2 + V_2^2 \sim \mathcal{U}([0, 1])$, on en déduit que :

$$\boxed{S = \sqrt{-2 \ln(V_1^2 + V_2^2)} \sim \mathcal{R}(1)}$$

De plus, avec $\Theta \sim \mathcal{U}([0, 2\pi])$, on sait d'après la question précédente que :

$$\boxed{T_1 = \frac{V_1}{\sqrt{V_1^2 + V_2^2}} \sim \cos(\Theta) \quad \text{et} \quad T_2 = \frac{V_2}{\sqrt{V_1^2 + V_2^2}} \sim \sin(\Theta)}$$

en reprenant les calculs pour T_2 par analogie avec T_1 .

Ainsi, comme $X = S T_1$ et $Y = S T_2$, par analogie avec la question 2, on en déduit très simplement que :

$$\boxed{(X, Y) \sim \mathcal{N}(0_2, I_2)}$$

2 Exercice 2

1. Pour prouver que le noyau de transition $P(x, A)$ est bien celui de la chaîne de Markov définie dans l'énoncé, nous devons vérifier trois conditions.

On doit effectivement vérifier $\forall A \in \mathcal{B}([0, 1])$:

- $x \mapsto P(x, A)$ est mesurable.
- $\forall x, P(x, \cdot)$ est une mesure de probabilité.
- $\mathbb{P}(X_{n+1} \in A \mid X_n = x) = P(x, A)$

On rappelle que le noyau de transition $P(x, A)$ est donné par :

$$P(x, A) = \begin{cases} x^2 \int_{A \cap [0, 1]} dt + (1 - x^2) \delta_{\frac{1}{k+1}}(A) & \text{si } x = \frac{1}{k} \\ \int_{A \cap [0, 1]} dt & \text{sinon} \end{cases}$$

où δ_α est la mesure de Dirac en α .

Étape 1 : Vérification de la mesurabilité de $x \mapsto P(x, A)$

Pour montrer que $x \mapsto P(x, A)$ est une fonction mesurable pour tout borélien $A \subset [0, 1]$, observons que $P(x, A)$ est défini différemment selon que $x = \frac{1}{k}$ pour un entier k ou non.

- Si $x = \frac{1}{k}$, alors $P(x, A) = x^2 \int_{A \cap [0, 1]} dt + (1 - x^2) \delta_{\frac{1}{k+1}}(A)$. Cette expression est une combinaison linéaire d'une intégrale et d'une masse de Dirac. Elle est donc mesurable en x car elle est constituée de fonctions continues et de fonctions indicatrices.
- Si $x \neq \frac{1}{k}$ pour tout entier k , alors $P(x, A) = \int_{A \cap [0, 1]} dt$, ce qui est constant par rapport à x et donc mesurable.

Étape 2 : Vérification que $P(x, \cdot)$ est une mesure de probabilité

- **Cas où $x = \frac{1}{k}$** : Dans ce cas, on a

$$P(x, [0, 1]) = x^2 \int_{[0, 1]} dt + (1 - x^2) \delta_{\frac{1}{k+1}}([0, 1]).$$

En évaluant chaque terme :

- $x^2 \int_{[0, 1]} dt = x^2 \times 1 = x^2$,
- $\delta_{\frac{1}{k+1}}([0, 1]) = 1$, donc $(1 - x^2) \delta_{\frac{1}{k+1}}([0, 1]) = 1 - x^2$.

Par conséquent,

$$P(x, [0, 1]) = x^2 + (1 - x^2) = 1.$$

Cela montre que $P(x, \cdot)$ est bien **une mesure de probabilité** lorsque $x = \frac{1}{k}$.

- **Cas où $x \neq \frac{1}{k}$ pour tout entier k** : Dans ce cas, $P(x, A) = \int_{A \cap [0, 1]} dt$, ce qui correspond à la mesure uniforme sur $[0, 1]$. Ainsi,

$$P(x, [0, 1]) = \int_{[0, 1]} dt = 1.$$

Par conséquent, $P(x, \cdot)$ est également une mesure de probabilité lorsque $x \neq \frac{1}{k}$.

Nous avons donc vérifié que $P(x, \cdot)$ est une mesure de probabilité pour chaque $x \in [0, 1]$.

Étape 3 : Calcul de $\mathbb{P}(X_{n+1} \in A \mid X_n = x)$

Pour prouver que $P(x, A)$ est le noyau de transition de la chaîne de Markov $(X_n)_{n \geq 0}$, nous devons montrer que la probabilité de transition $\mathbb{P}(X_{n+1} \in A \mid X_n = x)$ est égale à $P(x, A)$.

- **Cas où $x = \frac{1}{k}$ pour un entier $k \in \mathbb{N}^*$** : Dans ce cas, la transition est définie par deux événements possibles :

- Avec probabilité $1 - x^2 = 1 - \left(\frac{1}{k}\right)^2$, $X_{n+1} = \frac{1}{k+1}$.
- Avec probabilité $x^2 = \left(\frac{1}{k}\right)^2$, $X_{n+1} \sim \mathcal{U}([0, 1])$.

Donc, pour tout borélien $A \subset [0, 1]$, nous avons :

$$\begin{aligned} \mathbb{P}(X_{n+1} \in A \mid X_n = x) &= \left(\frac{1}{k}\right)^2 \int_{A \cap [0, 1]} dt + \left(1 - \frac{1}{k^2}\right) \int_{A \cap [0, 1]} \mathbb{1}_{\{\frac{1}{k+1}\}}(t) dt \\ &= \left(\frac{1}{k}\right)^2 \int_{A \cap [0, 1]} dt + \left(1 - \frac{1}{k^2}\right) \delta_{\frac{1}{k+1}}(A) \end{aligned}$$

Cette expression correspond bien à la formule donnée pour $P(x, A)$ dans le cas où $x = \frac{1}{k}$.

- **Cas où $x \neq \frac{1}{k}$** : Dans ce cas, $X_{n+1} \sim \mathcal{U}([0, 1])$, donc :

$$\mathbb{P}(X_{n+1} \in A \mid X_n = x) = \int_{A \cap [0, 1]} dt.$$

Cette expression correspond bien à la formule donnée pour $P(x, A)$ lorsque $x \neq \frac{1}{k}$.

Ainsi, nous avons vérifié que $\mathbb{P}(X_{n+1} \in A \mid X_n = x) = P(x, A)$ pour tout borélien $A \subset [0, 1]$.

Conclusion

$P(x, A)$ est bien le noyau de transition de la chaîne de Markov définie dans l'exercice.

2. L'objectif est de montrer que :

$$\forall A \in \mathcal{B}([0, 1]), \quad (\pi P)(A) = \pi(A)$$

En notant p la densité de π , qui est la distribution uniforme sur $[0, 1]$ on a :

$$\begin{aligned} (\pi P)(A) &= \int_{\mathbb{R}} p(A) P(x, A) dx \\ &= \int_0^1 P(x, A) dx && \text{(car } \pi \text{ est uniforme de densité } p(x) = 1) \\ &= \int_0^1 P(x, A) \mathbb{1}_Q(x) dx + \int_0^1 P(x, A) \mathbb{1}_{\overline{Q}}(x) dx && \text{(avec } Q = (\{\frac{1}{n}\})_{n \in \mathbb{N}^*}) \\ &= \int_0^1 P(x, A) \mathbb{1}_{\overline{Q}}(x) dx && \text{(car la mesure d'un point } \frac{1}{n} \text{ est nulle)} \\ &= \int_0^1 \int_{A \cap [0, 1]} dt dx && \text{(par def du noyau de transition } P) \\ &= \int_{A \cap [0, 1]} dx \\ &= \int_{\mathbb{R}} p(A) dx \\ &= \pi(A) \end{aligned}$$

□

3. Dans la question 2, nous avons prouvé que :

$$\forall A \in \mathcal{B}([0, 1]), \quad (\pi P)(A) = \pi(A),$$

où π est la mesure uniforme sur $[0, 1]$, de densité $p(x) = 1$ sur cet intervalle.

Calcul de $Pf(x)$

Pour $x \notin Q = \{\frac{1}{k}, k \in \mathbb{N}^*\}$, d'après la définition du noyau de transition $P(x, A) = \int_{A \cap [0, 1]} dt$, la variable X_1 suit une loi uniforme sur $[0, 1]$ indépendamment de $X_0 = x$. Par conséquent, pour une fonction mesurable bornée f , on a :

$$Pf(x) = \mathbb{E}[f(X_1) \mid X_0 = x] = \int_{[0, 1]} f(t) dt$$

Calcul de $P^n f(x)$ pour $n \geq 1$

Puisque $Pf(x) = \int_{[0,1]} f(t) dt$ pour tout $x \notin Q$, nous constatons que la chaîne de Markov se « mélange » immédiatement sur $[0, 1]$ pour ces valeurs de x . En d'autres termes, appliquer P plusieurs fois ne change pas le fait que X_n est uniformément distribué sur $[0, 1]$ dès $n = 1$.

Ainsi, pour tout $n \geq 1$, nous avons :

$$P^n f(x) = \int_{[0,1]} f(t) dt$$

Calcul de $\lim_{n \rightarrow \infty} P^n f(x)$

La limite $\lim_{n \rightarrow \infty} P^n f(x)$ correspond à l'espérance de f par rapport à la mesure invariante π . Puisque π est la loi uniforme sur $[0, 1]$, nous avons :

$$\lim_{n \rightarrow \infty} P^n f(x) = \int_{[0,1]} f(x) \pi(x) dx = \int_{[0,1]} f(x) dx$$

4. (a)
(b)

TP1_Colin_Coerchon

1^{er} Novembre 2024

1 TP 1 - Exercice 3

Élève : Colin Coërchon

```
[1]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import precision_score, recall_score, confusion_matrix
import seaborn as sns
```

1.1 Question 1

J'ai essayé de me baser sur les notations vues dans le cours :

- Hypothèse 0 : H espace de Hilbert et $\Theta_c \subset \Theta$ convexe fermé non vide.
- Hypothèse 1 : J convexe et différentiable sur Θ . $\theta^* \in \arg \min_{\theta \in \Theta_c} J(\theta)$
- Hypothèse 2 : On connaît g telle que : $\nabla J(\theta) = \mathbb{E}[g(\theta, W)]$

Algorithme du gradient stochastique :

1. **Input** : $\theta_0 \in \Theta_c$, (ε_k) décroît vers 0, la suite de pas

2. **Repeat** : $W_{k+1} \sim \mathbb{P}_W$ une réalisation suivant $\mathbb{P}_W$:

$$\theta^{k+1} = \text{proj}_{\Theta_c}(\theta^k - \varepsilon^k g(\theta^k, W_{k+1}))$$

3. **Stop** : critère d'arrêt : $\left| \frac{J(\theta^{k+1}) - J(\theta^k)}{J(\theta^k)} \right| < \text{tolérance}$

```
[73]: def stochastic_gradient_descent(grad_func, X, y, init_theta,
    ↪ learning_rate_schedule, max_iter=1000, tolerance=1e-6):
    n = X.shape[0]
    theta = init_theta
    history = [theta]

    for k in range(max_iter):
        i = np.random.randint(0, n)
        x_i = X[i]
        y_i = y[i]

        grad = grad_func(theta, x_i, y_i)
```

```

        epsilon_k = learning_rate_schedule(k)
        theta_next = theta - epsilon_k * grad

        if np.linalg.norm(theta_next - theta) / (np.linalg.norm(theta) + 1e-8) <
→tolerance:
            break

        theta = theta_next
        history.append(theta)

    return theta, history

```

1.2 Question 2

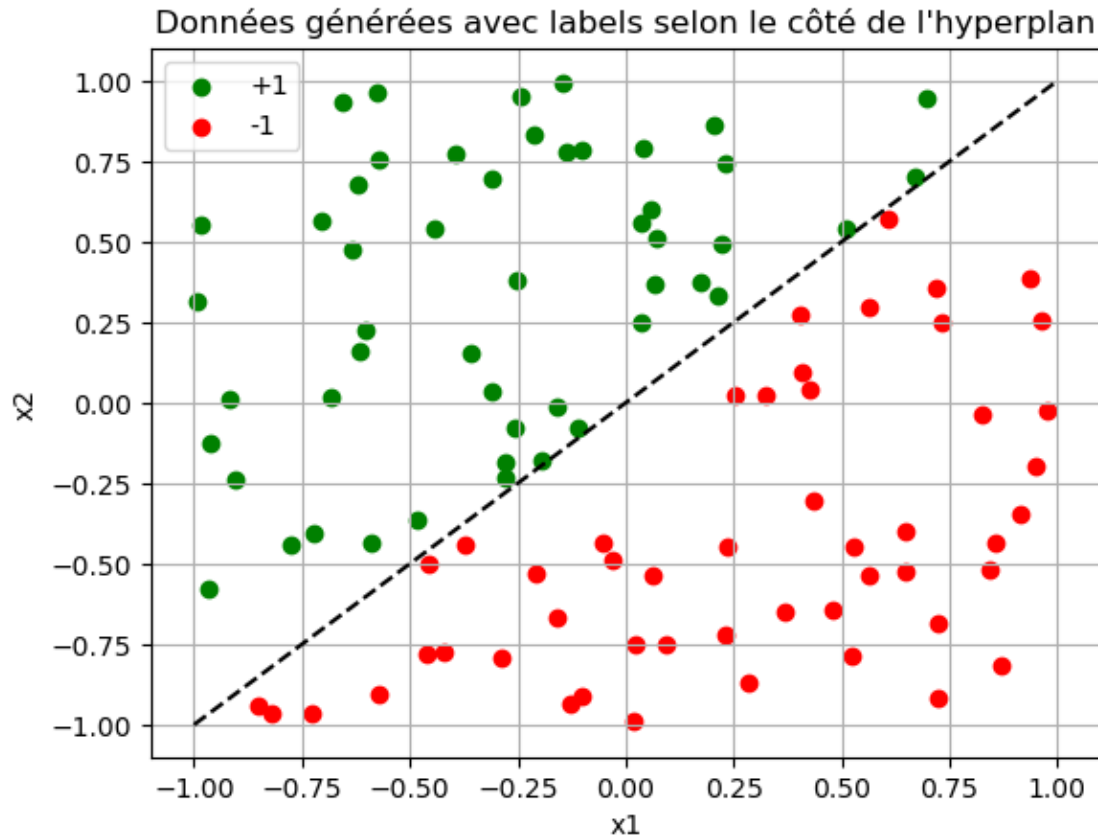
```

[74]: def generate_data(n, w_true):
        X = np.random.rand(n, 2) * 2 - 1
        y = np.sign(X @ w_true)
        return X, y

n = 100
w_true = np.array([-1, 1])
X, y = generate_data(n, w_true)

# Visualisation des données avec les deux classes
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='green', label='+1')
plt.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='red', label='-1')
x_vals = np.linspace(-1, 1, 100)
y_vals_real = x_vals # Hyperplan y = x
plt.plot(x_vals, y_vals_real, 'k--')
plt.legend()
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Données générées avec labels selon le côté de l'hyperplan")
plt.grid()
plt.show()

```



1.3 Question 3

```
[75]: def risk_gradient(theta, x, y):
    error = y - np.dot(theta, x)
    grad = -2 * error * x
    return grad

def learning_rate_schedule(k):
    return 0.01 / (1 + 0.0001 * k)

init_theta = np.random.uniform(-1, 1, 2)    # Initialisation aléatoire

theta_opt, theta_history = stochastic_gradient_descent(
    grad_func=risk_gradient,
    X=X,
    y=y,
    init_theta=init_theta,
    learning_rate_schedule=learning_rate_schedule
)
```

```

print("Vecteur w* estimé :", theta_opt)
print("Vecteur w~ réel :", w_true)
print("Distance entre w* et w~ :", np.linalg.norm(theta_opt - w_true))

plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='red', label='+1')
plt.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='green', label='-1')

x_vals = np.linspace(-1, 1, 100)
y_vals_real = x_vals
plt.plot(x_vals, y_vals_real, 'k--', label='Droite réelle (y = x)')

if theta_opt[1] != 0:
    slope_estimated = -theta_opt[0] / theta_opt[1]
    y_vals_estimated = slope_estimated * x_vals
    plt.plot(x_vals, y_vals_estimated, 'b-', label='Droite estimée')

plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.title("Comparaison entre la droite réelle et la droite estimée")
plt.grid()
plt.show()

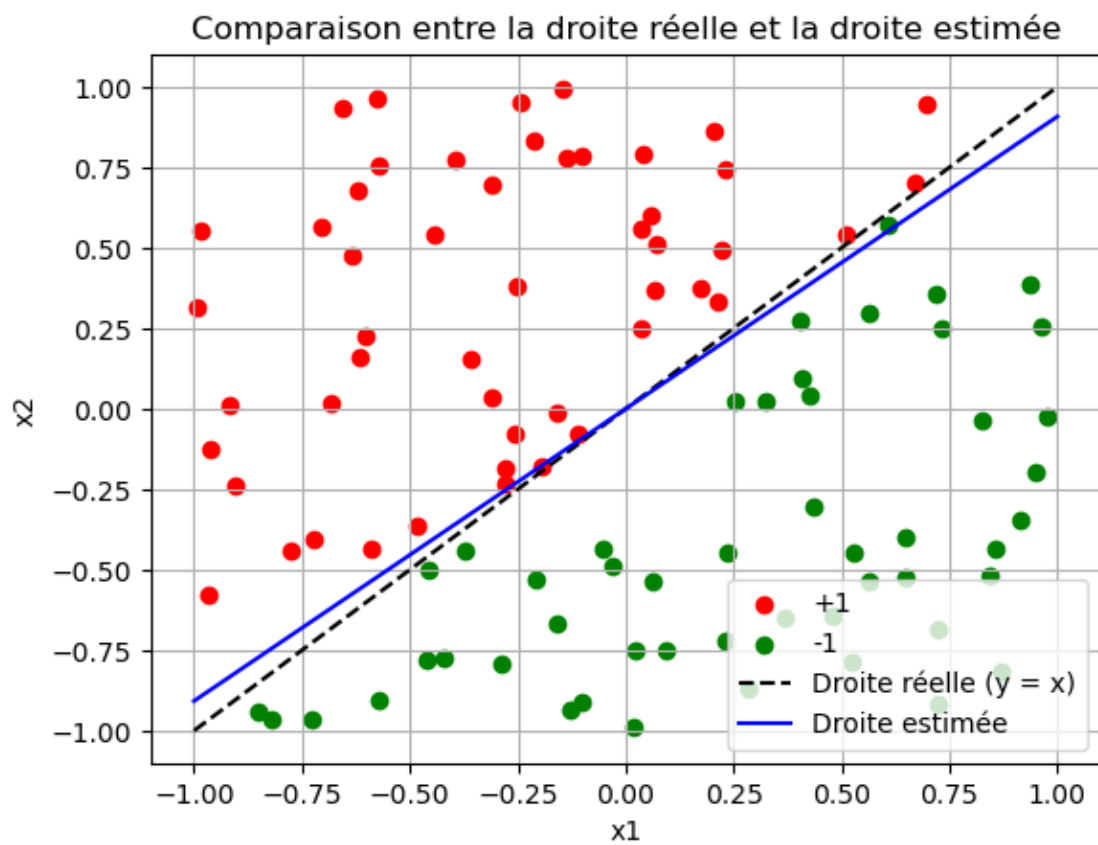
theta_history = np.array(theta_history)
plt.plot(theta_history[:, 0], label="Évolution de w1")
plt.plot(theta_history[:, 1], label="Évolution de w2")
plt.xlabel("Itérations")
plt.ylabel("Valeurs de theta")
plt.legend()
plt.title("Convergence de chaque composante de theta")
plt.grid()
plt.show()

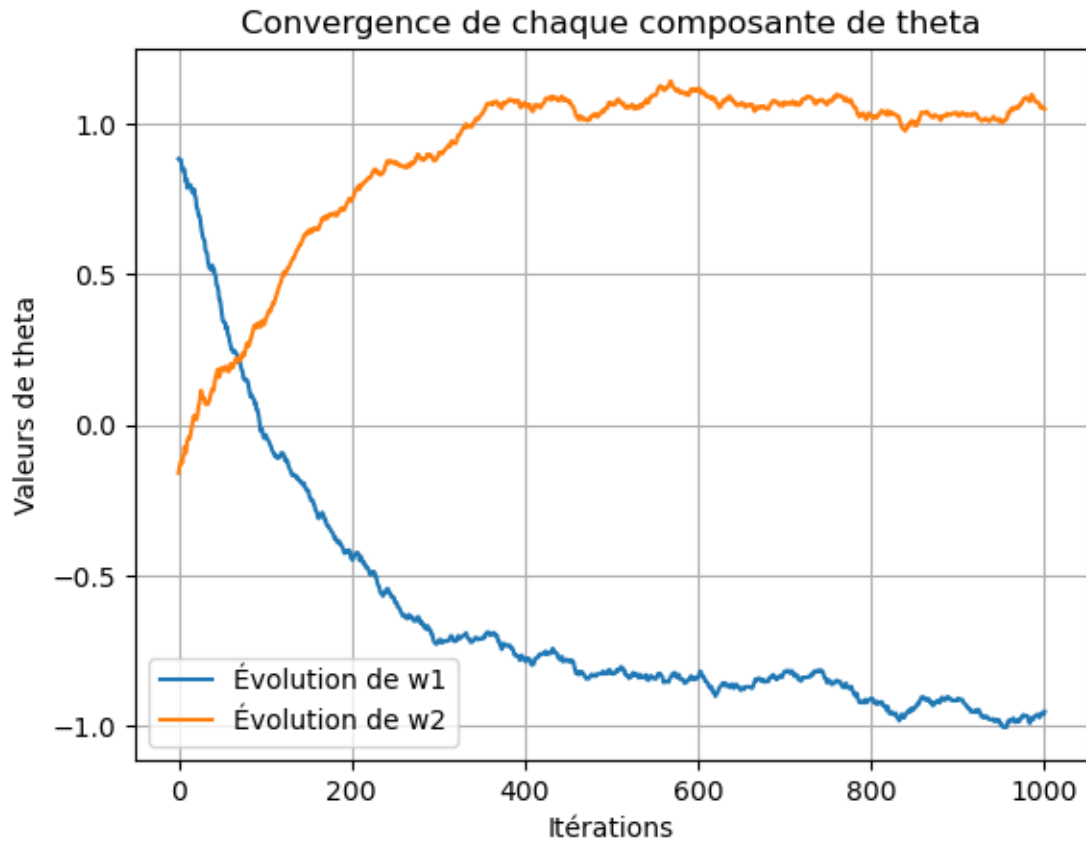
```

```

Vecteur w* estimé : [-0.95307357  1.04983485]
Vecteur w~ réel : [-1  1]
Distance entre w* et w~ : 0.06845145658223643

```





Les résultats sont plutôt bons, même si la convergence met parfois un peu de temps avant de parvenir à des valeurs cohérentes (cela dépend évidemment de l'initialisation de θ_0 qui est aléatoire). On arrive à une valeur de distance entre w^* et w plutôt bonne.

1.4 Question 4

```
[76]: # Ajout de bruit gaussien aux données
noise_std = 0.2
X_noisy = X + np.random.normal(0, noise_std, X.shape)

# Réinitialisation de theta pour l'optimisation avec les données bruitées
init_theta_noisy = np.random.uniform(-1, 1, 2)

# Exécution de l'algorithme avec les données bruitées
theta_opt_noisy, theta_history_noisy = stochastic_gradient_descent(
    grad_func=risk_gradient,
    X=X_noisy,
    y=y,
    init_theta=init_theta_noisy,
    learning_rate_schedule=learning_rate_schedule)
```

```

)

print("Vecteur  $w^*$  estimé avec bruit :", theta_opt_noisy)
print("Vecteur  $w^{\sim}$  réel :", w_true)
print("Distance entre  $w^*$  (bruité) et  $w^{\sim}$  réel :", np.linalg.norm(theta_opt_noisy_
    ↪ w_true))

plt.scatter(X_noisy[y == 1][:, 0], X_noisy[y == 1][:, 1], color='red', label='+1_
    ↪ (bruité)')
plt.scatter(X_noisy[y == -1][:, 0], X_noisy[y == -1][:, 1], color='green',_
    ↪ label='-1 (bruité)')

x_vals = np.linspace(-2, 2, 100)
y_vals_real = x_vals
plt.plot(x_vals, y_vals_real, 'k--', label='Droite réelle (y = x)')

if theta_opt_noisy[1] != 0:
    slope_estimated_noisy = -theta_opt_noisy[0] / theta_opt_noisy[1]
    y_vals_estimated_noisy = slope_estimated_noisy * x_vals
    plt.plot(x_vals, y_vals_estimated_noisy, 'b-', label='Droite estimée_
    ↪ (bruité)')

if theta_opt[1] != 0:
    slope_estimated = -theta_opt[0] / theta_opt[1]
    y_vals_estimated = slope_estimated * x_vals
    plt.plot(x_vals, y_vals_estimated, 'g--', label='Droite estimée (sans_
    ↪ bruit)')

plt.xlabel("x1")
plt.ylabel("x2")
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.legend()
plt.title("Comparaison entre la droite réelle, estimée (sans bruit) et estimée_
    ↪ (bruité)")
plt.grid()
plt.show()

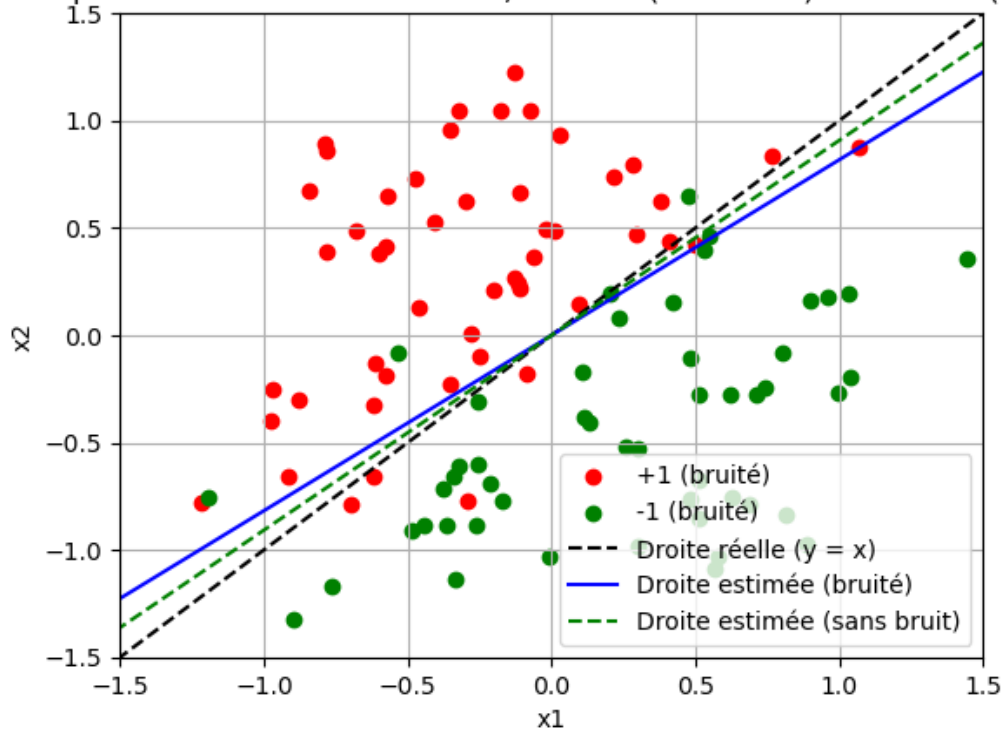
```

```

Vecteur  $w^*$  estimé avec bruit : [-0.82063142  1.00382989]
Vecteur  $w^{\sim}$  réel : [-1  1]
Distance entre  $w^*$  (bruité) et  $w^{\sim}$  réel : 0.1794094630762116

```

Comparaison entre la droite réelle, estimée (sans bruit) et estimée (bruité)



On teste pour différentes valeurs d'écart type ?

```
[77]: noise_levels = [0.1, 0.3, 0.7, 2]

fig, axes = plt.subplots(1, len(noise_levels), figsize=(18, 5), sharex=True,
    ↳sharey=True)

for idx, noise_std in enumerate(noise_levels):
    X_noisy = X + np.random.normal(0, noise_std, X.shape)
    init_theta_noisy = np.random.uniform(-1, 1, 2)

    theta_opt_noisy, _ = stochastic_gradient_descent(
        grad_func=risk_gradient,
        X=X_noisy,
        y=y,
        init_theta=init_theta_noisy,
        learning_rate_schedule=learning_rate_schedule
    )

    axes[idx].scatter(X_noisy[y == 1][:, 0], X_noisy[y == 1][:, 1], color='red',
    ↳label='+1 (bruité)')
```

```

axes[idx].scatter(X_noisy[y == -1][:, 0], X_noisy[y == -1][:, 1],
↳color='green', label='-1 (bruité)')

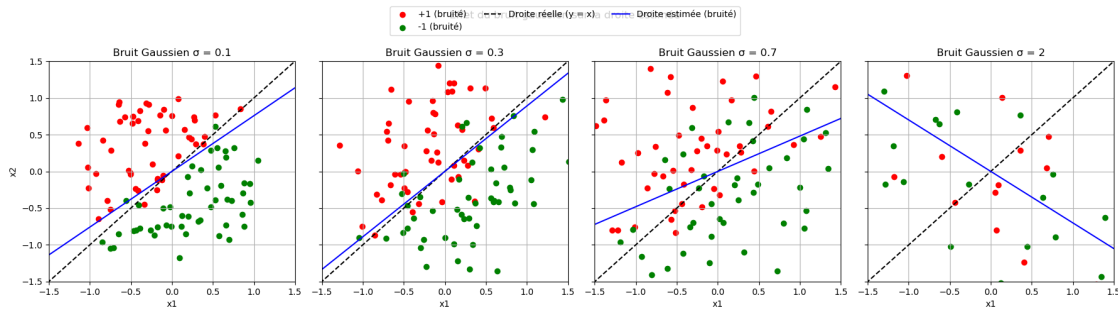
x_vals = np.linspace(-2, 2, 100)
y_vals_real = x_vals
axes[idx].plot(x_vals, y_vals_real, 'k--', label='Droite réelle (y = x)')

if theta_opt_noisy[1] != 0:
    slope_estimated_noisy = -theta_opt_noisy[0] / theta_opt_noisy[1]
    y_vals_estimated_noisy = slope_estimated_noisy * x_vals
    axes[idx].plot(x_vals, y_vals_estimated_noisy, 'b-', label='Droite_
↳estimée (bruité)')

axes[idx].set_title(f"Bruit Gaussien  $\sigma = \{noise\_std\}$ ")
axes[idx].set_xlim(-1.5, 1.5)
axes[idx].set_ylim(-1.5, 1.5)
axes[idx].set_xlabel("x1")
axes[idx].grid()

axes[0].set_ylabel("x2")
handles, labels = axes[0].get_legend_handles_labels()
fig.legend(handles, labels, loc="upper center", ncol=3)
fig.suptitle("Effet du bruit gaussien sur la droite estimée")
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

```



Les résultats sont étonnamment plutôt bons, du moment que le bruit n'est pas trop élevé. Jusqu'à $\sigma = 1$, les valeurs prédites w^* sont tout à fait pertinentes.

1.5 Question 5

```

[78]: # Chargement des données
data = pd.read_csv(r'wdbc.data', header=None)

# Prétraitement des données

```

```

data[1] = data[1].map({'M': -1, 'B': 1}) # M -> -1, B -> +1
y = data[1].values # La colonne de labels
X = data.iloc[:, 2:].values # Les colonnes de données sans ID ni label

# Normalisation des données
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

init_theta = np.random.uniform(-1, 1, X.shape[1])

def risk_gradient(theta, x, y):
    error = y - np.dot(theta, x)
    grad = -2 * error * x
    return grad

def learning_rate_schedule(k):
    return 0.01 / (1 + 0.0001 * k)

theta_opt, theta_history = stochastic_gradient_descent(
    grad_func=risk_gradient,
    X=X,
    y=y,
    init_theta=init_theta,
    learning_rate_schedule=learning_rate_schedule
)

print("Vecteur w* estimé :", theta_opt)
print("Nombre d'itérations :", len(theta_history))
print("Valeur finale du risque empirique :", np.mean([(y[i] - np.dot(theta_opt,
↪X[i])) ** 2 for i in range(len(y))]))

theta_history = np.array(theta_history)
plt.figure(figsize=(12, 6))
for i in range(X.shape[1]):
    plt.plot(theta_history[:, i], label=f"Évolution de w{i+1}")
plt.xlabel("Itérations")
plt.ylabel("Valeurs de theta")
plt.title("Convergence de chaque composante de theta")
plt.grid()
plt.show()

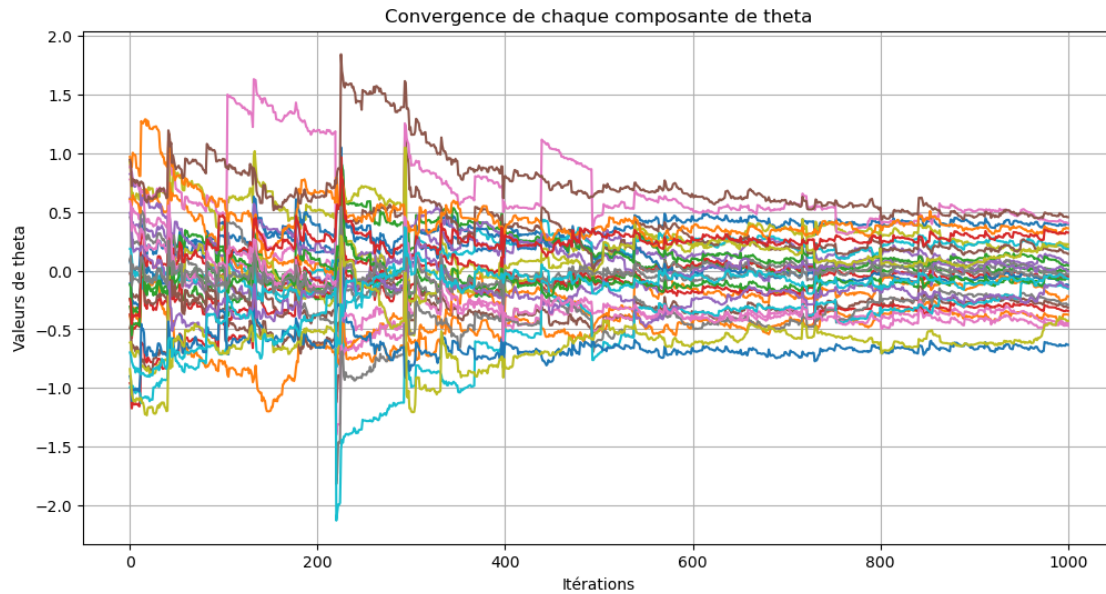
```

Vecteur w* estimé : [0.39488361 -0.37576228 0.00503893 -0.05238832 0.05993955
-0.31556966

-0.44997762 -0.03494324 0.19402095 0.18268566 -0.12311868 -0.21994772
0.04331271 -0.34347453 -0.00556477 0.14267558 0.40279905 -0.29183357
0.22295009 -0.15374864 -0.63250594 0.35991443 -0.07134829 0.32294198
-0.14008277 0.45466412 -0.47001657 -0.03238521 -0.42769137 -0.06908425]

Nombre d'itérations : 1001

Valeur finale du risque empirique : 0.5887622135662421



Nous allons maintenant évaluer la pertinence de ces résultats en effectuant, comme cela est présenté dans le petit texte, une estimation de la **précision** et du **rappel** de notre modèle.

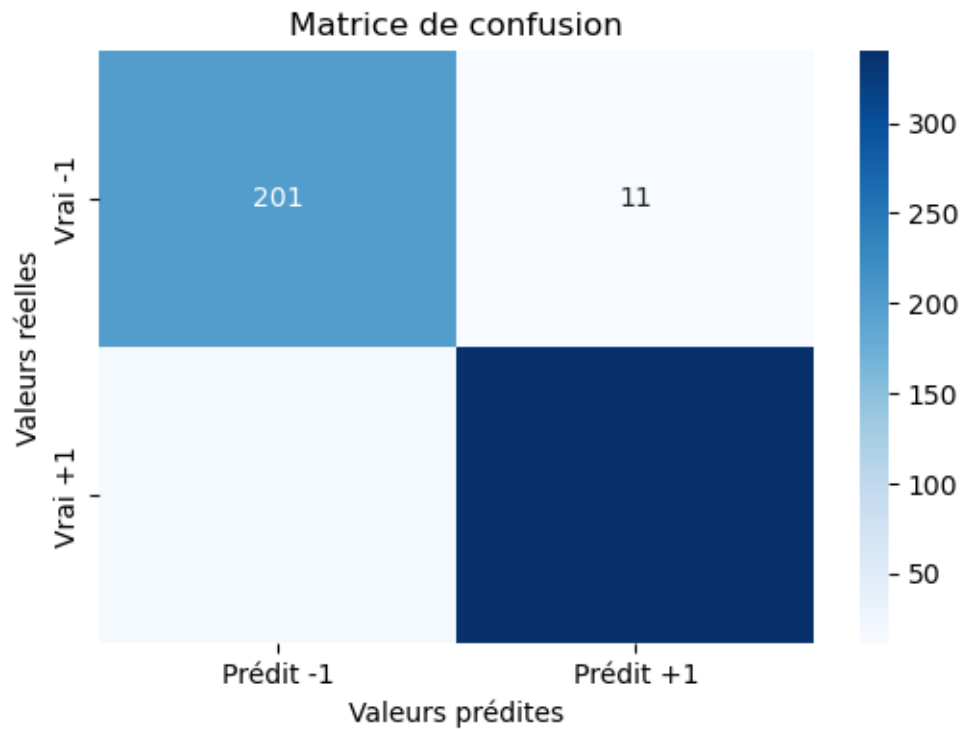
```
[79]: y_pred = np.sign(X @ theta_opt)

# Calcul de la précision et du rappel
precision = precision_score(y, y_pred)
recall = recall_score(y, y_pred)
print("Précision :", precision)
print("Rappel :", recall)

# Calcul et affichage de la matrice de confusion
conf_matrix = confusion_matrix(y, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Prédit ↵
↵-1', 'Prédit +1'], yticklabels=['Vrai -1', 'Vrai +1'])
plt.xlabel("Valeurs prédites")
plt.ylabel("Valeurs réelles")
plt.title("Matrice de confusion")
plt.show()
```

Précision : 0.9686609686609686

Rappel : 0.9523809523809523



Les résultats sont incroyables, alors que l'algorithme de descente de gradient stochastique n'est pas très complexe. On arrive rapidement à **une précision et un rappel au dessus de 0.9**, ce qui est extrêmement satisfaisant pour un algorithme de classification.