SY09 Printemps 2022
TD/TP 09 — Analyse discriminante de données gaussiennes

1 Partie pratique

On souhaite comparer les performances de trois modèles d'analyse discriminante (analyse discriminante quadratique, analyse discriminante linéaire, et classifieur bayésien naïf sous hypothèse de normalité des classes) sur des jeux de données simulées.

Pour chacun des jeux de données, la distribution conditionnelle à chaque classe est gaussienne, les paramètres pouvant en revanche changer d'un jeu de données à l'autre.

On utilisera les instructions suivantes pour charger ces modèles.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
```

1 Pour chacun des jeu de données bidimensionnels suivants :
- `data/SynthCross_n1000_p2.csv`,
- `data/SynthPara_n1000_p2.csv`,
- `data/SynthPlus_n1000_p2.csv`,

calculer les erreurs de validation croisée à 10 plis, comparer les performances des trois algorithmes et les interpréter.

Pour visualiser les erreurs conjointement, on pourra utiliser la fonction `sns.lineplot` après avoir généré un jeu de données regroupant les 10 erreurs de validation croisée pour chaque algorithme. À cette fin, on pourra d'abord définir un générateur `validation_errors` qui itère sur les modèles et génère pour chaque les 10 erreurs de validation croisée retournées par `cross_val_score`.

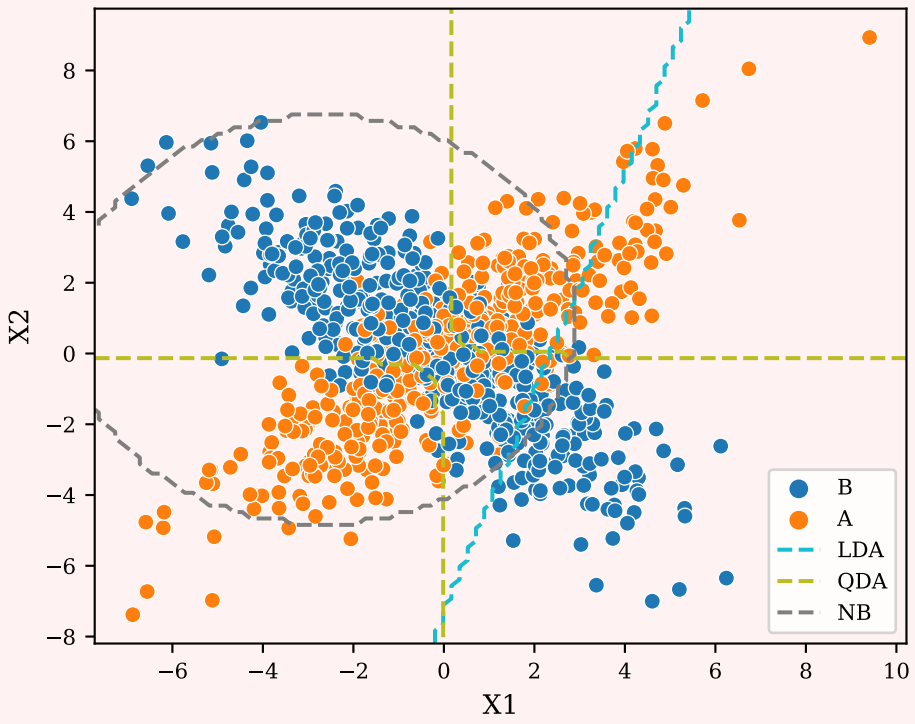
Pour expliquer ces résultats, on pourra s'appuyer sur la fonction `add_decision_boundary` définie au TP07. On pourra également définir une fonction `add_decision_boundaries` qui ajoute les frontières de décision pour chaque modèle.

On définit d'abord la fonction `validation_errors` qui va générer les erreurs de validation croisée à 10 plis pour chaque modèle.

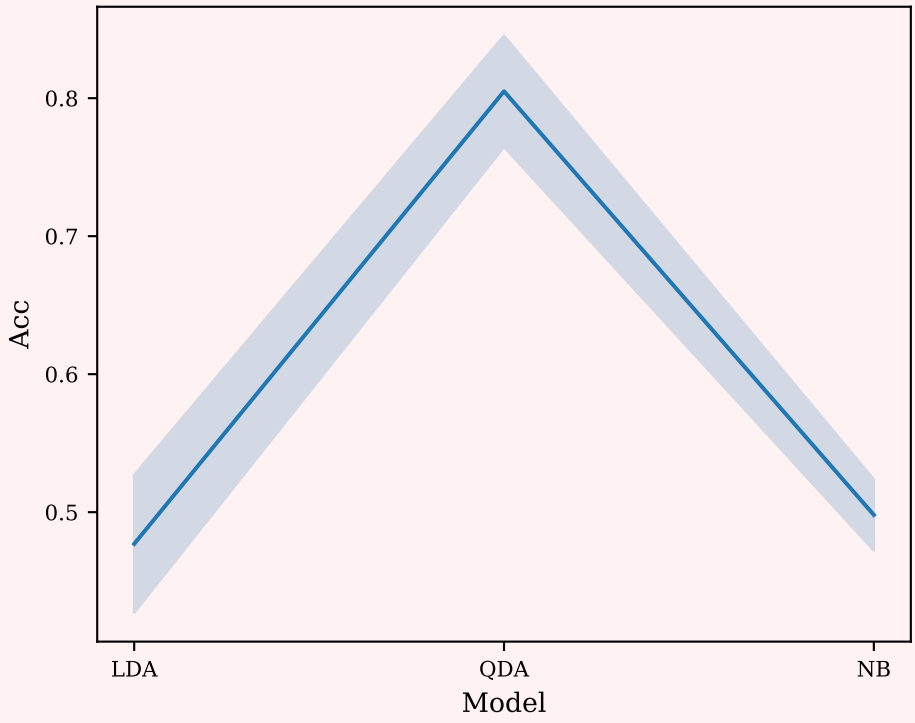
In [1]:
```
from sklearn.model_selection import cross_val_score

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB

models = [
    (LinearDiscriminantAnalysis, "LDA"),
    (QuadraticDiscriminantAnalysis, "QDA"),
    (GaussianNB, "NB"),
]

def validation_errors(df, models):
    y = df.z
    X = df.drop(columns=["z"])

    for model, name in models:
        for acc in cross_val_score(model(), X, y, cv=10):
            yield name, acc

def add_decision_boundaries(df, models):
    colors = sns.color_palette()
    for model, name in models:
        y = df.z
        X = df.drop(columns=["z"])
        cls = model()
        cls.fit(X, y)
        add_decision_boundary(cls, label=name, color=colors.pop(),
        ↪  region=False)
```

Pour chaque jeu de données bidimensionnels, on affiche le jeu de données, les frontières de décision et les performances.

In [2]:
```
df = pd.read_csv("data/SynthCross_n1000_p2.csv")
sns.scatterplot(x="X1", y="X2", data=df, hue="z")
add_decision_boundaries(df, models)
plt.show()
```



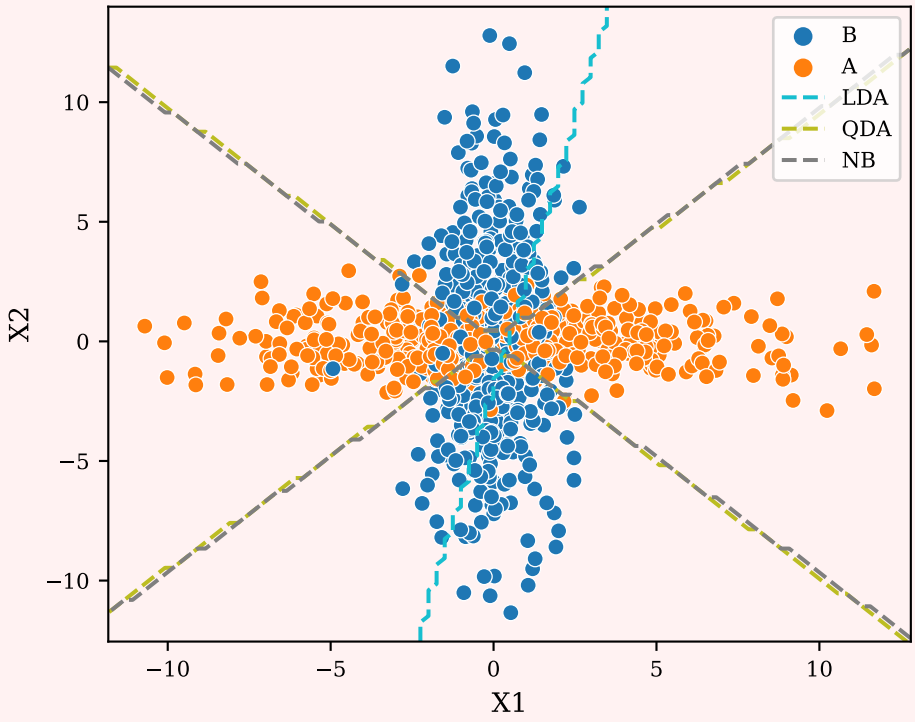
In [3]:
```
df = pd.DataFrame(validation_errors(df, models), columns=["Model",
↪ "Acc"])
sns.lineplot(x="Model", y="Acc", data=df, ci="sd")
plt.show()
```



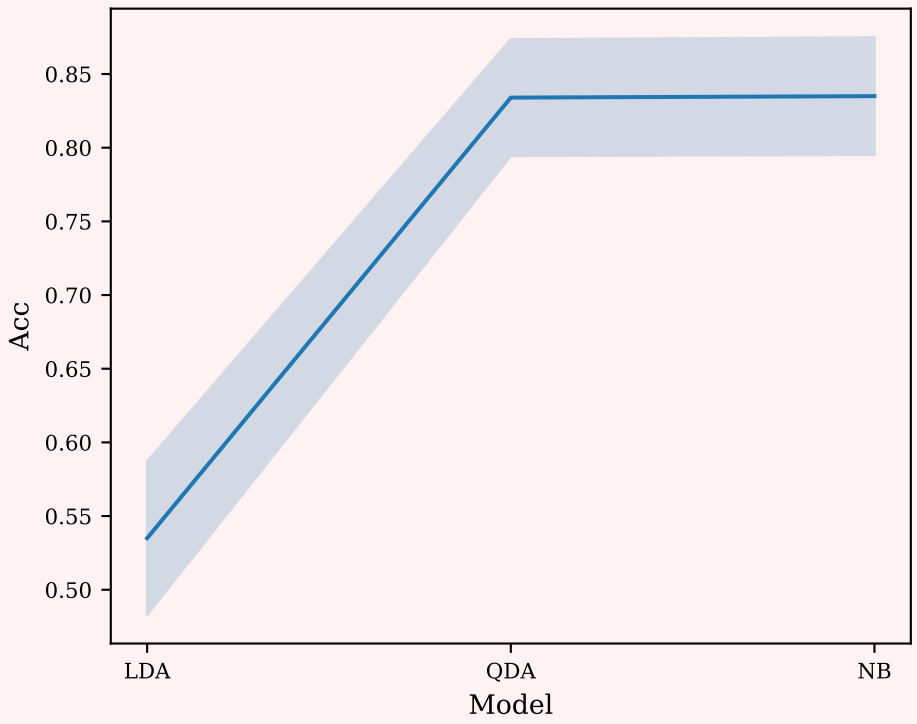
La forme en croix du jeu de données fait que ni la LDA ni l'algorithme Bayésien naïf n'arrive à séparer correctement les deux classes. Seule une courbe hyperbolique obtenue par la QDA permet

de séparer les deux classes. L'algorithme bayésien naïf peut théoriquement générer des frontières de décision de forme hyperbolique mais l'axe focal doit être parallèle à l'un des axes ce qui ne convient pas ici.

In [4]:
```
df = pd.read_csv("data/SynthPlus_n1000_p2.csv")
sns.scatterplot(x="X1", y="X2", data=df, hue="z")
add_decision_boundaries(df, models)
plt.show()
```

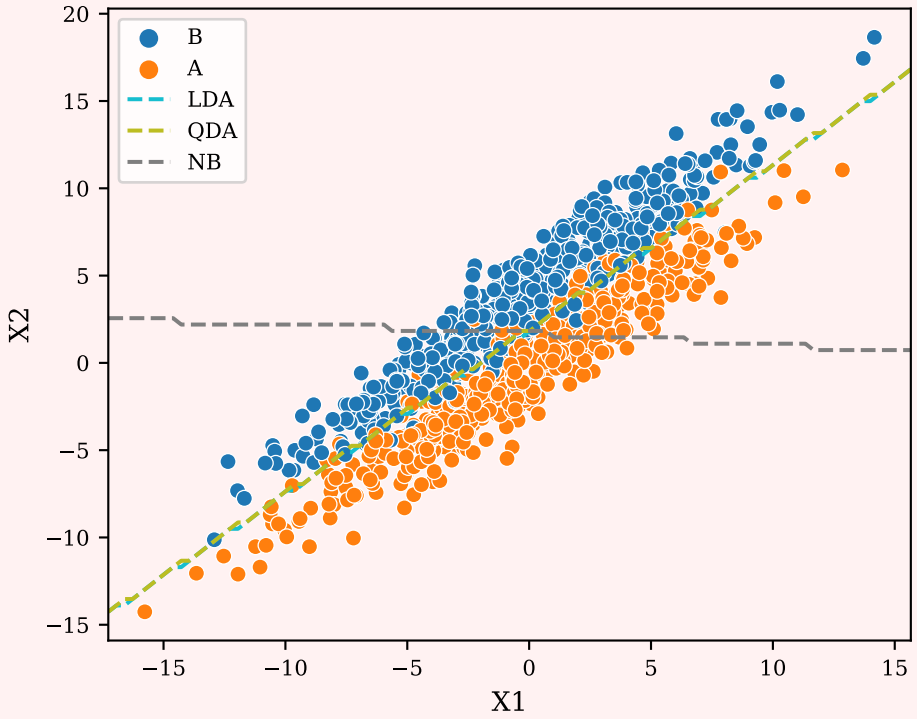


In [5]:
```
df = pd.DataFrame(validation_errors(df, models), columns=["Model",
↪  "Acc"])
sns.lineplot(x="Model", y="Acc", data=df, ci="sd")
plt.show()
```

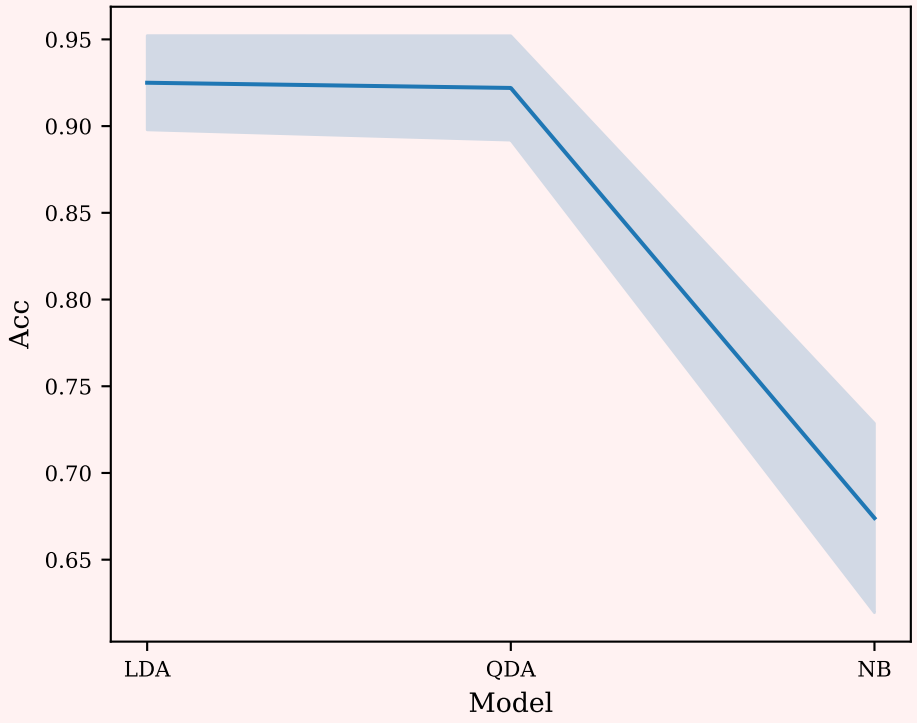


L'algorithme bayésien naïf et la QDA apprennent une frontière de décision de type hyperbolique qui sépare correctement les données. L'axe focal de l'hyperbole est cette fois parallèle à l'un des axes ce qui explique les bonnes performances de l'algorithme bayésien naïf.

In [6]:
```
df = pd.read_csv("data/SynthPara_n1000_p2.csv")
sns.scatterplot(x="X1", y="X2", data=df, hue="z")
add_decision_boundaries(df, models)
plt.show()
```



In [7]:
```
df = pd.DataFrame(validation_errors(df, models), columns=["Model",
↪ "Acc"])
sns.lineplot(x="Model", y="Acc", data=df, ci="sd")
plt.show()
```

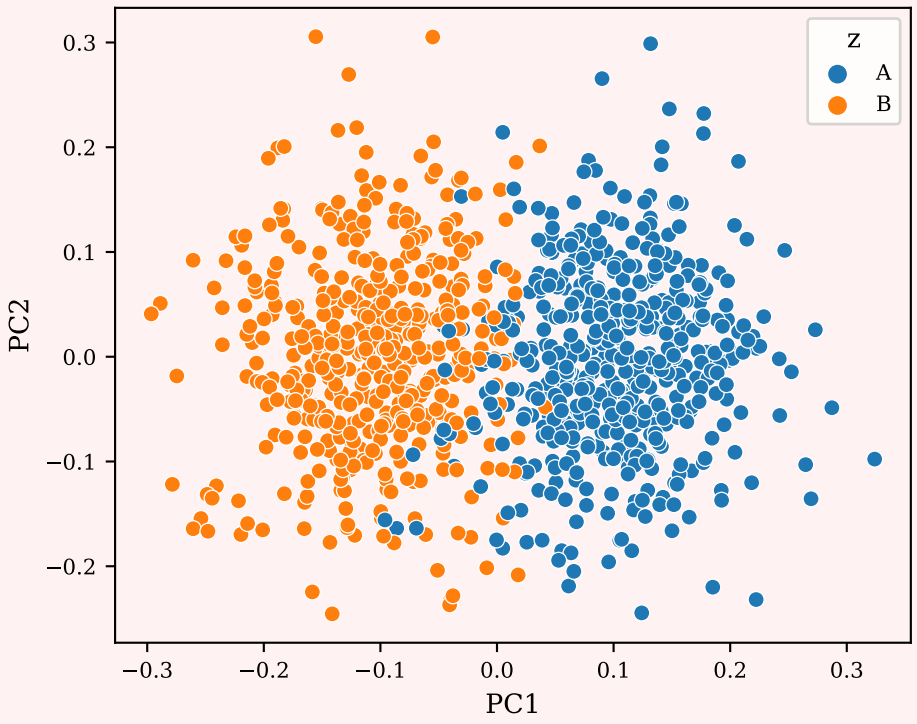


Les matrices de variance–covariance conditionnellement à la classe ne sont pas diagonales (les axes des gaussiennes ne sont pas alignés avec les axes).

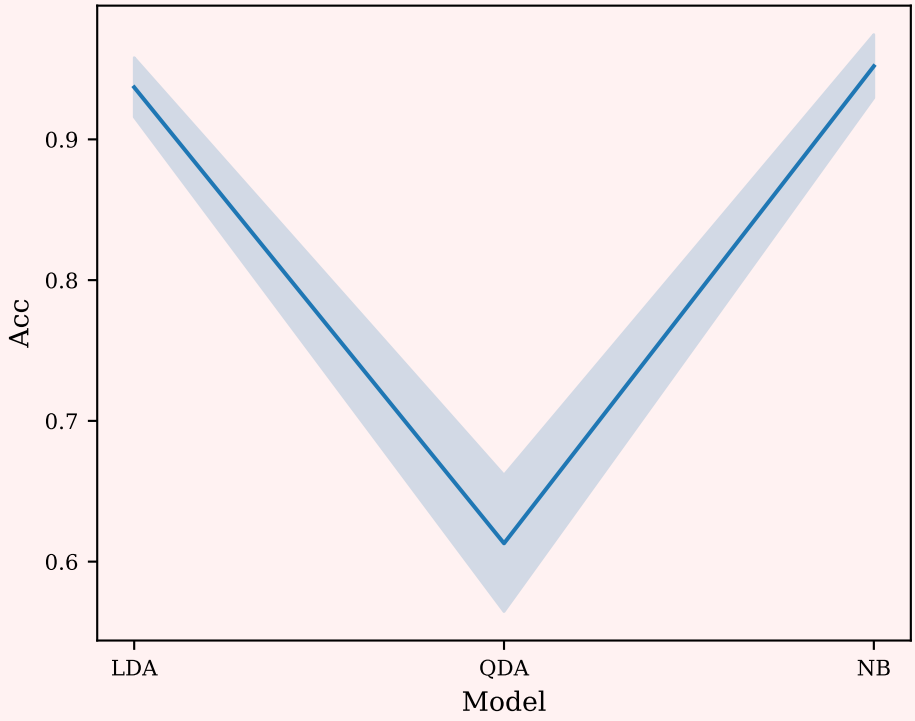
2 Recommencer avec les jeux de données suivants :
- `data/SynthBlob_n1000_p500.csv.gz`,
- `data/SynthPara_n1000_p500.csv.gz`,
- `data/SynthPlus_n1000_p500.csv.gz`.

Les trois jeux de données étant de dimension > 2, on pourra utiliser la fonction `scatterplot_pca` du TP06 pour les visualiser.

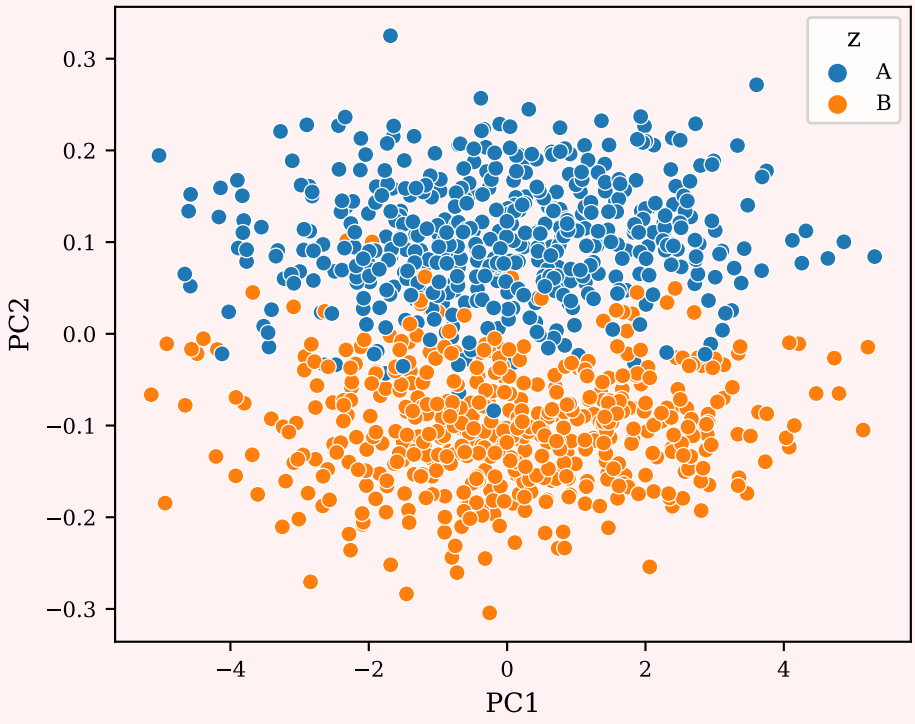
In [8]:
```
df = pd.read_csv("data/SynthBlob_n1000_p300.csv.gz")
scatterplot_pca(data=df, hue="z")
plt.show()
```



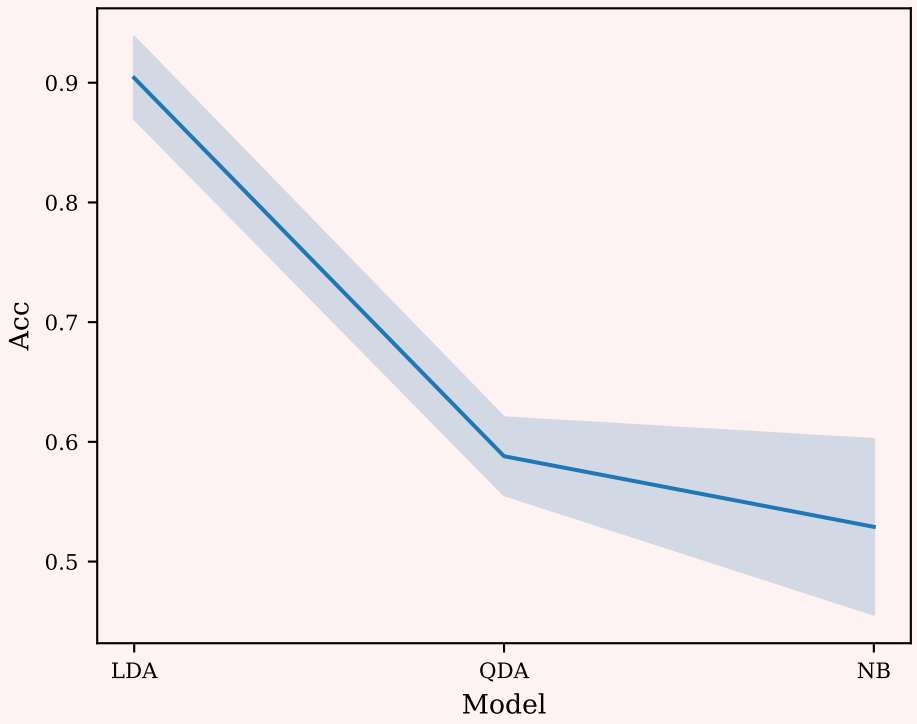
In [9]:
```
df = pd.DataFrame(validation_errors(df, models), columns=["Model",
↪ "Acc"])
sns.lineplot(x="Model", y="Acc", data=df, ci="sd")
plt.show()
```



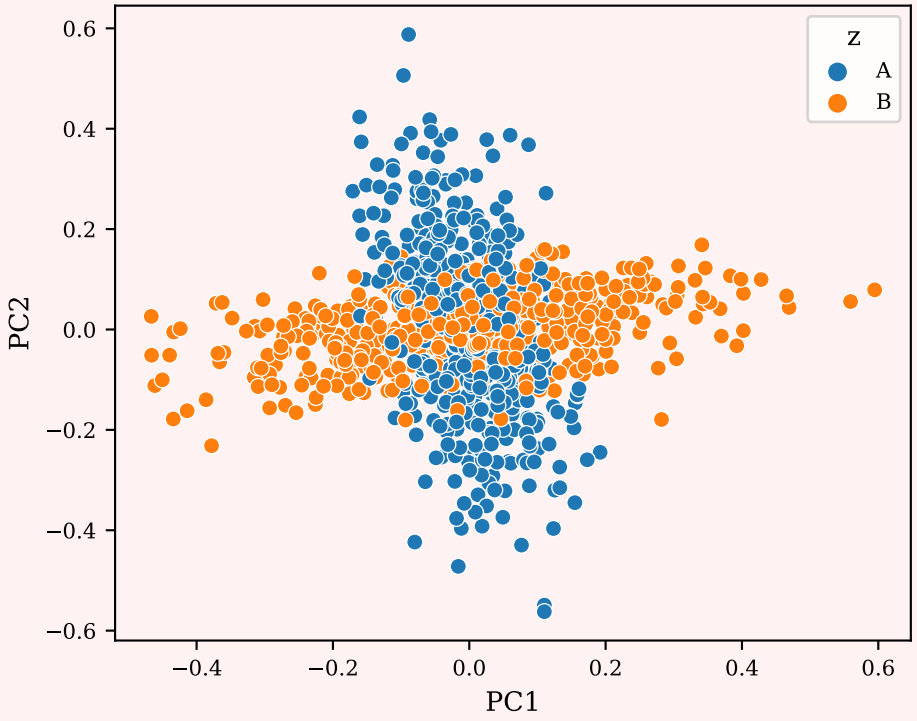
In [10]:
```
df = pd.read_csv("data/SynthPara_n1000_p300.csv.gz")
scatterplot_pca(data=df, hue="z")
plt.show()
```



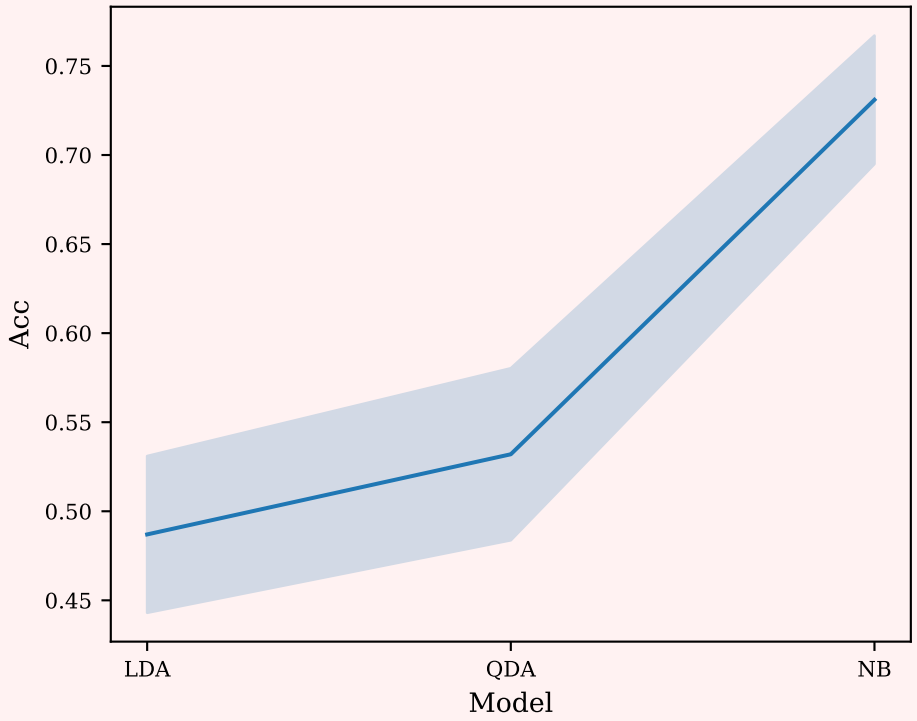
In [11]:
```
df = pd.DataFrame(validation_errors(df, models), columns=["Model",
↪  "Acc"])
sns.lineplot(x="Model", y="Acc", data=df, ci="sd")
plt.show()
```



In [12]:
```
df = pd.read_csv("data/SynthPlus_n1000_p300.csv.gz")
scatterplot_pca(data=df, hue="z")
plt.show()
```



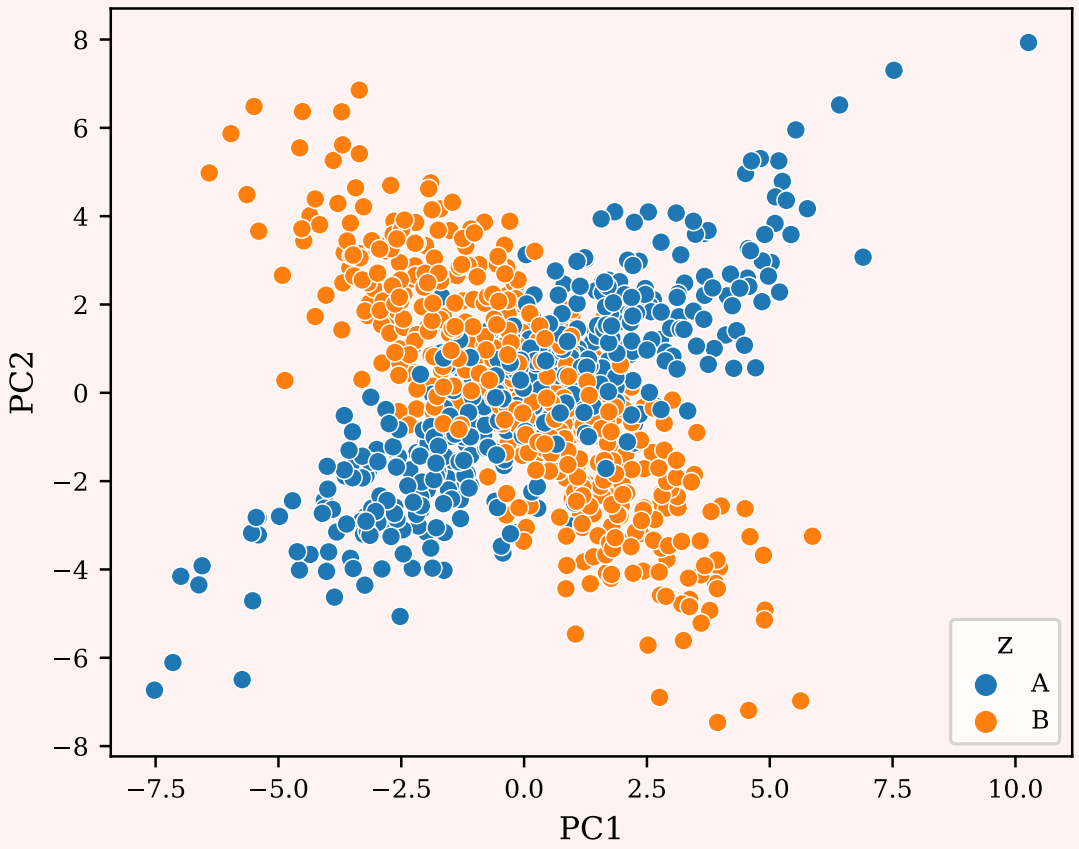

```
In [13]: df = pd.DataFrame(validation_errors(df, models), columns=["Model",
         ↪ "Acc"])
         sns.lineplot(x="Model", y="Acc", data=df, ci="sd")
         plt.show()
```



3 Construire un jeu de données sur lequel la précision des trois modèles n'excède pas 60%. Construire un classifieur simple « à la main » d'après ce que vous savez de ce jeu de données.

L'idée est de créer un jeu de données similaire à `SynthCross_n1000_p2.csv` mais en plus grande dimension pour handicaper l'analyse discriminante quadratique.

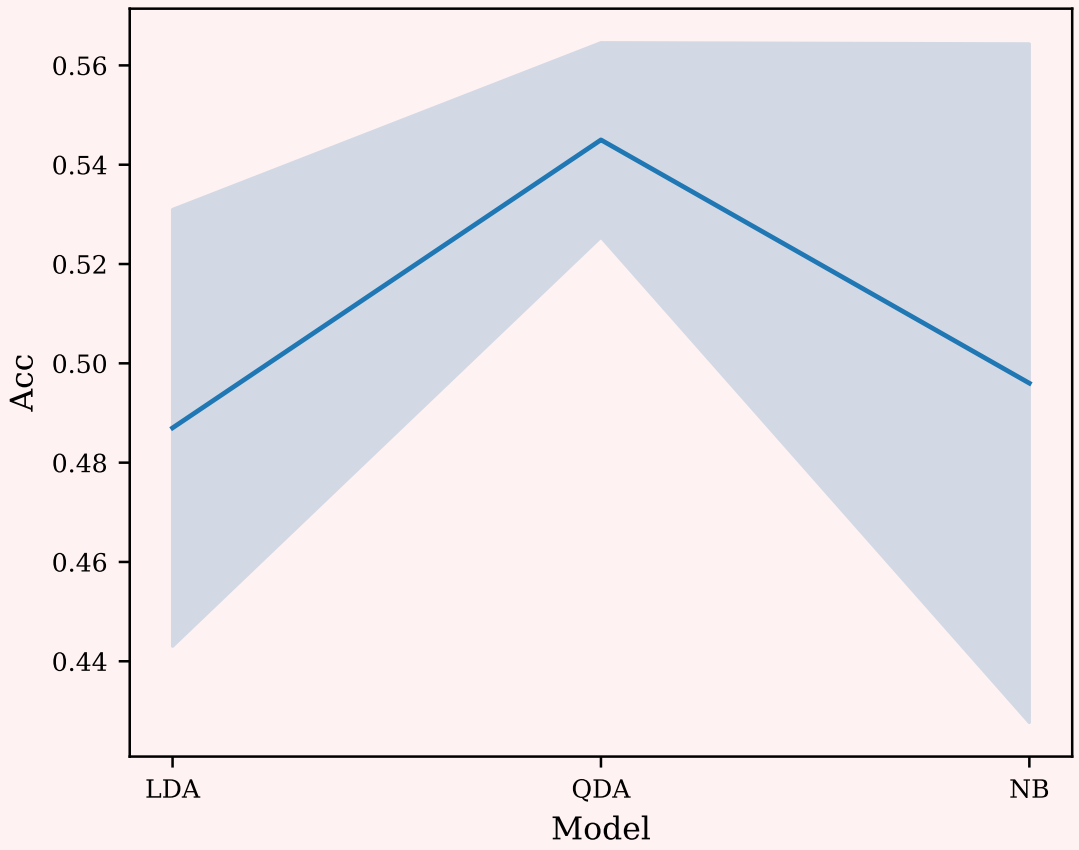
In [14]:
```
df = pd.read_csv("data/SynthCross_n1000_p2.csv")

rng = np.random.default_rng()
df300 = pd.concat((df, pd.DataFrame(.1*rng.normal(size=(1000, 298)))),
↪  axis=1)

scatterplot_pca(data=df300, hue="z")
plt.show()
```



In [15]:
```
df = pd.DataFrame(validation_errors(df300, models), columns=["Model",
↪  "Acc"])
```

```
Out [15]: /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
          ↪  will be raised in 1.2.
            warnings.warn(
          /usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:1688:
          ↪  FutureWarning: Feature names only support names that are all
          ↪  strings. Got feature names with dtypes: ['int', 'str']. An error
```

```
In [16]: sns.lineplot(x="Model", y="Acc", data=df, ci="sd")
         plt.show()
```



Il s'agit du jeu de données en croix. On sait qu'une hyperbole d'axe focal l'axe des abcisses est un bon classifieur.

```
In [17]: pred = pd.Series(np.where(df300.X1 * df300.X2 > 0, "A", "B"))
         np.mean(pred == df300.z)
```

```
Out [17]: 0.802
```

2 Partie théorique

Les jeux de données synthétiques utilisés au paragraphe 1 ont été obtenus par un processus génératif tel que décrit dans le TP08 :

1. tout d'abord, l'effectif n_1 de la classe ω_1 a été déterminé par tirage aléatoire suivant une loi binomiale $\mathcal{B}(n, \pi_1)$, avec $\pi_1 = 0.5$;
2. n_1 individus ont ensuite été générés dans la classe ω_1 suivant une loi normale bivariée $\mathcal{N}(\mu_1, \Sigma_1)$, et $n_2 = n - n_1$ dans la classe ω_2 suivant une loi normale bivariée $\mathcal{N}(\mu_2, \Sigma_2)$.

Questions.

4 Quelles sont les distributions marginales des variables X^1 et X^2 dans chaque classe ?

Le vecteur $\boldsymbol{X}$ suit une loi normale multivariée (bivariée) dans chacune des classes :

$$\boldsymbol{X} \underset{\omega_k}{\sim} \mathcal{N}(\mu_k, \Sigma_k), \quad \text{pour } k \in \{1, 2\}.$$

Par conséquent, tout sous-vecteur d'un vecteur aléatoire gaussien étant lui-même gaussien (les paramètres étant obtenus en sélectionnant les lignes et colonnes du vecteur espérance et de la matrice de covariance), on en déduit que

$$X^j \underset{\omega_k}{\sim} \mathcal{N}(\mu_{kj}, \sigma^2_{kj}),$$

où $j \in \{1, 2\}$ est l'indice de la variable considérée.

5 Calculer l'expression des courbes d'iso-densité dans la classe ω_k, en fonction de μ_k et Σ_k. À quoi correspondent ces courbes dans le cas général (Σ_k quelconque) ? Si Σ_k est diagonale, sphérique ?

La courbe d'iso-densité de la classe ω_k est définie par

$$\big\{\boldsymbol{x} \text{ t.q. } f_k(\boldsymbol{x}) = K\big\},$$

avec $0 < K \leq f_k(\mu_k)$, c'est-à-dire $0 < K \leq (2\pi)^{-\frac{p}{2}}|\Sigma_k|^{-\frac{1}{2}}$. Dans le cas général, cela donne l'équation d'une ellipse de centre μ_k :

$$\begin{aligned} f_k(\boldsymbol{x}) = K \quad &\Leftrightarrow \quad |\Sigma_k|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\mu_k)^T\Sigma_k^{-1}(\boldsymbol{x}-\mu_k)\right) = (2\pi)^{\frac{p}{2}}K, \\ &\Leftrightarrow \quad (\boldsymbol{x}-\mu_k)^T\Sigma_k^{-1}(\boldsymbol{x}-\mu_k) = -p\log(2\pi) - 2\log K - \log|\Sigma_k|; \end{aligned}$$

si $K = (2\pi)^{-\frac{p}{2}}|\Sigma_k|^{-\frac{1}{2}}$, la courbe d'iso-densité se réduit à un unique point :

$$f_k(\boldsymbol{x}) = K \Leftrightarrow \boldsymbol{x} = \mu_k;$$

si $K \leq 0$ ou $K > (2\pi)^{-\frac{p}{2}}|\Sigma_k|^{-\frac{1}{2}}$, il n'existe pas de valeur de $\boldsymbol{x}$ vérifiant l'égalité $f_k(\boldsymbol{x}) = K$, la courbe d'iso-densité est donc un ensemble vide.

En dimension $p = 2$, on peut aller plus loin en factorisant la matrice Σ_k :

$$\Sigma_k = \lambda_k P_k A_k P_k^T, \quad \lambda_k > 0, \quad P_k = \begin{pmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{pmatrix}, \quad A_k = \begin{pmatrix} a_k & 0 \\ 0 & 1/a_k \end{pmatrix}.$$

L'orientation de la classe et les longueurs des axes de l'ellipse dépendent des paramètres λ_k et a_k intervenant dans cette décomposition, comme le montre la figure 5.

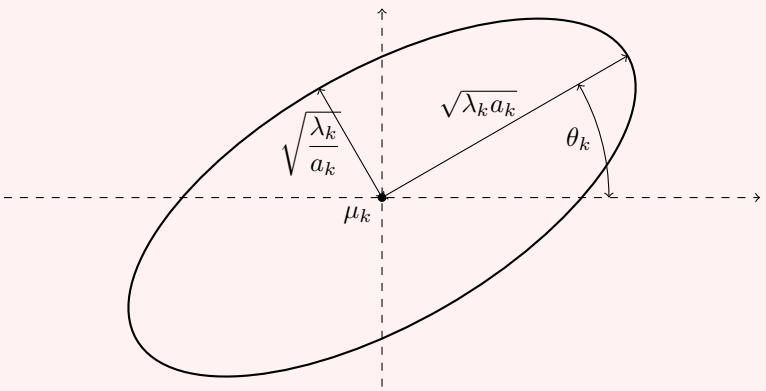


FIGURE 1 – Orientation d'une classe gaussienne de centre μ_k et de matrice de covariance Σ_k suivant la décomposition $\Sigma_k = \lambda_k P_k A_k P_k^T$.

6 Calculer l'expression de la frontière de décision de la règle de Bayes $\delta^\star$ dans le cas général.

La frontière de décision est définie par

$$\{\boldsymbol{x} \text{ t.q. } g_1(\boldsymbol{x}) = g_2(\boldsymbol{x})\} \Leftrightarrow \{\boldsymbol{x} \text{ t.q. } g_1(\boldsymbol{x}) - g_2(\boldsymbol{x}) = 0\},$$

où la fonction discriminante g_k est définie par

$$g_k(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x} - \mu_k)^T \Sigma_k^{-1}(\boldsymbol{x} - \mu_k) - \frac{1}{2}\log|\Sigma_k| + \log \pi_k - \frac{p}{2}\log(2\pi).$$

7 Représenter cette frontière et comparer aux frontières obtenues avec les trois modèles d'analyse discriminante pour le jeu de données `SynthCross_n1000_p2.csv`, généré avec les paramètres suivants :

$$\mu_1 = \begin{pmatrix}0\\0\end{pmatrix}, \quad \mu_2 = \begin{pmatrix}0\\0\end{pmatrix} \quad \Sigma_1 = \frac{1}{2}\begin{pmatrix}11 & 9\\9 & 11\end{pmatrix}, \quad \Sigma_2 = \frac{1}{2}\begin{pmatrix}11 & -9\\-9 & 11\end{pmatrix}.$$

Pour tracer une fonction implicite, on pourra s'inspirer du code suivant

```
# Les bornes
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# Discrétisation et création du tableau des abscisses et des ordonnées
resolution = 1000
xx = np.linspace(xlim[0], xlim[1], resolution)
yy = np.linspace(ylim[0], ylim[1], resolution)
X, Y = np.meshgrid(xx, yy)

# Calcul de la fonction implicite
Z = ...

# Tracé de la ligne séparant Z > 0 de Z < 0
plt.contour(X, Y, Z, levels=[0])
plt.show()
```

Il existe plusieurs possibilités, qui donnent rigoureusement les mêmes résultats. On peut calculer les fonctions discriminantes en utilisant les « vrais » paramètres, ou encore calculer les « vraies » probabilités a posteriori (là aussi, en utilisant les « vrais » paramètres).

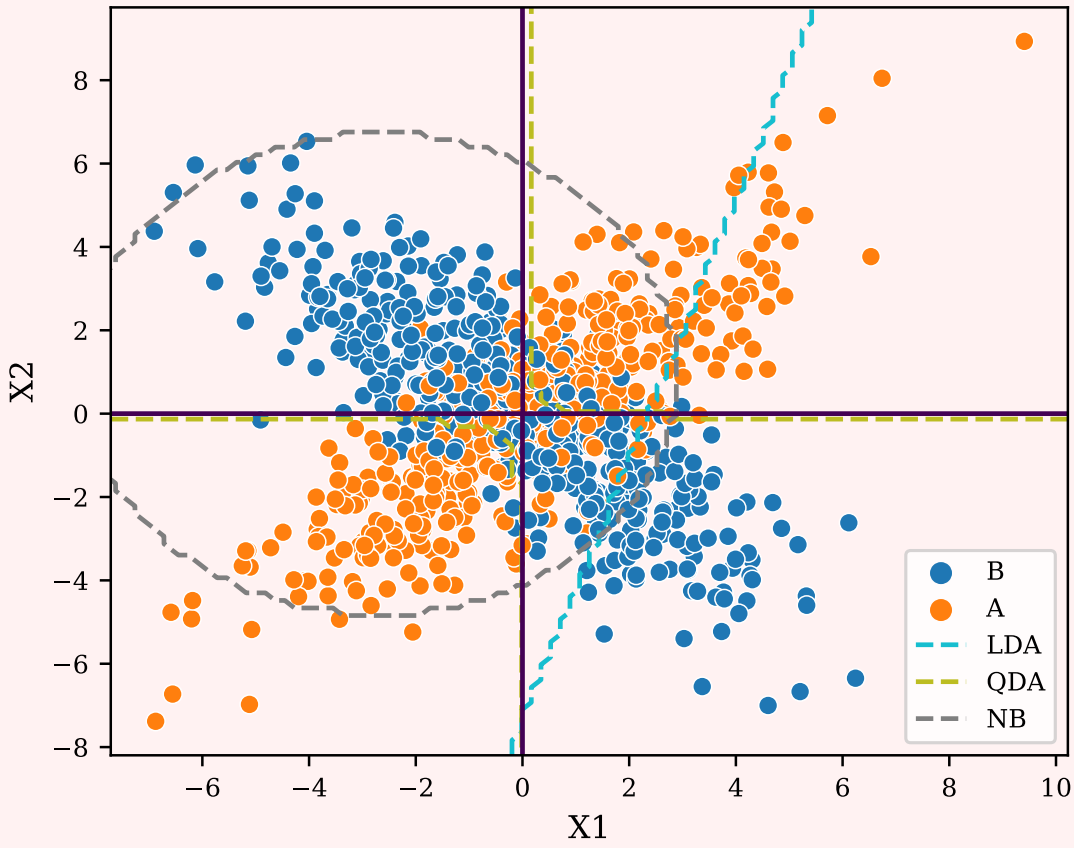
On en déduit ensuite la frontière de décision, dans le premier cas en traçant la courbe de niveau 0 de la surface $g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$, et dans le second cas en traçant la courbe de niveau 0.5 de la surface $\Pr(\omega_1|\boldsymbol{x})$.

En utilisant le fait que $\mu_1 = \mu_2 = 0$, $|\Sigma_1| = |\Sigma_2|$ et $\pi_1 = \pi_2$, et en remarquant que $\Sigma_1\Sigma_2 = \det\Sigma_1 I_2$, on trouve

$$\begin{aligned} g_1(\boldsymbol{x}) - g_2(\boldsymbol{x}) &= \boldsymbol{x}^T\left(\Sigma_1^{-1} - \Sigma_2^{-1}\right)\boldsymbol{x} \\ &= \frac{1}{|\Sigma_1|}\boldsymbol{x}^T(\Sigma_2 - \Sigma_1)\boldsymbol{x} \\ &= \frac{1}{|\Sigma_1|}\boldsymbol{x}^T\begin{pmatrix}0 & -18\\-18 & 0\end{pmatrix}\boldsymbol{x} \\ &= -\frac{36}{|\Sigma_1|}x_1x_2 \end{aligned}$$

La frontière de Bayes consiste donc en les droites $x_1 = 0$ et $x_2 = 0$.

```
In [18]: df = pd.read_csv("data/SynthCross_n1000_p2.csv")
         ax = sns.scatterplot(x="X1", y="X2", data=df, hue="z")
         add_decision_boundaries(df, models)
         xlim = ax.get_xlim()
         ylim = ax.get_ylim()

         resolution = 1000
         xx = np.linspace(xlim[0], xlim[1], resolution)
         yy = np.linspace(ylim[0], ylim[1], resolution)
         X, Y = np.meshgrid(xx, yy)

         Z = X * Y
         plt.contour(X, Y, Z, levels=[0])
         plt.show()
```



8 Calculer l'expression de la frontière de décision de la règle de Bayes $\delta^\star$ lorsque Σ_1 et Σ_2 sont diagonales ($\Sigma_k = \text{diag}(\sigma_{k1}^2, \sigma_{k2}^2)$, pour $k = 1, 2$).

On remarquera tout d'abord que si Σ_k est diagonale, $|\Sigma_k| = \sigma_{k1}^2\sigma_{k2}^2$. De plus, on a

$$(\boldsymbol{x} - \mu_k)^T \Sigma_k^{-1} (\boldsymbol{x} - \mu_k) = \left(\frac{x_1 - \mu_{k1}}{\sigma_{k1}}\right)^2 + \left(\frac{x_2 - \mu_{k2}}{\sigma_{k2}}\right)^2.$$

En utilisant le fait que $\pi_1 = \pi_2$, on a donc la frontière de décision suivante :

$$\left(\frac{x_1 - \mu_{11}}{\sigma_{11}}\right)^2 - \left(\frac{x_1 - \mu_{21}}{\sigma_{21}}\right)^2 + \left(\frac{x_2 - \mu_{12}}{\sigma_{12}}\right)^2 - \left(\frac{x_2 - \mu_{22}}{\sigma_{22}}\right)^2 + 2\log\frac{\sigma_{11}\sigma_{12}}{\sigma_{21}\sigma_{22}} = 0.$$

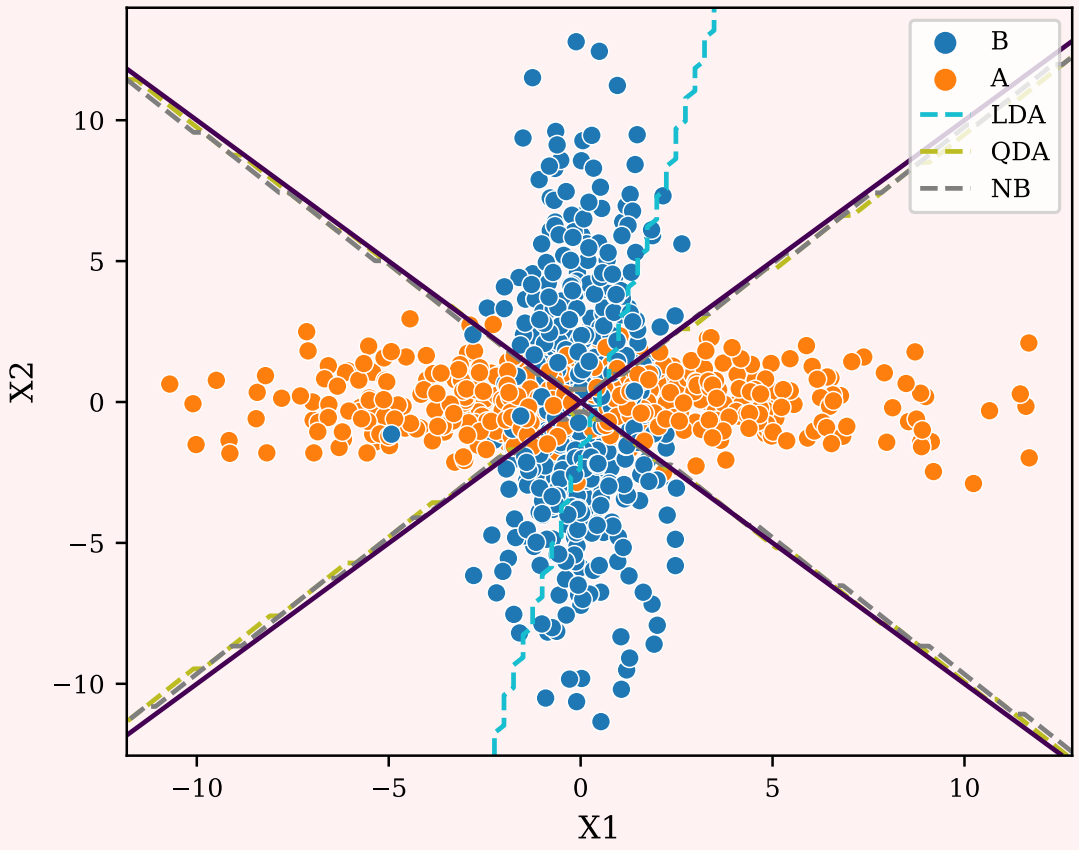
9 Représenter cette frontière et comparer aux frontières obtenues avec les trois modèles d'analyse discriminante pour le jeu de données `SynthPlus_n1000_p2.csv`, généré avec les paramètres suivants :

$$\mu_1 = \begin{pmatrix}0\\0\end{pmatrix}, \quad \mu_2 = \begin{pmatrix}0\\0\end{pmatrix}, \quad \Sigma_1 = \begin{pmatrix}15 & 0\\0 & 1\end{pmatrix}, \quad \Sigma_2 = \begin{pmatrix}1 & 0\\0 & 15\end{pmatrix}.$$

En remplaçant, on trouve

$$x_1^2 - x_2^2 = 0.$$

```
In [19]: df = pd.read_csv("data/SynthPlus_n1000_p2.csv")
         ax = sns.scatterplot(x="X1", y="X2", data=df, hue="z")
         add_decision_boundaries(df, models)
         xlim = ax.get_xlim()
         ylim = ax.get_ylim()

         resolution = 1000
         xx = np.linspace(xlim[0], xlim[1], resolution)
         yy = np.linspace(ylim[0], ylim[1], resolution)
         X, Y = np.meshgrid(xx, yy)

         Z = X**2-Y**2
         plt.contour(X, Y, Z, levels=[0])
         plt.show()
```



10 Calculer la frontière de décision de la règle de Bayes $\delta^\star$ lorsque $\Sigma_1 = \Sigma_2 = \Sigma$.

On a dans ce cas la frontière de décision

$$\left(\Sigma^{-1}(\mu_1-\mu_2)\right)^T\left(\boldsymbol{x}-\frac{\mu_1+\mu_2}{2}+\frac{\ln(\pi_1/\pi_2)}{(\mu_1-\mu_2)^T\Sigma^{-1}(\mu_1-\mu_2)}(\mu_1-\mu_2)\right)=0.$$

Après simplification (notamment en utilisant $\pi_1=\pi_2$ et $\mu_1=-\mu_2$), on obtient une frontière définie par

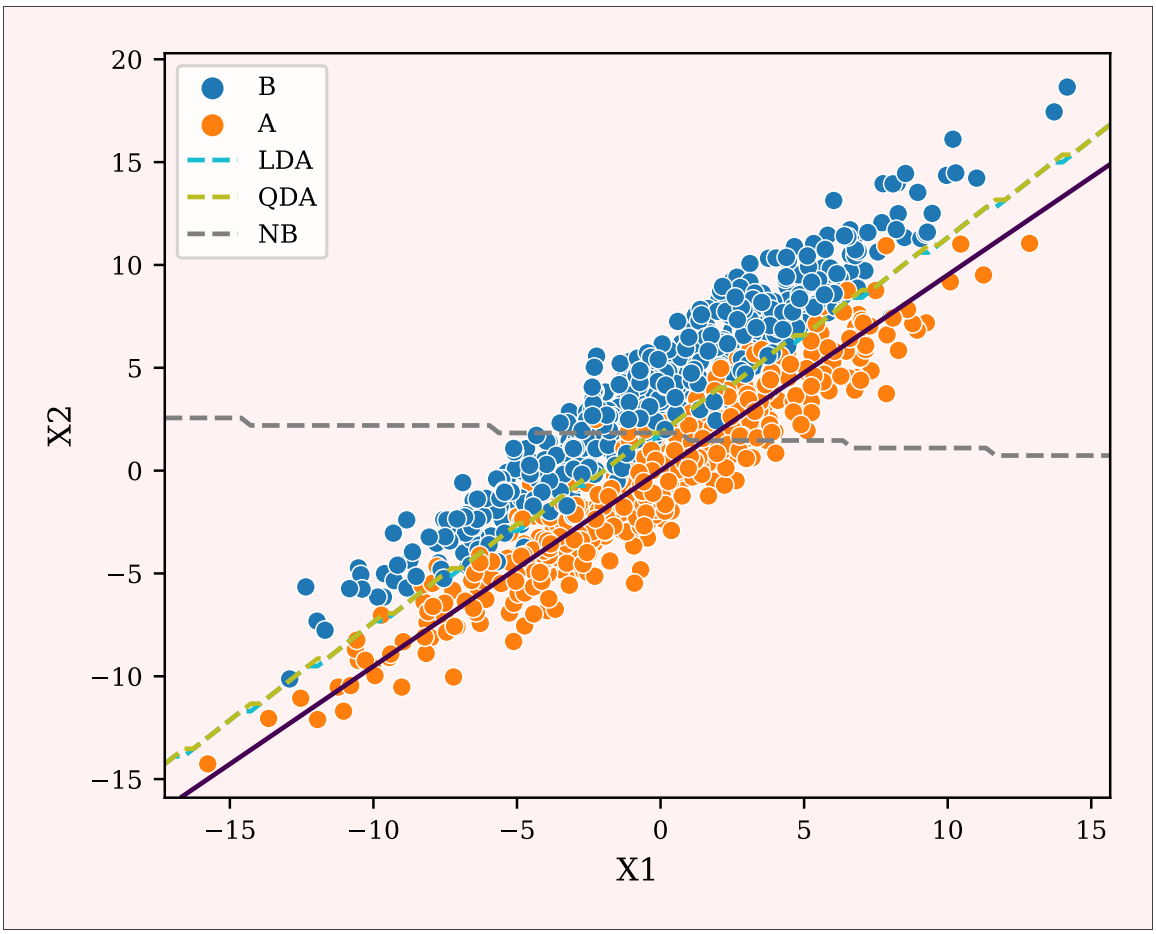
$$(\mu_1-\mu_2)^T\Sigma^{-1}\boldsymbol{x}=0.$$

11 Représenter cette frontière et comparer aux frontières obtenues avec les trois modèles d'analyse discriminante pour le jeu de données `SynthPara_n1000_p2.csv`, généré avec les paramètres suivants :

$$\mu_1=\begin{pmatrix}0\\-2\end{pmatrix},\quad \mu_2=\begin{pmatrix}0\\2\end{pmatrix},\quad \Sigma=\Sigma_1=\Sigma_2=\frac{1}{2}\begin{pmatrix}41&39\\39&41\end{pmatrix}.$$

En réduisant, on trouve $x_2=\frac{39}{41}x_1$.

```
In [20]: df = pd.read_csv("data/SynthPara_n1000_p2.csv")
         ax = sns.scatterplot(x="X1", y="X2", data=df, hue="z")
         add_decision_boundaries(df, models)
         xlim = ax.get_xlim()
         ylim = ax.get_ylim()

         resolution = 1000
         xx = np.linspace(xlim[0], xlim[1], resolution)
         yy = np.linspace(ylim[0], ylim[1], resolution)
         X, Y = np.meshgrid(xx, yy)

         Z = 39*X-41*Y
         plt.contour(X, Y, Z, levels=[0])
         plt.show()
```


B
A
LDA
QDA
NB
X1
X2
20
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15