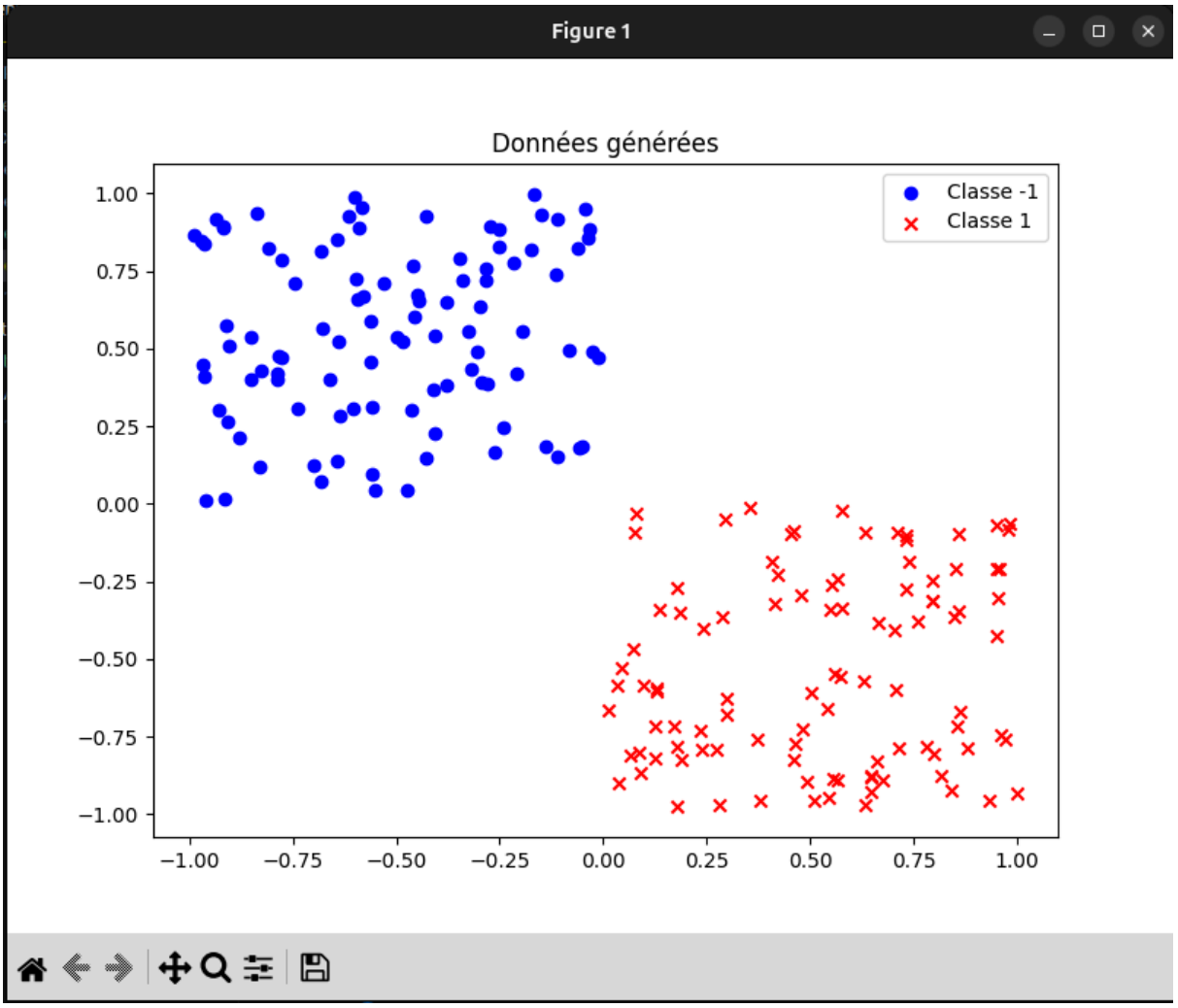
Anass EZZINE
4A FISA
Departement informatique

TP-Perceptron

1. Algorithme du perceptron pour la classification binaire

1.1. Commençons par générer et afficher un jeu de données en utilisant la fonction `generateData` fournie :



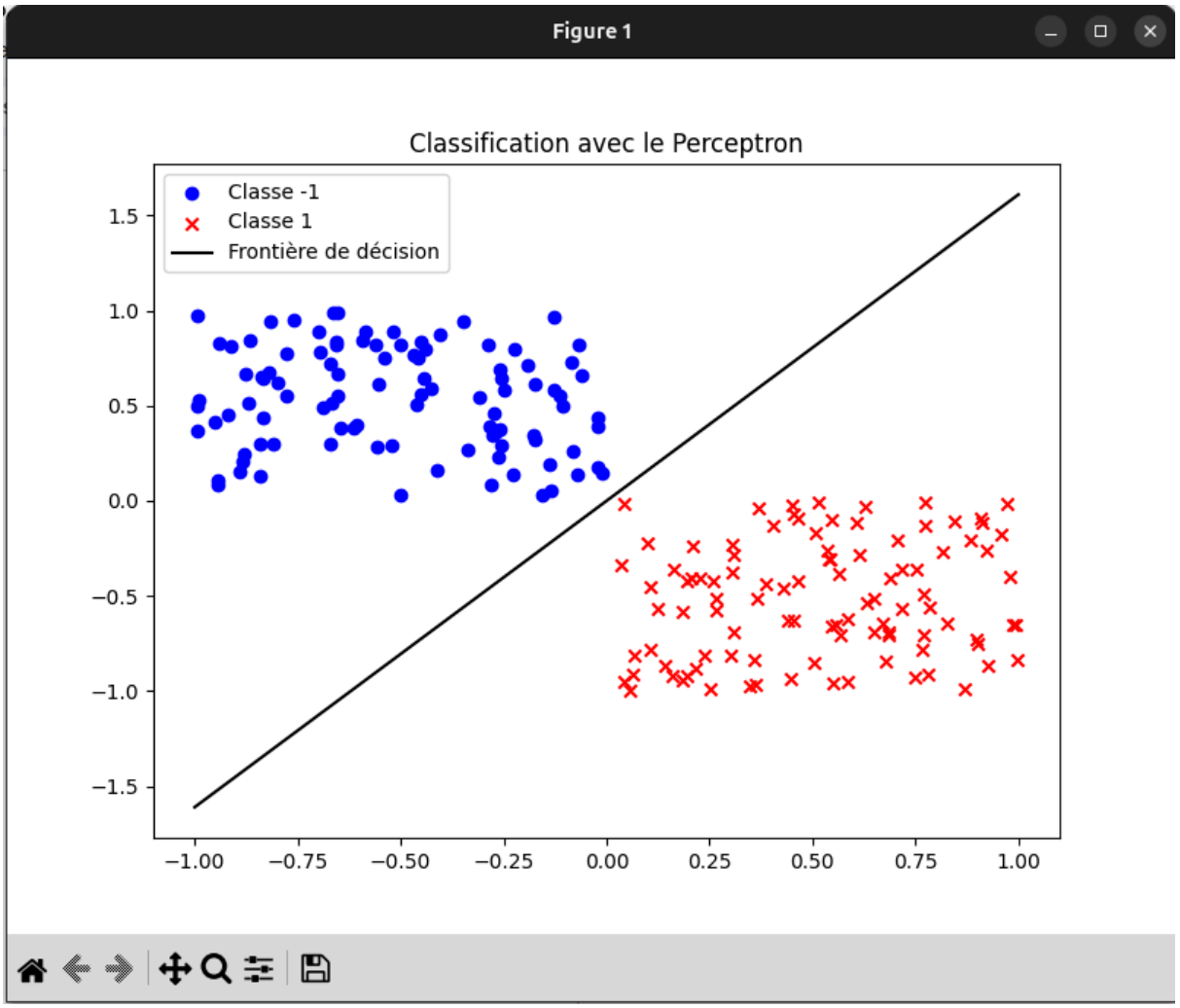
1.2.

Alors, voilà comment ça marche :
Au début, on part d'un vecteur de poids w initialisé à zéro, Ensuite, on répète une boucle : à chaque tour, on regarde chaque exemple de données. Si un point est mal classé (par exemple, il devrait être rouge mais le modèle le voit comme bleu), on ajuste w pour corriger l'erreur. On s'arrête seulement quand tous les exemples sont bien classés (plus d'erreurs !).
Pour montrer le résultat :

- Les points bleus sont ceux de la classe -1,
- Les points rouges, la classe 1,

Anass EZZINE
4A FISA
Departement informatique

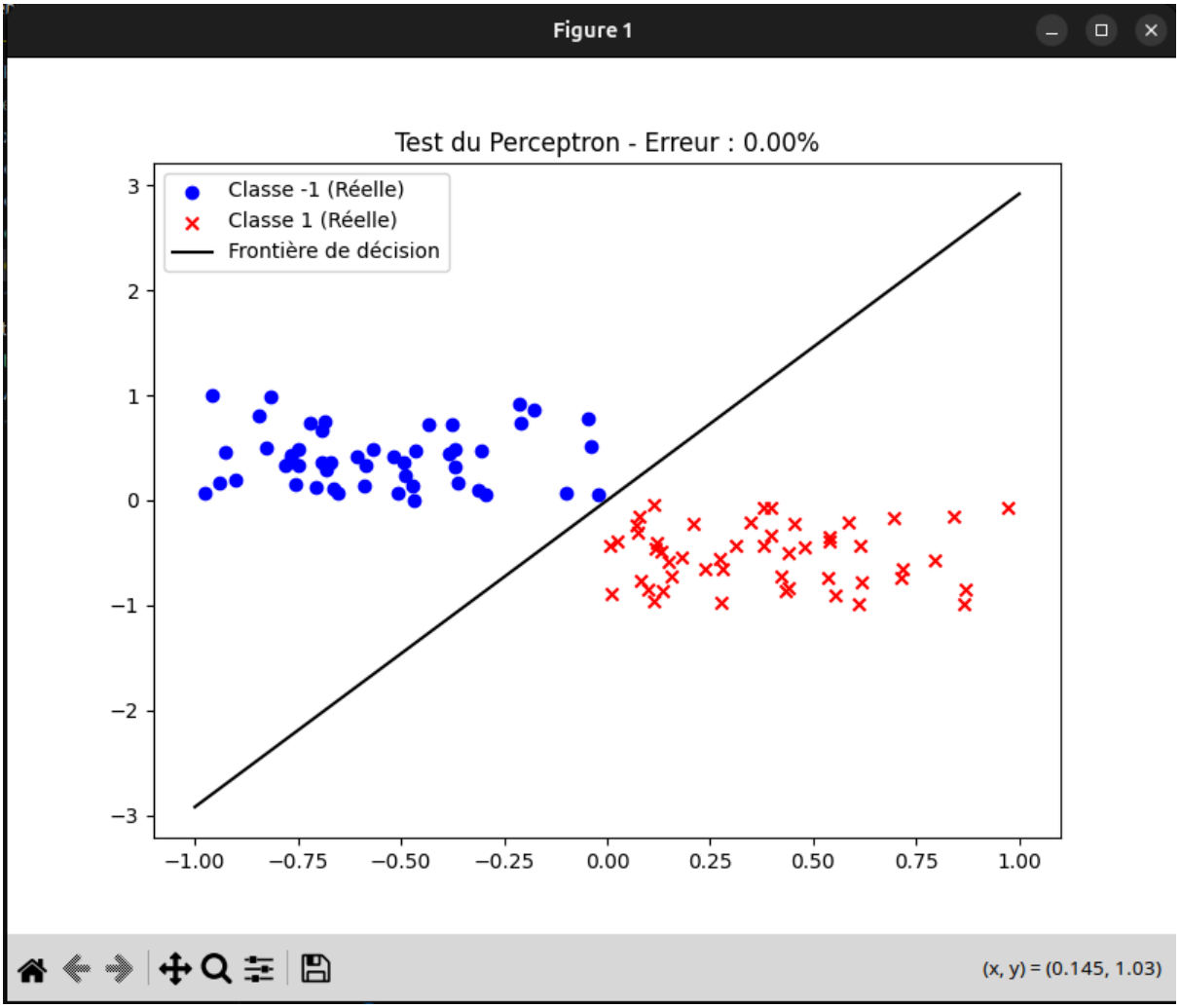
- La ligne noire, c'est la frontière (hyperplan) que le perceptron a apprise pour séparer les deux groupes.



1.3.

Si les données sont bien linéairement séparables, l'erreur doit être proche de 0%.Sinon, l'erreur peut être plus élevée.

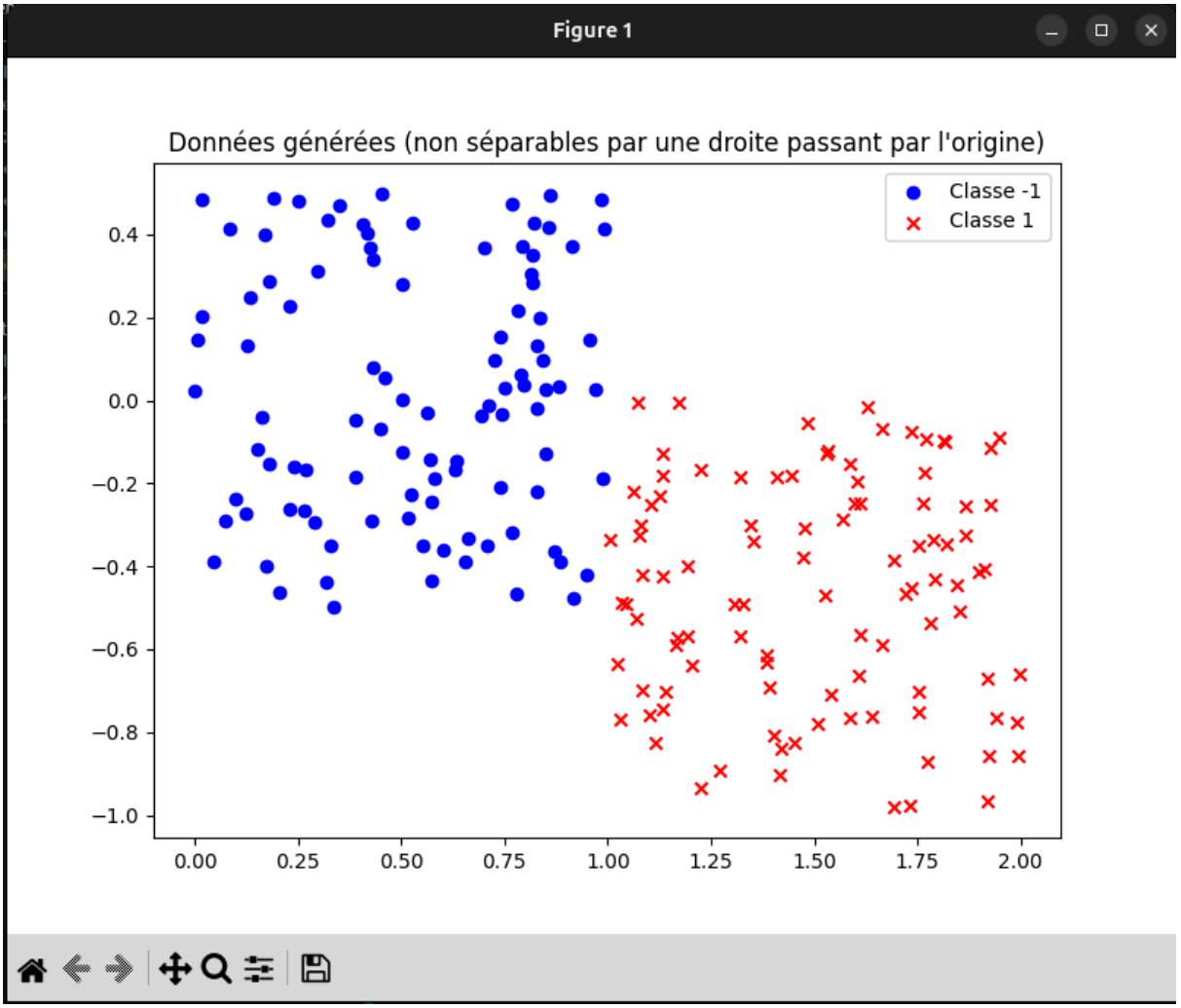
Anass EZZINE
4A FISA
Departement informatique



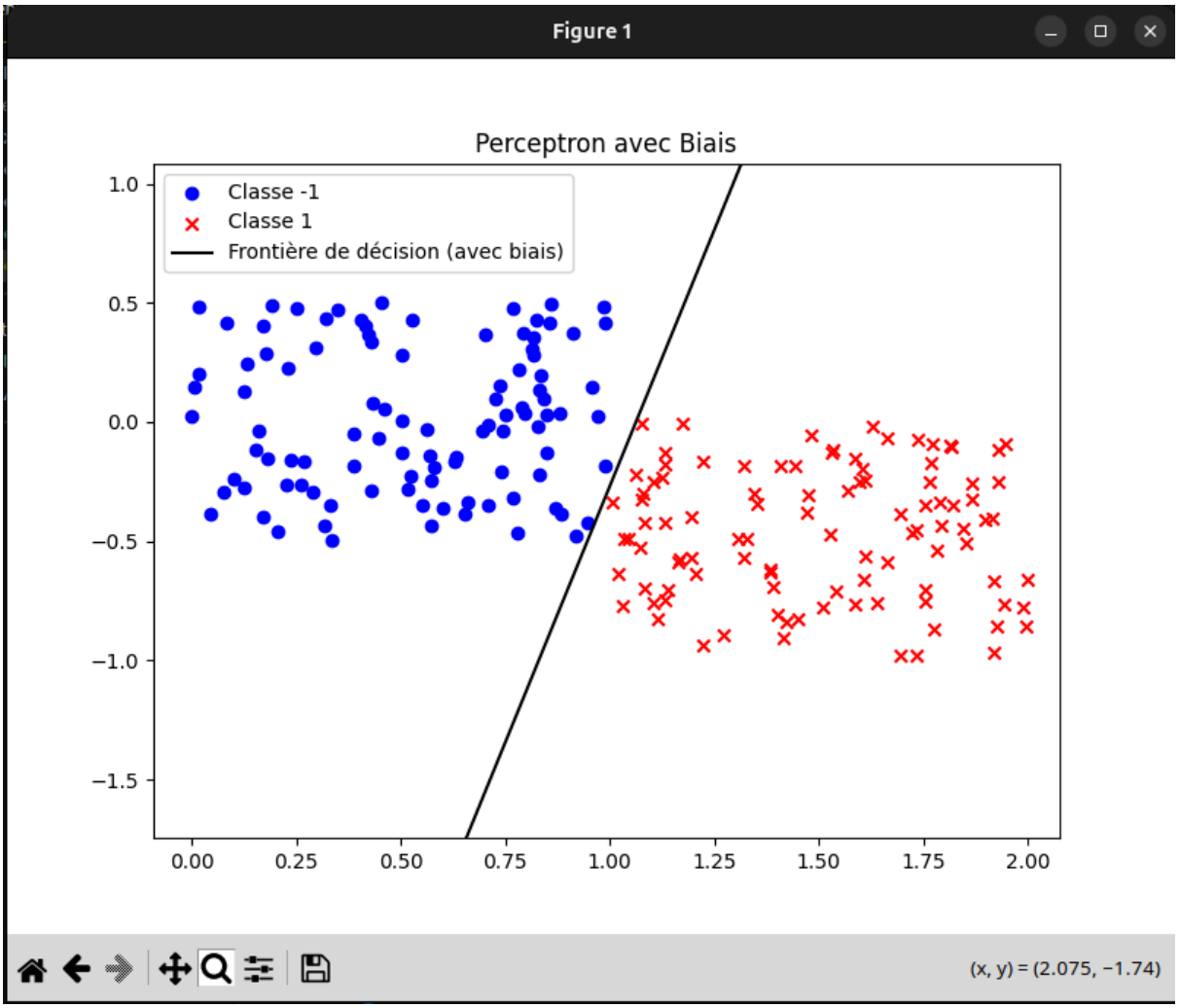
1.4.

Maintenant, passons à l'extension avec un biais, c'est-à-dire le cas où les données ne sont pas séparables par une droite passant par l'origine. Je modifie la fonction de génération des données (`generateData2`) pour qu'elles soient toujours séparables, mais par une droite qui ne passe pas par l'origine, puis je complète les échantillons en ajoutant une coordonnée constante `1` pour prendre en compte le biais.

Anass EZZINE
4A FISA
Departement informatique



Anass EZZINE
4A FISA
Departement informatique

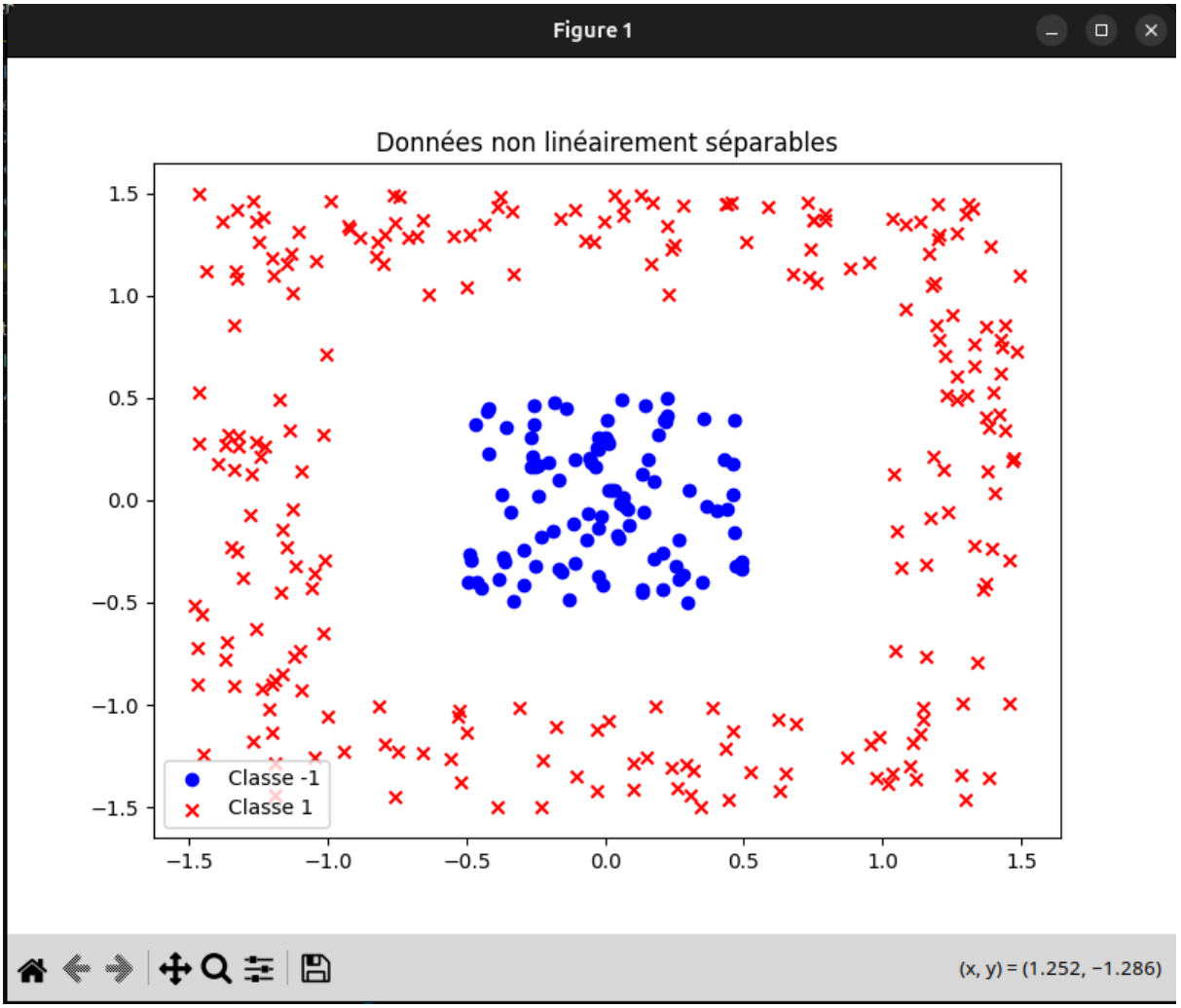


2. Perceptron `a noyau

2.1.

Je commence par la génération de données non linéairement séparables avec la fonction generateData3 :

Anass EZZINE
4A FISA
Departement informatique



2.2.

Je définit un plongement polynomial pour transformer les données :

```python
def polynomial_feature_mapping(X):
    X_poly = np.c_[np.ones(X.shape[0]), X[:, 0], X[:, 1], X[:, 0]**2, X[:, 0] * X[:, 1], X[:, 1]**2]
    return X_poly

# Transformation des données
X3_poly = polynomial_feature_mapping(X3)
```

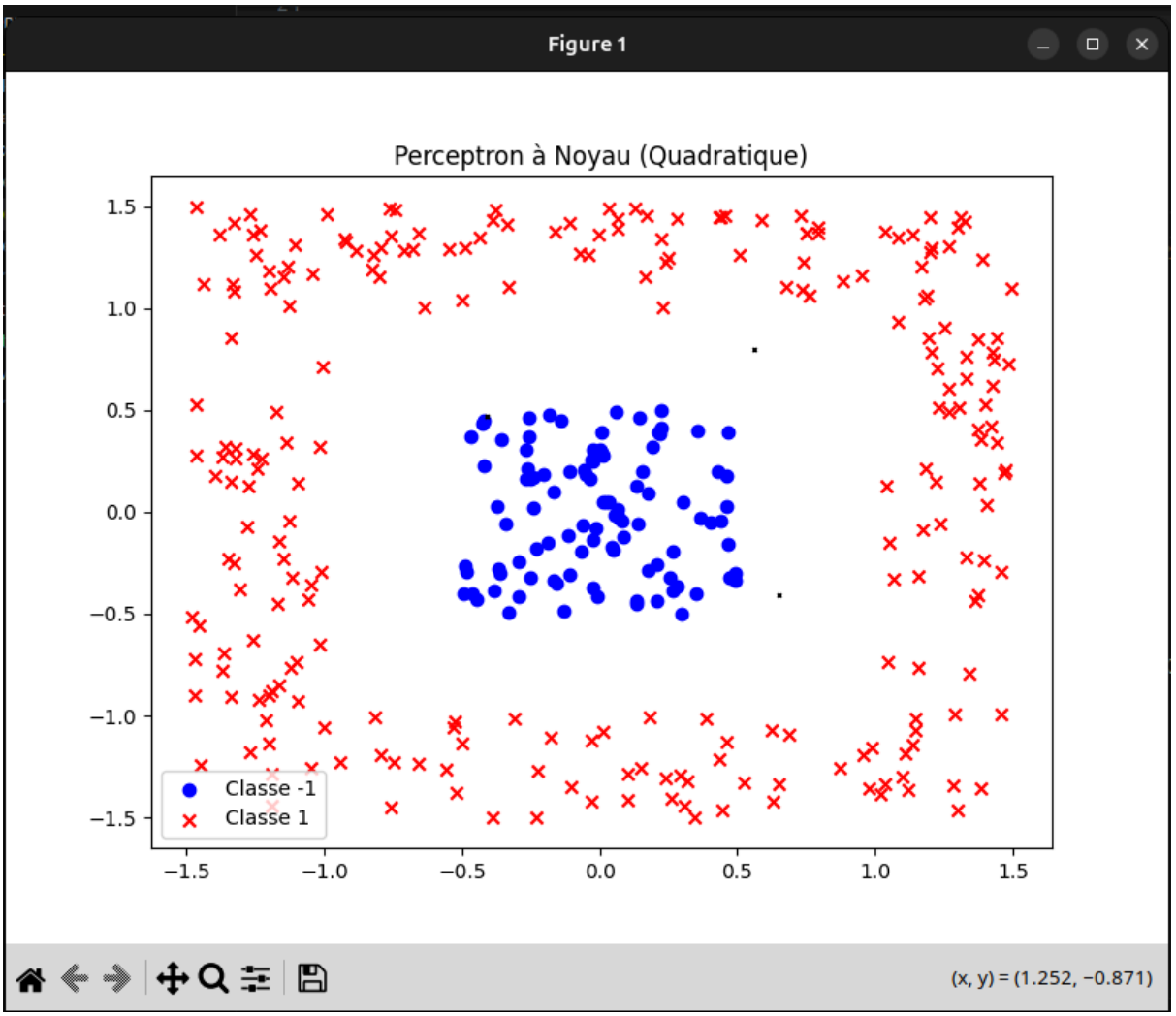
2.3.

Le perceptron standard stocke un vecteur de poids w, mais avec un noyau, on stocke plutôt un ensemble de coefficients α associés aux échantillons support

2.4.

Pour prédire la classe d'un nouveau point, on utilise : $f(x) = \sum_{i \in V} \alpha_i y_i k(x_i, x)$

Anass EZZINE
4A FISA
Departement informatique
On remarque que les données non séparables linéairement et que la frontière de séparation est courbée pour mieux séparer les classes:



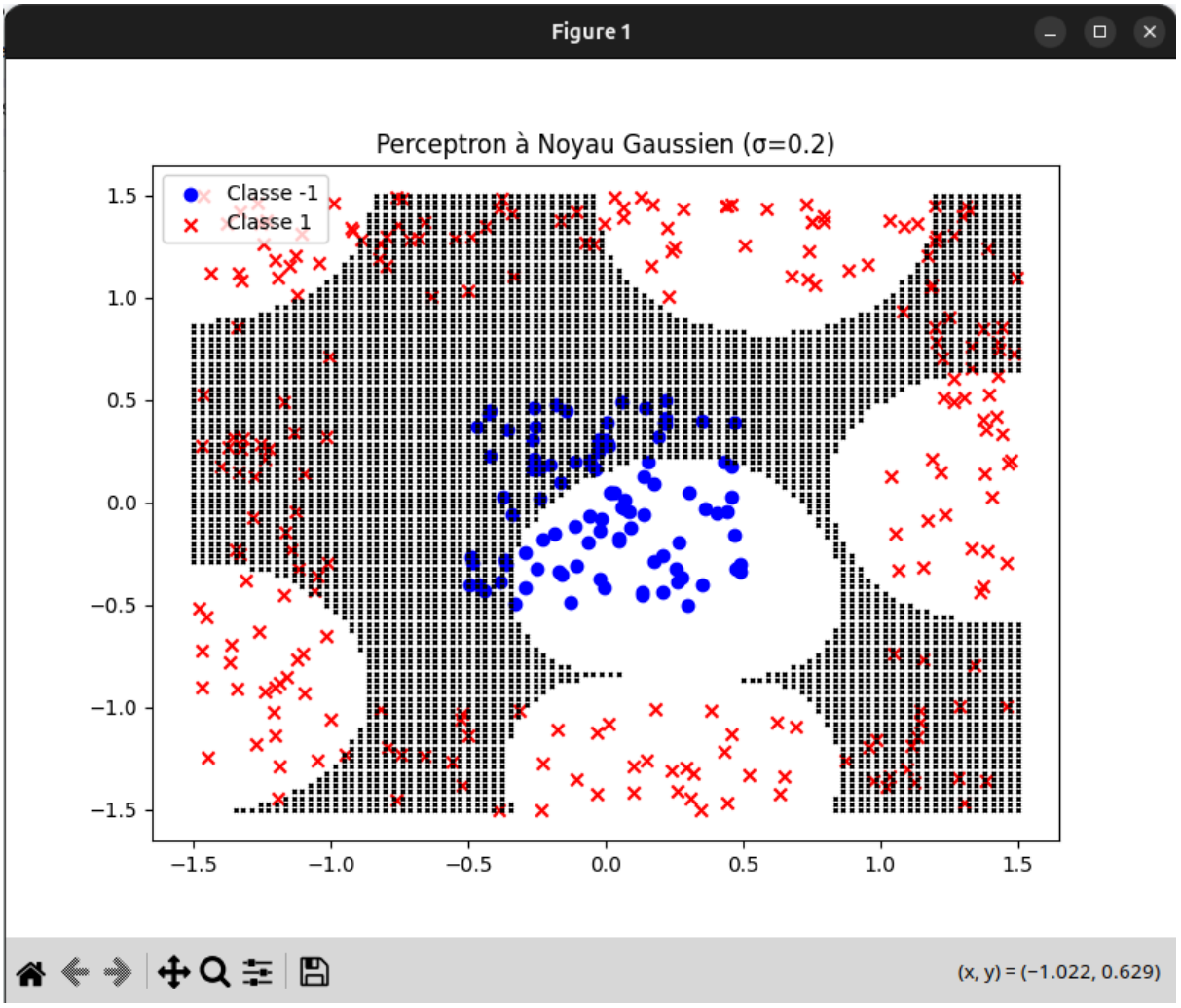
2.5.

On peut aussi tester un noyau Gaussien sur plusieur valeur sigma et on peut voir comment la séparation change pour chaque valeur :

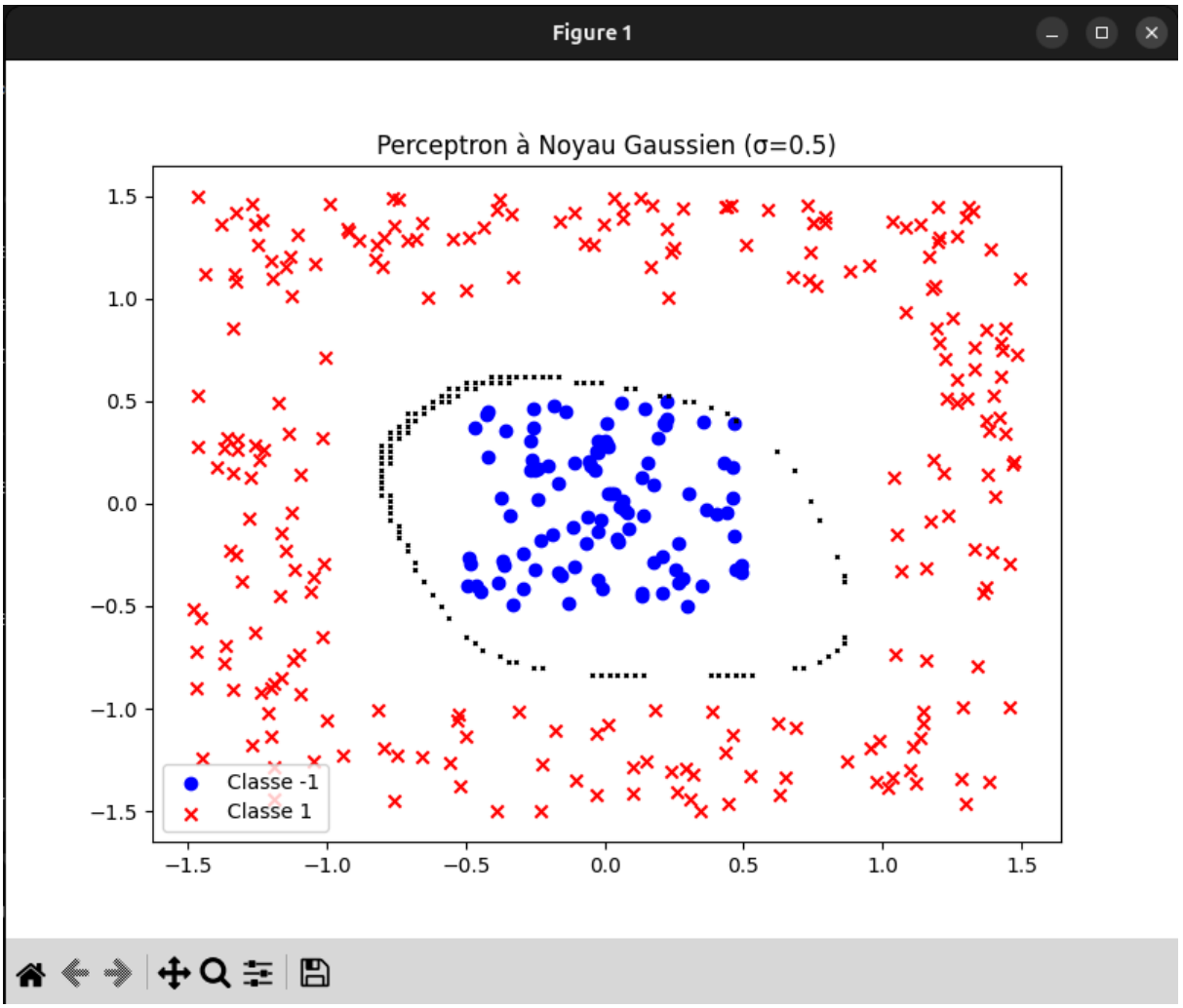
```
def kernel_gaussian(x, y, sigma=1):
    return np.exp(-np.linalg.norm(np.array(x) - np.array(y))**2 / (2 * sigma**2))

# Entraînement du perceptron avec noyau Gaussien
alphas_gauss, support_vectors_gauss = perceptron_kernel(X3, Y3, kernel_gaussian)
```

Anass EZZINE
4A FISA
Departement informatique

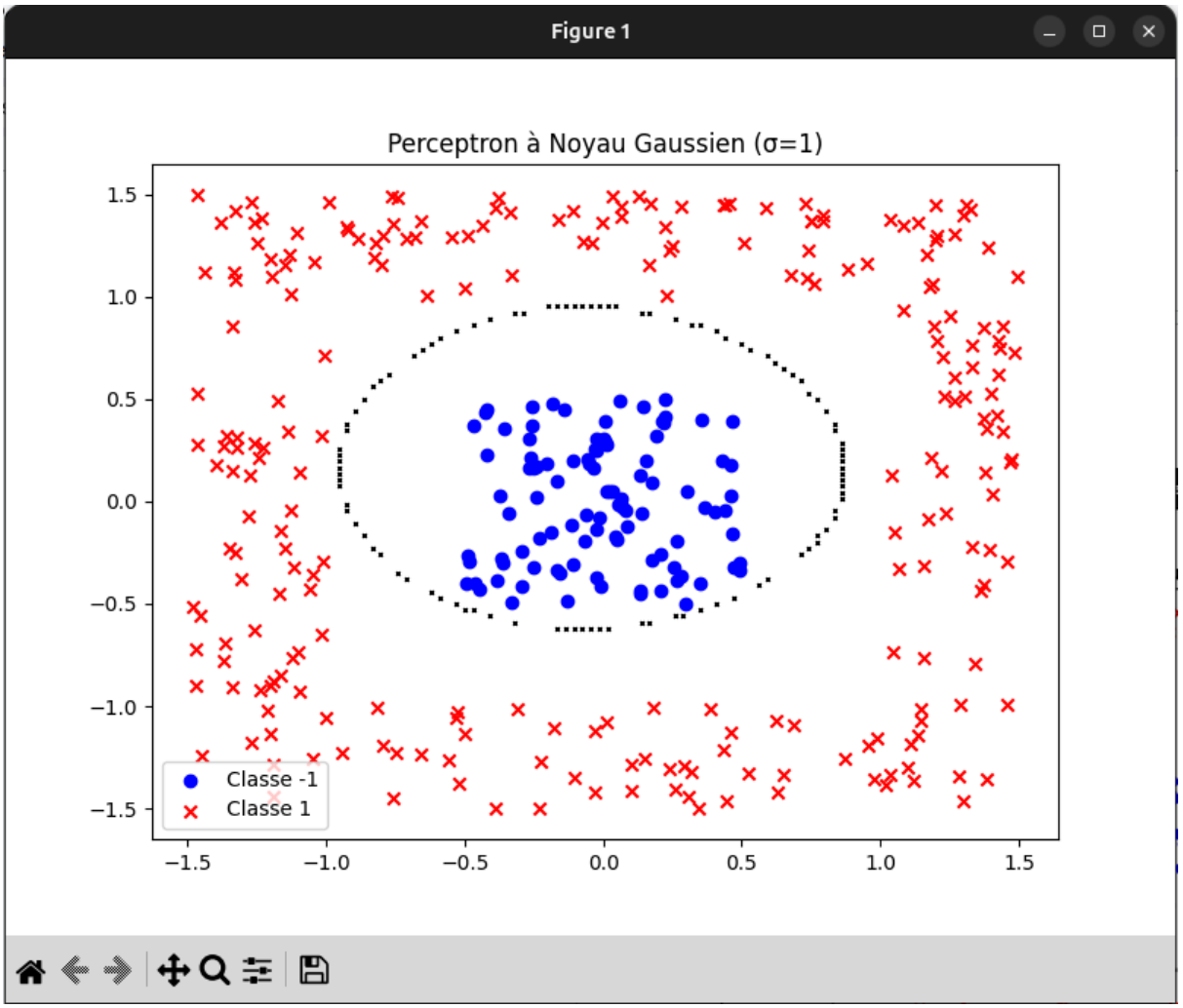
```python
# Expérimentation avec différents sigma
for sigma in [0.2, 0.5, 1, 2, 5]:
    alphas_gauss, support_vectors_gauss = perceptron_kernel(X3, Y3, lambda x, y: kernel_gaussian(x, y, sigma))

    plt.figure(figsize=(8, 6))
    plt.scatter(X3[Y3 == -1, 0], X3[Y3 == -1, 1], color='blue', marker='o', label='Classe -1')
    plt.scatter(X3[Y3 == 1, 0], X3[Y3 == 1, 1], color='red', marker='x', label='Classe 1')

    # Tracé de la frontière de séparation
    res = 100
    x_vals = np.linspace(-1.5, 1.5, res)
    y_vals = np.linspace(-1.5, 1.5, res)

    for x in range(res):
        for y in range(res):
            if abs(f_from_kernel(alphas_gauss, support_vectors_gauss, lambda x, y: kernel_gaussian(x, y, sigma), [x_vals[x], y_vals[y]])) < 0.01:
                plt.plot(x_vals[x], y_vals[y], 'kx', markersize=2)  # Points proches de la frontière

    plt.legend()
    plt.title(f"Perceptron à Noyau Gaussien (σ={sigma})")
    plt.show()
```



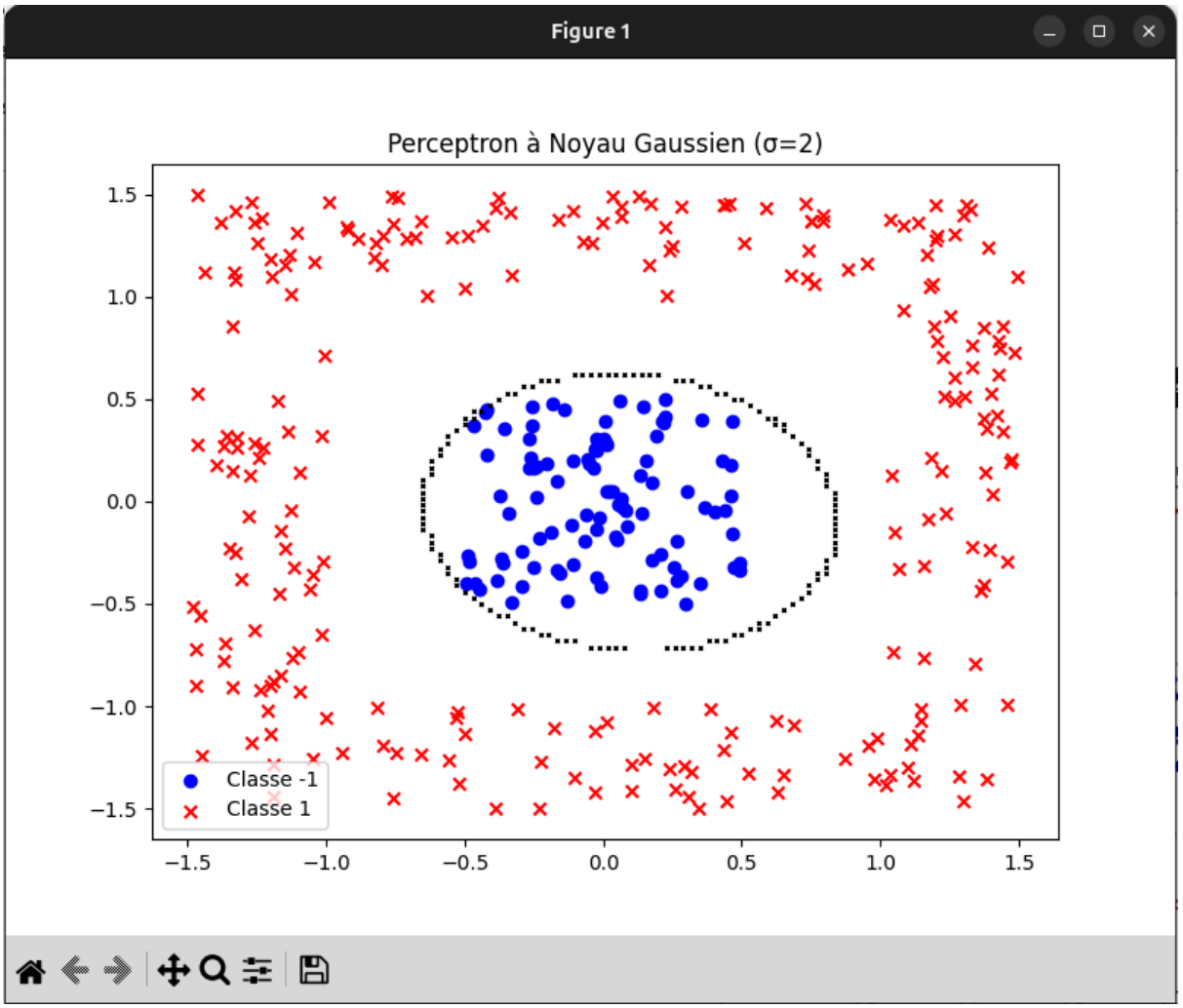
Anass EZZINE
4A FISA
Departement informatique



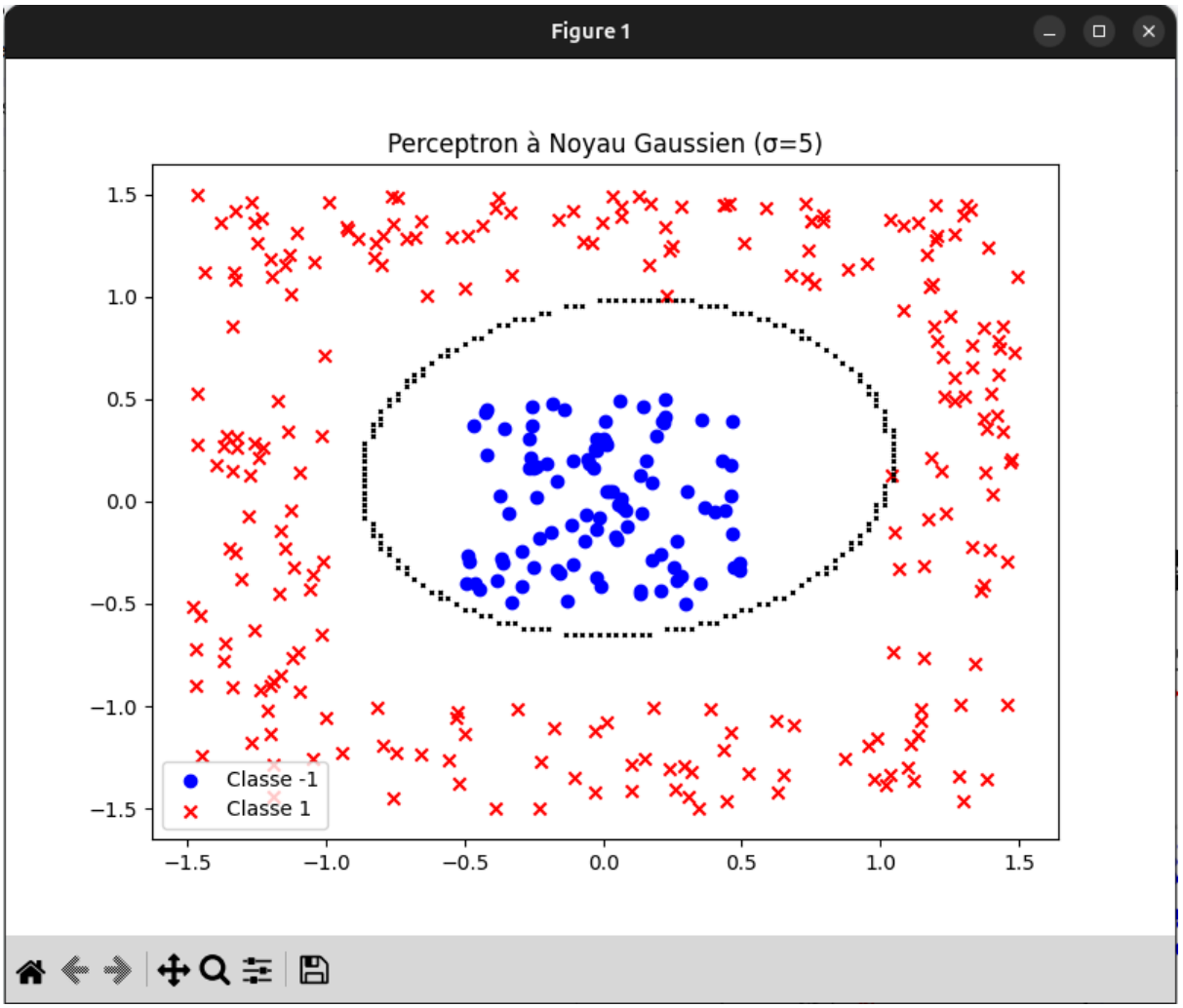
Anass EZZINE
4A FISA
Departement informatique



Anass EZZINE
4A FISA
Departement informatique



Anass EZZINE
4A FISA
Departement informatique



3. Exercice

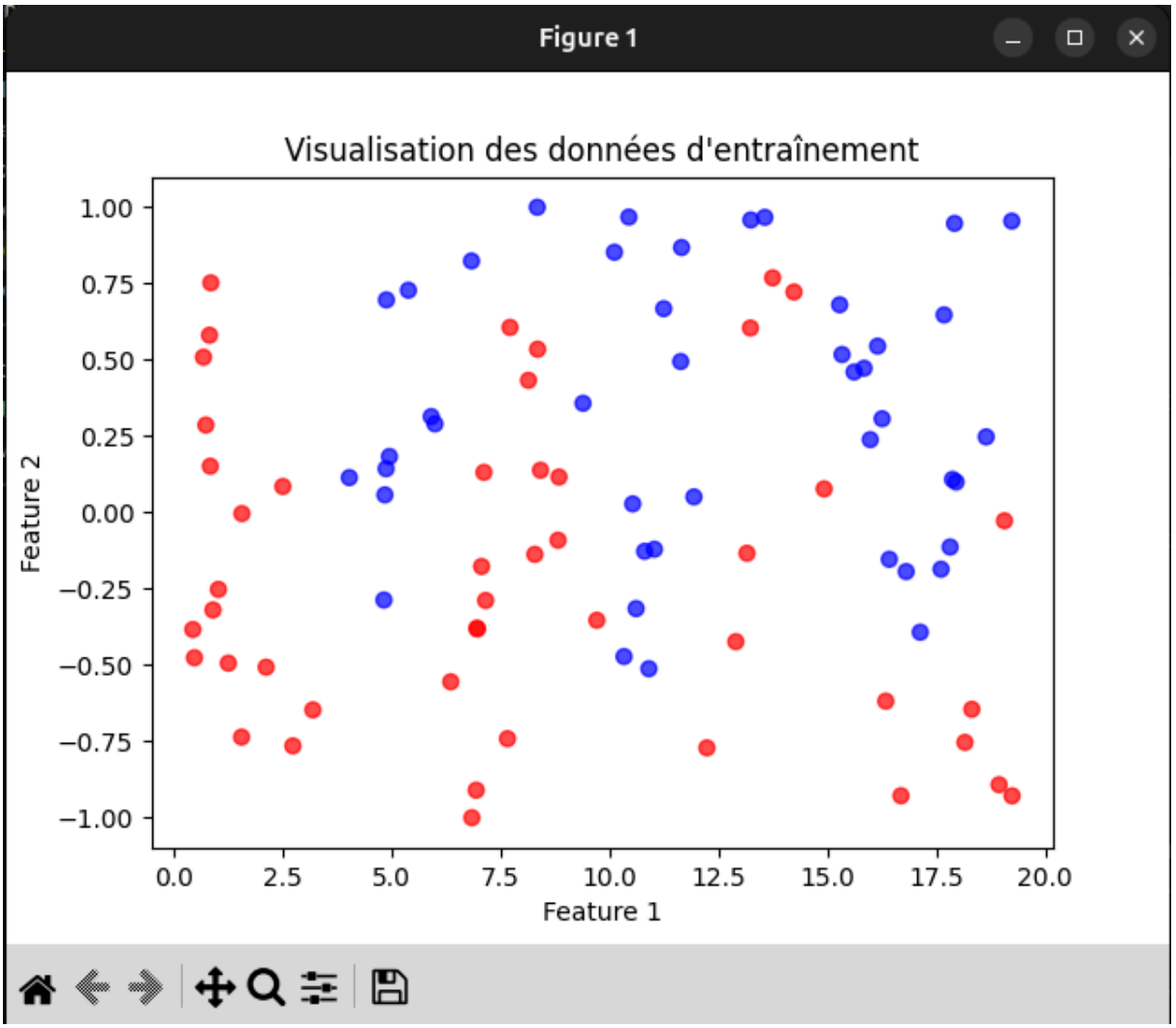
3.1.

Cette fonction mesure la précision d'un modèle avancé en s'appuyant sur plusieurs éléments : les coefficients (`coeffs`), un ensemble de supports (`support_set`) et une fonction `k`, qui peut agir comme un noyau dans une version améliorée du perceptron. Elle détermine la classe d'un échantillon en combinant linéairement les supports, puis applique la fonction `np.sign()` pour produire une classification binaire :

```python
def score(S, coeffs, support_set, k):
    correct_predictions = 0
    total_samples = len(S)

    for x, label in S:
        prediction = np.sign(sum(coeff * k(support, x) for coeff, support in zip(coeffs, support_set)))
        if prediction == label:
            correct_predictions += 1

    return correct_predictions / total_samples
```

3.2.

Anass EZZINE
4A FISA
Departement informatique




```
(venv) anass@anass-ezzine:~/Bureau/FISA/S8/AA/TP-Perceptron$ python exercice3.py
Accuracy: 62.50%
(venv) anass@anass-ezzine:~/Bureau/FISA/S8/AA/TP-Perceptron$
```