TP3 ATDN2

Alexis XIONG

Partie 1 : Optimisation Bayésienne

1. L'optimisation bayésienne est une technique d'optimisation qui a pour but de minimiser le nombre d'observations tout en convergeant rapidement vers la solution optimale. Elle se base sur trois caractéristiques fondamentales :

- Le processus Gaussien
- L'exploration et l'exploitation
- La fonction d'acquisition

Pour gérer les fonctions coûteuses, elle utilise un modèle probabiliste qui permet d'utiliser de manière efficace les données, la fonction d'acquisition permet de choisir les points important à évaluer tout en évitant les points inutiles et enfin la prédiction des potenciels optimums.

2. Le processus gaussien est une collection de variables aléatoires où chaque variable suit un loi normale multidimensionnelle. Un processus gaussien nous permet donc de définir une distribution de probabilité sur un ensemble de fonction.

Les processus gaussiens sont utilisés pour la modélisation de la fonction objectif car elle s'adapte à la complexité de la fonction objectif, elle peut etre linéaire, non linéaire où autre, les processus gaussiens vont quand même pour s'adapter à tout type de fonction. De plus, l'incertitude est également une raison pour laquelle les processus gaussiens sont utilisés, car elle permettent de prédire la valeur de la fonction objectif en donnant une incertitude à leur prédiction.

3. Les fonctions d'acquisition utilisent un modèle probabiliste pour estimer la fonction objectif et quantifier l'incertitude. Expected Improvement : Il mesure l'amélioration attendue par rapport à la meilleure valeur obtenue. UCB : L'Upper Confidence Bound utilise les points qui maximisent une borne supérieure de confiance sur la valeur de la fonction objective. Elle utilise la moyenne prédite et l'écart type prédite.

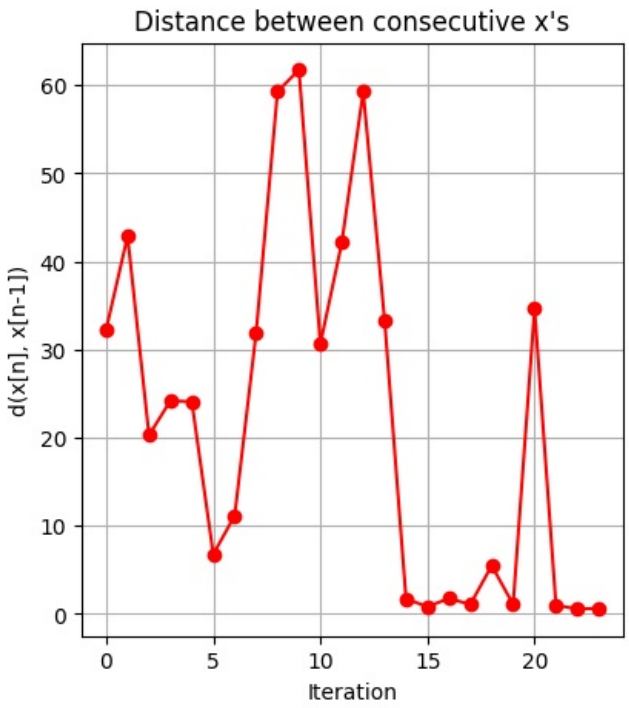
Compris entre Exploration/Exploitation : Les fonctions d'acquisition permettent de naviguer dans l'espace de recherche en tenant compte à la fois de la connaissance actuelle de la fonction objective et de l'incertitude associée à ce dernier. Elles permettent d'équilibrer l'exploration de nouvelles régions pour découvrir des optimas potentiels et l'exploration de régions prometteuses.

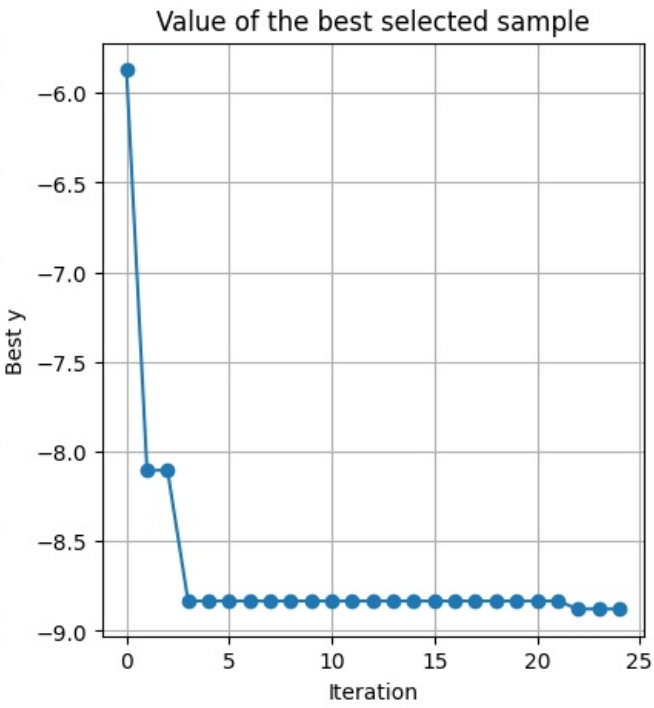
Questions 4) à 7) :

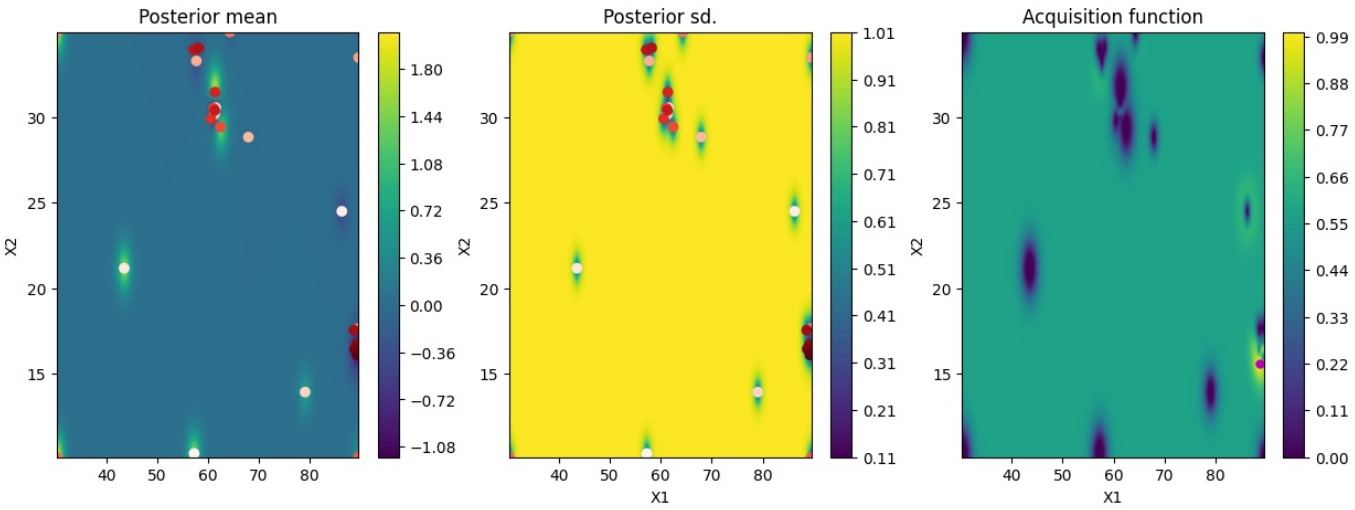
In [ ]:
```
import numpy as np
import pandas as pd
import GPy
import GPyOpt
import matplotlib.pyplot as plt

data = pd.read_csv('tp2_atdn_donnees.csv')

X = data[['Humidité (%)', 'Température (°C)']].values
y = data['Rendement agricole (t/ha)'].values.reshape(-1, 1)

# On définit la fonction objectif
def fonction_objectif(x):

    humidite = x[:, 0]
    temperature = x[:, 1]
    le_plus_proche = np.argmin(np.sum((X - x) ** 2, axis=1))
    return -y[le_plus_proche] # car on maximise le profit

bounds = [
    {'name': 'humidity', 'type': 'continuous', 'domain': (data['Humidité (%)'].min(), data['Humidité (%)'].max(
    {'name': 'temperature', 'type': 'continuous', 'domain': (data['Température (°C)'].min(), data['Température
]

# Utilisation de l'optimisation bayésienne
optimisation = GPyOpt.methods.BayesianOptimization(f = fonction_objectif, domain = bounds, model_type='GP', acq
optimisation.run_optimization(max_iter=30)

print("Meilleure humidité:", optimisation.x_opt[0])
print("Meilleure température:", optimisation.x_opt[1])
print("Rendement agricole maximal prédit:", -optimisation.fx_opt)

# Visualisation
optimisation.plot_convergence()
plt.show()
```

```
optimisation.plot_acquisition()
plt.show()
```

```
Meilleure humidité: 89.22740316526485
Meilleure température: 16.74545607995642
Rendement agricole maximal prédit: 8.877858327709873
```



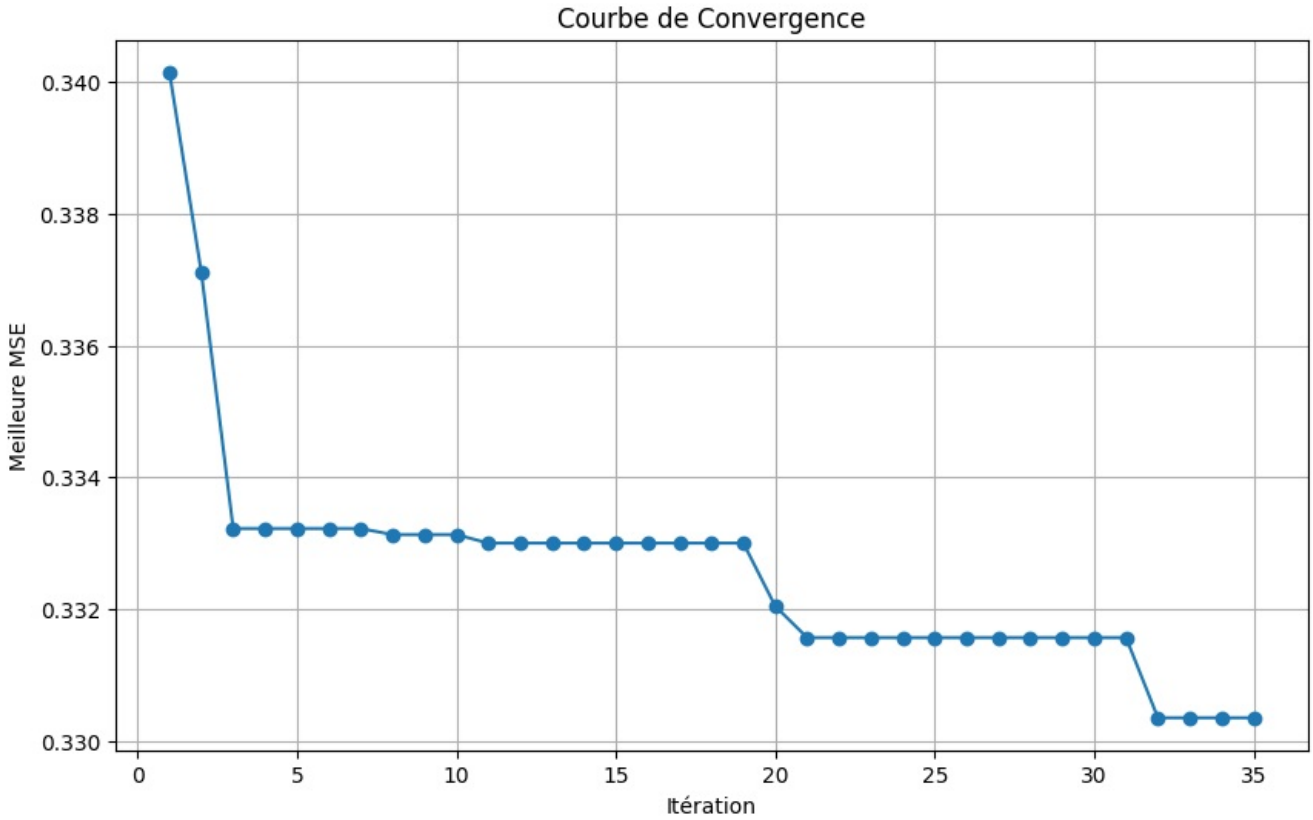


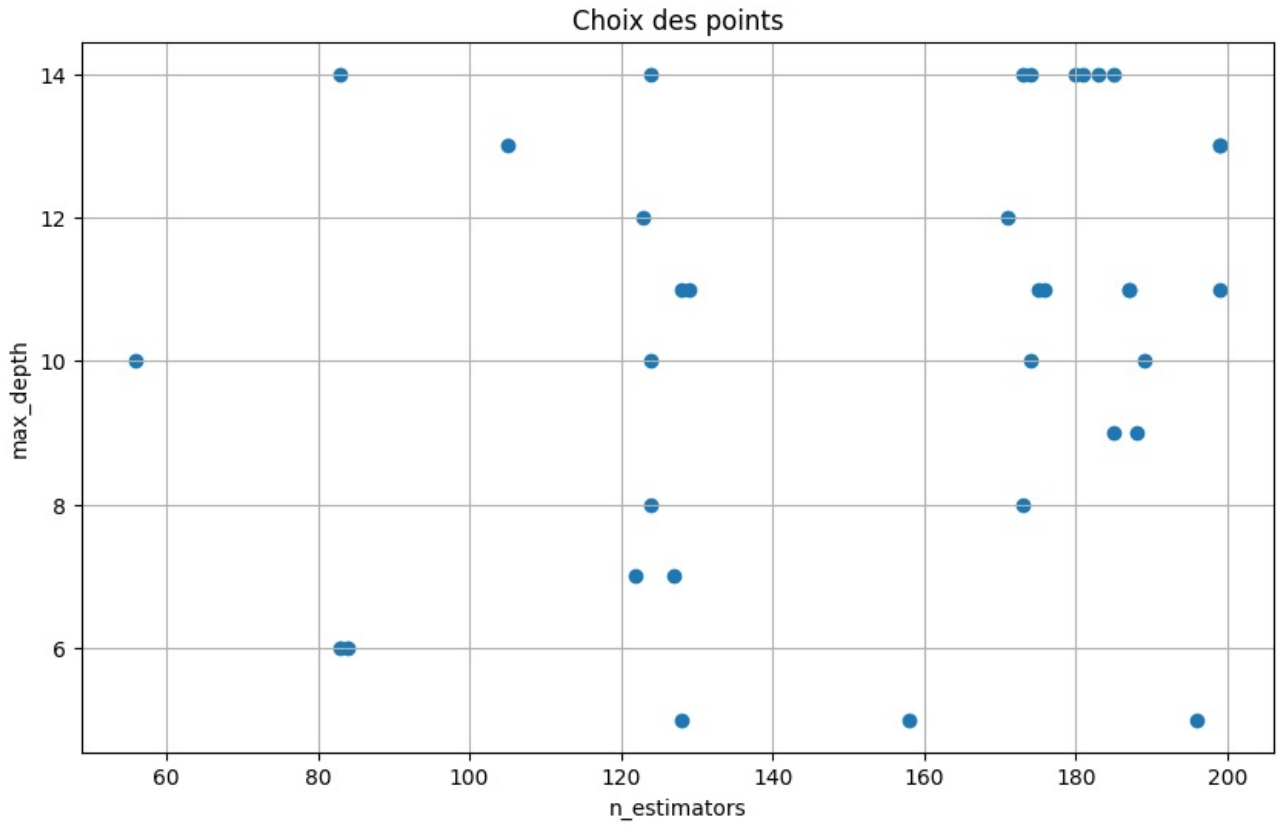


In [ ]:
```
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error


data = pd.read_csv('tp2_atdn_donnees.csv')

X = data[['Humidité (%)', 'Température (°C)', 'pH du sol', 'Précipitations (mm)']]
y = data['Rendement agricole (t/ha)']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# On définit la fonction objectif
def fonction_objectif(params):
    n_estimators = int(params[0, 0])
    max_depth = int(params[0, 1])
    min_samples_split = int(params[0, 2])

    rf = RandomForestRegressor(n_estimators=n_estimators, max_depth=max_depth, min_samples_split=min_samples_sp
    rf.fit(X_train, y_train)

    # Utilisation du MSE
    y_prediction = rf.predict(X_test)
    mse = mean_squared_error(y_test, y_prediction)
    return mse

espace_de_recherche = [
    {'name': 'n_estimators', 'type': 'discrete', 'domain': np.arange(50, 200)},
    {'name': 'max_depth', 'type': 'discrete', 'domain': np.arange(5, 15)},
```

```
    {'name': 'min_samples_split', 'type': 'discrete', 'domain': np.arange(2, 10)}
]

# On utilise l'optimisation bayésienne
optimisation_bayesienne = GPyOpt.methods.BayesianOptimization(f=fonction_objectif, domain=espace_de_recherche, a
optimisation_bayesienne.run_optimization(max_iter=30)

print("Optimisation bayésienne:")
print("Meilleurs paramètres:", optimisation_bayesienne.x_opt)
print("MSE:", optimisation_bayesienne.fx_opt)

# Grid Search
param_grid = {
    'n_estimators': np.arange(50, 200, 50),
    'max_depth': np.arange(5, 15, 5),
    'min_samples_split': [2, 5, 8]
}

grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, scoring='neg_mean_squared_error'
grid_search.fit(X_train, y_train)

print("-----------------------------------------------------")
print("Grid Search:")
print("Meilleurs paramètres:", grid_search.best_params_)
print("MSE:", -grid_search.best_score_)

# Random Search
param_dist = {
    'n_estimators': np.arange(50, 200),
    'max_depth': np.arange(5, 15),
    'min_samples_split': np.arange(2, 10)
}

random_search = RandomizedSearchCV(RandomForestRegressor(random_state=42), param_distributions=param_dist, n_it
random_search.fit(X_train, y_train)

print("-----------------------------------------------------")
print("\nRandom Search:")
print("Meilleurs paramètres:", random_search.best_params_)
print("MSE:", -random_search.best_score_)

# Courbe de convergence
courbe_convergence = pd.DataFrame({
    'Iteration': range(1, len(optimisation_bayesienne.Y) + 1),
    'Best MSE': np.minimum.accumulate(optimisation_bayesienne.Y).flatten()
})

# Visualisation
plt.figure(figsize=(10, 6))
plt.plot(courbe_convergence['Iteration'], courbe_convergence['Best MSE'], marker='o', linestyle='-')
plt.title('Courbe de Convergence')
plt.xlabel('Itération')
plt.ylabel('Meilleure MSE')
plt.grid(True)
plt.show()

choix_points = pd.DataFrame(optimisation_bayesienne.X, columns=['n_estimators', 'max_depth', 'min_samples_split
plt.figure(figsize=(10, 6))
plt.scatter(choix_points['n_estimators'], choix_points['max_depth'], marker='o', label='Points évalués')
plt.title('Choix des points')
plt.xlabel('n_estimators')
plt.ylabel('max_depth')
plt.grid(True)
plt.show()
```

```
Optimisation bayésienne:
Meilleurs paramètres: [124.  10.   4.]
MSE: 0.3303546115595213
-----------------------------------------------------
Grid Search:
Meilleurs paramètres: {'max_depth': 10, 'min_samples_split': 8, 'n_estimators': 150}
MSE: 0.32444791401028267
-----------------------------------------------------

Random Search:
Meilleurs paramètres: {'n_estimators': 78, 'min_samples_split': 3, 'max_depth': 9}
MSE: 0.32233966428983457
```





On remarque que l'optimisation bayésienne a un meilleur score que les autres modèles.

Avantages de l'optimisation bayésienne :

- L'optimisation bayésienne utilise un modèle probabiliste et explore l'espace de recherche de manière plus intelligente que les méthodes classiques comme Grid Search ou Random Search.
- Elle est particulièrement efficace lorsque l'évaluation de la fonction objective est coûteuse en temps et en ressources.
- Elle utilise l'incertitude, ce qui permet de faire un bon compromis entre l'exploration et l'exploitation.

Inconvénients de l'optimisation bayésienne :

- L'optimisation bayésienne est plus complexe à mettre en oeuvre par rapport à Grid Search ou Random Search. De plus, il a un temps d'éxécution qui est beaucoup plus long que ses derniers.
- Elle est sensible au choix de la fonction objectif, un mauvais choix peut entrainer de mauvaises performances.

Partie 2 : Modèles Bayésiens à Noyau

8. L'inférence bayésienne est une approche statistique qui met à jour nos croyances sur un événement ou un paramètre en fonction de nouvelles données. Il utilise le théorème de Bayes :

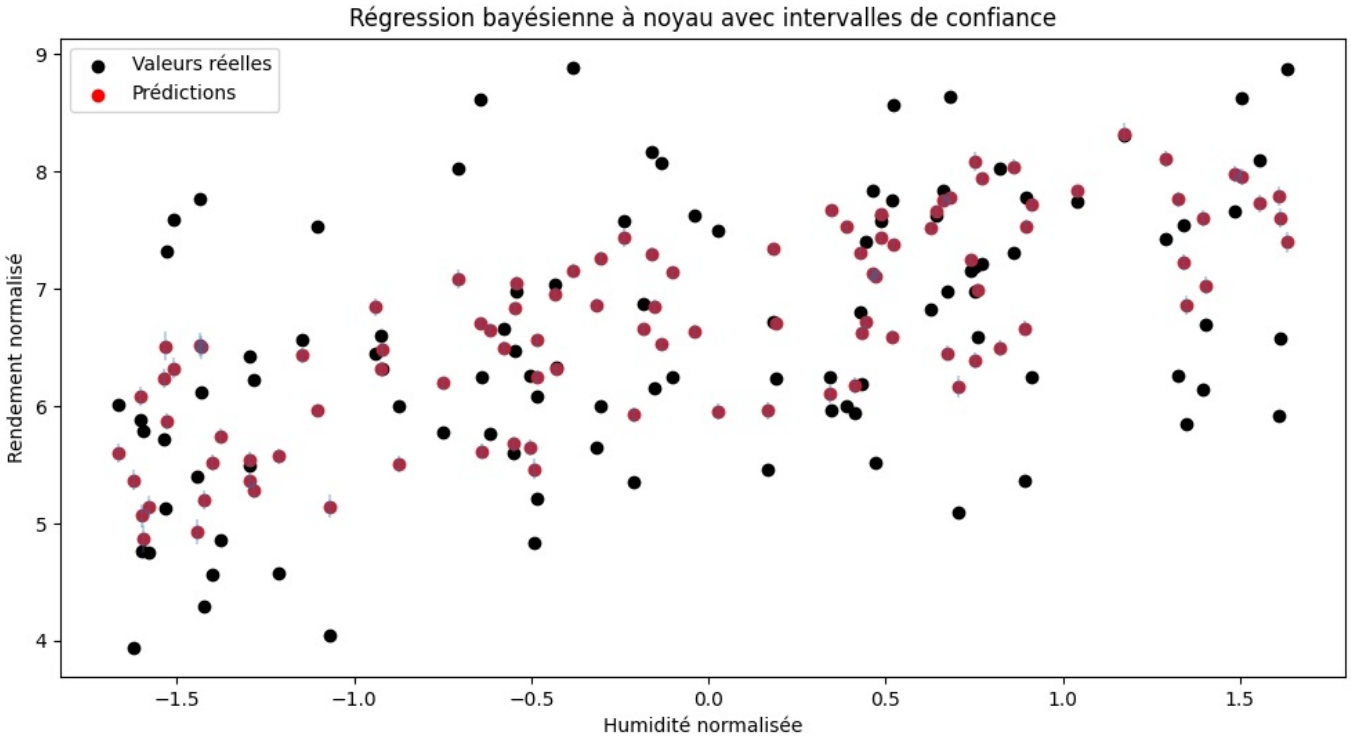
P(A|B) = P(B|A)*P(A)/P(B)

9. Les méthodes à noyau sont utilisées en apprentissage automatique, où les données sont projetées dans un espace à plusieurs dimensions pour les rendre séparables par des modèles linéaires. Les processus gaussiens utilisent une fonction de noyau pour définir la corrélation entre les points, ce qui permet d'estimer des distributions de probabilité sur des fonctions inconnues. Les noyaux permettent de modéliser des relations complexes entre les variables sans avoir à connaître la forme de la fonction.

10. La distribution à priori est la représentation de la distribution de probabilité des paramètres avant d'observer les données et la distribution à postériori est la représentation de la distribution de probabilité des paramètres après avoir observé les données. Dans notre cas, supposons que l'on souhaite prédire le rendement agricole en fonction du type de sol. A priori, on peut estimer que le rendement agricole est en général meilleur sur un sol sableux. Mais à priori, après avoir observé nos résultats ont peut utiliser le théorème de bayes pour obtenir la distribution à postériori.

Questions 11) à 15) :

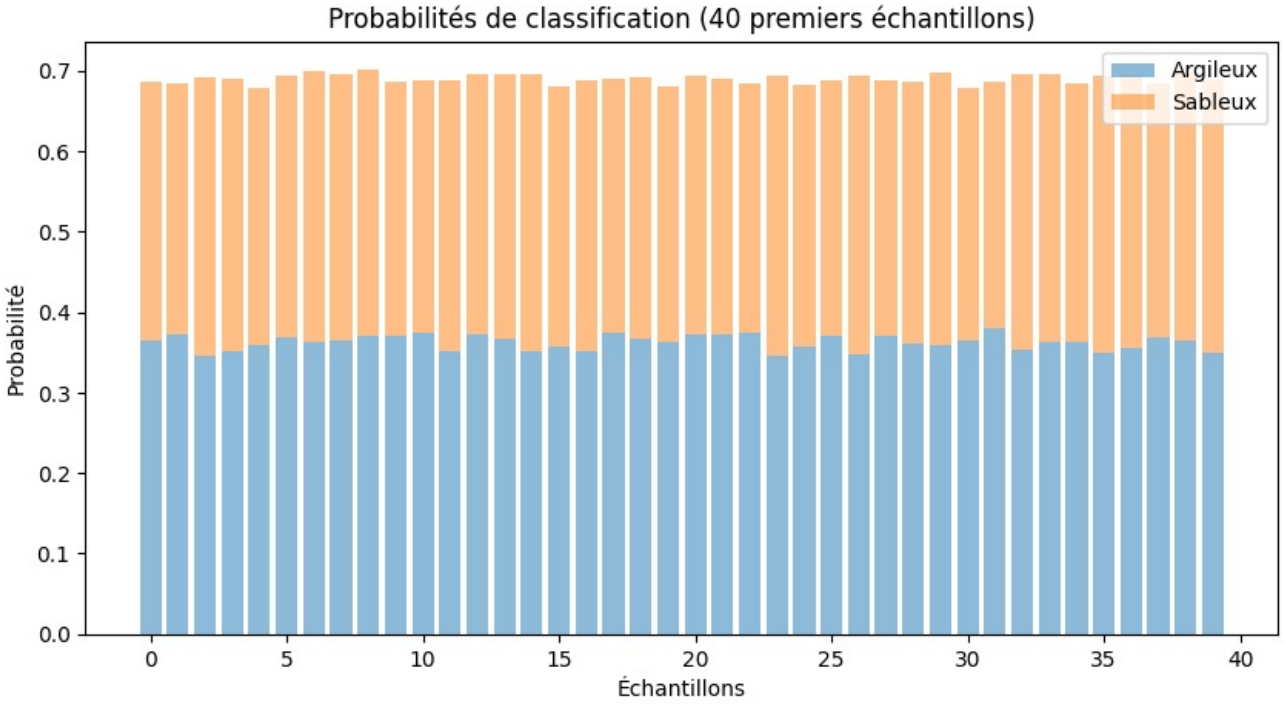
In [ ]:
```
import matplotlib.pyplot as plt
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

data = pd.read_csv('tp2_atdn_donnees.csv')
X = data[['Humidité (%)', 'Température (°C)']].values
y = data['Rendement agricole (t/ha)'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalisation
Normaliser = StandardScaler()
X_train = Normaliser.fit_transform(X_train)
X_test = Normaliser.transform(X_test)

# On choisit un noyau RBF
kernel = RBF(length_scale=1.0, length_scale_bounds=(1e-1, 10.0))

# On utilise le processus gaussien
gp = GaussianProcessRegressor(kernel=kernel, alpha=0.1, n_restarts_optimizer=10)
gp.fit(X_train, y_train)
y_prediction, sigma = gp.predict(X_test, return_std=True)

# Visualisation
plt.figure(figsize=(12, 6))
plt.scatter(X_test[:, 0], y_test, c='k', label='Valeurs réelles')
plt.scatter(X_test[:, 0], y_prediction, c='r', label='Prédictions')
plt.errorbar(X_test[:, 0], y_prediction, yerr=1.96*sigma, fmt='o', alpha=0.3)
plt.xlabel('Humidité normalisée')
plt.ylabel('Rendement normalisé')
plt.title('Régression bayésienne à noyau avec intervalles de confiance')
plt.legend()
plt.show()
```



In [ ]:

```python
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.metrics import accuracy_score, classification_report

data = pd.read_csv('tp2_atdn_donnees.csv')

X = data[['Humidité (%)', 'Température (°C)']].values
y = data['Type de sol'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalisation
Normaliser = StandardScaler()
X_train = Normaliser.fit_transform(X_train)
X_test = Normaliser.transform(X_test)

# On utilise un noyau RBF et on utilise le classifier processus gaussien
kernel = RBF(length_scale=1.0, length_scale_bounds=(1e-1, 10.0))
gpc = GaussianProcessClassifier(kernel=kernel, random_state=42)
gpc.fit(X_train, y_train)
y_pred_gpc = gpc.predict(X_test)
accuracy_gpc = accuracy_score(y_test, y_pred_gpc)

print(f"Accuracy de la classification bayésienne à noyau : {accuracy_gpc:.2f}")
print("Classification Report pour la classification bayésienne à noyau :")
print(classification_report(y_test, y_pred_gpc))

# SVM
svm = SVC(kernel='linear', probability=True)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"Accuracy du SVM classique : {accuracy_svm:.2f}")
print("Classification Report pour le SVM classique :")
print(classification_report(y_test, y_pred_svm))

# Visualisation des probabilités
probs = gpc.predict_proba(X_test)
plt.figure(figsize=(10, 5))
plt.bar(np.arange(len(probs[:40])), probs[:40, 0], alpha=0.5, label='Argileux')
plt.bar(np.arange(len(probs[:40])), probs[:40, 1], bottom=probs[:40, 0], alpha=0.5, label='Sableux')
plt.xlabel('Échantillons')
plt.ylabel('Probabilité')
plt.title('Probabilités de classification (40 premiers échantillons)')
plt.legend()
plt.show()
```

```
Accuracy de la classification bayésienne à noyau : 0.30
Classification Report pour la classification bayésienne à noyau :
              precision    recall  f1-score   support

    Argileux       0.31      0.97      0.47        31
    Limoneux       0.00      0.00      0.00        35
     Sableux       0.00      0.00      0.00        34

    accuracy                           0.30       100
   macro avg       0.10      0.32      0.16       100
weighted avg       0.09      0.30      0.14       100

Accuracy du SVM classique : 0.31
Classification Report pour le SVM classique :
              precision    recall  f1-score   support

    Argileux       0.31      1.00      0.47        31
    Limoneux       0.00      0.00      0.00        35
     Sableux       0.00      0.00      0.00        34

    accuracy                           0.31       100
   macro avg       0.10      0.33      0.16       100
weighted avg       0.10      0.31      0.15       100
```

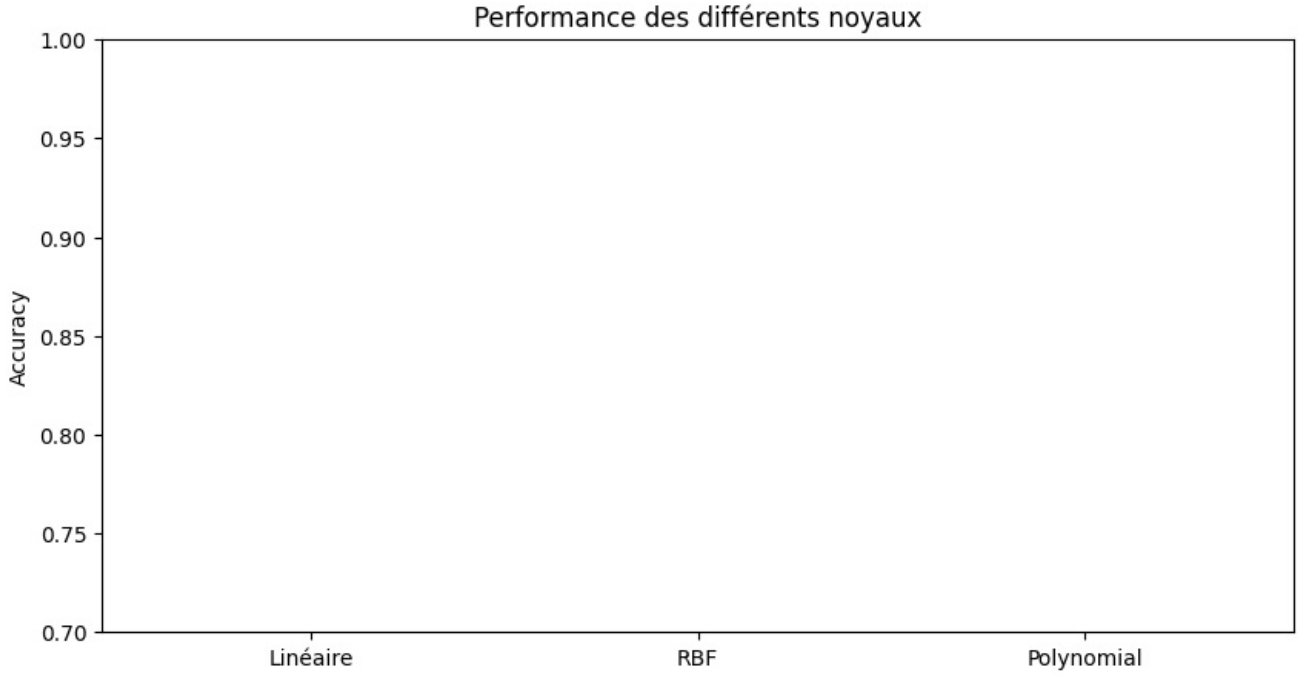


En comparant les différents types de sol, le sol argileux semble être le plus performant. De plus, en comparant, les modèles, on remarque que le SVM et le GP semble être similaire.

Les zones où le modèle est moins confiant sont celles où les probabilités prédites pour les différentes classes sont proches les unes des autres.

In [61]:
```python
from sklearn.gaussian_process.kernels import DotProduct, RBF

data = pd.read_csv('tp2_atdn_donnees.csv')

X = data[['Humidité (%)', 'Température (°C)']].values
y = data['Type de sol'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalisation
Normaliser = StandardScaler()
X_train = Normaliser.fit_transform(X_train)
X_test = Normaliser.transform(X_test)

# Liste des kernels
kernels = {
    'Linéaire': DotProduct(),
    'RBF': RBF(length_scale=1.0, length_scale_bounds=(1e-1, 10.0)),
    'Polynomial': DotProduct() ** 3
}

# Application des kernels avec le classifieur processus gaussien
results = {}
for name, kernel in kernels.items():
```

```
    gpc_test = GaussianProcessClassifier(kernel=kernel, random_state=42)
    gpc_test.fit(X_train, y_train)
    y_pred = gpc_test.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc

# Visualisation
plt.figure(figsize=(10, 5))
plt.bar(results.keys(), results.values())
plt.ylim(0.7, 1.0)
plt.ylabel('Accuracy')
plt.title('Performance des différents noyaux')
plt.show()

# Comparaison des noyaux
print("Comparaison des noyaux :")
for name, acc in results.items():
    print(f"{name: <10} : {acc:.3f}")
```




```
Comparaison des noyaux :
Linéaire   : 0.300
RBF        : 0.300
Polynomial : 0.310
```

J'ai essayé de reproduire le noyau polynomial mais sans réussite, par conséquent je ne peux que répondre partiellement à la question 14.

Différence entre les noyaux :

- Noyau linéaire : les données sont linéairements séparables
- Noyau polynomiale : les données sont séparables par une séparation polynomiale
- Noyau RBF : les données ne sont pas linéaires.

En fonction du type de noyaux, les résultats seront entièrement différents. En effet, choisir un noyau Linéaire suppose une corrélation entre les variables. Choisir un noyau polynomial signifie que les données peuvent être capturées car elles ont des corrélations polynomiales entre elles. Enfin, le choix d'un noyau RBF sera lorsque les variables ne sont pas ou peu corrélées et que la relation entre la variable cible et les autres variables ne sont pas linéaires.

Au niveau de l'influence de la distribution à priori, il combine les connaissances a priori avec les données observées pour faire des prédictions.

Conclusion

Nous avons pu constater à travers ce TP que l'approche bayésienne (optimisation bayésienne et classificateur bayésien) offrait des avantages plus intéréssants par rapport aux autres modèles malgré son temps d'exécution plus long que les autres. Il utilise la loi à priori afin d'optimiser ses prédictions tout en utilisant la loi de Bayes afin d'avoir un résulatat à postériori. Nous avons également vu que le choix des noyaux était très important car les résultats pouvaient être mauvais si l'on ne choisissait pas le bon noyau.