Projet CC2 Statistiques 2

Twens CENAT, Mohamed CHAIB, Benjamin GRES, Maxime HUDEBINE

Mai 2024

1 Analyse descriptive

1.1 Graphique des 100 premières observations

Soit la série $(X_t)_{1\leq t\leq 1000}$ à analyser, traçons le graphique des 100 premières observations de la série. pour cela on utilise le code Python suivant :

```
df_first_100 = df.head(100)
plt.figure(figsize=(10, 6))
plt.plot(df_first_100['Index'], df_first_100['Value'], marker='o', linestyle='-', color='b')
plt.title('Graphique des 100 premieres observations de la serie temporelle (Xt)')
plt.xlabel('Index')
plt.ylabel('Value')
plt.grid(True)
```

Listing 1: Représentation graphique des 100 premières observations

Le code compilé nous fournit le graphique suivant :

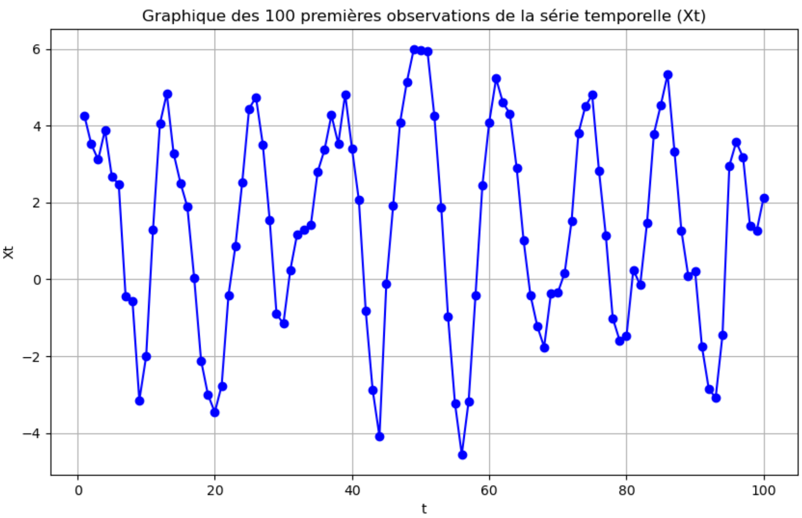

Figure 1: Graphique des 100 premières observations de la série temporelle (Xt)

Le graphique des 100 premières observations de la série temporelle montre des variations significatives au fil du temps. Il semble y avoir des périodes de hausse et de baisse, ce qui pourrait indiquer une tendance sous-jacente. La série ne semble pas osciller autour d'une moyenne et ne varie pas de façon constante, ce qui traduit une non-stationnarité. De plus, il y a présence de pics, cela peut indiquer la présence d'une saisonnalité, à confirmer par le calcul de la fonction d'auto-corrélation de la série.

1.2 Fonction d'auto-corrélation de la série X_t

Traçons la fonction d'auto-corrélation de $(X_t)_{1\leq t\leq 1000}$ en compilant les codes Python suivants :

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf

# Graphe d'autocorrelation des 1000 observations de $(X_t)$
plt.figure(figsize=(10, 6))
plot_acf(df['Value'], lags=40)
plt.title('Graphe d\'autocorrelation des 1000 observations de (X_t)')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.show()
```

Listing 2: Code Python pour générer le graphe d'autocorrélation

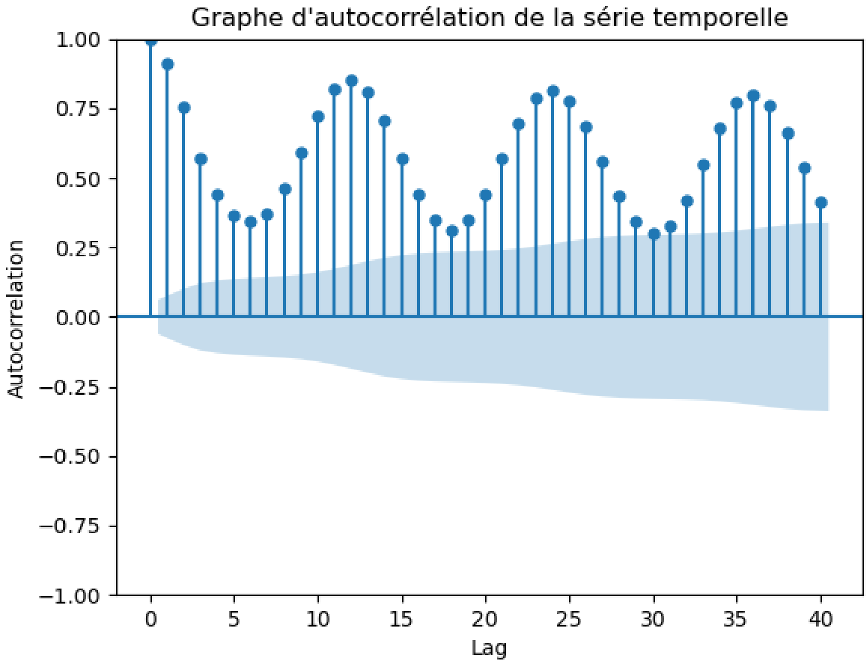

Figure 2: Graphe d'autocorrélation des 1000 observations de X_t

Le graphe d'autocorrélation montre les corrélations des valeurs de la série avec leurs décalages successifs. S'il y a une décroissance lente dans les valeurs de l'autocorrélation, cela indique une non-stationnarité de la série. Au contraire, si la courbe décroît rapidement vers zéro à mesure que le décalage augmente, alors il y a stationnarité de la série.

Observations :

Cela ne semble pas être le cas ici, la série n'est donc pas stationnaire Les autocorrélations sont au-dessus du seuil de signification (zone bleue), les observations sont alors corrélées sur de longues périodes.

Test de Dickey-Fuller

Lorsque la fonction d'autocovariance décroît lentement, on peut penser qu'il y a une racine unitaire.
Le test de Dickey-Fuller vise à tester l'hypothèse nulle $H_0 : \gamma = 0$ contre l'hypothèse alternative $H_1 : \gamma < 0$.

$$\begin{cases} H_0 : \ (X_t)_{1\leq t\leq 1000} \text{ a une racine unitaire} \\ H_1 : \ (X_t)_{1\leq t\leq 1000} \text{ n'a pas de racine unitaire} \end{cases}$$

Fixons un seuil d'erreur de 0.05, les résultats compilés en Python sont les suivants :

```python
# Test ADF ('Test de Dickey-Fuller augmente')
adf.test = adfuller(df['Value'])
```

```python

adf_output = pd.Series(result[0:4], index=['ADF Statistic', 'p-value', '#Lags Used', '
    Number of Observations Used'])
for key, value in result[4].items():
    adf_output[f'Critical Value ({key})'] = value

print(adf_output)
```

Listing 3: Test de Dickey-Fuller

```
# Resultats
ADF Statistic                    -1.017082
p-value                           0.746983
#Lags Used                       22.000000
Number of Observations Used     977.000000
 Critical Value (1%)             -3.437061
 Critical Value (5%)             -2.864503
 Critical Value (10%)            -2.568348
```

Listing 4: Résultats

Les résultats du test de Dickey-Fuller nous fournit une p-value de 0.746983 ce qui est bien supérieur au seuil de signification de 0.05. Par conséquent, nous ne pouvons pas rejeter l'hypothèse nulle. En d'autres termes, il y a des preuves insuffisantes pour conclure que la série temporelle est stationnaire.

On en conclut par observations que $(X_t)_{1\leq t\leq 1000}$ n'est pas stationnaire

2 Stationnarisation par estimation et modélisation.

2.1 calcul des estimations $\hat{\mathrm{m}}(\mathrm{t})$ et $\hat{\mathrm{s}}(\mathrm{t})$

2.1.1 Calcul de $\hat{\mathrm{m}}(\mathrm{t})$

On sait d'après l'énoncé que s'il y a une saisonnalité, celle-ci est déterministe en fonction de t : $s(t)$ et de période $d = 12$, et que si cette série à une tendance, celle-ci
déterministe en fonction de t : $m(t)$ et qu'elle est forcément linéaire: $m(t) = a + bt$.
La période est paire, $d = 2q$, alors d'après le cours, le calcul de $\hat{\mathrm{m}}(\mathrm{t})$ se fait par la formule suivante :

$$m_t = \frac{0.5x_{t-q} + x_{t-q+1} + \cdots + x_{t+q-1} + 0.5x_{t+q}}{d}, \quad q < t \leq n - q \tag{1}$$

En compilant ce calcul en Python, cela donne :

```python
# D finir la p riode, ici suppos e   12 mois
d = 12
q = d // 2 if d % 2 == 0 else (d - 1) // 2

# D finir une fonction pour la moyenne mobile centr e
def moyenne_mobile_centree(x, fenetre):
    return np.convolve(x, np.ones(fenetre), 'valid') / fenetre

# Calculer la tendance
if d % 2 == 0:  # p riode paire
    tendance = moyenne_mobile_centree(data['Xt'], d)
    tendance = np.concatenate(([np.nan]*q, tendance, [np.nan]*q))
else:  # p riode impaire
    tendance = data['Xt'].rolling(window=d, center=True).mean()

# Ajuster la longueur de la tendance
if len(tendance) < len(data):
    tendance = np.concatenate((tendance, [np.nan]*(len(data)-len(tendance))))
elif len(tendance) > len(data):
    tendance = tendance[:len(data)]

data['tendance'] = tendance
print(tendance)

# Calculer les d viations par rapport   la tendance
data['deviation'] = data['Xt'] - data['tendance']
```

Listing 5: calcul de $\hat{\mathrm{m}}(\mathrm{t})$

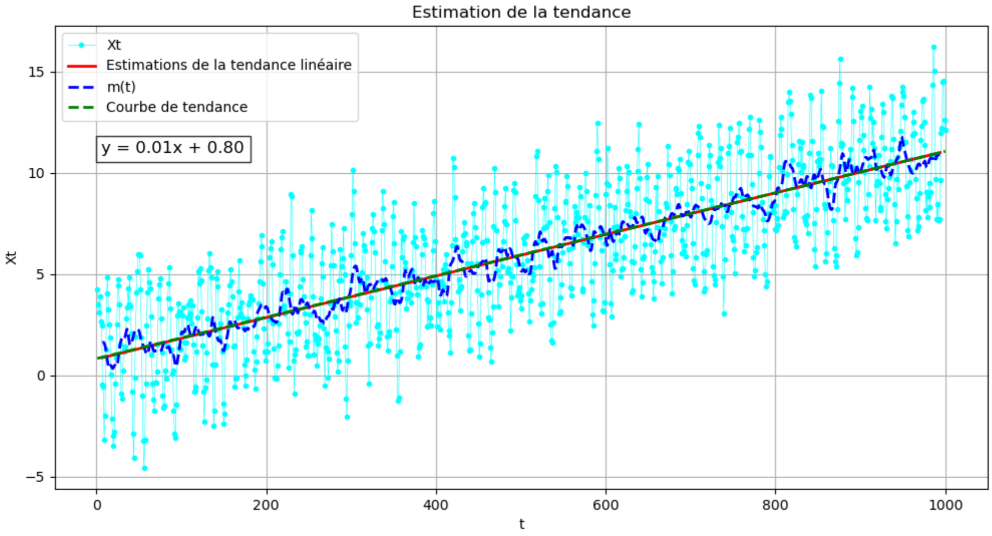

Figure 3: Estimation de la tendance $\hat{m}(t)$ de X_t

	t	Xt	m_t
0	7	-0.4353	1.6199
1	8	-0.5543	1.6341
2	9	-3.1529	1.5987
3	10	-1.999	1.49
4	11	1.303	1.2965
0	...	...	...
100	107	2.8671	2.5062
101	108	3.5064	2.4478
102	109	2.8257	2.2692
103	110	3.086	2.105
104	111	3.7122	1.98
105	112	2.5476	1.8989
0	...	...	...
600	607	2.9064	6.8363
601	608	5.2026	6.6588
602	609	7.3248	6.6056
603	610	8.3177	6.6955
604	611	5.7232	6.7998
605	612	5.985	6.9047
0	...	...	...
983	990	7.7159	10.9056
984	991	8.4571	10.9658
985	992	7.6912	10.9014
986	993	7.6988	10.7321
987	994	9.6415	10.6872

Figure 4: Estimation de la tendance $\hat{m}(t)$ de X_t

Les valeurs obtenues de $\hat{m}(t)$ ainsi que la courbe bleue sur le graphe dans la page suivante montre que la tendance présente bien une croissance tout-à-fait linéaire malgré les fluctuations.

2.1.2 Calcul de $\hat{s}(t)$

On estime à présent la composante saisonnière. D'après le cours, pour $k = 1, \ldots, d$, on calcule la moyenne w_k des déviations

$$\{(x_{k+jd} - \hat{m}_{k+jd}), \quad q < k + jd \leq n - q\} \tag{2}$$

Puisque les moyennes des déviations ne somment pas nécessairement à 0, on estime alors le composant s_k de la saison comme

$$\hat{s}_k = w_k - \frac{\sum_{i=1}^{d} w_i}{d}$$

et

$$\hat{s}_k = \hat{s}_{k-d}, \quad k > d$$

Les données désaisonnalisées sont alors définies comme

$$d_t := x_t - \hat{s}_t$$

Finalement, on réestime la tendance avec la méthode de la section précédente.
Cela donne compilé en python :

```
# Calculer les deviations par rapport  la tendance
data['deviation'] = data['Xt'] - data['tendance']

# Initialiser le tableau des composantes saisonnieres
composante_saisonniere = np.zeros(d)

# Calculer les deviations moyennes pour chaque periode
for k in range(d):
    valeurs_periode = data['deviation'][k::d]
    composante_saisonniere[k] = valeurs_periode.mean()

# Ajuster la composante saisonni re pour qu'elle somme  zero
composante_saisonniere -= composante_saisonniere.mean()

# Etendre la composante saisonniere pour correspondre  la longueur des donnees
saison_etendue = np.tile(composante_saisonniere, len(data) // d + 1)[:len(data)]

data['saisonnier'] = saison_etendue
```

Listing 6: calcul de $\hat{s}(t)$

On obtient les résultats suivants :

	t	s_t
0	1.0	-2.5314
1	2.0	-2.5512
2	3.0	-1.9489
3	4.0	-0.8622
4	5.0	0.581
99	100.0	-0.8622
100	101.0	0.581
101	102.0	1.791
102	103.0	2.7219
103	104.0	2.8525
104	105.0	2.1229
599	600.0	-2.0105
600	601.0	-2.5314
601	602.0	-2.5512
602	603.0	-1.9489
603	604.0	-0.8622
604	605.0	0.581
995	996.0	-2.0105
996	997.0	-2.5314
997	998.0	-2.5512
998	999.0	-1.9489
999	1000.0	-0.8622

Figure 5: Quelques valeurs de $\hat{s}(t)$

il est normal que les données désaisonnalisées fluctuent davantage car elles ne contiennent plus la composante saisonnière régulière, qui a été supprimée. Ce qui reste sont les variations irrégulières, la tendance et les éventuels bruits aléatoires.

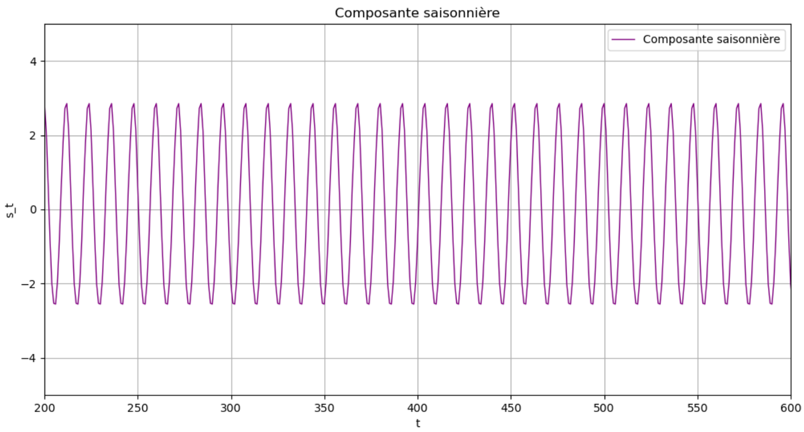

Figure 6: Estimation de la composante saisonnière ŝ(t)

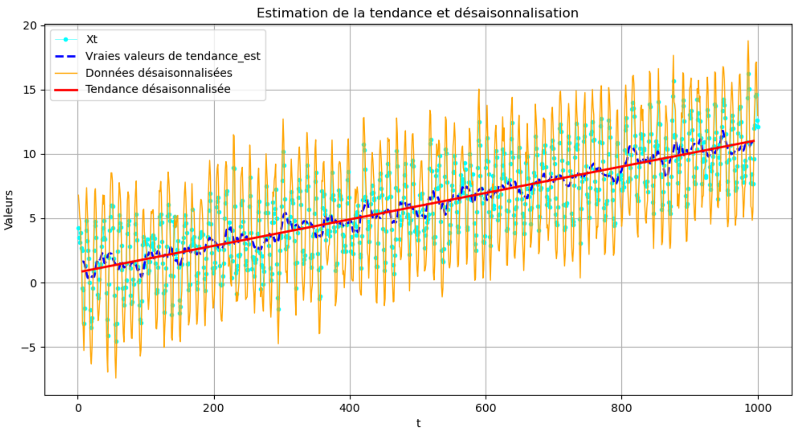

Figure 7: Estimation des données désaisonnalisées

2.2 Graphique des 100 premières valeurs de $(Y_t)_{1\leq t\leq 1000} = (X_t - \hat{m}_t - \hat{s}_t)_{1\leq t\leq 1000}$

Il suffit de retrancher à $(X_t)_{1\leq t\leq 1000}$ les valeurs obtenues de $\hat{m}_t$ et de $\hat{s}_t$
en code Python, cela donne :

```python
# Calculer les donnees desaisonnalisees
data['desaisonnalise'] = data['Xt'] - data['saisonnier']

# Calculer Yt
data['Yt'] = data['desaisonnalise'] - data['tendance']

# Tracer les 100 premieres valeurs de Yt
plt.figure(figsize=(12, 6))
plt.plot(df['t'][:100], df['Y_t'][:100], label='Y_t', color='blue', marker='.', 
    linewidth=0.4)
plt.xlabel('t')
plt.ylabel('Y_t')
plt.title('100 premieres valeurs de Yt = Xt - mt - st')
plt.legend()
plt.grid(True)
```

Listing 7: calcul de $(Y_t)_{1\leq t\leq 1000}$

Soit le graphique suivant :

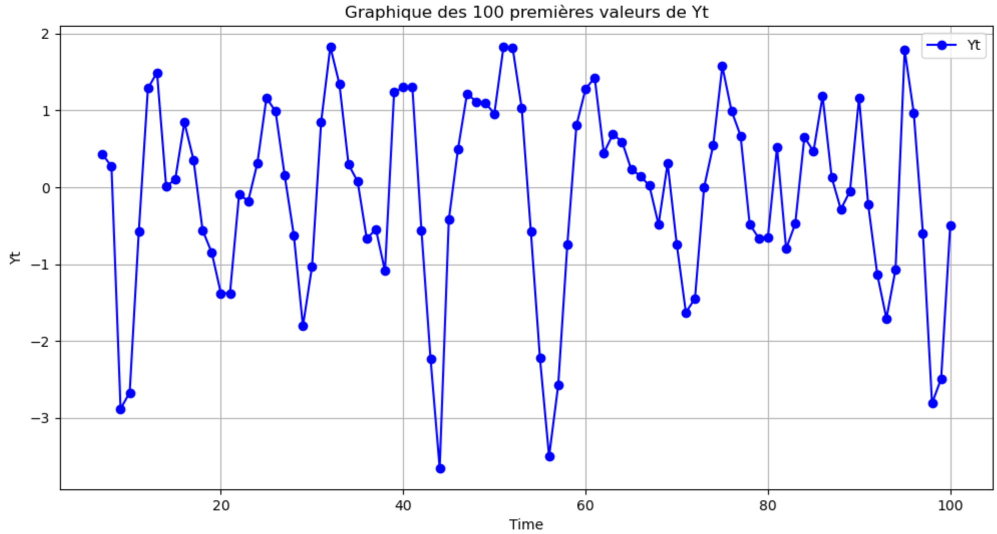

Figure 8: Graphique des 100 premières valeurs de Y_t

La série $(Y_t)_{1\leq t\leq 1000}$ présente des fluctuations significatives autour de la ligne de base (0). Cela indique que la série a une variabilité notable sans une tendance claire à la hausse ou à la baisse sur cette période. De plus, la série semble plus aléatoire, suggérant que les composantes saisonnières ont été bien retirées.

2.3 Fonction d'autocorrélation de $(Y_t)_{1\leq t\leq 1000}$

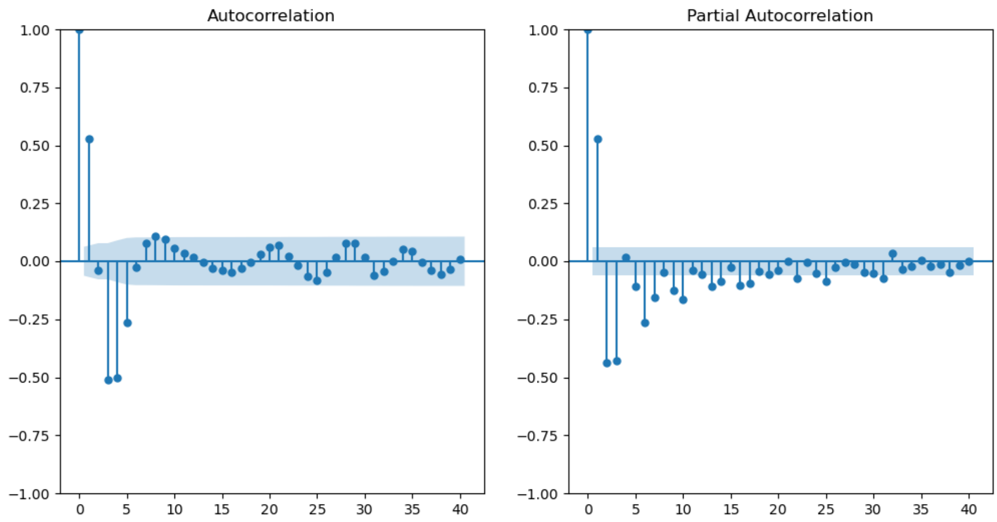

Figure 9: 100 premières valeurs de $(Y_t)_{1\leq t\leq 1000}$

Conjecture (analyse des graphes de l'ACF et du PACF) :

Les graphiques indiquent tous les deux une décroissance rapide des fonctions d'autocorrélation et d'autocorrélation partielle, ce qui suggère que notre série $(Y_t)_{1\leq t\leq 1000}$ a bien été stationnarisée. On peut également émettre une hypothèse concernant l'ordre de l'ARMA à estimer par la fonction d'autocovariance qui approche de zéro à partir de $h = 2$ malgré les fluctuations sur les 3 pas suivants, et le graphe d'autocorrélation partielle approche de zéro à partir du pas $h = 4$, cela donne un avant-goût de l'ordre du modèle ARMA qui nous donneraît un ARMA(2,4). Néanmoins, il faut passer par des critères de sélection pour trouver l'ordre optimal.

2.4 Estimation d'un modèle ARMA(p,q) du processus $(Y_t)_{1\leq t\leq 1000}$

Les graphiques des fonctions d'autocorrélation et d'autocorrélation partielle montre des valeurs significatives jusqu'au lag 3. Cela suggère que l'ordre p et q pourraient être autour de 3

On choisit l'ordre du modèle grâce au principe de parcimonie (c'est un compromis entre la valeur de la vraisemblance et du nombre de paramètres utilisés pour le modèle).
La sélection de cet ordre peut être automatisée grâce à un critère d'information comme BIC (Bayesian Information Criteria) ou AIC (Akaike Information Criteria)

On représente ce processus sous forme d'un algorithme itératif, qui va sélectionner le meilleur ordre en utilisant les critères AIC et BIC, ce qui nous donne :

```
aic_values = []
bic_values = []
orders = []

for p in range(5):
    for q in range(5):
        try:
            model = ARIMA(data['Yt'].dropna(), order=(p, 0, q))
            model_fit = model.fit()
            aic_values.append(model_fit.aic)
            bic_values.append(model_fit.bic)
            orders.append((p, 0, q))
        except Exception as e:
            print(f"Erreur pour ARIMA({p},{q}): {e}")
            continue

# Trouver les ordres avec les AIC et BIC les plus bas
best_aic_order = orders[np.argmin(aic_values)]
best_bic_order = orders[np.argmin(bic_values)]

print(f"Meilleur ordre selon AIC: {best_aic_order}")
print(f"Meilleur ordre selon BIC: {best_bic_order}")

# Estimer le modele ARMA avec les meilleurs ordres trouves
model_arma = ARIMA(data['Yt'].dropna(), order=best_bic_order)
model_arma_fit = model_arma.fit()
residus_arma = model_arma_fit.resid
```

Listing 8: Estimation d'un modele ARMA par les critères AIC et BIC

Le critère AIC sélectionne un modèle ARMA(4,3), tandis que le critère BIC sélectionne un modèle ARMA(2,3), Conservons le modèle sélectionné par BIC, et utilisons le pour estimer les prédictions sur les 100 prochaines valeurs de $(X_t)_{1001\leq t\leq 1100}$

```

# Estimer le modele ARMA avec les meilleurs ordres trouves
model_arma = ARIMA(data['Yt'].dropna(), order=best_bic_order)
model_arma_fit = model_arma.fit()

# Prevoir les 100 prochaines valeurs
forecast_residual = model_arma_fit.forecast(steps=100)

# Combiner les residus avec la tendance et la saisonnalite
prevision_tendance = np.polyval(np.polyfit(data['tendance'].dropna().index, data['
    tendance'].dropna(), 1), range(len(data), len(data) + 100))
cycle_saisonnier = np.tile(composante_saisonniere, 9)[:100]

prevision = prevision_tendance + cycle_saisonnier + forecast_residual

# Tracer la prevision
plt.figure(figsize=(14, 7))
plt.plot(data['t'], data['Xt'], label='Serie Originale')
```

```
plt.plot(range(len(data), len(data) + 100), prevision, label='Prevision', color='red')
plt.legend()
plt.title('Prevision des 100 valeurs suivantes de Xt')
plt.show()

# Creer un DataFrame pour afficher les valeurs prevues
valeurs_prevision = pd.DataFrame(prevision, columns=['Prevision'], index=range(len(data)
    + 1, len(data) + 101))

# Afficher les valeurs prevues
print(valeurs_prevision)
```

Listing 9: Prévision des 100 prochaines observations $(X_t)_{1001 \leq t \leq 1100}$

Le graphique suivant caractérise les prévisions :

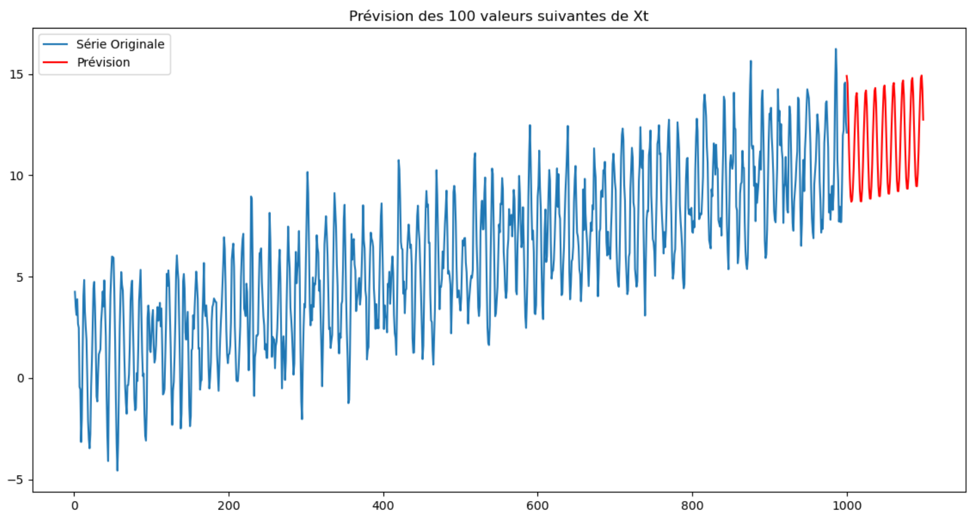

Figure 10: Prédiction des 100 nouvelles observations $(Y_t)_{1 \leq t \leq 1000}$

Les prévisions suivent de façon linéaire la tendance déjà estimée, ce qui est bon signe au-niveau de la significativité des valeurs prédites.

3 Modélisation d'un SARIMA

Soit le processus $(Z_t)_{1 \leq t \leq 1000-d} := (\nabla_d X_t)_{d \leq t \leq 1000} = ((I - B^d)X_t)_{d \leq t \leq 100}$

3.1 Graphique des 100 premières valeurs de $(Z_t)_{1 \leq t \leq 1000-d}$

Différenciation de la série : La différenciation est une technique utilisée pour rendre une série temporelle stationnaire en supprimant les tendances et les saisonnalités. La différenciation avec une période d consiste à calculer la différence entre chaque valeur et la valeur d'il y a d périodes. Mathématiquement, cela s'écrit :

$$Z_t = X_t - X_{t-d} \tag{3}$$

Dans notre cas, $d = 12$, donc nous calculons : $Z_t = X_t - X_{t-12}$
En compilant notre calcul en Python, on obtient :

```
d = 12
Z_t = series_values.diff(periods=d).dropna()

# Tracer les 100 1eres valeurs de Zt
plt.figure(figsize=(12, 6))
```

```
plt.plot(Z_t[:100], marker='o')
plt.title('100 premieres observations de Z_t (d=12)')
plt.xlabel('Time')
plt.ylabel('Z_t')
plt.grid(True)
plt.show()
```

Listing 10: Calcul des 100 premières observations de la série Z_t

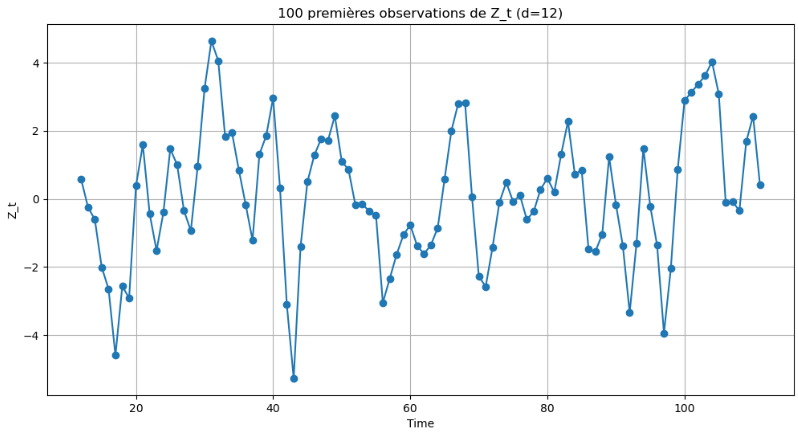

Figure 11: Graphique des 100 premières valeurs de Z_t

Implémentation de la fonction d'autocorrélation de Z_t:

```
    #Tracer la fonctions d'autocorrelation de Zt
plt.figure(figsize=(12, 6))
plot_acf(Z_t, lags=40)
plt.title('Autocorrelation de Z_t (d=12)')
plt.xlabel('Lags')
plt.ylabel('Autocorrelation')
plt.grid(True)
plt.show()
```

Listing 11: Fonction d'autocorrélation de $(Z_t)_{1\leq t\leq 1000-d}$

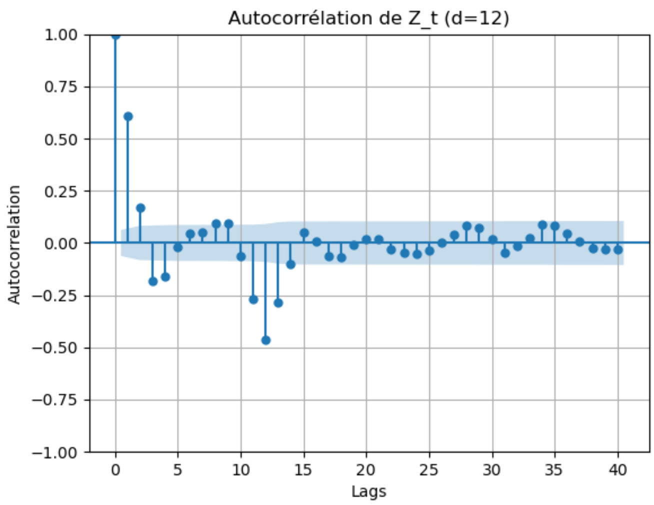

Figure 12: Fonction d'autocorrélation de Z_t

En observant le graphique d'autocorrélation, nous pouvons identifier si des patterns saisonniers ou des dépendances temporelles persistent dans la série différenciée.

Dans ce cas-ci, malgré une plus forte fluctuation aux pas $h \in \{11, 12, 13\}$, la fonction se stabilise rapidement autour de 0, en en déduit donc que Z_t est considéré comme stationnaire. Utilisons le test de Dickey-Fuller pour le montrer:

Test ADF : La statistique ADF et la p-value associée nous aident à déterminer si la série différenciée $(Z_t)_{1\leq t\leq 1000-d}$ est stationnaire.

Si la p-value est inférieure au seuil 0.05, nous rejetons l'hypothèse nulle et concluons que la série est stationnaire. Si la p-value est supérieure au seuil, nous ne pouvons pas rejeter l'hypothèse nulle, ce qui suggère que la série n'est pas stationnaire. En résumé, ces étapes nous permettent d'analyser la stationnarité de la série différenciée et de mieux comprendre les propriétés temporelles de $(Z_t)_{1\leq t\leq 1000-d}$.

```
result = adfuller(Z_t)
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')

ADF Statistic: -9.935511645704251
p-value: 2.7504682474762874e-17
```

Listing 12: Test ADF pour la stationnarité de $(Z_t)_{1\leq t\leq 1000-d}$

Au vu du test qui affiche une p-value nettement inférieure à 0.05, nous voyons donc que la serie Z_t est stationnaire

3.2 Estimation d'un modèle SARIMA pour $(X_t)_{1\leq t\leq 1000}$

Cette algorithme a pour but de tester tout les ordres entre 0 et 5 afin de voir quelle ordre est le plus optimisé pour le modele SARIMA , c'est un algorithme iteratif qui teste toute les combinaisons possibles d'ordre. Paramètres p,d,q : Les paramètres (p, d, q) représentent les ordres des composants autoregressif, différenciation et moyenne mobile du modèle SARIMA non saisonnier.

Paramètres (P, D, Q) : Les paramètres (P, D, Q) représentent les ordres des composants saisonniers autoregressif, différenciation et moyenne mobile du modèle SARIMA saisonnier.

Critère d'Information d'Akaike (AIC) : La valeur d'AIC est utilisée pour comparer les modèles. Un modèle avec une valeur d'AIC plus faible est préféré car il indique un meilleur équilibre entre la qualité de l'ajustement du modèle et sa complexité

```
    # Define the p, d, q ranges for non-seasonal and seasonal components
p = d = q = range(0, 5)
P = D = Q = range(0, 5)
s = 12

# Generate all different combinations of p, d, q triplets
pdq = list(itertools.product(p, [0], q))
seasonal_pdq = list(itertools.product(P, [1], Q, [s]))

# Function to perform grid search for SARIMA parameters
def sarima_grid_search(data, pdq, seasonal_pdq):
    best_aic = float("inf")
    best_pdq = None
    best_seasonal_pdq = None
    best_model = None

    for param in pdq:
        for param_seasonal in seasonal_pdq:
            try:
                model = sm.tsa.statespace.SARIMAX(data,
                                                  order=param,
                                                  seasonal_order=param_seasonal,
                                                  enforce_stationarity=False,
                                                  enforce_invertibility=False)
                results = model.fit()

                if results.aic < best_aic:
                    best_aic = results.aic
                    best_pdq = param
                    best_seasonal_pdq = param_seasonal
                    best_model = results
            except:
                continue
```

```python

    return best_model, best_pdq, best_seasonal_pdq, best_aic

# Perform grid search
best_model, best_pdq, best_seasonal_pdq, best_aic = sarima_grid_search(data['Value'],
    pdq, seasonal_pdq)

best_pdq, best_seasonal_pdq, best_aic
```

Listing 13: Estimation des paramètres du SARIMA

On voit donc que avec l'algorithme iteratif les meilleurs ordres sur SARIMA sont : 5 1 5 1 1 1 12, avec un AIC de 2738

```python
# La serie a une saisonalite de 12
# Algorithme iteratif pour trouver les meilleurs odres
p = 5
d = 1
q = 5
P = 1
D = 1
Q = 1
m = 12


sarima_model = sm.tsa.SARIMAX(series_values, order=(p, d, q), seasonal_order=(P, D, Q, m
    ), enforce_stationarity=False, enforce_invertibility=False)
sarima_results = sarima_model.fit()


print(sarima_results.summary())
```

3.3 Estimation des performances de ce modèle sur les 100 prochaines observations $(X_t)_{1001\leq t\leq 1100}$

```python
#Prevision des 100 prochaines valeurs
forecast_steps = 100
forecast = sarima_results.get_forecast(steps=forecast_steps)
forecast_values = forecast.predicted_mean
```

4 Conclusion

Il paraitraît évident que le SARIMA soit jugé 'meilleur' dans le sens ou les données d'origine ne sont pas stationnaire et le modèle présente une saisonnalité, donc SARIMA seraît généralement plus approprié car il prend en compte les composantes saisonnières. De plus de par ses composantes saisonnières, SARIMA est plus flexible et peut capturer une gamme plus large de structures temporelles. On pourraît néanmoins des métriques telles que la somme des carrés des résidus pour évaluer les performances des modèles estimés.

```python
residus_sarima = sarima_results.resid

# Calculer la somme des carres des residus pour les deux modeles
ssr_arma = np.sum(residus_arma**2)
ssr_sarima = np.sum(residus_sarima**2)
```

Listing 14: Calcul des MSE

Les résultats sont 603.5767077717986 pour l'ARMA et 1032.3036339978794 pour le SARIMA, donc les performances sont à priori plus élevées pour le modèle estimé par un ARMA(2,3) que pour un SARIMA(5,1,5)$\times$(1,1,1)$_{12}$. Donc le modèle estimé par un ARMA(2,3) serait plus adapté pour capturer les tendances et les structures temporelles de notre série. De plus, il a tendance à éviter le sur-ajustement (*Overfitting*) contrairement à un SARIMA qui peut capturer le bruit ou les composantes saisonnières.

On en conclut que le modèle estimé par un ARMA nous semble être le meilleur .