Time series project

May 7, 2024

[4]:
```
import pandas as pd
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.vector_ar.var_model import VAR
from statsmodels.tsa.stattools import adfuller
from statsmodels.tools.eval_measures import rmse, aic
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

[6]:
```
import warnings
warnings.filterwarnings('ignore')
```

[7]:
```
data=pd.read_excel('datauk.xlsx')
```

[8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63 entries, 0 to 62
Data columns (total 5 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Année                         63 non-null     datetime64[ns]
 1   Dépenses (% du PIB)           51 non-null     float64
 2   Masse monétaire (% du PIB)    63 non-null     float64
 3   Croissance du PIB (% annuel)  62 non-null     float64
 4   Inflation, IPC (% annuel)     63 non-null     float64
dtypes: datetime64[ns](1), float64(4)
memory usage: 2.6 KB
```

[9]:
```
df=data.dropna()
```

[10]:
```
df.rename(columns = {'Dépenses (% du PIB)':'depenses','Masse monétaire (% du␣
↪PIB)':'Mass_Mon','Croissance du PIB (% annuel)':'PIB','Inflation, IPC␣
↪(% annuel)':'IPC'}, inplace = True)
```

[11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 51 entries, 12 to 62
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Année     51 non-null     datetime64[ns]
 1   depenses  51 non-null     float64
 2   Mass_Mon  51 non-null     float64
 3   PIB       51 non-null     float64
 4   IPC       51 non-null     float64
dtypes: datetime64[ns](1), float64(4)
memory usage: 2.4 KB
```

[12]:
```
df.describe().T
```

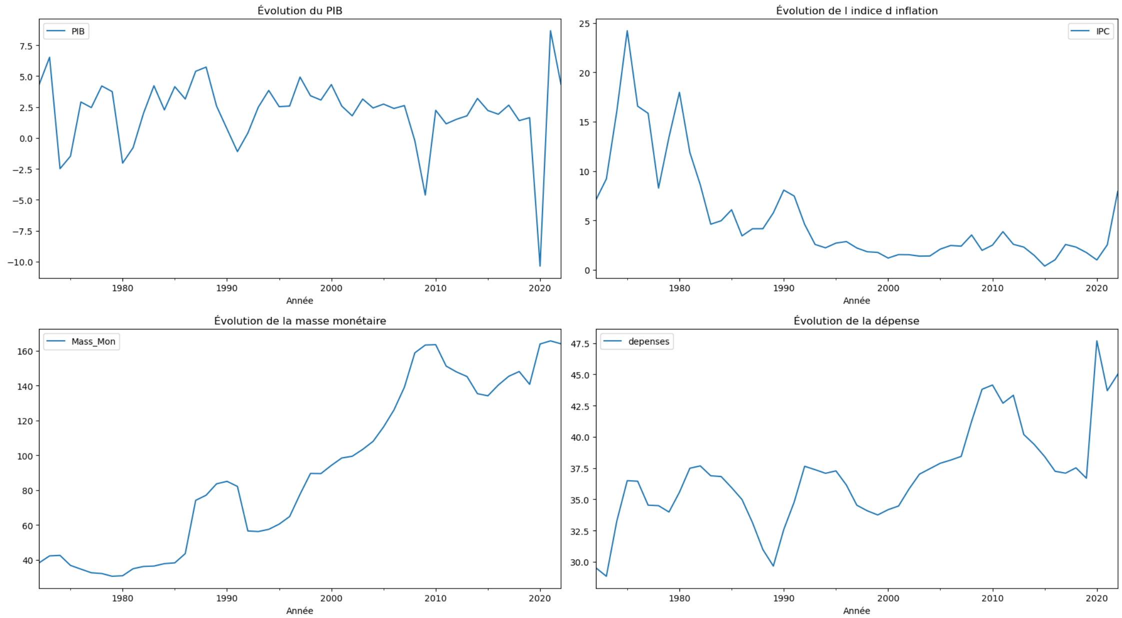
[12]:
```
          count       mean        std        min        25%        50%  \
depenses   51.0  36.927618   3.928926  28.838079  34.507635  36.881620
Mass_Mon   51.0  91.185677  47.241943  30.453995  42.307287  84.960198
PIB        51.0   2.143492   2.897470 -10.359901   1.577274   2.531670
IPC        51.0   5.291625   5.227952   0.368047   2.025434   2.697495

                 75%         max
depenses   38.010740   47.677124
Mass_Mon  139.654479  165.625401
PIB         3.581643    8.674904
IPC         7.266441   24.207288
```

1 Visualiser les données

[13]:
```
df.columns
```

[13]:
```
Index(['Année', 'depenses', 'Mass_Mon', 'PIB', 'IPC'], dtype='object')
```

[14]:
```
fig, axs = plt.subplots(2, 2, figsize=(18, 10))

df.plot(x='Année', y='PIB', ax=axs[0, 0])
df.plot(x='Année', y='IPC', ax=axs[0, 1])
df.plot(x='Année', y='Mass_Mon', ax=axs[1, 0])
df.plot(x='Année', y='depenses', ax=axs[1, 1])

axs[0,0].set_title('Évolution du PIB')
axs[0,1].set_title('Évolution de l indice d inflation')
axs[1,0].set_title('Évolution de la masse monétaire')
axs[1,1].set_title('Évolution de la dépense')

plt.tight_layout()
plt.show()
```



[15]:
```python
df.set_index('Année', inplace=True)
```

2 Stationnarité

[16]:
```python
# Faire les tests de Dickey-Fuller augmenté
for column in df.columns:
    result = adfuller(df[column])
    print(f"\nResults for column {column}:")
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    print('Critical Values:')
    for key, value in result[4].items():
        print(f'\t{key}: {value:.3f}')
```

```
Results for column depenses:
ADF Statistic: -0.01788946499528467
p-value: 0.9570273974381989
Critical Values:
        1%: -3.601
        5%: -2.935
        10%: -2.606

Results for column Mass_Mon:
ADF Statistic: -0.3034413347165586
p-value: 0.9250155201169266
Critical Values:
```

```
        1%: -3.585
        5%: -2.928
        10%: -2.602

Results for column PIB:
ADF Statistic: -6.8423903498917245
p-value: 1.7775876148768182e-09
Critical Values:
        1%: -3.568
        5%: -2.921
        10%: -2.599

Results for column IPC:
ADF Statistic: -2.0482518404718397
p-value: 0.26582499942353527
Critical Values:
        1%: -3.589
        5%: -2.930
        10%: -2.603
```

Travaillons pour une seuil de confiance de 5%:

Sur la base des résultats fournis pour le test ADF (Augmented Dickey-Fuller) pour la colonne PIB :

- Statistique ADF : -6.8423903498917245
- valeur p : 1.7775876148768182e-09 (environ 0)

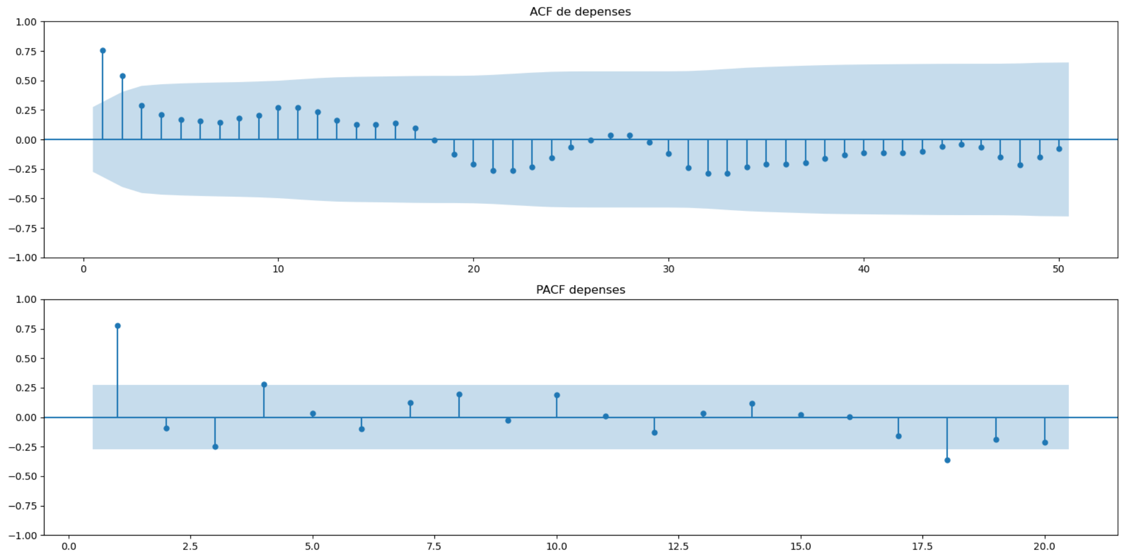
Les valeurs critique: - Valeur critique à 5 % : -2.921

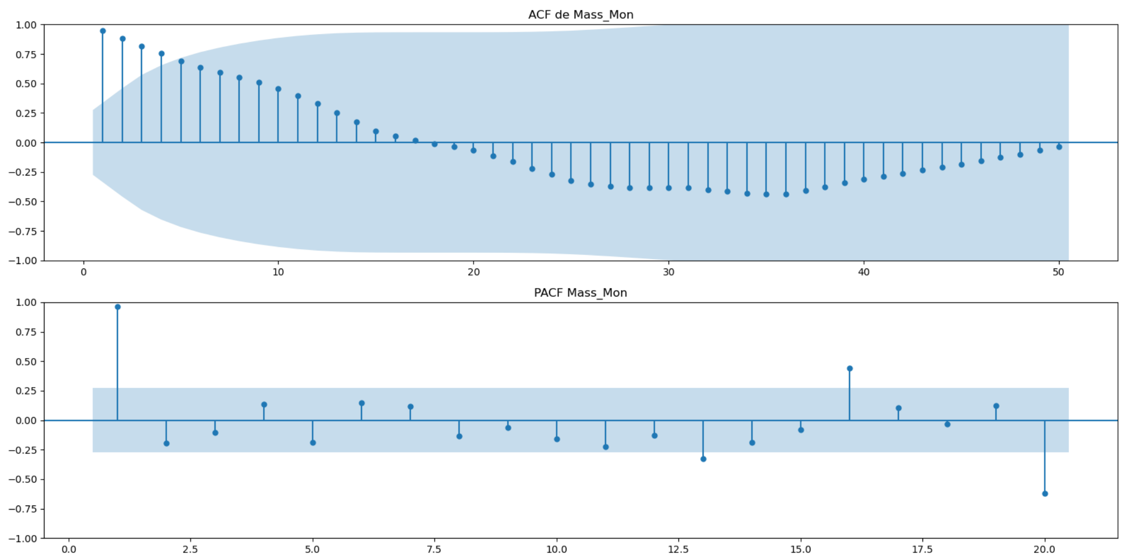
Interprétation :

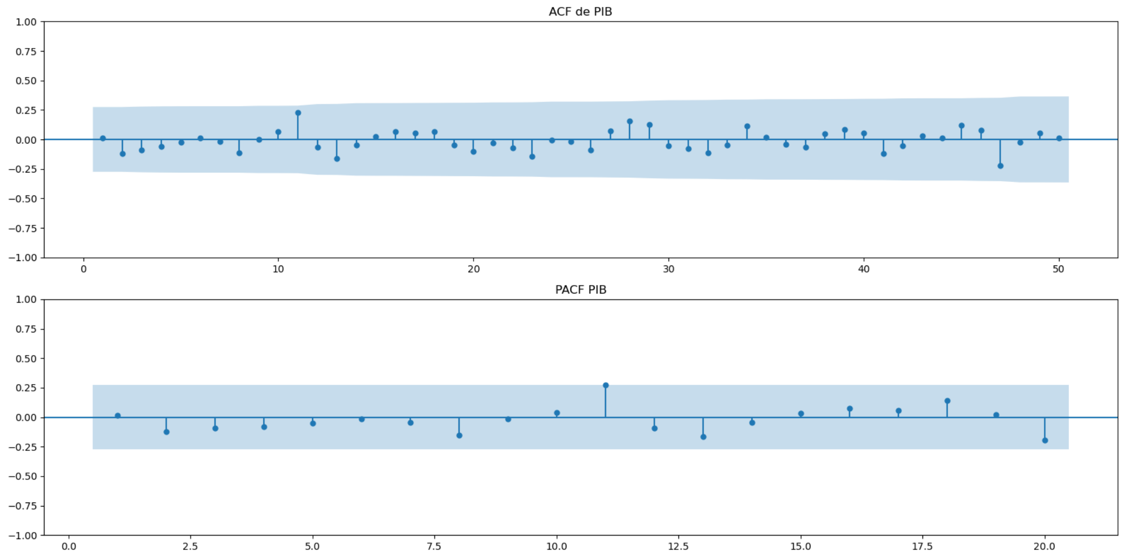
1. Statistique ADF : La statistique ADF est -6.8423903498917245. Cette valeur est fortement négative.
2. Valeur p : La valeur p est approximativement 0, spécifiquement 1.7775876148768182e-09, ce qui est très proche de zéro. Nous avons suffisamment de preuves pour conclure que la série du PIB est stationnaire.

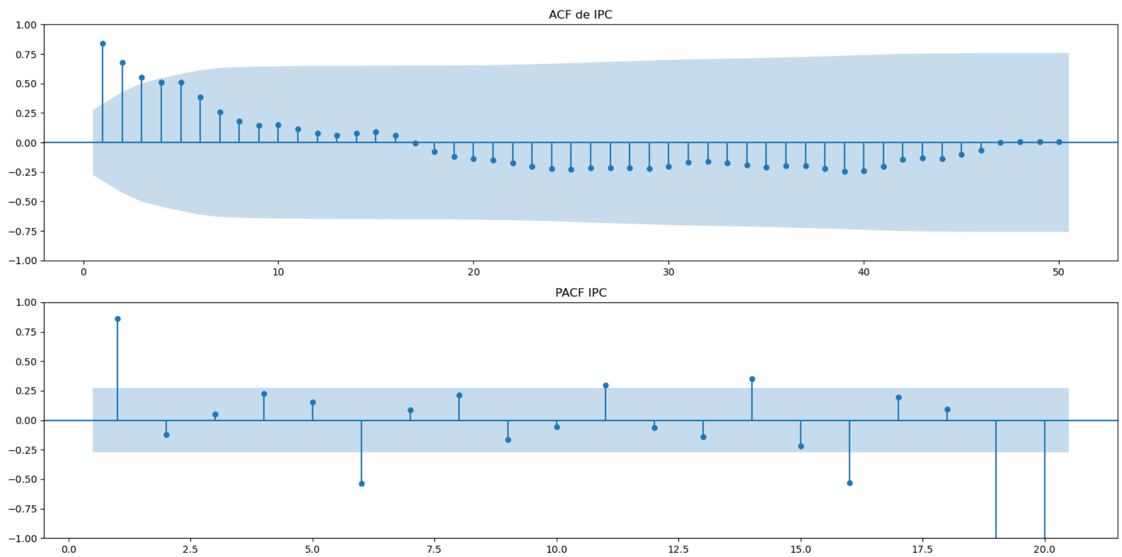
Idem: on peut conclure que le reste des séries (depenses, Mass_Mon, IPC) sont non stationnaire.

[99]:
```python
# Faire les graphiques ACF et PACF pour chaque colonne
for column in df.columns:
    fig, (ax1, ax2) = plt.subplots(2,1, figsize=(16,8))
    plot_acf(df[column], lags=50, zero=False, ax=ax1)
    ax1.set_title(f"ACF de {column}")
    plot_pacf(df[column], lags=20, zero=False, ax=ax2)
    ax2.set_title(f"PACF {column}")
    plt.tight_layout()
    plt.show()
```


ACF de depenses
PACF depenses


ACF de Mass_Mon
PACF Mass_Mon





3 Transformation pour rendre la série stationnaire

[100]:
```
dff=df[['depenses','Mass_Mon','IPC']]
```

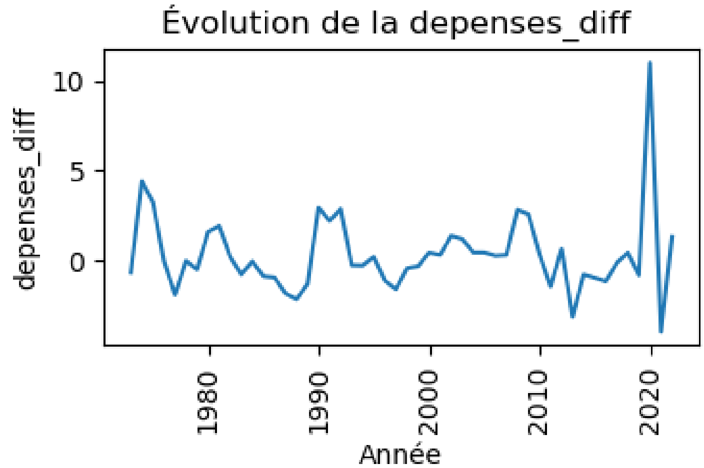
[101]:
```
# Différencier les séries de données et les ajouter au DataFrame
df_diff = dff.diff().dropna()
df_diff.columns = [f"{col}_diff" for col in dff.columns]
df = pd.concat([df, df_diff], axis=1)
df.head()
```

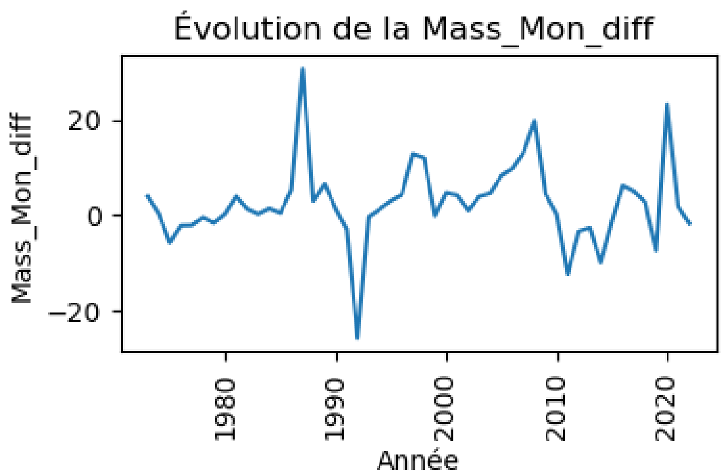
[101]:
```
            depenses   Mass_Mon       PIB        IPC  depenses_diff  \
Année
1972-01-01  29.511858  38.185852  4.321668   7.071098            NaN
1973-01-01  28.838079  42.169886  6.523848   9.196033      -0.673779
1974-01-01  33.253575  42.444689 -2.484404  16.044011       4.415496
1975-01-01  36.489009  36.697659 -1.473649  24.207288       3.235434
1976-01-01  36.444959  34.562379  2.910266  16.559523      -0.044050

            Mass_Mon_diff  IPC_diff
Année
1972-01-01            NaN       NaN
1973-01-01       3.984033  2.124935
1974-01-01       0.274803  6.847978
1975-01-01      -5.747030  8.163276
1976-01-01      -2.135280 -7.647765
```

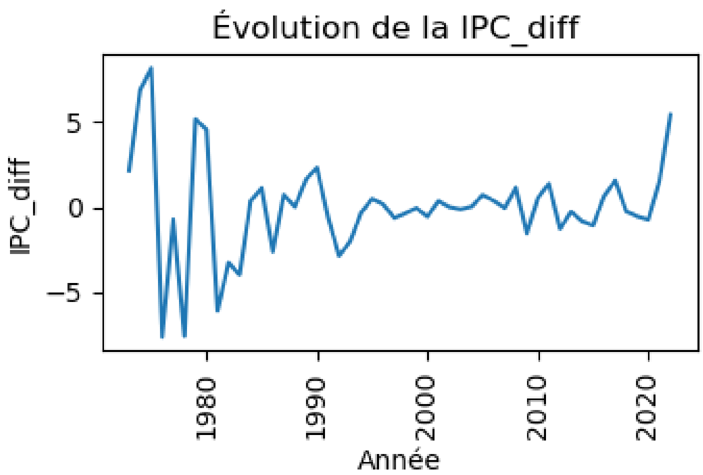
[102]:
```
df=df.dropna()
df.head()
```

[102]:
```
            depenses   Mass_Mon       PIB        IPC  depenses_diff  \
Année
1973-01-01  28.838079  42.169886  6.523848   9.196033      -0.673779
1974-01-01  33.253575  42.444689 -2.484404  16.044011       4.415496
1975-01-01  36.489009  36.697659 -1.473649  24.207288       3.235434
1976-01-01  36.444959  34.562379  2.910266  16.559523      -0.044050
1977-01-01  34.528167  32.479976  2.457751  15.840267      -1.916793

            Mass_Mon_diff  IPC_diff
Année
1973-01-01       3.984033  2.124935
1974-01-01       0.274803  6.847978
1975-01-01      -5.747030  8.163276
1976-01-01      -2.135280 -7.647765
1977-01-01      -2.082404 -0.719256
```

[103]:
```
for i in df_diff.columns:
    plt.figure(figsize=(4,2))
    sns.lineplot(y=df[i],x = df.index,linewidth = 1.5)
    plt.xlabel ('Année')
    plt.ylabel (i)
    plt.title('Évolution de la {} '.format(i))
    plt.xticks(rotation=90)
    plt.show()
```


Évolution de la depenses_diff
depenses_diff
10
5
0
1980
1990
2000
2010
2020
Année


Évolution de la Mass_Mon_diff
Mass_Mon_diff
20
0
−20
1980
1990
2000
2010
2020
Année

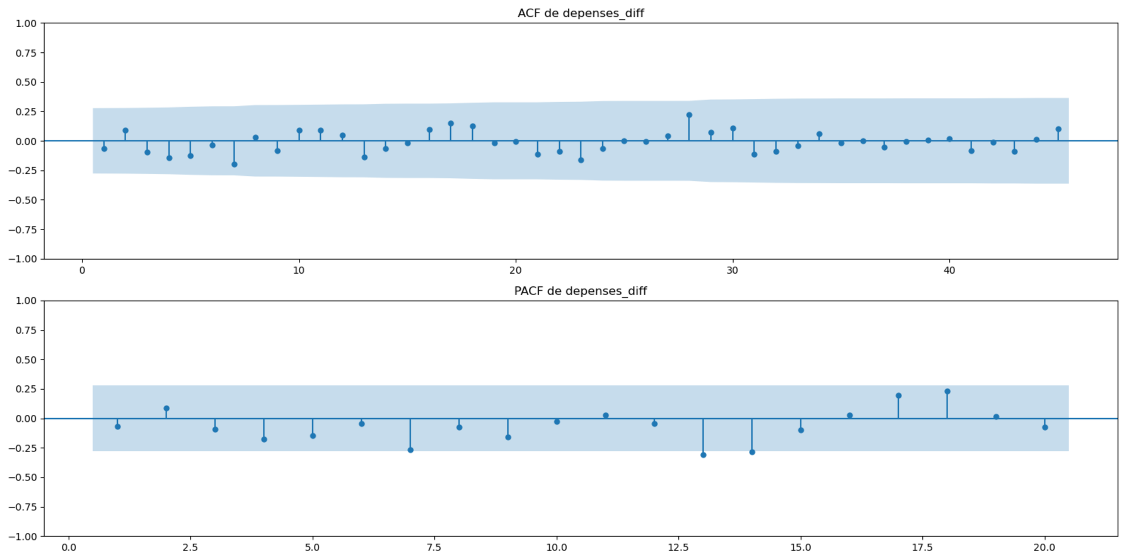


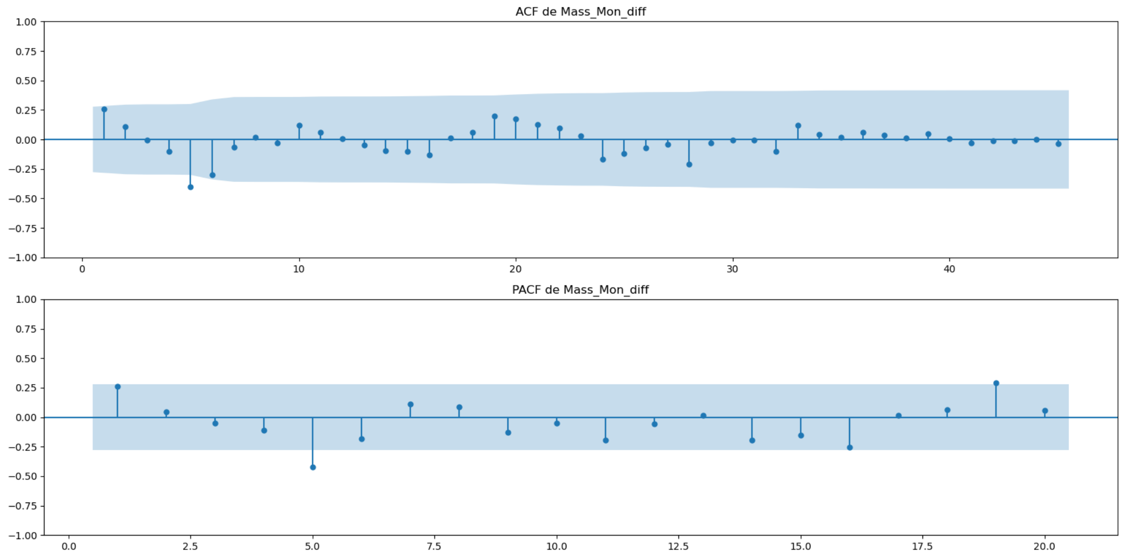
[104]:
```
# Faire les tests de Dickey-Fuller augmenté pour les séries différenciées
for column in df_diff.columns:
    result = adfuller(df[column])
    print(f"\nResults for column {column}:")
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    print('Critical Values:')
    for key, value in result[4].items():
        print(f'\t{key}: {value:.3f}')
```

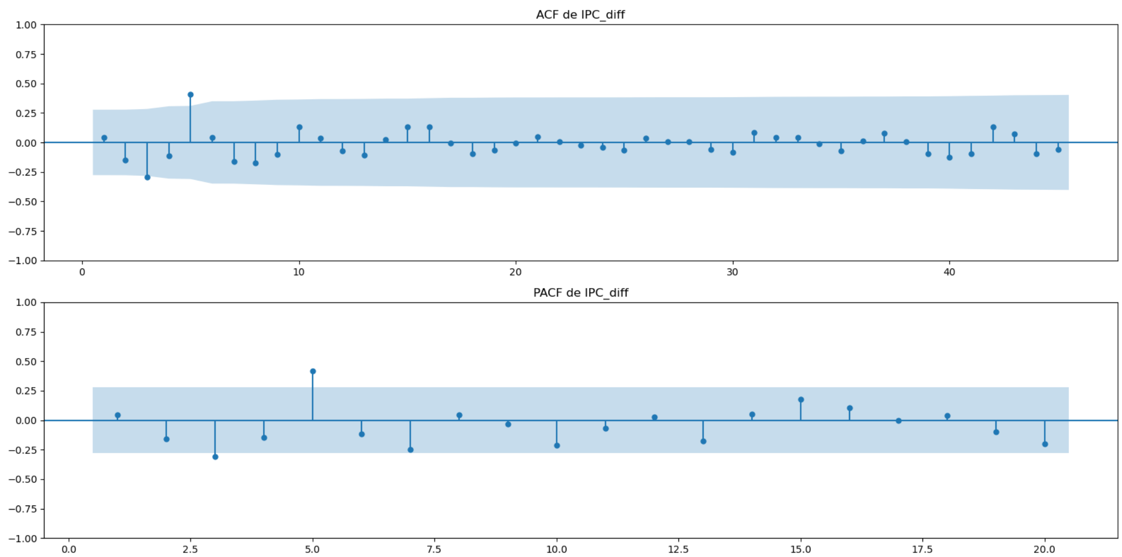
```
Results for column depenses_diff:
ADF Statistic: -4.167977524602791
p-value: 0.0007461448757894758
Critical Values:
        1%: -3.601
        5%: -2.935
        10%: -2.606

Results for column Mass_Mon_diff:
ADF Statistic: -4.583590916426238
p-value: 0.0001384703714606517
Critical Values:
        1%: -3.585
        5%: -2.928
        10%: -2.602

Results for column IPC_diff:
ADF Statistic: -1.3855675908734404
p-value: 0.5891753067726766
```

```
Critical Values:
	1%: -3.589
	5%: -2.930
	10%: -2.603
```

[105]:
```python
# Faire les graphiques ACF et PACF pour chaque colonne différenciée
for column in df_diff.columns:
    fig, (ax1, ax2) = plt.subplots(2,1, figsize=(16,8))
    plot_acf(df[column], lags=45, zero=False, ax=ax1)
    ax1.set_title(f"ACF de {column}")
    plot_pacf(df[column], lags=20, zero=False, ax=ax2)
    ax2.set_title(f"PACF de {column}")
    plt.tight_layout()
    plt.show()
```



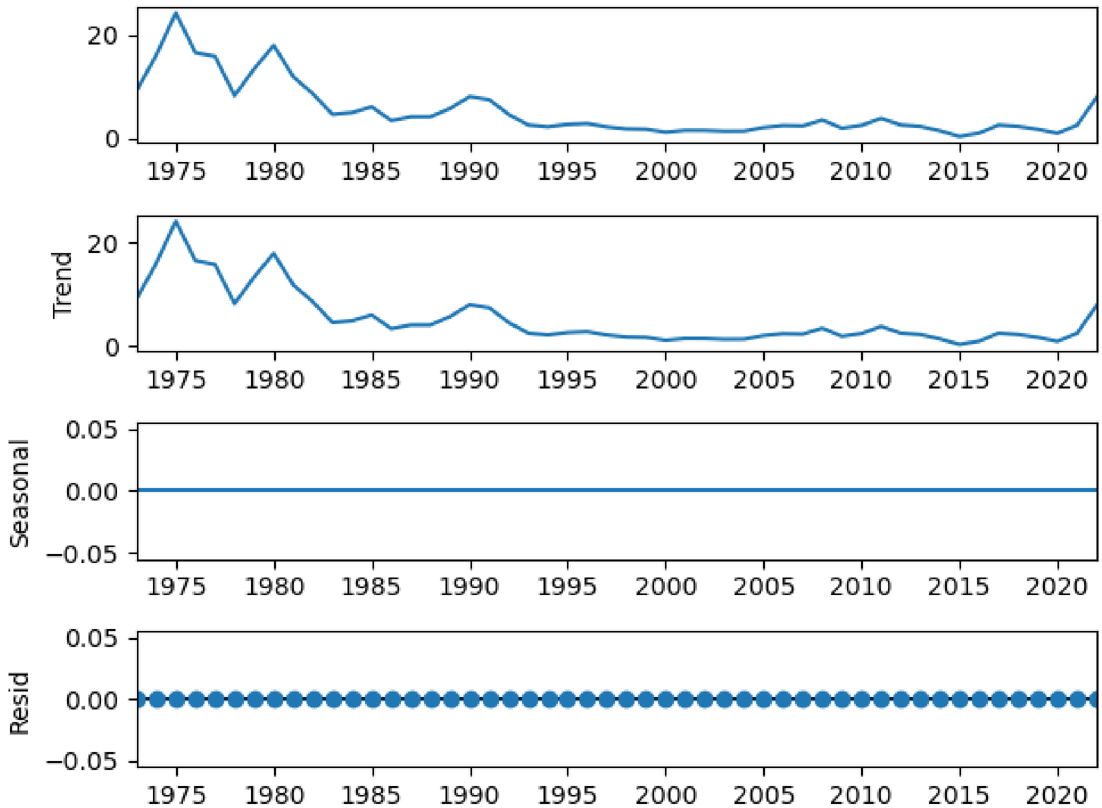




4 Saisonabilité

[106]:
```
analysis = df[['IPC']].copy()
decompose_result_mult = seasonal_decompose(analysis, model="additive")

trend = decompose_result_mult.trend
seasonal = decompose_result_mult.seasonal
residual = decompose_result_mult.resid
```

```
decompose_result_mult.plot();
```



5 Division de la base en train et en base test

[121]:
```
dfd= pd.concat([df['PIB'], df_diff], axis=1).dropna()
dfd.head()
```

[121]:
```
                  PIB  depenses_diff  Mass_Mon_diff  IPC_diff
Année
1973-01-01   6.523848      -0.673779       3.984033  2.124935
1974-01-01  -2.484404       4.415496       0.274803  6.847978
1975-01-01  -1.473649       3.235434      -5.747030  8.163276
1976-01-01   2.910266      -0.044050      -2.135280 -7.647765
1977-01-01   2.457751      -1.916793      -2.082404 -0.719256
```

[123]:
```
# Définir le point de division
split_point = int(len(dfd) * 0.9)  # 90% des données pour l'entraînement

# Diviser les données
df_train = dfd.iloc[:split_point]
```

```
df_test = dfd.iloc[split_point:]

# Vérifier les tailles des ensembles d'entraînement et de test
print('Training set:', len(df_train))
print('Test set:', len(df_test))
```

```
Training set: 45
Test set: 5
```

[128]:
```
# Choisir l'ordre optimal du modèle VAR
model = VAR(df_train)

# Choisir l'ordre optimal p pour le modèle VAR
order_selection = model.select_order(maxlags=5)
order_selection.summary()
```

[128]: <class 'statsmodels.iolib.table.SimpleTable'>

[129]:
```
# L'ordre optimal est donné par le critère d'information d'Akaike (AIC) ou le␣
 ↪critère d'information bayésien (BIC)
optimal_order_aic = order_selection.aic
optimal_order_bic = order_selection.bic
optimal_order_fpe = order_selection.fpe
optimal_order_hqic = order_selection.hqic

print(f"P optimal avec AIC: {optimal_order_aic}")
print(f"P optimal avec BIC: {optimal_order_bic}")
print(f"P optimal avec FPE: {optimal_order_fpe}")
print(f"P optimal avec HQIC: {optimal_order_hqic}")
```

```
P optimal avec AIC: 5
P optimal avec BIC: 0
P optimal avec FPE: 5
P optimal avec HQIC: 1
```

6 Estimation du modèle

[130]:
```
# Créer le modèle VAR
model = VAR(df_train)

# Choisir un ordre 5
results = model.fit(5)

# Voir un sommaire des résultats
print(results.summary())
```

```
  Summary of Regression Results
==================================
```

```
Model:                         VAR
Method:                        OLS
Date:           Thu, 18, Apr, 2024
Time:                     00:50:47
--------------------------------------------------------------------
No. of Equations:         4.00000    BIC:                    9.84930
Nobs:                     40.0000    HQIC:                   7.58501
Log likelihood:          -269.083    FPE:                    869.916
AIC:                      6.30266    Det(Omega_mle):         160.841
--------------------------------------------------------------------
Results for equation PIB
================================================================================
===
                      coefficient       std. error           t-stat
prob
--------------------------------------------------------------------------------
---
const                   -0.109529         1.121097           -0.098
0.922
L1.PIB                  -0.006526         0.265196           -0.025
0.980
L1.depenses_diff        -0.984657         0.352963           -2.790
0.005
L1.Mass_Mon_diff        -0.051472         0.038504           -1.337
0.181
L1.IPC_diff             -0.596072         0.162949           -3.658
0.000
L2.PIB                   0.188380         0.274972            0.685
0.493
L2.depenses_diff         0.411660         0.380119            1.083
0.279
L2.Mass_Mon_diff         0.003304         0.038936            0.085
0.932
L2.IPC_diff             -0.259035         0.192577           -1.345
0.179
L3.PIB                   0.643228         0.262403            2.451
0.014
L3.depenses_diff         0.435794         0.363963            1.197
0.231
L3.Mass_Mon_diff        -0.010868         0.036727           -0.296
0.767
L3.IPC_diff             -0.071195         0.160297           -0.444
0.657
L4.PIB                  -0.042996         0.310708           -0.138
0.890
L4.depenses_diff         0.183206         0.375655            0.488
0.626
L4.Mass_Mon_diff         0.015471         0.036019            0.430
```

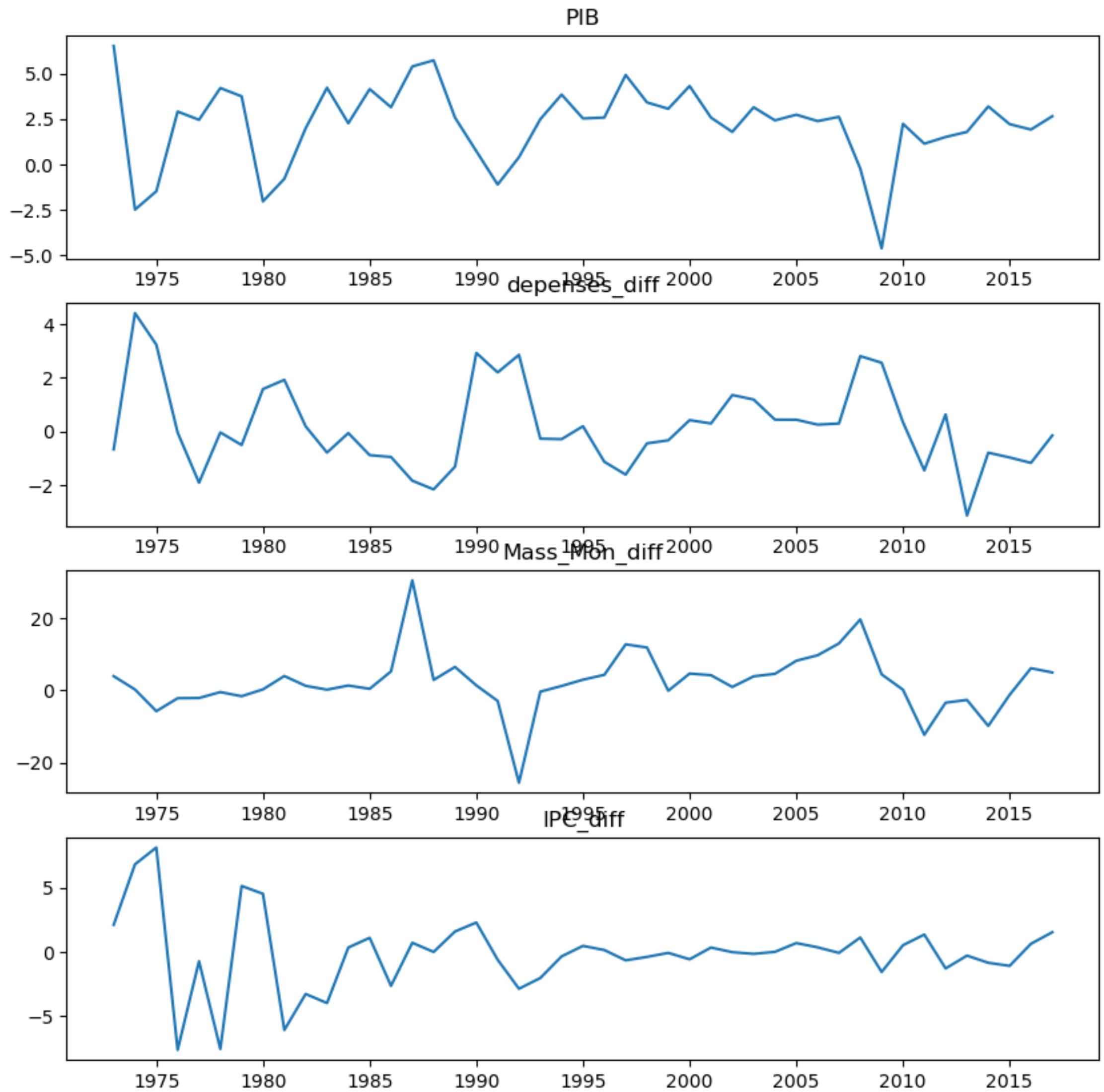
```
0.668
L4.IPC_diff           -0.117863         0.126476           -0.932
0.351
L5.PIB                 0.073938         0.325502            0.227
0.820
L5.depenses_diff      -0.057117         0.366284           -0.156
0.876
L5.Mass_Mon_diff       0.010429         0.036464            0.286
0.775
L5.IPC_diff           -0.286288         0.113914           -2.513
0.012
================================================================================
===

Results for equation depenses_diff
================================================================================
===
                    coefficient       std. error           t-stat
prob
--------------------------------------------------------------------------------
---
const                 -1.243930         0.855363           -1.454
0.146
L1.PIB                -0.373795         0.202336           -1.847
0.065
L1.depenses_diff       0.020326         0.269300            0.075
0.940
L1.Mass_Mon_diff       0.015174         0.029377            0.517
0.605
L1.IPC_diff            0.143768         0.124325            1.156
0.248
L2.PIB                 0.179997         0.209796            0.858
0.391
L2.depenses_diff      -0.092272         0.290019           -0.318
0.750
L2.Mass_Mon_diff      -0.038675         0.029707           -1.302
0.193
L2.IPC_diff           -0.017474         0.146931           -0.119
0.905
L3.PIB                 0.047975         0.200205            0.240
0.811
L3.depenses_diff       0.101518         0.277693            0.366
0.715
L3.Mass_Mon_diff      -0.001223         0.028021           -0.044
0.965
L3.IPC_diff           -0.075728         0.122302           -0.619
0.536
L4.PIB                 0.482416         0.237061            2.035
```

```
0.042
L4.depenses_diff         0.101983         0.286614            0.356
0.722
L4.Mass_Mon_diff         0.008512         0.027481            0.310
0.757
L4.IPC_diff              0.079014         0.096498            0.819
0.413
L5.PIB                   0.256587         0.248348            1.033
0.302
L5.depenses_diff         0.240911         0.279463            0.862
0.389
L5.Mass_Mon_diff         0.016280         0.027821            0.585
0.558
L5.IPC_diff              0.198134         0.086913            2.280
0.023
================================================================================
===

Results for equation Mass_Mon_diff
================================================================================
===
                      coefficient       std. error           t-stat
prob
--------------------------------------------------------------------------------
---
const                   -5.660473         5.160944           -1.097
0.273
L1.PIB                   0.911604         1.220823            0.747
0.455
L1.depenses_diff         0.962035         1.624856            0.592
0.554
L1.Mass_Mon_diff         0.217141         0.177252            1.225
0.221
L1.IPC_diff             -1.276125         0.750133           -1.701
0.089
L2.PIB                   3.289188         1.265830            2.598
0.009
L2.depenses_diff         0.758480         1.749870            0.433
0.665
L2.Mass_Mon_diff        -0.079139         0.179242           -0.442
0.659
L2.IPC_diff              0.875704         0.886526            0.988
0.323
L3.PIB                  -0.230692         1.207966           -0.191
0.849
L3.depenses_diff         3.234652         1.675497            1.931
0.054
L3.Mass_Mon_diff         0.119318         0.169071            0.706
```

```
0.480
L3.IPC_diff             0.616638         0.737926            0.836
0.403
L4.PIB                 -0.123757         1.430336           -0.087
0.931
L4.depenses_diff       -1.033858         1.729322           -0.598
0.550
L4.Mass_Mon_diff        0.097503         0.165813            0.588
0.557
L4.IPC_diff            -0.653509         0.582231           -1.122
0.262
L5.PIB                 -0.268840         1.498442           -0.179
0.858
L5.depenses_diff        1.086108         1.686178            0.644
0.519
L5.Mass_Mon_diff       -0.469614         0.167861           -2.798
0.005
L5.IPC_diff            -0.550382         0.524402           -1.050
0.294
================================================================================
===

Results for equation IPC_diff
================================================================================
===
                      coefficient       std. error           t-stat
prob
--------------------------------------------------------------------------------
---
const                  -1.343160         1.215891           -1.105
0.269
L1.PIB                  0.342179         0.287619            1.190
0.234
L1.depenses_diff        0.023547         0.382808            0.062
0.951
L1.Mass_Mon_diff        0.062514         0.041760            1.497
0.134
L1.IPC_diff            -0.159727         0.176727           -0.904
0.366
L2.PIB                  0.017910         0.298223            0.060
0.952
L2.depenses_diff        0.100442         0.412260            0.244
0.808
L2.Mass_Mon_diff       -0.022530         0.042229           -0.534
0.594
L2.IPC_diff             0.065686         0.208861            0.314
0.753
L3.PIB                 -0.161083         0.284590           -0.566
```

```
0.571
L3.depenses_diff        -0.099625         0.394738           -0.252
0.801
L3.Mass_Mon_diff         0.052811         0.039832            1.326
0.185
L3.IPC_diff             -0.148486         0.173851           -0.854
0.393
L4.PIB                   0.154319         0.336980            0.458
0.647
L4.depenses_diff        -0.452550         0.407419           -1.111
0.267
L4.Mass_Mon_diff         0.034962         0.039065            0.895
0.371
L4.IPC_diff             -0.066674         0.137171           -0.486
0.627
L5.PIB                  -0.079176         0.353025           -0.224
0.823
L5.depenses_diff         0.550934         0.397255            1.387
0.165
L5.Mass_Mon_diff        -0.004776         0.039547           -0.121
0.904
L5.IPC_diff              0.447457         0.123546            3.622
0.000
================================================================================
===

Correlation matrix of residuals
                      PIB  depenses_diff  Mass_Mon_diff  IPC_diff
PIB              1.000000      -0.690335      -0.022932  0.248272
depenses_diff   -0.690335       1.000000      -0.078252 -0.308152
Mass_Mon_diff   -0.022932      -0.078252       1.000000  0.309592
IPC_diff         0.248272      -0.308152       0.309592  1.000000
```

[135]:
```
results.plot()
plt.show()
```



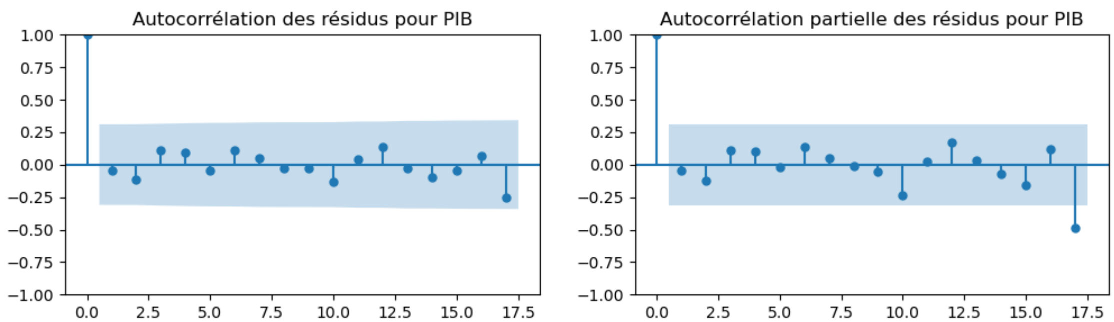
7 Validation du modèle

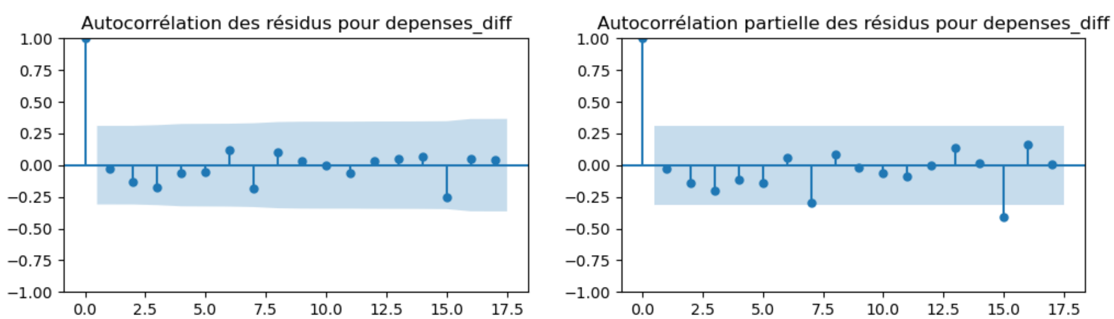
[136]:
```python
# Obtenir les résidus
residuals = results.resid
residuals.head()
```

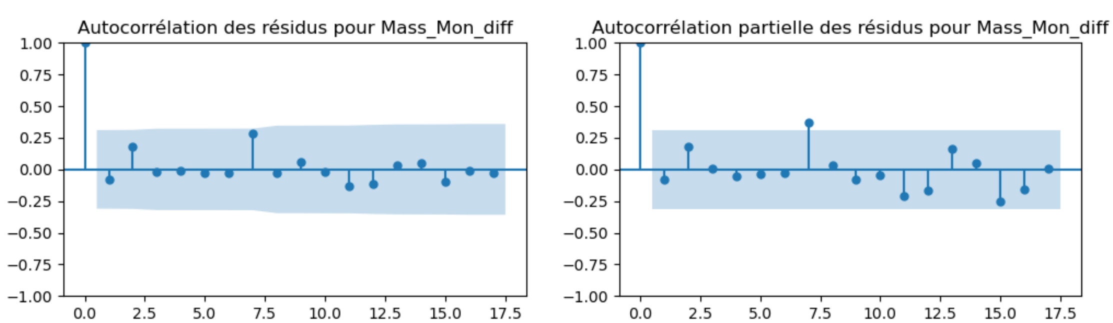
[136]:
```
                  PIB  depenses_diff  Mass_Mon_diff  IPC_diff
Année
1978-01-01 -0.607868       0.114010       0.425423 -2.222838
1979-01-01 -0.177103      -0.028790      -2.181848  0.215909
1980-01-01 -1.006621      -0.074270       3.638977 -0.319160
1981-01-01 -0.778222       0.152255      -1.342012 -1.756115
```

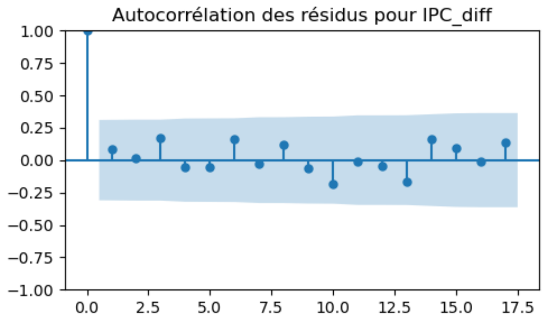
```
1982-01-01 -1.532983       1.360965      -4.830124 -1.539489
```

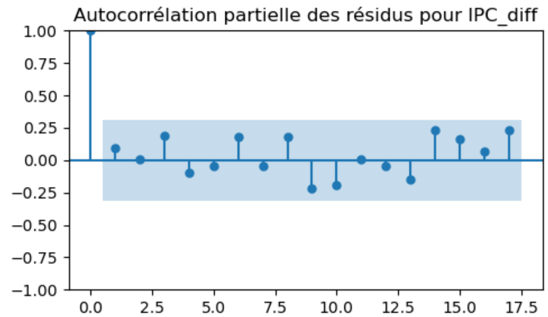
[137]:
```python
# Tracer l'ACF et le PACF pour chaque série de résidus
for col in residuals.columns:
    fig, axes = plt.subplots(1, 2, figsize=(12, 3))

    plot_acf(residuals[col], ax=axes[0])
    axes[0].set_title(f'Autocorrélation des résidus pour {col}')

    plot_pacf(residuals[col], ax=axes[1])
    axes[1].set_title(f'Autocorrélation partielle des résidus pour {col}')

plt.show()
```











[138]:
```
from statsmodels.stats.stattools import jarque_bera
from statsmodels.stats.diagnostic import acorr_ljungbox
# Effectuer des tests sur les résidus
for col in residuals.columns:
    print(f"Résidus pour {col}:")
    jb_test = jarque_bera(residuals[col])
    print(f"Test de Jarque-Bera: statistic={jb_test[0]}, p-value={jb_test[1]}")
```

```
Résidus pour PIB:
Test de Jarque-Bera: statistic=1.9059317485845881, p-value=0.3855956974812551
Résidus pour depenses_diff:
Test de Jarque-Bera: statistic=2.1044742399081877, p-value=0.3491557714050238
Résidus pour Mass_Mon_diff:
Test de Jarque-Bera: statistic=1.7449668760734254, p-value=0.41791239990028173
Résidus pour IPC_diff:
Test de Jarque-Bera: statistic=2.7136207894529063, p-value=0.2574807304901465
```

[139]:
```
# Préparer un tableau pour stocker les résultats
results_df = pd.DataFrame(columns=['Variable', 'Lag', 'LB Statistic', 'LB␣
↪P-value'])

# Effectuer le test de Ljung-Box pour des décalages de 1 à 12
for col in residuals.columns:
    for lag in range(1, 13):
        lb_test = acorr_ljungbox(residuals[col], lags=[lag], return_df=True)
        results_df = results_df.append({
            'Variable': col,
            'Lag': lag,
            'LB Statistic': lb_test['lb_stat'].values[0],
            'LB P-value': lb_test['lb_pvalue'].values[0]
        }, ignore_index=True)

# Afficher les résultats
```

```
print(results_df)
```

```
         Variable Lag  LB Statistic  LB P-value
0             PIB   1      0.096761    0.755751
1             PIB   2      0.709175    0.701463
2             PIB   3      1.264977    0.737466
3             PIB   4      1.698864    0.790924
4             PIB   5      1.812667    0.874411
5             PIB   6      2.372771    0.882426
6             PIB   7      2.511812    0.926205
7             PIB   8      2.561811    0.958786
8             PIB   9      2.616649    0.977580
9             PIB  10      3.540358    0.965710
10            PIB  11      3.623642    0.979663
11            PIB  12      4.690470    0.967527
12  depenses_diff   1      0.036890    0.847690
13  depenses_diff   2      0.784358    0.675583
14  depenses_diff   3      2.166080    0.538660
15  depenses_diff   4      2.358316    0.670173
16  depenses_diff   5      2.506311    0.775544
17  depenses_diff   6      3.162226    0.788228
18  depenses_diff   7      4.813343    0.682730
19  depenses_diff   8      5.326830    0.722141
20  depenses_diff   9      5.393058    0.798787
21  depenses_diff  10      5.393179    0.863415
22  depenses_diff  11      5.642614    0.896115
23  depenses_diff  12      5.701439    0.930378
24  Mass_Mon_diff   1      0.287804    0.591631
25  Mass_Mon_diff   2      1.663023    0.435391
26  Mass_Mon_diff   3      1.679474    0.641507
27  Mass_Mon_diff   4      1.687122    0.793056
28  Mass_Mon_diff   5      1.729041    0.885221
29  Mass_Mon_diff   6      1.776083    0.939098
30  Mass_Mon_diff   7      5.935356    0.547318
31  Mass_Mon_diff   8      5.968670    0.650741
32  Mass_Mon_diff   9      6.143811    0.725443
33  Mass_Mon_diff  10      6.164239    0.801281
34  Mass_Mon_diff  11      7.226793    0.780431
35  Mass_Mon_diff  12      7.995430    0.785487
36       IPC_diff   1      0.337762    0.561124
37       IPC_diff   2      0.346063    0.841111
38       IPC_diff   3      1.680647    0.641245
39       IPC_diff   4      1.828997    0.767173
40       IPC_diff   5      1.969633    0.853331
41       IPC_diff   6      3.324634    0.767136
42       IPC_diff   7      3.357120    0.850119
43       IPC_diff   8      4.070265    0.850729
44       IPC_diff   9      4.272726    0.892563
```

```
45        IPC_diff   10      6.148646     0.802624
46        IPC_diff   11      6.153666     0.862917
47        IPC_diff   12      6.287597     0.900896
```

8 Prédiction train et test

[140]:
```
# Prévoir les valeurs sur l'ensemble d'entraînement
# Prédictions sur les ensembles d'entraînement et de test
train_pred = results.fittedvalues
test_pred = results.forecast(df_train.values, steps=len(df_test))
```

[142]:
```
forecast = results.forecast(df_train.values, len(df_test))
```

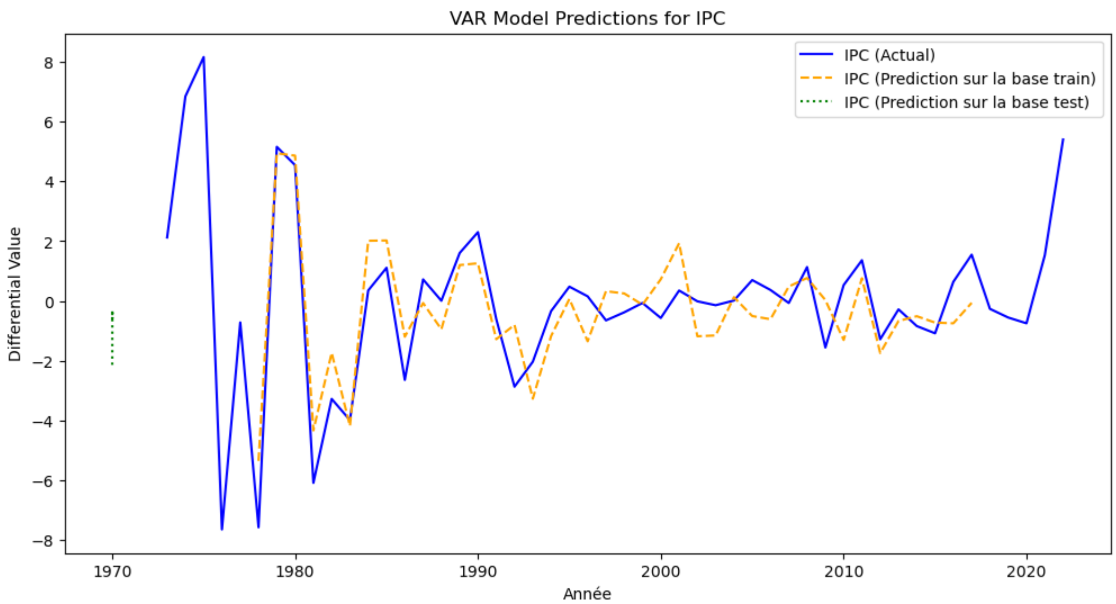
[160]:
```
#train_rmse = rmse(df_train, train_pred)
```

[143]:
```
forecast
```

[143]:
```
array([[-0.30418627, -0.73044023, -5.73882813, -2.13858285],
       [ 3.10057656, -0.17420198,  5.99134857, -1.62824441],
       [ 2.59773517, -1.27797102, -1.46818597, -0.30837701],
       [ 1.88939742,  0.01138868, -6.32393935, -0.5876255 ],
       [ 2.40642024, -0.64119103, -1.14970416, -0.38906814]])
```

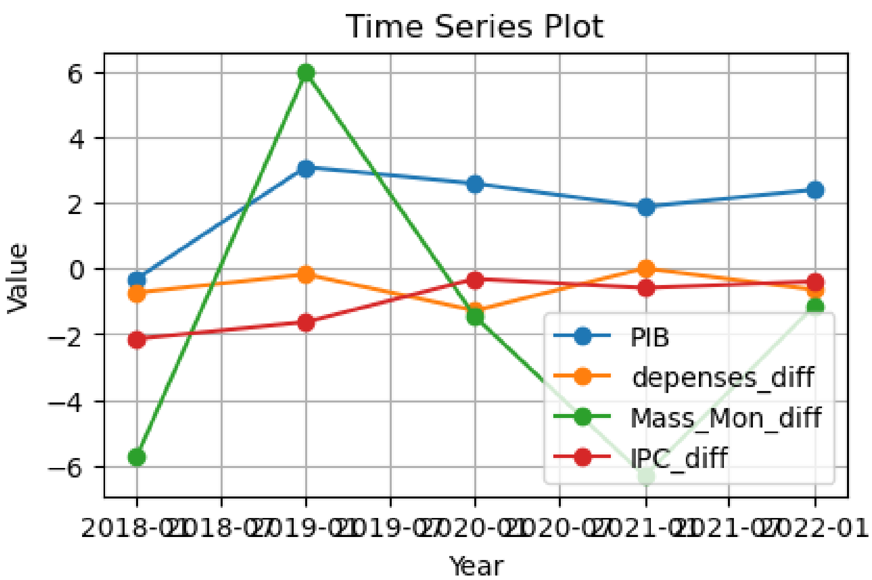
[145]:
```
# Créer un DataFrame avec les prédictions
train_pred_df = pd.DataFrame(train_pred, columns=df_train.columns)
#train_pred_df['Année'] = df_diff.index[4:len(df_train)]  # Ajouter les dates␣
↪correspondant à l'ensemble d'entraînement
#train_pred_df.set_index('Année', inplace=True)  # Utiliser les dates comme␣
↪indices

test_pred_df = pd.DataFrame(test_pred, columns=df_train.columns)
#test_pred_df['Année'] = df_diff.index[len(df_train):]  # Ajouter les dates␣
↪correspondant à l'ensemble d'entraînement
#test_pred_df.set_index('Année', inplace=True)  # Utiliser les dates comme␣
↪indices
```

[149]:
```
plt.figure(figsize=(12, 6))
plt.plot(df_diff.index, df_diff['IPC_diff'], label='IPC (Actual)', color='blue')
plt.plot(train_pred_df.index, train_pred_df['IPC_diff'], label='IPC (Prediction␣
↪sur la base train)', linestyle='dashed', color='orange')
plt.plot(test_pred_df.index, test_pred_df['IPC_diff'], label='IPC (Prediction␣
↪sur la base test)', linestyle='dotted', color='green')
plt.xlabel('Année')
plt.ylabel('Differential Value')
plt.title('VAR Model Predictions for IPC')
plt.legend()
plt.show()
```



[151]:
```
cols = ['PIB', 'depenses_diff', 'Mass_Mon_diff', 'IPC_diff']
forc = pd.DataFrame(forecast, index = df_test.index, columns = cols)
cols2 = forc.columns
```

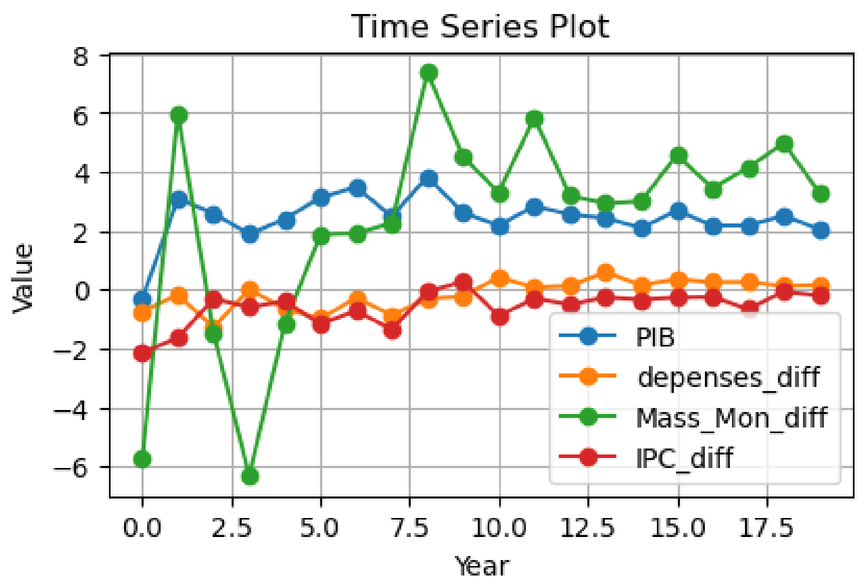
[152]:
```
plt.figure(figsize=(5, 3))
for column in forc.columns:
    plt.plot(forc.index, forc[column], marker='o', label=column)

plt.title('Time Series Plot')
plt.xlabel('Year')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```



[154]:
```
forecast = results.forecast(df_train.values, 20)
```

[155]:
```
cols = ['PIB', 'depenses_diff', 'Mass_Mon_diff', 'IPC_diff']
forc = pd.DataFrame(forecast, columns = cols)
cols2 = forc.columns
```

[157]:
```
plt.figure(figsize=(5, 3))
for column in forc.columns:
    plt.plot(forc.index, forc[column], marker='o', label=column)

plt.title('Time Series Plot')
plt.xlabel('Year')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```



[158]:
```
test_rmse = rmse(df_test, test_pred)
```

[161]:
```
test_rmse
```

[161]:
```
array([ 6.67462882,  5.86640774, 13.58467749,  2.92890357])
```

[162]:
```
# Calculate absolute error
absolute_error = abs(df_test.values - test_pred)

# Calculate absolute percentage error
absolute_percentage_error = absolute_error / df_test.values * 100

# Calculate mean of absolute percentage errors
mape = absolute_percentage_error.mean()

print("MAPE for test set:", mape)
```

```
MAPE for test set: 25.09414585104277
```