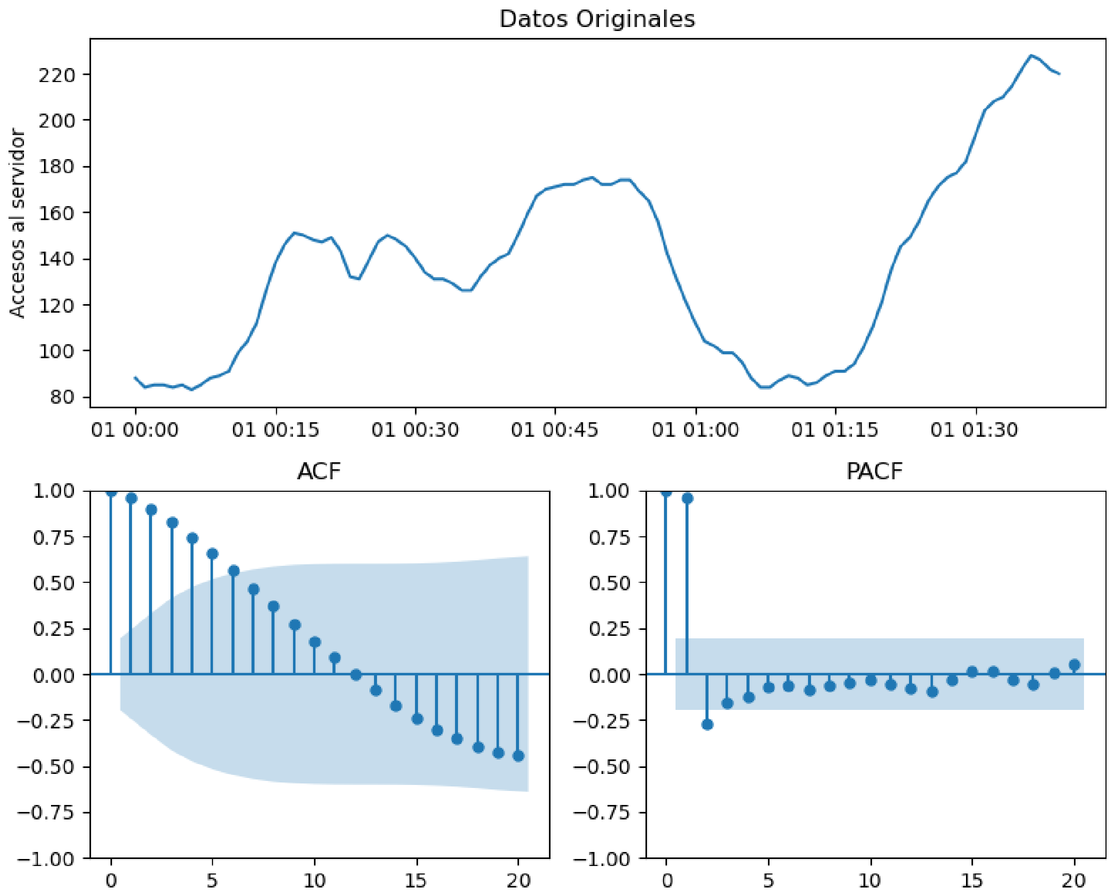
project_final_clean

January 13, 2025

[3]:
```python
# Función para graficar una serie temporal, su ACF y PACF
# Importar bibliotecas necesarias para gráficos, manipulación de datos y␣
 ↪análisis de series temporales
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from numpy import float128  # Para cálculos de alta precisión
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf  # Para gráficas␣
 ↪de ACF y PACF
import statsmodels.tsa.stattools as st  # Herramientas estadísticas para series␣
 ↪temporales
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller  # Prueba de Dickey-Fuller para␣
 ↪estacionariedad

# Variable para alternar entre datos artificiales y datos reales
test_with_artificial = False

# Función para generar datos sintéticos siguiendo un modelo ARIMA
def artificial_arima(p=np.array([]), d=0, q=np.array([]), f=lambda x: x, n=100,␣
 ↪m=0):
    """
    Genera datos sintéticos basados en un modelo ARIMA para validar el método␣
 ↪en datos reales.

    Parámetros:
    - p: Coeficientes del modelo AR (Autoregresivo).
    - d: Número de diferencias acumulativas para hacer la serie estacionaria.
    - q: Coeficientes del modelo MA (Media Móvil).
    - f: Función de transformación aplicada a los datos generados.
    - n: Número de puntos en la serie temporal.
    - m: Media del ruido blanco agregado.

    Retorna:
    - Serie transformada basada en los parámetros proporcionados.
    """
```

```python
    a = np.random.normal(0, 1, n)  # Generar ruido blanco con media 0 y␣
↪varianza 1
    W = np.zeros(n)  # Inicializar la serie temporal

    for t in range(n):
        if t < len(p) or t < len(q):  # Manejar índices fuera de rango
            W[t] = 0
        else:
            # Aplicar componentes AR y MA usando productos escalares
            W[t] = -W[t-len(p):t] @ p[::-1] + a[t] + a[t-len(q):t] @ q[::-1]

    for d_c in range(d):  # Aplicar diferenciación acumulativa d veces
        W = np.cumsum(W)

    W += m  # Agregar media a la serie
    return f(W)  # Aplicar la transformación final

# Si se activa `test_with_artificial`, generar datos sintéticos
if test_with_artificial:
    n = 1000  # Número de puntos en la serie sintética
    points = artificial_arima(
        p=np.array([0]),  # Coeficientes AR
        q=np.array([0]),  # Coeficientes MA
        d=1,  # Diferenciación
        f=lambda x: 1.01**x,  # Función de transformación exponencial
        n=n,  # Tamaño de la serie
        m=0  # Media
    )
    # Crear un índice de fechas mensual empezando desde 1800
    dates = pd.date_range(start='1800-01-01', periods=n, freq='MS')
    # Crear un DataFrame con los datos generados y el índice temporal
    time_series_df = pd.DataFrame({'points': 1.7**points}, index=dates)
else:
    # Usar un dataset real: WWWusage
    freq = 'min'  # Frecuencia de minutos
    time_series_df = pd.read_csv('../data/WWWusage.csv', header=0)  # Leer el␣
↪archivo CSV
    time_series_df['points'] = time_series_df['WWWusage'] # Extraer la columna␣
↪relevante

    # Crear un índice de tiempo ficticio para compatibilidad con otras␣
↪bibliotecas
    start_date = '2023-01-01 00:00:00'
    time_series_df['datetime'] = pd.to_datetime(start_date) + pd.
↪to_timedelta(time_series_df['time'] - 1, unit='m')
    time_series_df.set_index('datetime', inplace=True)  # Establecer 'datetime'␣
↪como índice
```

```python
    time_series_df.index.freq = freq  # Asignar frecuencia temporal al índice
def plot_series(series, series_title, alpha=0.05):
    """
    Grafica una serie temporal junto con sus funciones ACF y PACF.

    Parámetros:
    - series: Serie temporal a graficar.
    - series_title: Título para la gráfica de la serie temporal.
    - alpha: Nivel de significancia para los intervalos de confianza en ACF y␣
 ↪PACF.
    """
    fig = plt.figure(figsize=(8, 6.5))  # Crear figura de tamaño personalizado
    gs = fig.add_gridspec(2, 2)  # Crear un diseño de 2 filas y 2 columnas

    # Gráfica de la serie temporal
    ax0 = fig.add_subplot(gs[0, :])  # Primera fila ocupa ambas columnas
    ax0.plot(series)
    ax0.set_title(series_title)
    ax0.set_ylabel('Accesos al servidor')

    # Gráfica de ACF
    ax1 = fig.add_subplot(gs[1, 0])  # Segunda fila, primera columna
    plot_acf(series, ax=ax1, alpha=alpha)
    ax1.set_title("ACF")

    # Gráfica de PACF
    ax2 = fig.add_subplot(gs[1, 1])  # Segunda fila, segunda columna
    plot_pacf(series, ax=ax2, alpha=alpha)
    ax2.set_title("PACF")

    plt.tight_layout()  # Ajustar diseño
    plt.show()  # Mostrar gráficos


# fraction of data that should be training data
train_fraction = len(time_series_df) -10


# Graficar la serie temporal con su ACF y PACF
plot_series(time_series_df['points'], "Datos Originales")
```



Utiliza un método completamente automático de biblioteca para determinar el modelo ARIMA que maximice la precisión, de modo que podamos comparar el resultado de nuestro método con el resultado de este método.

[ ]:

```

```

[4]:

```
if False:  # Cambia a `True` para ejecutar este bloque de código
    import pmdarima as pm  # Biblioteca para ajuste automático de modelos ARIMA/
 ↪SARIMA

    # Ajustar automáticamente el mejor modelo ARIMA/SARIMA
    model = pm.auto_arima(
        time_series_df['points'],  # Columna de datos de la serie temporal
        seasonal=False,            # Desactiva el componente estacional
        stepwise=False,            # Desactiva el algoritmo de búsqueda paso a␣
 ↪paso (más exhaustivo)
        trace=True,                # Muestra detalles del proceso de ajuste
        max_p=3,                   # Máximo valor de p (orden autoregresivo)
        max_q=3,                   # Máximo valor de q (orden de media móvil)
```

```
        max_d=2,                    # Máximo número de diferenciaciones (d)
        start_d=2,                  # Comienza probando con una diferenciación␣
↪inicial d=2
        test='adf',                 # Prueba de Dickey-Fuller para verificar␣
↪estacionariedad
        max_order=10                # Límite máximo para p + q + P + Q
    )

    # Mostrar un resumen del mejor modelo ajustado
    print(model.summary())

    # Pronosticar valores futuros (por ejemplo, para los próximos 12 períodos)
    forecast = model.predict(n_periods=12)  # Genera predicciones para 12␣
↪períodos futuros
```

Prueba la estacionariedad utilizando el test de raíz unitaria de Dickey-Fuller aumentado.

[5]:
```
# Verificar la estacionariedad de la serie temporal
# La hipótesis alternativa de la prueba ADF es que la serie es estacionaria
adf_result = st.adfuller(time_series_df['points'])  # Realizar la prueba ADF
print(adf_result[1])
```

```
0.12441935447109453
```

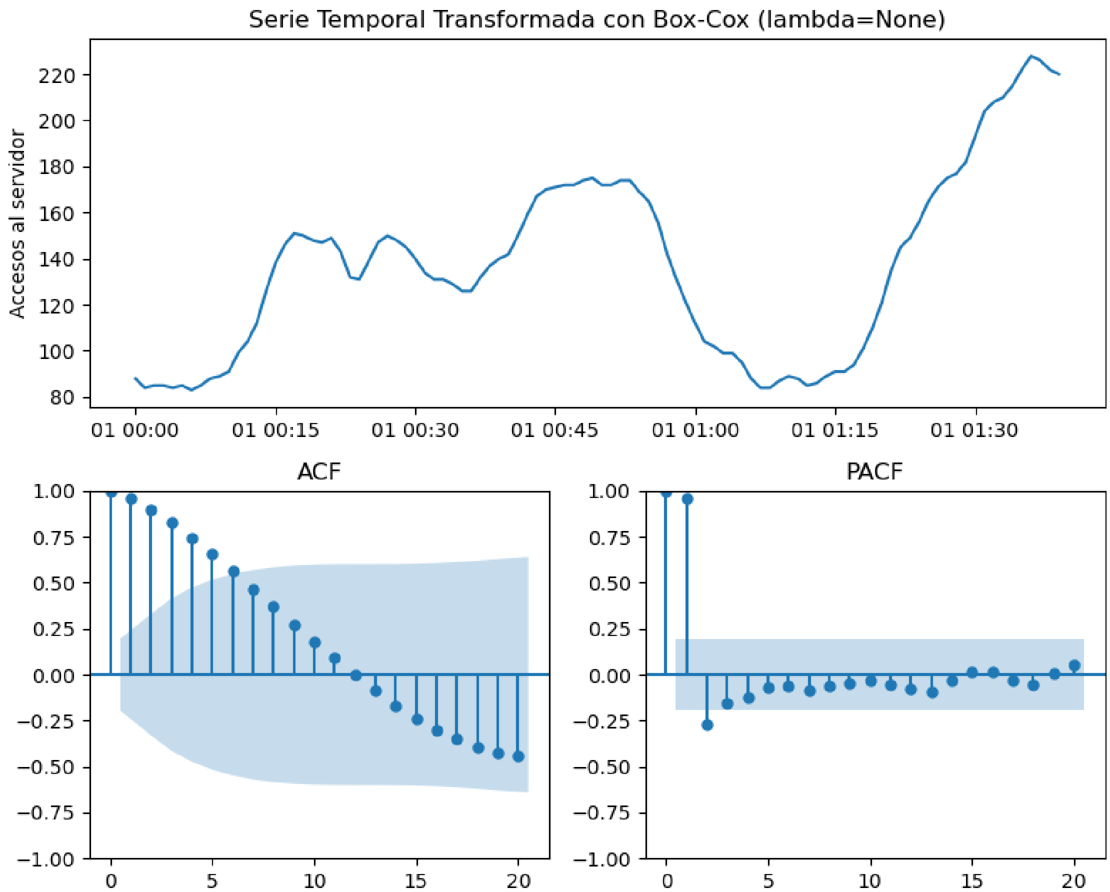
p value > 0.05 => no stationaridad

Aplica la transformación de Box-Cox y usa scipy para estimar el parámetro óptimo de Box-Cox, lambda.

Decidimos no utilizar esta transformación, ya que después de diferenciar los datos, la serie temporal parece bastante estable. Además, obtuvimos mejores resultados al no aplicar ninguna transformación.

[6]:
```
from scipy import stats  # Importar herramientas estadísticas de SciPy

# Variable para controlar si se aplica la transformación Box-Cox
use_trafo = False

# Verificar si se aplica la transformación Box-Cox
if use_trafo:
    # Aplicar la transformación Box-Cox y estimar lambda automáticamente si no␣
↪se especifica
    lmbda = None  # Valor inicial de lambda; si es None, se estima␣
↪automáticamente
    if lmbda is None:
        # Aplicar Box-Cox y estimar lambda óptimo
        transformed_prices, lmbda = stats.boxcox(time_series_df["points"])
    else:
        # Aplicar Box-Cox con un valor específico de lambda
```

```
        transformed_prices = stats.boxcox(time_series_df["points"], lmbda=lmbda)

    # Almacenar los valores transformados en el DataFrame
    time_series_df['T_points'] = transformed_prices
    print("Lambda usado: ", lmbda)  # Mostrar el valor de lambda utilizado
else:
    # Si no se aplica la transformación, conservar la serie original
    lmbda = None
    time_series_df['T_points'] = time_series_df['points']

# Graficar la serie temporal (transformada o sin transformar)
plot_series(time_series_df['T_points'], f"Serie Temporal Transformada con␣
↪Box-Cox (lambda={lmbda})")
```



Diferencia los datos hasta que las pruebas de estacionariedad sean positivas.

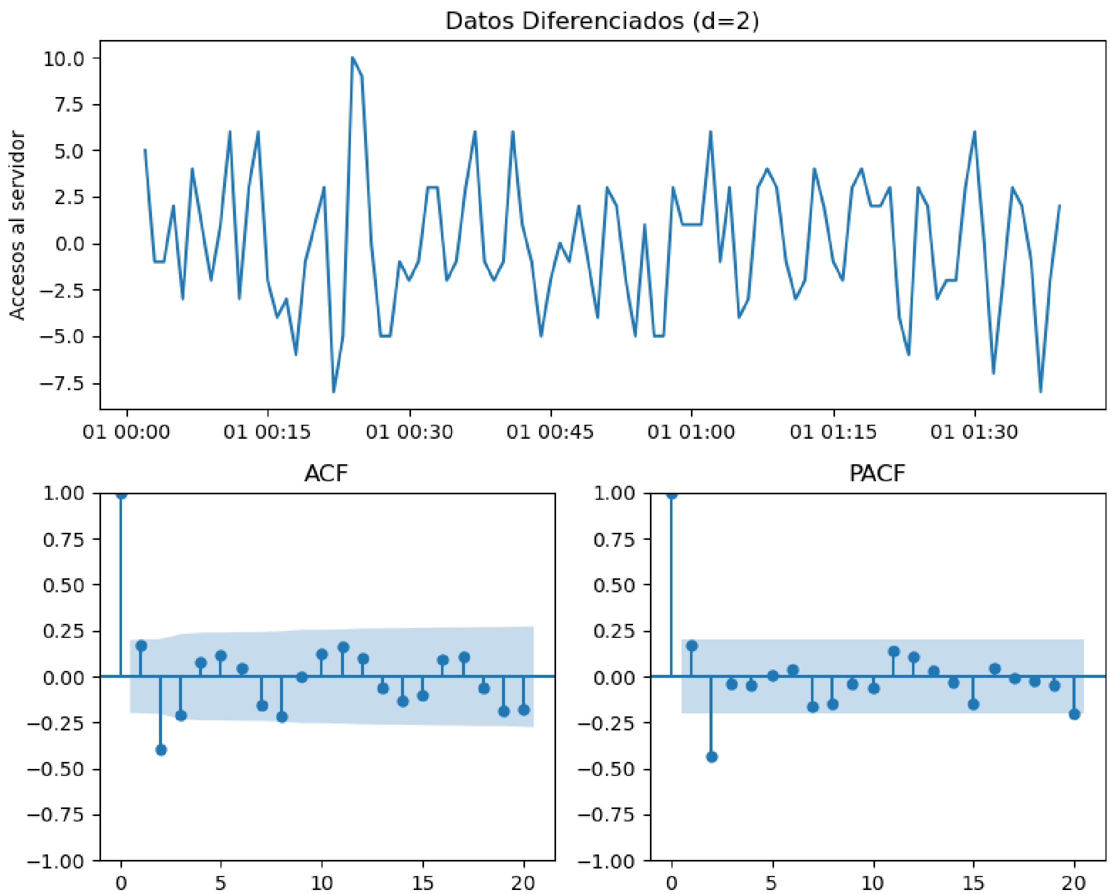
[7]:
```
from statsmodels.sandbox.archive import tsa  # Importar herramientas para␣
↪análisis de series temporales (opcional)
```

```python
# Nivel de significancia para la prueba ADF
alpha = 0.05

# Seleccionar la serie transformada sin valores nulos
current_series = time_series_df['T_points'].dropna()
d = 0  # Contador de diferenciaciones aplicadas

# Función para obtener el p-valor de la prueba ADF
def get_adf_p_value(series):
    """
    Realiza la prueba Dickey-Fuller Aumentada (ADF) y retorna el p-valor.

    Parámetros:
    - series: Serie temporal a analizar.

    Retorna:
    - p-valor de la prueba ADF.
    """
    adf_result = st.adfuller(series.dropna())  # Realizar la prueba ADF
    return adf_result[1]  # Retornar el p-valor

# Iterar hasta que la serie sea estacionaria o se alcance el máximo de␣
↪diferenciaciones
while get_adf_p_value(current_series) >= alpha:
    d += 1  # Incrementar el contador de diferenciaciones
    # Aplicar diferenciación de primer orden
    current_series = current_series.diff()  # Calcula la diferencia entre␣
↪valores consecutivos

    # Imprimir el progreso y el p-valor después de cada diferenciación
    print(f"Después de {d} diferenciación(es), el p-valor de ADF es:␣
↪{get_adf_p_value(current_series)}")

# Agregar la serie diferenciada al DataFrame original
time_series_df['diff_points'] = current_series.copy()

# Verificar si la serie es estacionaria después de la diferenciación
if get_adf_p_value(current_series) < alpha:
    print(f"La serie es estacionaria después de {d} diferenciación(es).")
else:
    print("Se alcanzó el máximo de diferenciaciones sin lograr estacionariedad.
↪")
```

```
Después de 1 diferenciación(es), el p-valor de ADF es: 0.07026846015272693
Después de 2 diferenciación(es), el p-valor de ADF es: 2.843428755547158e-17
La serie es estacionaria después de 2 diferenciación(es).
```

Grafica la nueva serie temporal y las funciones de ACF y PCF.

```python
[8]: # Seleccionar la serie diferenciada eliminando valores nulos generados por la operación diff
data = time_series_df['diff_points'].dropna()

# Graficar la serie diferenciada junto con las gráficas de ACF y PACF
plot_series(data, f"Datos Diferenciados (d={d})", alpha=0.05)
```



Imprime todos los rezagos que son significativamente diferentes de cero.

```python
[9]: data = time_series_df['T_points'].dropna()

def print_significant_lags(data, alpha=0.05, nlags=30):
    """
    Calculate and print all significantly non-zero lags using ACF.

    Parameters:
    -----------
    data : array-like
        The time series data
```

```
    alpha : float, default=0.05
        Significance level for the confidence intervals
    nlags : int, default=40
        Number of lags to calculate

    Returns:
    --------
    tuple
        - List of significant lag indices
        - ACF values for significant lags
        - Confidence intervals
    """
    # Calculate ACF with confidence intervals
    acf_values, acf_confint = st.acf(data, alpha=alpha, fft=True, nlags=nlags,␣
↪adjusted=True)
    pacf_values, pacf_confint = st.pacf(data, alpha=alpha, nlags=nlags)

    # The confidence intervals come as [lower, upper] for each lag
    # If 0 is not in [lower, upper], the lag is significant
    acf_significant_lags = []
    acf_significant_values = []

    print(f"\nSignificant lags at {alpha*100}% significance level:")
    print("----------------------------------------")
    print("Lag | ACF Value | Confidence Interval")
    print("----------------------------------------")

    for lag in range(len(acf_values)):
        lower_ci = acf_confint[lag][0]
        upper_ci = acf_confint[lag][1]

        # Check if 0 is outside the confidence interval
        if (lower_ci > 0) or (upper_ci < 0):
            acf_significant_lags.append(lag)
            acf_significant_values.append(acf_values[lag])
            print(f"{lag:3d} | {acf_values[lag]:9.3f} | [{lower_ci:6.3f},␣
↪{upper_ci:6.3f}]")


    # The confidence intervals come as [lower, upper] for each lag
    # If 0 is not in [lower, upper], the lag is significant
    pacf_significant_lags = []
    pacf_significant_values = []

    print(f"\nSignificant lags at {alpha*100}% significance level:")
    print("----------------------------------------")
    print("Lag | PACF Value | Confidence Interval")
```

```python
    print("----------------------------------------")

    for lag in range(len(pacf_values)):
        lower_ci = pacf_confint[lag][0]
        upper_ci = pacf_confint[lag][1]

        # Check if 0 is outside the confidence interval
        if (lower_ci > 0) or (upper_ci < 0):
            pacf_significant_lags.append(lag)
            pacf_significant_values.append(acf_values[lag])
            print(f"{lag:3d} | {pacf_values[lag]:10.3f} | [{lower_ci:6.3f},␣
↪{upper_ci:6.3f}]")


    if not acf_significant_lags and not pacf_significant_lags:
        print("No significant lags found.")

    return (acf_significant_lags, acf_significant_values, acf_confint),␣
↪(pacf_significant_lags, pacf_significant_values, pacf_confint)
sig = print_significant_lags(data)
```

```
Significant lags at 5.0% significance level:
----------------------------------------
Lag | ACF Value | Confidence Interval
----------------------------------------
  0 |     1.000 | [ 1.000,  1.000]
  1 |     0.970 | [ 0.774,  1.166]
  2 |     0.920 | [ 0.587,  1.252]
  3 |     0.854 | [ 0.435,  1.273]
  4 |     0.777 | [ 0.296,  1.258]
  5 |     0.692 | [ 0.164,  1.219]
  6 |     0.601 | [ 0.040,  1.162]

Significant lags at 5.0% significance level:
----------------------------------------
Lag | PACF Value | Confidence Interval
----------------------------------------
  0 |      1.000 | [ 1.000,  1.000]
  1 |      0.970 | [ 0.774,  1.166]
  2 |     -0.356 | [-0.552, -0.160]
```

A partir de los rezagos significativos en la ACF y PACF en el lag 2, y de los gráficos de ACF y PACF, podemos proponer estos modelos, que tienen como máximo 2 parámetros AR o como máximo 2 parámetros MA:

- **ARIMA(0,2,0)** (para fines de comparación)
- **ARIMA(0,2,2)**

- **ARIMA(2,2,0)**
- **ARIMA(2,2,2)**

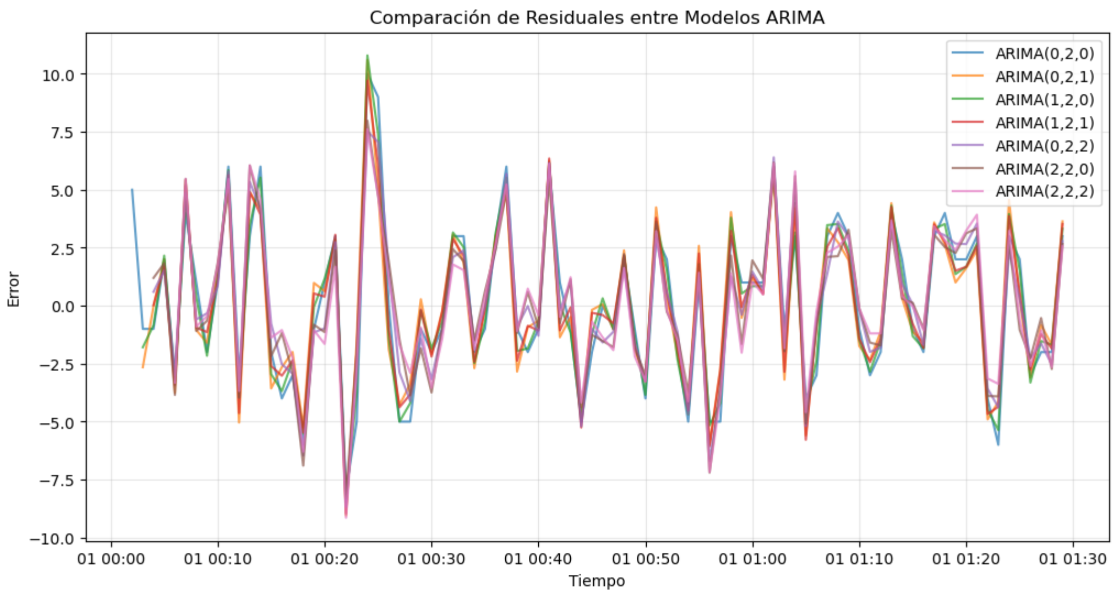
0.1 Estimate Parameters

[10]:
```python
from statsmodels.tsa.arima.model import ARIMA  # Importar el modelo ARIMA

# Seleccionar la serie transformada y eliminar valores nulos
data = time_series_df['T_points'].dropna()

# Definir los modelos ARIMA sugeridos
suggested_models = np.array([
    [0, d, 0],  # Modelo ARIMA(0,d,0)
    [0, d, 1],  # Modelo ARIMA(0,d,1)
    [1, d, 0],  # Modelo ARIMA(1,d,0)
    [1, d, 1],  # Modelo ARIMA(1,d,1)
    [0, d, 2],  # Modelo ARIMA(0,d,2)
    [2, d, 0],  # Modelo ARIMA(2,d,0)
    [2, d, 2],  # Modelo ARIMA(2,d,2)
])

# Inicializar listas para almacenar resultados y modelos ajustados
results = []  # Lista para almacenar métricas de ajuste (AIC, BIC)
error_dfs = []  # Lista para almacenar los residuales de cada modelo
fitted_models = []  # Lista para almacenar los modelos ajustados

# Iterar sobre los modelos sugeridos y ajustar cada uno
for p, d, q in suggested_models:
    # Ajustar el modelo ARIMA con los parámetros actuales (p, d, q)
    train_data = data[:train_fraction]
    model = ARIMA(train_data, order=(p, d, q))
    fitted = model.fit()

    # Guardar las métricas de rendimiento (AIC y BIC)
    results.append({
        'order': f"ARIMA({p},{d},{q})",
        'aic': fitted.aic,
        'bic': fitted.bic,
        'p': p,
        'd': d,
        'q': q
    })

    # Imprimir los resultados del modelo
    print(f"\nARIMA({p},{d},{q}):")
    print(f"AIC: {fitted.aic:.2f}")
    print(f"BIC: {fitted.bic:.2f}")
```

```
    # Obtener los residuales del modelo ajustado
    residuals = pd.DataFrame(fitted.resid)[p+d+q:]  # Excluir los primeros␣
 ↪valores que dependen de datos iniciales
    residuals.columns = [f'ARIMA({p},{d},{q})']  # Etiquetar los residuales con␣
 ↪el nombre del modelo
    error_dfs.append(residuals)

    # Almacenar el modelo ajustado
    fitted_models.append(fitted)

# Combinar todos los residuales en un único DataFrame
all_errors = pd.concat(error_dfs, axis=1)

# Crear un DataFrame para los modelos ajustados
fitted_models_df = pd.DataFrame({
    'model_order': [f"ARIMA({p},{d},{q})" for p, d, q in suggested_models],
    'fitted_model': fitted_models
})

# Graficar los residuales de los modelos ajustados
plt.figure(figsize=(12, 6))
for column in all_errors.columns:
    plt.plot(all_errors.index, all_errors[column], label=column, alpha=0.7)
plt.legend()
plt.title('Comparación de Residuales entre Modelos ARIMA')
plt.xlabel('Tiempo')
plt.ylabel('Error')
plt.grid(True, alpha=0.3)
plt.show()
```

```
ARIMA(0,2,0):
AIC: 473.92
BIC: 476.39

ARIMA(0,2,1):
AIC: 469.59
BIC: 474.55

ARIMA(1,2,0):
AIC: 473.72
BIC: 478.67

ARIMA(1,2,1):
AIC: 469.42
BIC: 476.85
```

```
ARIMA(0,2,2):
AIC: 464.44
BIC: 471.87

ARIMA(2,2,0):
AIC: 461.10
BIC: 468.53

ARIMA(2,2,2):
AIC: 464.71
BIC: 477.10
```



0.2 Verify 8 supuestos

0.2.1 Tests on residuals, Supuestos 1-4

[11]:
```python
import statsmodels.api as sm  # Importar herramientas para modelos estadísticos
from scipy.stats import shapiro, jarque_bera, ttest_1samp  # Pruebas de␣
 ↪normalidad y t-test
from statsmodels.stats.diagnostic import het_breuschpagan, acorr_ljungbox  #␣
 ↪Pruebas de homocedasticidad e independencia

# Inicializar una lista para almacenar los resultados de las pruebas de␣
 ↪residuales
residuals_tests = []

# Función para evaluar los residuales de un modelo
```

```python
def test_residuals(residuals, model_name, alpha=0.05):
    """
    Realiza pruebas estadísticas en los residuales de un modelo y guarda los␣
 ↪resultados.

    Parámetros:
    -----------
    residuals : array-like
        Residuales del modelo a evaluar.
    model_name : str
        Nombre del modelo para etiquetar los resultados.
    alpha : float, opcional
        Nivel de significancia para las pruebas (por defecto 0.05).
    """
    results = {}

    # 1. Prueba de media cercana a 0 (t-test), (we assume normality which is␣
 ↪evaluated in another test (eat), assume iid)
    p_value = ttest_1samp(residuals, 0).pvalue
    results['mean_close_to_0'] = p_value > alpha  # La media está cerca de 0 si␣
 ↪p-valor > alpha
    print(f"{model_name}: Media de residuales = {np.mean(residuals):.4f},␣
 ↪p-valor = {p_value:.4f}")

    # 2. Prueba de varianza constante (homocedasticidad) usando Breusch-Pagan␣
 ↪(assume independent samples, is eat)
    _, pvalue, _, _ = het_breuschpagan(residuals, sm.add_constant(np.
 ↪arange(len(residuals))))
    results['constant_variance'] = pvalue > alpha  # Pasa si p-valor > alpha
    print(f"{model_name}: Homocedasticidad (p-valor de Breusch-Pagan) = {pvalue:
 ↪.4f}")

    # 3. Pruebas de normalidad (Shapiro-Wilk y Jarque-Bera) (assumes␣
 ↪independent observations, )
    _, shapiro_pvalue = shapiro(residuals)  # Prueba Shapiro-Wilk
    jb_stat, jb_pvalue = jarque_bera(residuals)  # Prueba Jarque-Bera
    results['normal_distribution'] = shapiro_pvalue > alpha and jb_pvalue >␣
 ↪alpha  # Ambas deben pasar
    print(f"{model_name}: Normalidad (p-valor Shapiro-Wilk) = {shapiro_pvalue:.
 ↪4f}")
    print(f"{model_name}: Normalidad (p-valor Jarque-Bera) = {jb_pvalue:.4f}")

    # 4. Prueba de independencia usando Ljung-Box (assumes identical␣
 ↪distribution, actually tests
    # uncorrelatedness)
    lb_test = acorr_ljungbox(residuals, lags=[10], return_df=True)
```

```python
    pvalue_ljungbox = lb_test['lb_pvalue'].values[0]
    results['independent_errors'] = pvalue_ljungbox > alpha  # Pasa si p-valor␣
 ↪> alpha
    print(f"{model_name}: Independencia (p-valor de Ljung-Box) =␣
 ↪{pvalue_ljungbox:.4f}")

    # Guardar los resultados en la lista
    residuals_tests.append({'model': model_name, 'results': results})

    print("-" * 40)


# Iterar sobre cada columna en el DataFrame `all_errors` (que contiene los␣
 ↪residuales de los modelos)
for column in all_errors.columns:
    print(f"Analizando modelo: {column}")
    residuals = all_errors[column].dropna()  # Eliminar valores NaN si existen
    test_residuals(residuals, column)  # Aplicar las pruebas a los residuales


# Resumen de los resultados de las pruebas de residuales
print("\nResumen de las Pruebas de Suposiciones de Residuales:")
for result in residuals_tests:
    model_name = result['model']
    tests = result['results']
    print(f"{model_name}:")
    print(f"  Media cercana a 0: {'Pasa' if tests['mean_close_to_0'] else 'No␣
 ↪pasa'}")
    print(f"  Varianza constante: {'Pasa' if tests['constant_variance'] else␣
 ↪'No pasa'}")
    print(f"  Distribución normal: {'Pasa' if tests['normal_distribution'] else␣
 ↪'No pasa'}")
    print(f"  Errores independientes: {'Pasa' if tests['independent_errors']␣
 ↪else 'No pasa'}")
    print("-" * 40)
```

```
Analizando modelo: ARIMA(0,2,0)
ARIMA(0,2,0): Media de residuales = 0.1023, p-valor = 0.7878
ARIMA(0,2,0): Homocedasticidad (p-valor de Breusch-Pagan) = 0.1193
ARIMA(0,2,0): Normalidad (p-valor Shapiro-Wilk) = 0.1051
ARIMA(0,2,0): Normalidad (p-valor Jarque-Bera) = 0.5914
ARIMA(0,2,0): Independencia (p-valor de Ljung-Box) = 0.0242
----------------------------------------
Analizando modelo: ARIMA(0,2,1)
ARIMA(0,2,1): Media de residuales = 0.0309, p-valor = 0.9328
ARIMA(0,2,1): Homocedasticidad (p-valor de Breusch-Pagan) = 0.1529
ARIMA(0,2,1): Normalidad (p-valor Shapiro-Wilk) = 0.6090
```

```
ARIMA(0,2,1): Normalidad (p-valor Jarque-Bera) = 0.7061
ARIMA(0,2,1): Independencia (p-valor de Ljung-Box) = 0.2344
----------------------------------------
Analizando modelo: ARIMA(1,2,0)
ARIMA(1,2,0): Media de residuales = 0.0350, p-valor = 0.9256
ARIMA(1,2,0): Homocedasticidad (p-valor de Breusch-Pagan) = 0.1463
ARIMA(1,2,0): Normalidad (p-valor Shapiro-Wilk) = 0.3401
ARIMA(1,2,0): Normalidad (p-valor Jarque-Bera) = 0.5081
ARIMA(1,2,0): Independencia (p-valor de Ljung-Box) = 0.0314
----------------------------------------
Analizando modelo: ARIMA(1,2,1)
ARIMA(1,2,1): Media de residuales = 0.0634, p-valor = 0.8621
ARIMA(1,2,1): Homocedasticidad (p-valor de Breusch-Pagan) = 0.1371
ARIMA(1,2,1): Normalidad (p-valor Shapiro-Wilk) = 0.9337
ARIMA(1,2,1): Normalidad (p-valor Jarque-Bera) = 0.8782
ARIMA(1,2,1): Independencia (p-valor de Ljung-Box) = 0.2585
----------------------------------------
Analizando modelo: ARIMA(0,2,2)
ARIMA(0,2,2): Media de residuales = 0.0943, p-valor = 0.7906
ARIMA(0,2,2): Homocedasticidad (p-valor de Breusch-Pagan) = 0.1371
ARIMA(0,2,2): Normalidad (p-valor Shapiro-Wilk) = 0.8997
ARIMA(0,2,2): Normalidad (p-valor Jarque-Bera) = 0.9479
ARIMA(0,2,2): Independencia (p-valor de Ljung-Box) = 0.5100
----------------------------------------
Analizando modelo: ARIMA(2,2,0)
ARIMA(2,2,0): Media de residuales = 0.0922, p-valor = 0.7911
ARIMA(2,2,0): Homocedasticidad (p-valor de Breusch-Pagan) = 0.0995
ARIMA(2,2,0): Normalidad (p-valor Shapiro-Wilk) = 0.4672
ARIMA(2,2,0): Normalidad (p-valor Jarque-Bera) = 0.9459
ARIMA(2,2,0): Independencia (p-valor de Ljung-Box) = 0.8563
----------------------------------------
Analizando modelo: ARIMA(2,2,2)
ARIMA(2,2,2): Media de residuales = 0.1001, p-valor = 0.7776
ARIMA(2,2,2): Homocedasticidad (p-valor de Breusch-Pagan) = 0.0871
ARIMA(2,2,2): Normalidad (p-valor Shapiro-Wilk) = 0.4625
ARIMA(2,2,2): Normalidad (p-valor Jarque-Bera) = 0.9966
ARIMA(2,2,2): Independencia (p-valor de Ljung-Box) = 0.8741
----------------------------------------

Resumen de las Pruebas de Suposiciones de Residuales:
ARIMA(0,2,0):
  Media cercana a 0: Pasa
  Varianza constante: Pasa
  Distribución normal: Pasa
  Errores independientes: No pasa
----------------------------------------
ARIMA(0,2,1):
  Media cercana a 0: Pasa
```

```
  Varianza constante: Pasa
  Distribución normal: Pasa
  Errores independientes: Pasa
----------------------------------------
ARIMA(1,2,0):
  Media cercana a 0: Pasa
  Varianza constante: Pasa
  Distribución normal: Pasa
  Errores independientes: No pasa
----------------------------------------
ARIMA(1,2,1):
  Media cercana a 0: Pasa
  Varianza constante: Pasa
  Distribución normal: Pasa
  Errores independientes: Pasa
----------------------------------------
ARIMA(0,2,2):
  Media cercana a 0: Pasa
  Varianza constante: Pasa
  Distribución normal: Pasa
  Errores independientes: Pasa
----------------------------------------
ARIMA(2,2,0):
  Media cercana a 0: Pasa
  Varianza constante: Pasa
  Distribución normal: Pasa
  Errores independientes: Pasa
----------------------------------------
ARIMA(2,2,2):
  Media cercana a 0: Pasa
  Varianza constante: Pasa
  Distribución normal: Pasa
  Errores independientes: Pasa
----------------------------------------
```

0.2.2 Supuesto 5: Modelo Parsimonioso

[12]:
```
# Inicializar listas para almacenar modelos que aprueban o fallan la prueba de␣
↪significancia
models_pass = []  # Modelos cuyos intervalos de confianza NO contienen cero
models_fail = []  # Modelos cuyos intervalos de confianza contienen cero

# Iterar sobre cada modelo ajustado en el DataFrame `fitted_models_df`
for index, row in fitted_models_df.iterrows():
    # Obtener el orden del modelo y el objeto del modelo ajustado
    model_order = row['model_order']
    fitted_model = row['fitted_model']
```

```
    # Obtener los intervalos de confianza para los parámetros del modelo
    conf_intervals = fitted_model.conf_int()
    parameters = fitted_model.params  # Obtener los valores estimados de los␣
 ↪parámetros

    # Imprimir los parámetros del modelo
    print(f"\nParámetros para {model_order}:")
    print(parameters)

    # Imprimir los intervalos de confianza de los parámetros
    print(f"\nIntervalos de Confianza para {model_order}:")
    print(conf_intervals)
    print("-" * 50)

    # Verificar si los intervalos de confianza NO contienen cero
    intervals_do_not_contain_zero = (conf_intervals[0] > 0) |␣
 ↪(conf_intervals[1] < 0)

    if intervals_do_not_contain_zero.all():
        # Si TODOS los intervalos de confianza no contienen cero
        models_pass.append(model_order)
    else:
        # Si ALGÚN intervalo de confianza contiene cero
        models_fail.append(model_order)

# Imprimir el resumen de resultados
print("\nResumen de Significancia de Parámetros de Modelos ARIMA Basado en␣
 ↪Intervalos de Confianza:")
print("==================================================")
print("Modelos que Aprobaron (Intervalos de Confianza NO contienen cero):")
print(models_pass)
print("\nModelos que Fallaron (Intervalos de Confianza contienen cero):")
print(models_fail)
```

```
Parámetros para ARIMA(0,2,0):
sigma2    12.488598
dtype: float64

Intervalos de Confianza para ARIMA(0,2,0):
               0          1
sigma2  8.620549  16.356647
--------------------------------------------------

Parámetros para ARIMA(0,2,1):
ma.L1      0.378011
```

```
sigma2    11.601834
dtype: float64

Intervalos de Confianza para ARIMA(0,2,1):
               0          1
ma.L1   0.177472   0.578550
sigma2  8.183782  15.019886
--------------------------------------------------

Parámetros para ARIMA(1,2,0):
ar.L1      0.158697
sigma2    12.176288
dtype: float64

Intervalos de Confianza para ARIMA(1,2,0):
               0          1
ar.L1  -0.025871   0.343264
sigma2  8.448016  15.904560
--------------------------------------------------

Parámetros para ARIMA(1,2,1):
ar.L1     -0.293201
ma.L1      0.601858
sigma2    11.318881
dtype: float64

Intervalos de Confianza para ARIMA(1,2,1):
               0          1
ar.L1  -0.746875   0.160473
ma.L1   0.227128   0.976589
sigma2  7.868230  14.769532
--------------------------------------------------

Parámetros para ARIMA(0,2,2):
ma.L1      0.126651
ma.L2     -0.328996
sigma2    10.682678
dtype: float64

Intervalos de Confianza para ARIMA(0,2,2):
               0          1
ma.L1  -0.086842   0.340144
ma.L2  -0.516662  -0.141331
sigma2  7.320527  14.044828
--------------------------------------------------

Parámetros para ARIMA(2,2,0):
ar.L1      0.226779
```

```
ar.L2     -0.394230
sigma2    10.273513
dtype: float64

Intervalos de Confianza para ARIMA(2,2,0):
               0          1
ar.L1   0.025078   0.428480
ar.L2  -0.574273  -0.214187
sigma2  6.970457  13.576570
--------------------------------------------------

Parámetros para ARIMA(2,2,2):
ar.L1      0.445028
ar.L2     -0.203014
ma.L1     -0.245049
ma.L2     -0.256755
sigma2    10.222930
dtype: float64

Intervalos de Confianza para ARIMA(2,2,2):
               0          1
ar.L1  -0.054274   0.944331
ar.L2  -0.699500   0.293471
ma.L1  -0.737975   0.247877
ma.L2  -0.706690   0.193180
sigma2  7.065923  13.379938
--------------------------------------------------

Resumen de Significancia de Parámetros de Modelos ARIMA Basado en Intervalos de
Confianza:
==================================================
Modelos que Aprobaron (Intervalos de Confianza NO contienen cero):
['ARIMA(0,2,0)', 'ARIMA(0,2,1)', 'ARIMA(2,2,0)']

Modelos que Fallaron (Intervalos de Confianza contienen cero):
['ARIMA(1,2,0)', 'ARIMA(1,2,1)', 'ARIMA(0,2,2)', 'ARIMA(2,2,2)']
```

0.2.3 Supuesto 6, modelos crean seria estationario y invertible

[13]:
```
# Inicializar listas para almacenar los modelos que pasan o fallan la prueba de␣
↪estacionariedad
admisible_models = []  # Modelos que cumplen con raíces fuera del círculo␣
↪unitario
non_admisible_models = []  # Modelos que no cumplen con raíces fuera del␣
↪círculo unitario

# Iterar sobre cada modelo ajustado en el DataFrame `fitted_models_df`
```

```python
for index, row in fitted_models_df.iterrows():
    # Obtener el orden del modelo y el objeto del modelo ajustado
    model_order = row['model_order']
    fitted_model = row['fitted_model']

    # Obtener los parámetros MA (Media Móvil) del modelo
    ma_params = [param for param in fitted_model.params.index if 'ma.L' in␣
 ↪param]
    ma_coefficients = [fitted_model.params[param] for param in ma_params]  #␣
 ↪Extraer los valores de los coeficientes MA

    # Calcular las raíces del polinomio MA: 1 - p1*x - p2*x^2 - ...
    ma_roots = np.roots(([1] + [-coeff for coeff in ma_coefficients])[::-1])  #␣
 ↪Negar los coeficientes para el cálculo

    # Obtener los parámetros AR (Autoregresivo) del modelo
    ar_params = [param for param in fitted_model.params.index if 'ar.L' in␣
 ↪param]
    ar_coefficients = [fitted_model.params[param] for param in ar_params]  #␣
 ↪Extraer los valores de los coeficientes AR

    # Calcular las raíces del polinomio AR: 1 - p1*x - p2*x^2 - ...
    ar_roots = np.roots(([1] + [-coeff for coeff in ar_coefficients])[::-1])  #␣
 ↪Negar los coeficientes para el cálculo

    # Verificar si todas las raíces están fuera del círculo unitario (valor␣
 ↪absoluto > 1)
    if np.all(np.abs(ma_roots) > 1) and np.all(np.abs(ar_roots) > 1):
        admisible_models.append(model_order)  # Agregar el modelo a la lista de␣
 ↪modelos admisibles
    else:
        non_admisible_models.append(model_order)  # Agregar el modelo a la␣
 ↪lista de modelos no admisibles

# Imprimir los resultados de la prueba
print("\nResumen de la Prueba de Admisibilidad Basada en Raíces de los␣
 ↪Polinomios MA y AR:")
print("====================================================")
print("Modelos que Pasaron (Modelos Admisibles con Raíces Fuera del Círculo␣
 ↪Unitario):")
print(admisible_models)
print("\nModelos que Fallaron (Modelos No Admisibles con Raíces Dentro o en el␣
 ↪Círculo Unitario):")
print(non_admisible_models)
```

```
Resumen de la Prueba de Admisibilidad Basada en Raíces de los Polinomios MA y
```

```
AR:
==================================================
Modelos que Pasaron (Modelos Admisibles con Raíces Fuera del Círculo Unitario):
['ARIMA(0,2,0)', 'ARIMA(0,2,1)', 'ARIMA(1,2,0)', 'ARIMA(1,2,1)', 'ARIMA(0,2,2)',
'ARIMA(2,2,0)', 'ARIMA(2,2,2)']

Modelos que Fallaron (Modelos No Admisibles con Raíces Dentro o en el Círculo
Unitario):
[]
```

0.2.4 Supuesto 7,

verifica si las correlaciones entre los parámetros estimados son pequeñas.

[14]:
```python
from statsmodels.stats.moment_helpers import cov2corr  # Importar función para␣
 ↪convertir matriz de covarianza a correlación

# Iterar sobre cada modelo ARIMA en el DataFrame `fitted_models_df`
for idx, row in fitted_models_df.iterrows():
    model_name = row['model_order']  # Nombre del modelo (orden ARIMA)
    fitted_model = row['fitted_model']  # Objeto del modelo ajustado

    # Extraer la matriz de varianza-covarianza de los parámetros estimados
    cov_matrix = fitted_model.cov_params()  # Matriz de covarianza de los␣
 ↪parámetros

    # Convertir la matriz de covarianza a una matriz de correlación
    correlation_matrix = cov2corr(cov_matrix)

    # Crear un DataFrame para hacer la matriz de correlación más legible
    correlation_df = pd.DataFrame(correlation_matrix,
                                  index=fitted_model.param_names,  # Usar␣
 ↪nombres de parámetros como índices
                                  columns=fitted_model.param_names)  # Usar␣
 ↪nombres de parámetros como columnas

    # Mostrar la matriz de correlación
    print(f"\nMatriz de Correlación de Parámetros para {model_name}:")
    print(correlation_df)
```

```
Matriz de Correlación de Parámetros para ARIMA(0,2,0):
        sigma2
sigma2     1.0

Matriz de Correlación de Parámetros para ARIMA(0,2,1):
           ma.L1    sigma2
ma.L1   1.000000  0.080977
```

```
sigma2  0.080977  1.000000

Matriz de Correlación de Parámetros para ARIMA(1,2,0):
           ar.L1    sigma2
ar.L1   1.000000  0.136053
sigma2  0.136053  1.000000

Matriz de Correlación de Parámetros para ARIMA(1,2,1):
           ar.L1     ma.L1    sigma2
ar.L1   1.000000 -0.898173  0.106209
ma.L1  -0.898173  1.000000 -0.063483
sigma2  0.106209 -0.063483  1.000000

Matriz de Correlación de Parámetros para ARIMA(0,2,2):
           ma.L1     ma.L2    sigma2
ma.L1   1.000000  0.267239  0.123591
ma.L2   0.267239  1.000000 -0.096439
sigma2  0.123591 -0.096439  1.000000

Matriz de Correlación de Parámetros para ARIMA(2,2,0):
           ar.L1     ar.L2    sigma2
ar.L1   1.000000 -0.157797  0.236934
ar.L2  -0.157797  1.000000 -0.019350
sigma2  0.236934 -0.019350  1.000000

Matriz de Correlación de Parámetros para ARIMA(2,2,2):
           ar.L1     ar.L2     ma.L1     ma.L2    sigma2
ar.L1   1.000000 -0.323665 -0.899500  0.254358  0.174623
ar.L2  -0.323665  1.000000  0.305872 -0.919857  0.017758
ma.L1  -0.899500  0.305872  1.000000 -0.296871 -0.093611
ma.L2   0.254358 -0.919857 -0.296871  1.000000 -0.098382
sigma2  0.174623  0.017758 -0.093611 -0.098382  1.000000
```

0.2.5 Supuesto 8

Verificar si los valores atípicos tienen influencia en el resultado. No es necesario ya que no tenemos valores atípicos.

0.3 Calcular la previsión y los intervalos de confianza

Tomamos el modelo ARIMA(2,2,0) para la predicción, ya que es el único que pasó los 8 supuestos (también tuvo correlaciones pequeñas entre los coeficientes).

[15]:
```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy import special
```

```python
# Definir el nivel de confianza para los intervalos
confidence_level = 0.95  # Nivel de confianza del 95%

# Lista de modelos que se usarán para los pronósticos
models_to_forecast = ['ARIMA(2,2,0)', 'ARIMA(0,2,0)']

def forecast(data, models_to_forecast, start, end, lmbda=None):
    """
    Realiza pronósticos para los modelos especificados y compara con los datos␣
 ↪originales.

    Parámetros:
    -----------
    data : pandas.Series
        Serie temporal original.
    models_to_forecast : list
        Lista de nombres de los modelos ARIMA que se utilizarán para el␣
 ↪pronóstico.
    start : int
        Índice inicial del período de pronóstico.
    end : int
        Índice final del período de pronóstico.
    lmbda : float, opcional
        Parámetro de transformación Box-Cox (None si no se aplica).

    Retorno:
    --------
    None. Imprime resultados y genera gráficos.
    """
    forecast_results = {}  # Diccionario para almacenar los resultados de los␣
 ↪pronósticos

    # Recuperar los datos originales para graficar
    data_to_plot = data[start - (end - start) * 2:]  # Incluye un período␣
 ↪adicional para contexto

    # Crear un rango de fechas para los períodos pronosticados
    if start < len(data):
        start_date = data.index[start]  # Fecha inicial dentro de los datos
        forecast_index = pd.date_range(start=start_date, periods=np.abs(end -␣
 ↪start) + 1, freq=freq)
    else:
        last_date = data.index[-1]  # Fecha final en los datos originales
        overlap = start - len(data) + 1
        forecast_index = pd.date_range(start=last_date, periods=np.abs(end -␣
 ↪start) + overlap + 1, freq=freq)[overlap:]
```

```python
    # Iterar sobre cada modelo para realizar pronósticos
    for model_name in models_to_forecast:
        # Obtener el modelo ajustado del DataFrame
        fitted_model = fitted_models_df.loc[fitted_models_df['model_order'] == model_name, 'fitted_model'].iloc[0]

        # Realizar pronósticos
        forecast = fitted_model.get_prediction(start=start, end=end)
        forecast_mean = forecast.predicted_mean  # Valores pronosticados
        forecast_ci = forecast.conf_int(alpha=1 - confidence_level)  # Intervalos de confianza ajustados

        # Aplicar la transformación inversa de Box-Cox si corresponde
        if lmbda is not None:
            var = np.var(fitted_model.resid)
            correction_factor = np.exp(var / 2) if lmbda == 0 else (0.5 + np.sqrt(1 - 2 * lmbda * (lmbda - 1) *
                (1 + lmbda * forecast_mean) ** (-1) * var) / 2) ** (1 / lmbda)
            forecast_mean = special.inv_boxcox(forecast_mean, lmbda) * correction_factor
            forecast_ci['lower T_points'] = forecast_ci['lower T_points'].apply(lambda x: special.inv_boxcox(x, lmbda)) * correction_factor
            forecast_ci['upper T_points'] = forecast_ci['upper T_points'].apply(lambda x: special.inv_boxcox(x, lmbda)) * correction_factor

        # Almacenar los resultados del pronóstico
        forecast_results[model_name] = {'forecast_mean': forecast_mean, 'forecast_ci': forecast_ci}

        # Imprimir los resultados del pronóstico
        print(f"\nPronóstico para {model_name} para los próximos {end - start} períodos con Intervalo de Confianza del {confidence_level * 100:.1f}%:")
        print(forecast_mean)
        print("\nIntervalos de Confianza:")
        print(forecast_ci)

        # Comparar con los datos originales
        print("\nComparación del Pronóstico con los Datos Originales:")
        print("Fecha\t\tPronóstico\tDatos Originales\tDiferencia")
        for date, forecast_value in zip(forecast_index, forecast_mean):
            if date in data.index:
                original_value = data.loc[date]
                difference = original_value - forecast_value
                print(f"{date.strftime('%Y-%m-%d')}\t{forecast_value:.2f}\t\t{original_value:.2f}\t\t{difference:.2f}")
```

```
            else:
                print(f"{date.strftime('%Y-%m-%d')}\t{forecast_value:.
↪2f}\t\tDatos No Disponibles")
        print("-" * 40)

    # Generar gráficos de los pronósticos
    plt.figure(figsize=(10, 6))
    plt.plot(data_to_plot.index, data_to_plot, label='Datos Originales',␣
↪color='blue')

    colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
    for i, (model_name, result) in enumerate(forecast_results.items()):
        forecast_mean = result['forecast_mean']
        forecast_ci = result['forecast_ci']
        plt.plot(forecast_index, forecast_mean, label=f'Pronóstico -␣
↪{model_name}', color=colors[i], marker='o')
        plt.fill_between(forecast_index, forecast_ci.iloc[:, 0], forecast_ci.
↪iloc[:, 1], color=colors[i], alpha=0.2)

    plt.title(f'Comparación de Pronósticos para {models_to_forecast} con␣
↪Intervalo de Confianza del {confidence_level * 100:.1f}%')
    plt.xlabel('Tiempo')
    plt.ylabel('Accesos al Servidor')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()
```

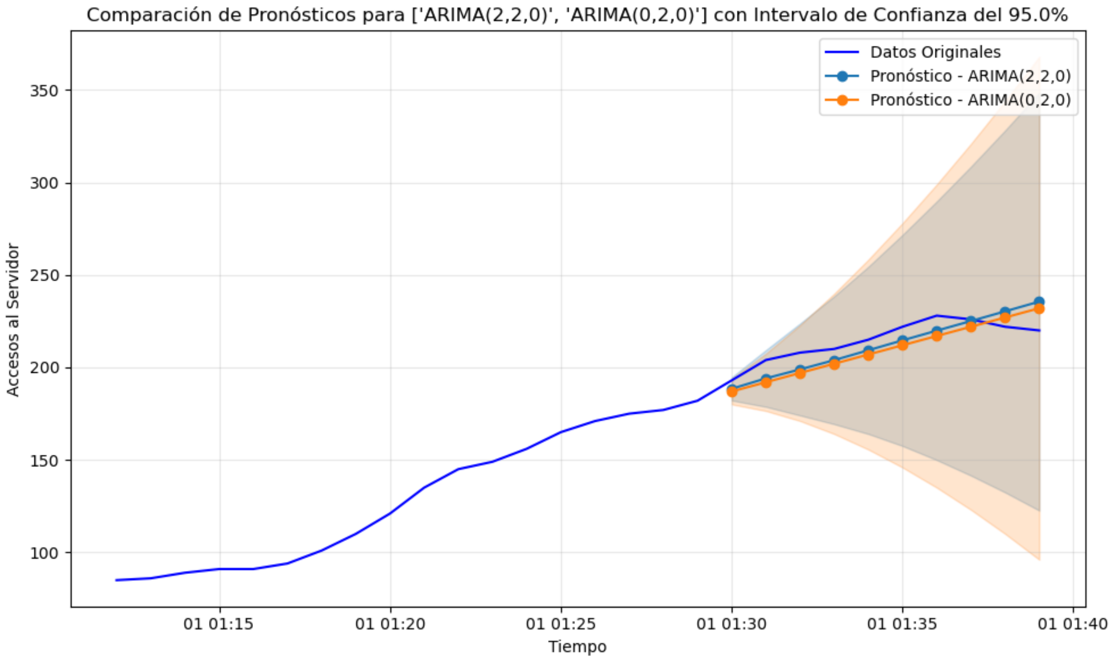
0.3.1 Predecir los valores dentro de la muestra

[16]:
```
# Definir el número de períodos que deseas pronosticar
num_predictions = 10  # Número de períodos a pronosticar

# Determinar el índice del último dato disponible en la serie temporal
end = len(time_series_df['points']) - 1  # Último índice de los datos␣
↪disponibles

# Calcular el índice inicial del período a pronosticar
start = end - (num_predictions - 1)  # Comenzar desde los últimos␣
↪`num_predictions` períodos

# Llamar a la función forecast para realizar los pronósticos
forecast(
    time_series_df['points'],  # Serie temporal original
    models_to_forecast,        # Lista de modelos ARIMA a utilizar para los␣
↪pronósticos
    start=start,               # Índice inicial del período de pronóstico
```

```
    end=end,                    # Índice final del período de pronóstico
    lmbda=lmbda                 # Parámetro de tran
)
```

```
Pronóstico para ARIMA(2,2,0) para los próximos 9 períodos con Intervalo de
Confianza del 95.0%:
2023-01-01 01:30:00    188.468796
2023-01-01 01:31:00    194.087994
2023-01-01 01:32:00    198.935478
2023-01-01 01:33:00    203.942890
2023-01-01 01:34:00    209.290804
2023-01-01 01:35:00    214.652887
2023-01-01 01:36:00    219.883949
2023-01-01 01:37:00    225.079710
2023-01-01 01:38:00    230.319120
2023-01-01 01:39:00    235.582344
Freq: min, Name: predicted_mean, dtype: float64

Intervalos de Confianza:
                     lower T_points  upper T_points
2023-01-01 01:30:00      182.186656      194.750936
2023-01-01 01:31:00      178.753211      209.422777
2023-01-01 01:32:00      174.094920      223.776036
2023-01-01 01:33:00      169.370685      238.515096
2023-01-01 01:34:00      164.063398      254.518210
2023-01-01 01:35:00      157.570662      271.735112
2023-01-01 01:36:00      149.964698      289.803199
2023-01-01 01:37:00      141.558715      308.600706
2023-01-01 01:38:00      132.467819      328.170421
2023-01-01 01:39:00      122.654174      348.510514

Comparación del Pronóstico con los Datos Originales:
Fecha           Pronóstico      Datos Originales        Diferencia
2023-01-01      188.47          193.00          4.53
2023-01-01      194.09          204.00          9.91
2023-01-01      198.94          208.00          9.06
2023-01-01      203.94          210.00          6.06
2023-01-01      209.29          215.00          5.71
2023-01-01      214.65          222.00          7.35
2023-01-01      219.88          228.00          8.12
2023-01-01      225.08          226.00          0.92
2023-01-01      230.32          222.00          -8.32
2023-01-01      235.58          220.00          -15.58
----------------------------------------

Pronóstico para ARIMA(0,2,0) para los próximos 9 períodos con Intervalo de
Confianza del 95.0%:
```

```
2023-01-01 01:30:00    187.0
2023-01-01 01:31:00    192.0
2023-01-01 01:32:00    197.0
2023-01-01 01:33:00    202.0
2023-01-01 01:34:00    207.0
2023-01-01 01:35:00    212.0
2023-01-01 01:36:00    217.0
2023-01-01 01:37:00    222.0
2023-01-01 01:38:00    227.0
2023-01-01 01:39:00    232.0
Freq: min, Name: predicted_mean, dtype: float64

Intervalos de Confianza:
                     lower T_points  upper T_points
2023-01-01 01:30:00      180.073642      193.926358
2023-01-01 01:31:00      176.512193      207.487807
2023-01-01 01:32:00      171.083941      222.916059
2023-01-01 01:33:00      164.062775      239.937225
2023-01-01 01:34:00      155.632754      258.367246
2023-01-01 01:35:00      145.926756      278.073244
2023-01-01 01:36:00      135.046227      298.953773
2023-01-01 01:37:00      123.071820      320.928180
2023-01-01 01:38:00      110.069619      343.930381
2023-01-01 01:39:00       96.095042      367.904958

Comparación del Pronóstico con los Datos Originales:
Fecha           Pronóstico      Datos Originales        Diferencia
2023-01-01      187.00          193.00          6.00
2023-01-01      192.00          204.00          12.00
2023-01-01      197.00          208.00          11.00
2023-01-01      202.00          210.00          8.00
2023-01-01      207.00          215.00          8.00
2023-01-01      212.00          222.00          10.00
2023-01-01      217.00          228.00          11.00
2023-01-01      222.00          226.00          4.00
2023-01-01      227.00          222.00          -5.00
2023-01-01      232.00          220.00          -12.00
----------------------------------------
```



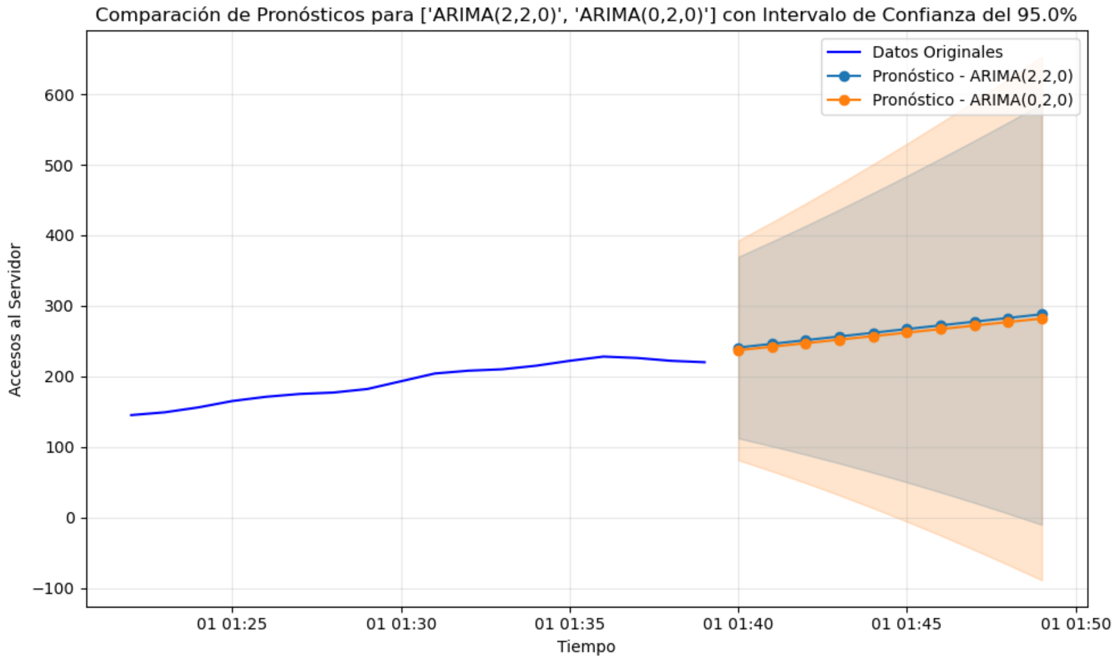
0.3.2 Predecir los valores fuera de la muestra

[17]:
```
# Definir el número de períodos que deseas pronosticar hacia el futuro
num_predictions = 10  # Número de períodos a pronosticar

# Calcular el índice de inicio y final para los pronósticos futuros
start = len(time_series_df['points'])  # Índice inmediatamente después de los␣
↪datos disponibles
end = start + num_predictions - 1  # Índice final de los pronósticos (10␣
↪períodos hacia el futuro)

# Llamar a la función forecast para realizar los pronósticos
forecast(
    time_series_df['points'],  # Serie temporal original
    models_to_forecast,        # Lista de modelos ARIMA a utilizar para los␣
↪pronósticos
    start=start,               # Índice inicial del período de pronóstico␣
↪(futuro)
    end=end,                   # Índice final del período de pronóstico (futuro)
    lmbda=lmbda                # Parámetro de transformación Box-Cox (si se␣
↪aplicó previamente)
)
```

```
Pronóstico para ARIMA(2,2,0) para los próximos 9 períodos con Intervalo de
Confianza del 95.0%:
```

```
2023-01-01 01:40:00    240.833762
2023-01-01 01:41:00    246.073113
2023-01-01 01:42:00    251.314383
2023-01-01 01:43:00    256.560845
2023-01-01 01:44:00    261.807727
2023-01-01 01:45:00    267.052659
2023-01-01 01:46:00    272.296982
2023-01-01 01:47:00    277.541936
2023-01-01 01:48:00    282.787273
2023-01-01 01:49:00    288.032449
Freq: min, Name: predicted_mean, dtype: float64

Intervalos de Confianza:
                     lower T_points  upper T_points
2023-01-01 01:40:00      112.106605      369.560918
2023-01-01 01:41:00      100.873014      391.273212
2023-01-01 01:42:00       89.001530      413.627236
2023-01-01 01:43:00       76.514137      436.607552
2023-01-01 01:44:00       63.423287      460.192168
2023-01-01 01:45:00       49.746598      484.358720
2023-01-01 01:46:00       35.504454      509.089510
2023-01-01 01:47:00       20.714078      534.369794
2023-01-01 01:48:00        5.389285      560.185262
2023-01-01 01:49:00      -10.457077      586.521974

Comparación del Pronóstico con los Datos Originales:
Fecha           Pronóstico      Datos Originales        Diferencia
2023-01-01      240.83          Datos No Disponibles
2023-01-01      246.07          Datos No Disponibles
2023-01-01      251.31          Datos No Disponibles
2023-01-01      256.56          Datos No Disponibles
2023-01-01      261.81          Datos No Disponibles
2023-01-01      267.05          Datos No Disponibles
2023-01-01      272.30          Datos No Disponibles
2023-01-01      277.54          Datos No Disponibles
2023-01-01      282.79          Datos No Disponibles
2023-01-01      288.03          Datos No Disponibles
----------------------------------------

Pronóstico para ARIMA(0,2,0) para los próximos 9 períodos con Intervalo de
Confianza del 95.0%:
2023-01-01 01:40:00    237.0
2023-01-01 01:41:00    242.0
2023-01-01 01:42:00    247.0
2023-01-01 01:43:00    252.0
2023-01-01 01:44:00    257.0
2023-01-01 01:45:00    262.0
2023-01-01 01:46:00    267.0
```

```
2023-01-01 01:47:00    272.0
2023-01-01 01:48:00    277.0
2023-01-01 01:49:00    282.0
Freq: min, Name: predicted_mean, dtype: float64

Intervalos de Confianza:
                     lower T_points  upper T_points
2023-01-01 01:40:00       81.195429      392.804571
2023-01-01 01:41:00       65.411826      418.588174
2023-01-01 01:42:00       48.780267      445.219733
2023-01-01 01:43:00       31.332712      472.667288
2023-01-01 01:44:00       13.097765      500.902235
2023-01-01 01:45:00       -5.898778      529.898778
2023-01-01 01:46:00      -25.633503      559.633503
2023-01-01 01:47:00      -46.085027      590.085027
2023-01-01 01:48:00      -67.233722      621.233722
2023-01-01 01:49:00      -89.061491      653.061491

Comparación del Pronóstico con los Datos Originales:
Fecha           Pronóstico      Datos Originales        Diferencia
2023-01-01      237.00          Datos No Disponibles
2023-01-01      242.00          Datos No Disponibles
2023-01-01      247.00          Datos No Disponibles
2023-01-01      252.00          Datos No Disponibles
2023-01-01      257.00          Datos No Disponibles
2023-01-01      262.00          Datos No Disponibles
2023-01-01      267.00          Datos No Disponibles
2023-01-01      272.00          Datos No Disponibles
2023-01-01      277.00          Datos No Disponibles
2023-01-01      282.00          Datos No Disponibles
----------------------------------------
```


Comparación de Pronósticos para ['ARIMA(2,2,0)', 'ARIMA(0,2,0)'] con Intervalo de Confianza del 95.0%
Datos Originales
Pronóstico - ARIMA(2,2,0)
Pronóstico - ARIMA(0,2,0)
Accesos al Servidor
600
500
400
300
200
100
0
−100
01 01:25
01 01:30
01 01:35
01 01:40
01 01:45
01 01:50
Tiempo