In [119]:
```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima_process import ArmaProcess
import matplotlib.pyplot as plt
import datetime
import io
import datetime
import matplotlib.lines as mlines
import statsmodels.formula.api as smf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from pmdarima.arima import auto_arima
from statsmodels.tsa.arima.model import ARIMA
import scipy.stats as st
```

1

In [2]:
```
df = pd.read_csv("desktop/JPYUSD.csv",parse_dates = True, index_col = 0)
```

In [3]:
```
df
```

Out[3]:

	Japan
1985-01-01	0.003927
1985-02-01	0.003854
1985-03-01	0.003960
1985-04-01	0.003964
1985-05-01	0.003971
...	...
2022-07-01	0.007519
2022-08-01	0.007214
2022-09-01	0.006909
2022-10-01	0.006746
2022-11-01	0.007205

455 rows × 1 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 455 entries, 1985-01-01 to 2022-11-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Japan   455 non-null    float64
dtypes: float64(1)
memory usage: 7.1 KB
```

In [5]:
```
df80 = df.iloc[:int(len(df)*0.8)]
```

In [6]:
```
df80
```

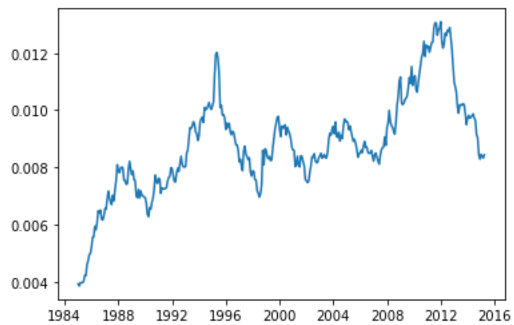
Out[6]:

	Japan
1985-01-01	0.003927
1985-02-01	0.003854
1985-03-01	0.003960
1985-04-01	0.003964
1985-05-01	0.003971
...	...
2014-12-01	0.008289
2015-01-01	0.008459
2015-02-01	0.008385
2015-03-01	0.008326
2015-04-01	0.008438

364 rows × 1 columns

In [8]:
```
plt.plot(df80)
```

Out[8]: [<matplotlib.lines.Line2D at 0x7faf40e98f40>]



In [9]:
```
df80.diff()
```

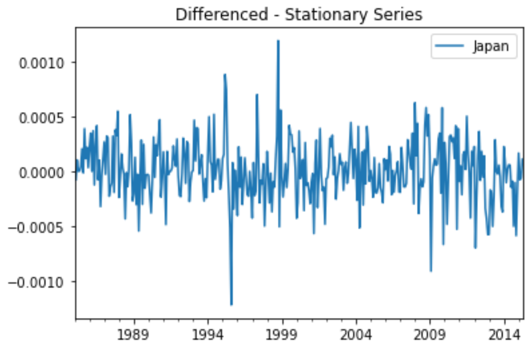
Out[9]:

	Japan
1985-01-01	NaN
1985-02-01	-0.000073
1985-03-01	0.000107
1985-04-01	0.000004
1985-05-01	0.000006
...	...
2014-12-01	-0.000170
2015-01-01	0.000170
2015-02-01	-0.000074
2015-03-01	-0.000059
2015-04-01	0.000112

364 rows × 1 columns

In [10]:
```
df80.diff().plot()
plt.title("Differenced - Stationary Series")
```

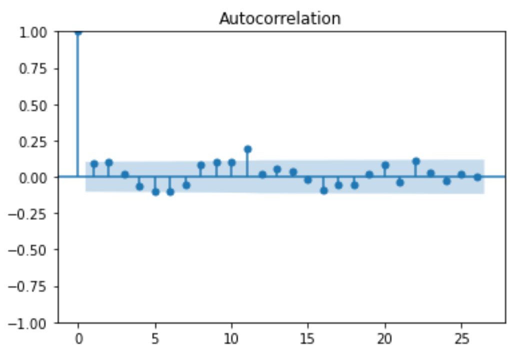
Out[10]: Text(0.5, 1.0, 'Differenced - Stationary Series')

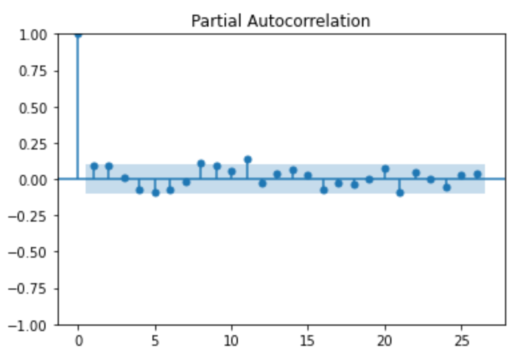


The differenced training data looks stationary as it is mean converting.

In [12]:
```
plot_acf(df80['Japan'].diff().dropna())
plot_pacf(df80['Japan'].diff().dropna(), method='ywm')
plt.plot()
```

Out[12]: []





In [18]:
```
model3 = ARIMA(df80.dropna(), order=(1,1,0)).fit()
model3.summary()
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning:
No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning:
No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning:
No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning: Ma
ximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

Out[18]:

SARIMAX Results

Dep. Variable:	Japan	**No. Observations:**	364
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	2461.065
Date:	Wed, 25 Jan 2023	**AIC**	-4918.130
Time:	17:13:32	**BIC**	-4910.341
Sample:	01-01-1985	**HQIC**	-4915.034
	- 04-01-2015		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0991	0.044	2.267	0.023	0.013	0.185
sigma2	7.537e-08	4.21e-09	17.901	0.000	6.71e-08	8.36e-08

Ljung-Box (L1) (Q):	0.04	**Jarque-Bera (JB):**	41.03
Prob(Q):	0.83	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.49	**Skew:**	-0.09
Prob(H) (two-sided):	0.03	**Kurtosis:**	4.64

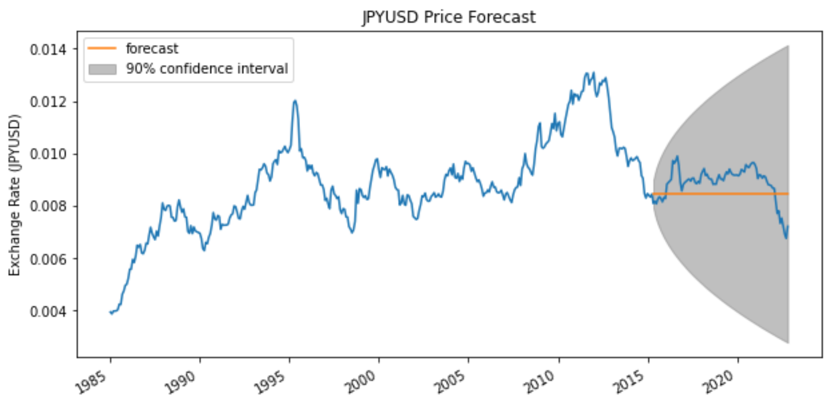
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [19]:
```
forecast2 = model3.get_forecast(12)

# get the 90% confidence interval for the forecast
yhat_conf_int2 = forecast2.conf_int(alpha=0.1)
yhat_conf_int2["mean"] = forecast2.predicted_mean
yhat_conf_int2
```

Out[19]:

	lower Japan	upper Japan	mean
2015-05-01	0.007998	0.008901	0.008449
2015-06-01	0.007779	0.009121	0.008450
2015-07-01	0.007613	0.009288	0.008450
2015-08-01	0.007475	0.009426	0.008450
2015-09-01	0.007353	0.009548	0.008450
2015-10-01	0.007244	0.009657	0.008450
2015-11-01	0.007144	0.009757	0.008450
2015-12-01	0.007051	0.009850	0.008450
2016-01-01	0.006964	0.009937	0.008450
2016-02-01	0.006882	0.010019	0.008450
2016-03-01	0.006804	0.010097	0.008450
2016-04-01	0.006729	0.010172	0.008450

In [20]:
```
from statsmodels.graphics.tsaplots import plot_predict

# Plot the data and the forecast
fig, ax = plt.subplots(figsize = (10, 5))
plt.title("JPYUSD Price Forecast")
plt.plot(df['Japan'])
plt.ylabel("Exchange Rate (JPYUSD)")
plot_predict(model3, ax=ax, start = "2015-05-01", end = "2022-11-01", alpha=0.1)
plt.legend()
plt.show()
```



In [31]:
```
def evaluate_arima_model(X, arima_order):
    # prepare training dataset
    train_size = int(len(X) * 0.8)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    # make predictions
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit()
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    # calculate out of sample error
    test = pd.DataFrame(test)
    test["predictions"] = predictions
    return test
y = evaluate_arima_model(df['Japan'], (1,1,0))
```

```
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning:
Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning:
Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning:
Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning:
Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning:
Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning:
Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
/Users/youssefmahmoud/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning:
```

In [32]:
```
y
```

Out[32]:

	Japan	predictions
2015-05-01	0.008081	0.008449
2015-06-01	0.008167	0.008046
2015-07-01	0.008066	0.008175
2015-08-01	0.008251	0.008057
2015-09-01	0.008337	0.008268
...	...	...
2022-07-01	0.007519	0.007272
2022-08-01	0.007214	0.007538
2022-09-01	0.006909	0.007186
2022-10-01	0.006746	0.006880
2022-11-01	0.007205	0.006730

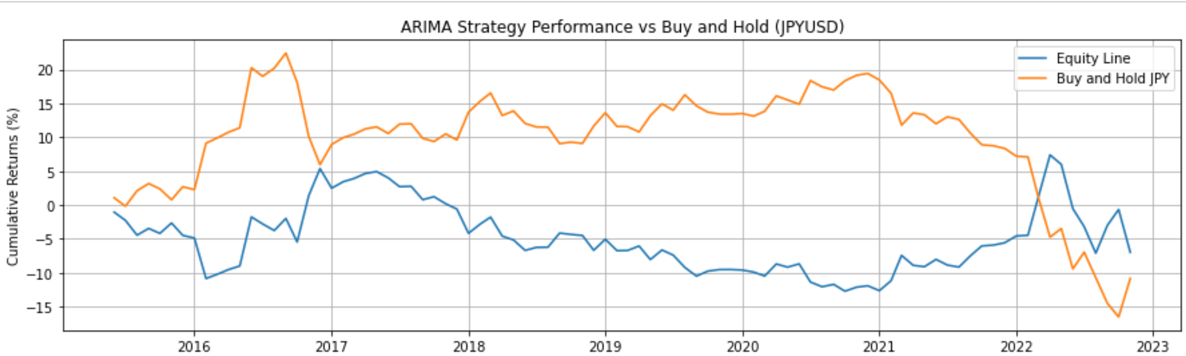
91 rows × 2 columns

In [33]:
```
y["Signals"] = np.where(y["predictions"]>y['Japan'].shift(), 1,-1)


y["returns"] = np.log(y["Japan"]/y["Japan"].shift())


y["strategy returns"] = y["Signals"]*y["returns"]


y["Cumulative Returns"] = (np.exp(y["strategy returns"].cumsum())-1)*100
```

In [34]:
```
y["Cumulative Returns"]
```

Out[34]:
```
2015-05-01          NaN
2015-06-01    -1.050505
2015-07-01    -2.263728
2015-08-01    -4.447558
2015-09-01    -3.451806
                 ...
2022-07-01    -3.187663
2022-08-01    -7.105975
2022-09-01    -2.997755
2022-10-01    -0.652115
2022-11-01    -6.978640
Name: Cumulative Returns, Length: 91, dtype: float64
```

In [35]:
```
plt.figure(figsize = (15, 4))
plt.plot(y["Cumulative Returns"])
plt.plot((np.exp(y["returns"].cumsum())-1)*100)
plt.ylabel("Cumulative Returns (%)")
plt.title("ARIMA Strategy Performance vs Buy and Hold (JPYUSD)")
plt.legend(["Equity Line", "Buy and Hold JPY"])
plt.grid()
```



In [46]:
```
y["Cumulative Returns"]/(100+1)
```

Out[46]:
```
2015-05-01         NaN
2015-06-01   -0.010401
2015-07-01   -0.022413
2015-08-01   -0.044035
2015-09-01   -0.034176
                ...
2022-07-01   -0.031561
2022-08-01   -0.070356
2022-09-01   -0.029681
2022-10-01   -0.006457
2022-11-01   -0.069095
Name: Cumulative Returns, Length: 91, dtype: float64
```

In [47]:
```
P = 1000000
A = (((y["Cumulative Returns"]/100)+1)*P)[-1]
t = (len(y)/12)
CCROR = np.log(A/P)/t
print(CCROR*100)
```

```
-0.9539477975994095
```

In [48]:
```
((A/P)**(1/t)-1)*100 # Annualized
```

Out[48]:
```
-0.9494121496260166
```

2

In [39]:
```
y['error'] =y['Japan']- y['predictions']
y
```

Out[39]:

	Japan	predictions	Signals	returns	strategy returns	Cumulative Returns	error
2015-05-01	0.008081	0.008449	-1	NaN	NaN	NaN	-0.000368
2015-06-01	0.008167	0.008046	-1	0.010561	-0.010561	-1.050505	0.000121
2015-07-01	0.008066	0.008175	1	-0.012337	-0.012337	-2.263728	-0.000108
2015-08-01	0.008251	0.008057	-1	0.022598	-0.022598	-4.447558	0.000194
2015-09-01	0.008337	0.008268	1	0.010367	0.010367	-3.451806	0.000068
...	...	...	...	...	...	...	...
2022-07-01	0.007519	0.007272	-1	0.026927	-0.026927	-3.187663	0.000247
2022-08-01	0.007214	0.007538	1	-0.041315	-0.041315	-7.105975	-0.000323
2022-09-01	0.006909	0.007186	-1	-0.043275	0.043275	-2.997755	-0.000277
2022-10-01	0.006746	0.006880	-1	-0.023894	0.023894	-0.652115	-0.000134
2022-11-01	0.007205	0.006730	-1	0.065799	-0.065799	-6.978640	0.000475

91 rows × 7 columns

In [66]:
```
s_current = np.log(y["Japan"]).reset_index(drop=True)
s_current = s_current.rename('s_current')
s_current
```

Out[66]:
```
0    -4.818263
1    -4.807703
2    -4.820040
3    -4.797442
4    -4.787075
        ...
86   -4.890349
87   -4.931664
88   -4.974939
89   -4.998833
90   -4.933034
Name: s_current, Length: 91, dtype: float64
```

In [65]:
```
df3 = pd.DataFrame(s_current, columns = ['s_current'])
```

In [67]:
```
df3
```

Out[67]:

	s_current
0	-4.818263
1	-4.807703
2	-4.820040
3	-4.797442
4	-4.787075
...	...
86	-4.890349
87	-4.931664
88	-4.974939
89	-4.998833
90	-4.933034

In [96]:
```
s_future = s_current.shift(1)
```

In [113]:
```
s_future
```

Out[113]:
```
0          NaN
1    -4.818263
2    -4.807703
3    -4.820040
4    -4.797442
        ...
86   -4.917277
87   -4.890349
88   -4.931664
89   -4.974939
90   -4.998833
Name: s_current, Length: 91, dtype: float64
```

In [114]:
```
s_future1 = s_future.fillna(0)
```

In [115]:
```
s_future1
```

Out[115]:
```
0     0.000000
1    -4.818263
2    -4.807703
3    -4.820040
4    -4.797442
        ...
86   -4.917277
87   -4.890349
88   -4.931664
89   -4.974939
90   -4.998833
Name: s_current, Length: 91, dtype: float64
```

In [75]:
```
s_change = s_future - s_current
s_change = s_change.rename('s_change')
s_change
```

Out[75]:
```
0          NaN
1    -0.010561
2     0.012337
3    -0.022598
4    -0.010367
        ...
86   -0.026927
87    0.041315
88    0.043275
89    0.023894
90   -0.065799
Name: s_change, Length: 91, dtype: float64
```

In [109]:
```
df6 = pd.DataFrame(s_change.fillna(0), columns = ['s_change'])
```

In [110]:
```
df6
```

Out[110]:

	s_change
0	0.000000
1	-0.010561
2	0.012337
3	-0.022598
4	-0.010367
...	...
86	-0.026927
87	0.041315
88	0.043275
89	0.023894
90	-0.065799

91 rows × 1 columns

In [125]:
```
P1 = len(y['error'])
MSE_T = np.sum(np.square(y['error']))/P1
MSE_T
```

Out[125]: 4.368784909818687e-08

In [126]:
```
MSE_R = np.sum(np.square([df6['s_change']]))/P1
MSE_R
```

Out[126]: 0.0005809296335330899

In [122]:
```
error_R = df6['s_change'].reset_index(drop=True)
error_T = y['error'].reset_index(drop=True)
tmp = np.square(error_R) - np.square(error_T) - (MSE_R - MSE_T)
V_hat = np.sum(np.square(tmp))/P1

## Statistic
DMW = (MSE_R - MSE_T)/np.sqrt(V_hat/P1)

print('Since the DMW statitsic is equal to ' + str(DMW) + ',' + ' which is more than the critical value (' + str(round(
print('we reject the null hypothesis that the MP model does not outperform the random walk model.')
```

```
Since the DMW statitsic is equal to 4.734497242845056, which is more than the critical value (1.28),
we reject the null hypothesis that the MP model does not outperform the random walk model.
```

In [124]:
```
tmp2 = np.sum(np.square(df6['s_change']))/P1
CW = (MSE_R - MSE_T + tmp2)/np.sqrt(V_hat/P1)

print('Since the CW statitsic is equal to ' + str(CW) + ',' + ' which is more than the critical value (' + str(round(st
print('we reject the null hypothesis that the MP model does not outperform the random walk model.')
```

```
Since the CW statitsic is equal to 9.469350562477024, which is more than the critical value (1.28),
we reject the null hypothesis that the MP model does not outperform the random walk model.
```