npk4jhysa

March 10, 2023

[ ]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

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

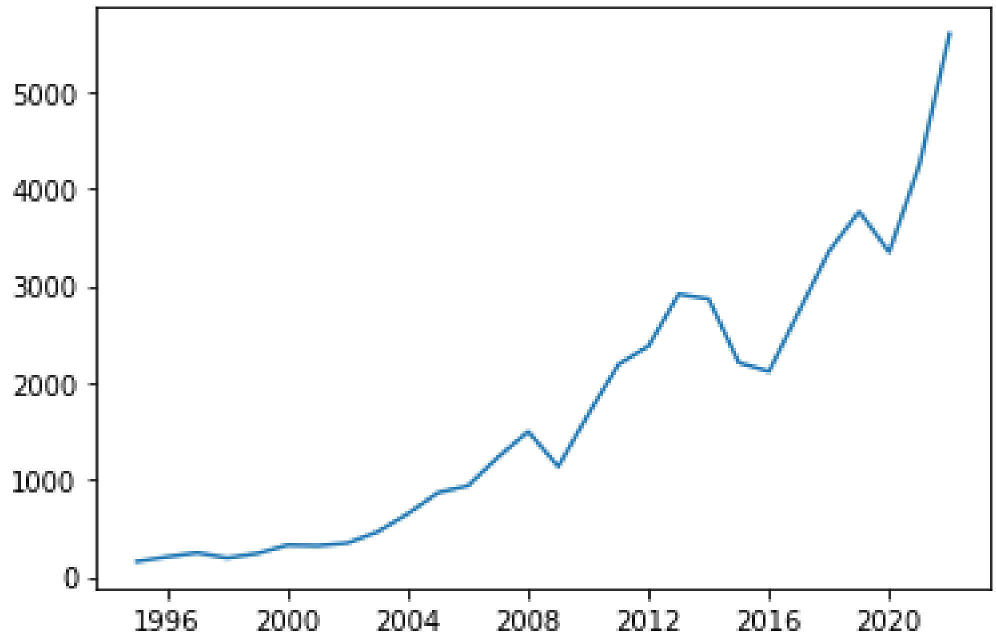
1 Import Data

[ ]:
```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving export_by_years_95_22.csv to export_by_years_95_22.csv
```

[ ]:
```
import io
df = pd.read_csv(io.BytesIO(uploaded['export_by_years_95_22.csv']))
df.set_index('Date', inplace=True)
df.index = pd.to_datetime(df.index, format='%Y')
fig = plt.figure()
plt.plot(df)
#plt.savefig('export.png')
```



2 Finding ARIMA model's orders of the terms: p, d, q

Check if the series is stationary - Augmented Dickey Fuller test

[ ]:
```
from statsmodels.tsa.stattools import adfuller
import statsmodels.api as sm
from numpy import log
```

[ ]:
```
test = sm.tsa.adfuller(df)
print('adf: ', test[0])
print('p-value: ', test[1])
print('Critical values: ', test[4])
if test[0]> test[4]['5%']:
    print('Non-stationary time series')
else:
    print('Stationary time series')
```

```
adf:  1.944258319115555
p-value:  0.9985959908497772
Critical values:  {'1%': -3.7238633119999998, '5%': -2.98648896, '10%':
-2.6328004}
Non-stationary time series
```

[ ]:
```
df1diff = df.diff(periods=1).dropna()
```

[ ]:
```
test = sm.tsa.adfuller(df1diff)
print('adf: ', test[0])
print('p-value: ', test[1])
print('Critical values: ', test[4])
if test[0]> test[4]['5%']:
    print('Non-stationary time series')
else:
    print('Stationary time series')
```

```
adf:  -3.792399919280002
p-value:  0.002986037972090526
Critical values:  {'1%': -3.7238633119999998, '5%': -2.98648896, '10%':
-2.6328004}
Stationary time series
```

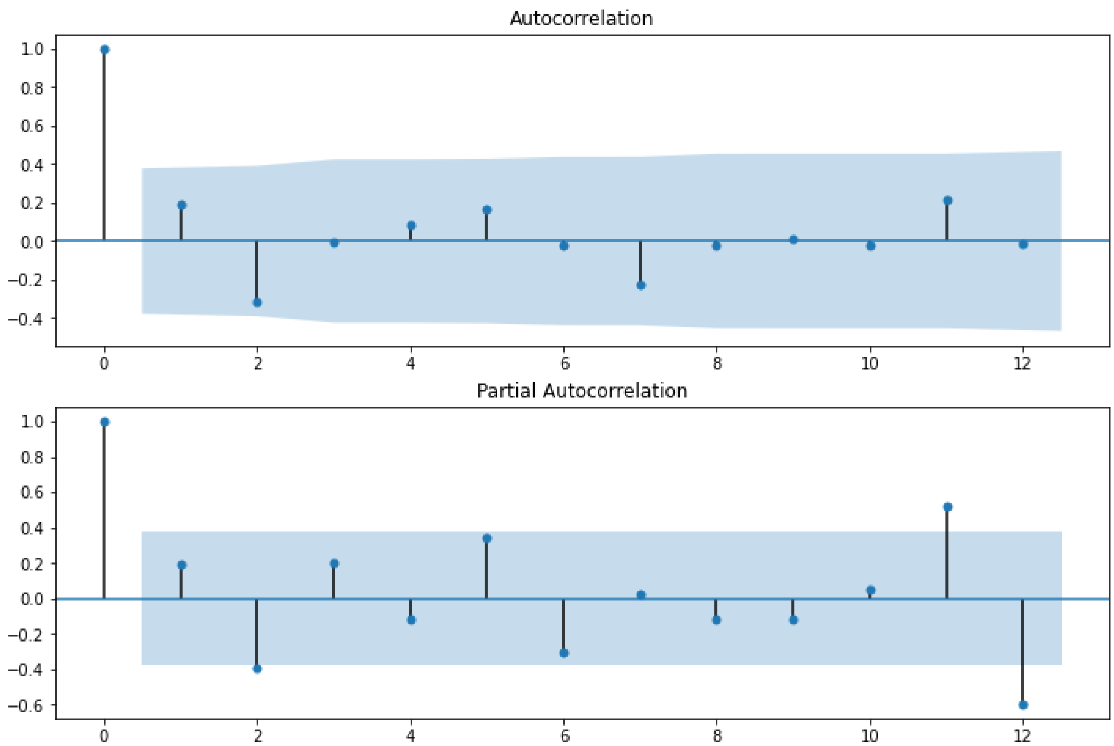
[ ]:
```
fig = plt.figure()
plt.plot(df1diff)
#plt.savefig('export_1diff.png')
```

d=1

Correlogram

[ ]:
```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df1diff.values.squeeze(),lags=12, ax=ax1)
ax2 = fig.add_subplot(212)
```

```
fig = sm.graphics.tsa.plot_pacf(df1diff, lags=12, ax=ax2)
#plt.savefig('correlogram.png')
```



q=1 p=1

3 Build the model

[ ]:
```
model = sm.tsa.ARIMA(df, order=(1,1,1)).fit(full_output=False, disp=0)
print(model.summary())
```

```
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency AS-
JAN will be used.
  warnings.warn('No frequency information was'
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency AS-
JAN will be used.
  warnings.warn('No frequency information was'
                             ARIMA Model Results
==============================================================================
Dep. Variable:               D.Export   No. Observations:                   27
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -196.388
```

```
Method:                       css-mle   S.D. of innovations            332.931
Date:                Wed, 08 Feb 2023   AIC                            400.776
Time:                        11:03:53   BIC                            405.960
Sample:                    01-01-1996   HQIC                           402.318
                         - 01-01-2022
================================================================================
==
                     coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const            200.0188     90.640      2.207      0.027      22.368
377.669
ar.L1.D.Export    -0.4028      0.182     -2.213      0.027      -0.760
-0.046
ma.L1.D.Export     1.0000      0.115      8.724      0.000       0.775
1.225
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -2.4826           +0.0000j            2.4826            0.5000
MA.1           -1.0000           +0.0000j            1.0000            0.5000
-----------------------------------------------------------------------------
```

The Ljung–Box Q test to test whether any of a group of autocorrelations of a time series are different from zero.

[ ]:
```
q_test = sm.tsa.stattools.acf(model.resid, qstat=True)
print(pd.DataFrame({'Q-stat':q_test[1], 'p-value':q_test[2]}))
```

```
      Q-stat   p-value
0   0.053316  0.817390
1   1.198731  0.549160
2   1.200881  0.752793
3   1.255290  0.868914
4   2.082932  0.837555
5   2.109079  0.909398
6   3.287016  0.857243
7   3.311945  0.913285
8   3.515592  0.940315
9   4.183748  0.938677
10  7.860827  0.725714
11  8.232539  0.766704
12  8.620770  0.800944
13  8.903053  0.837210
14  8.933045  0.880992
15  8.965974  0.914808
```

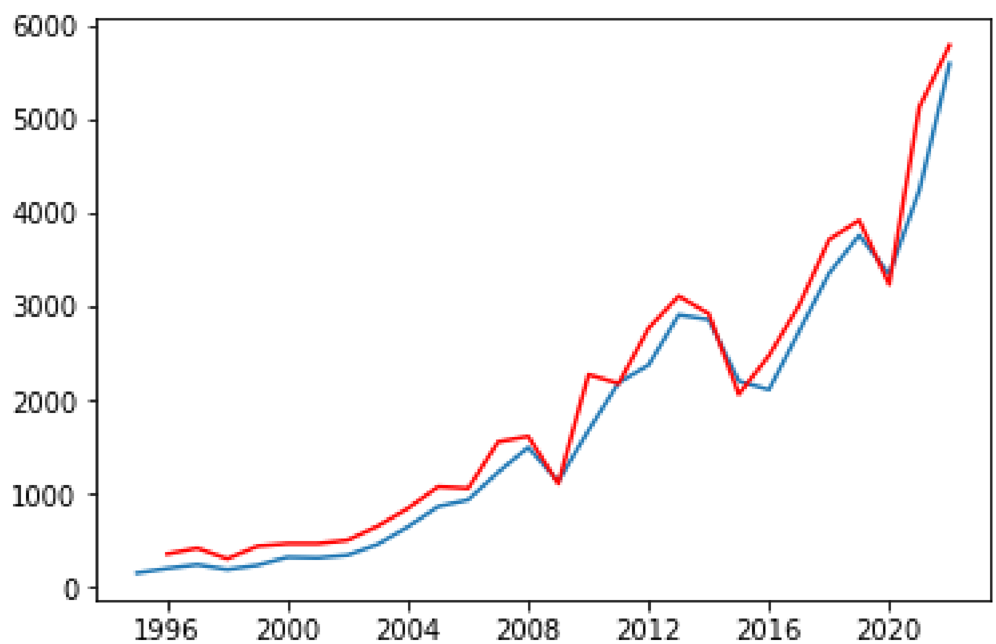
```
16   9.056098  0.938485
17   9.092376  0.957564
18   9.108904  0.971627
19   9.854171  0.970740
20  10.585360  0.970266
21  10.620030  0.979758
22  11.435196  0.978293
23  11.887348  0.981147
24  13.512254  0.969583
25  13.893367  0.974379
```

The value of this statistic and p-values indicate that the hypothesis of the randomness of the residuals is not rejected, and most likely this process is "white noise".

4 Predict and Metrics

```
[ ]: from sklearn.metrics import r2_score
```

```
[ ]: pred = model.predict(1, 28, typ='levels', dynamic=False).shift(-1)[:-1]
     trn = df
     fig = plt.figure()
     plt.plot(trn)
     plt.plot(pred, color='red')
     #plt.savefig('metrics.png')
```



[ ]:
```
def forecast_accuracy(forecast, actual):
    r2 = r2_score(actual, forecast)
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))
    me = np.mean(forecast - actual)
    mae = np.mean(np.abs(forecast - actual))
    mpe = np.mean((forecast - actual)/actual)
    rmse = np.mean((forecast - actual)**2)**.5
    corr = np.corrcoef(forecast, actual)[0,1]
    mins = np.amin(np.hstack([forecast[:,None], actual[:,None]]), axis=1)
    maxs = np.amax(np.hstack([forecast[:,None], actual[:,None]]), axis=1)
    minmax = 1 - np.mean(mins/maxs)
    return({'r2':r2, 'mape':mape, 'me':me, 'mae': mae,
            'mpe': mpe, 'rmse':rmse,'corr':corr, 'minmax':minmax})
metrics = pd.DataFrame(list(forecast_accuracy(pred.values, trn[1:].values.
 ↪flatten()).items()), columns=['Metrics', 'Value'])
print(metrics)
#metrics.to_excel('metrics.xls', index=False)
```

```
  Metrics       Value
0      r2    0.959751
1    mape    0.265651
2      me  203.415473
3     mae  224.327097
4     mpe    0.256613
5    rmse  286.539521
6    corr    0.991273
7  minmax    0.183541
```

5 Forecasting

[ ]:
```
fc, se, conf = model.forecast(3, alpha=0.05)
```

[ ]:
```
fi = ['2022', '2023', '2024']
fc_series = pd.Series(fc)
lower_series = pd.Series(conf[:, 0])
upper_series = pd.Series(conf[:, 1])

fc_series.index = pd.to_datetime(fi, format='%Y')
lower_series.index = pd.to_datetime(fi, format='%Y')
upper_series.index = pd.to_datetime(fi, format='%Y')
```

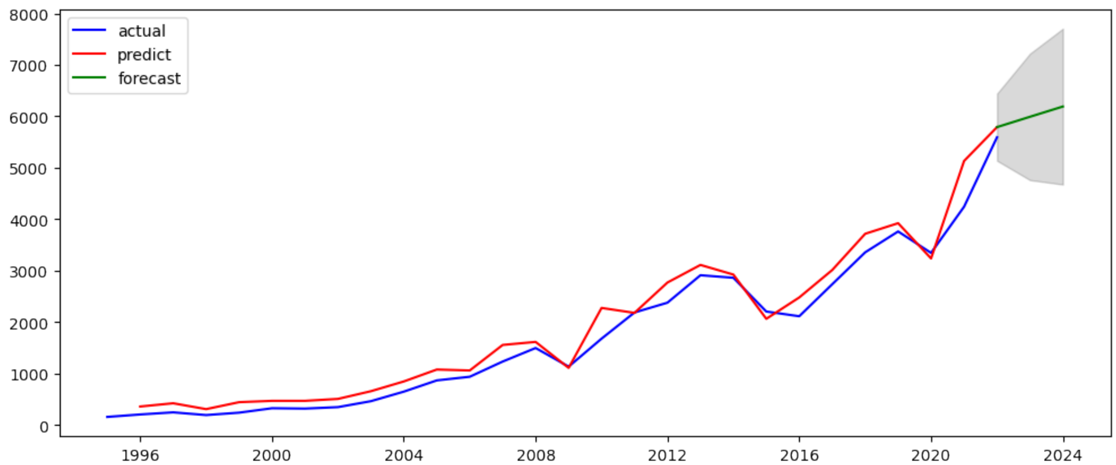
[ ]:
```
fc_df = pd.DataFrame({'Date':['2022', '2023', '2024'], 'forecast':fc_series.
 ↪values, 'conf. intervals lower':lower_series.values, 'conf. intervals upper':
 ↪upper_series.values})
fc_df[1:]
```

[ ]:

	Date	forecast	conf. intervals lower	conf. intervals upper
1	2023	5991.366036	4761.719147	7221.012926
2	2024	6190.957865	4675.797756	7706.117975

[ ]:

```
fig = plt.figure(figsize=(12,5), dpi=100)
plt.plot(df, color='blue', label='actual')
plt.plot(pred, color='red', label='predict')
plt.plot(fc_series, color='green', label='forecast')
plt.fill_between(lower_series.index, lower_series, upper_series, color='k',␣
 ↪alpha=.15)
plt.legend(loc="upper left")
#plt.savefig('result.png')
```



mbkkp9wlm

March 10, 2023

[ ]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

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

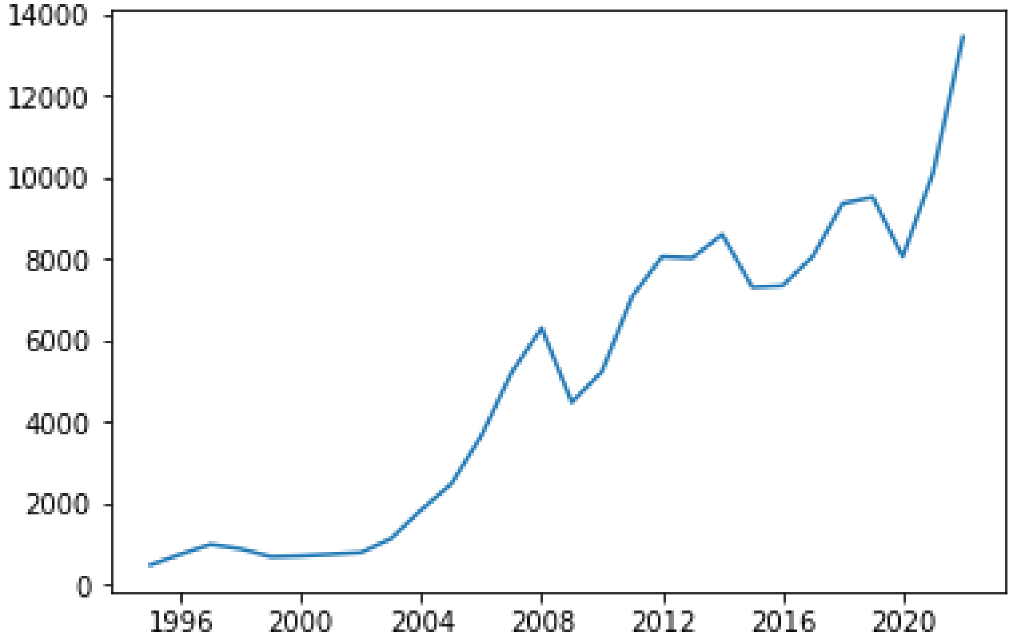
1 Import Data

[ ]:
```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving import_by_years_95_22.csv to import_by_years_95_22.csv
```

[ ]:
```
import io
df = pd.read_csv(io.BytesIO(uploaded['import_by_years_95_22.csv']))
df.set_index('Date', inplace=True)
df.index = pd.to_datetime(df.index, format='%Y')
fig = plt.figure()
plt.plot(df)
#plt.savefig('import.png')
```



2 Finding ARIMA model's orders of the terms: p, d, q

Check if the series is stationary - Augmented Dickey Fuller test

[ ]:
```
from statsmodels.tsa.stattools import adfuller
import statsmodels.api as sm
from numpy import log
```

[ ]:
```
test = sm.tsa.adfuller(df)
print('adf: ', test[0])
print('p-value: ', test[1])
print('Critical values: ', test[4])
if test[0]> test[4]['5%']:
    print('Non-stationary time series')
else:
    print('Stationary time series')
```

```
adf:  0.6260836746022841
p-value:  0.9882412090145923
Critical values:  {'1%': -3.7238633119999998, '5%': -2.98648896, '10%':
-2.6328004}
Non-stationary time series
```

[ ]:
```
df1diff = df.diff(periods=1).dropna()
```

```
[ ]: test = sm.tsa.adfuller(df1diff)
     print('adf: ', test[0])
     print('p-value: ', test[1])
     print('Critical values: ', test[4])
     if test[0]> test[4]['5%']:
         print('Non-stationary time series')
     else:
         print('Stationary time series')
```

```
adf:  -4.5164380377267035
p-value:  0.00018362005836327426
Critical values:  {'1%': -3.7238633119999998, '5%': -2.98648896, '10%':
-2.6328004}
Stationary time series
```

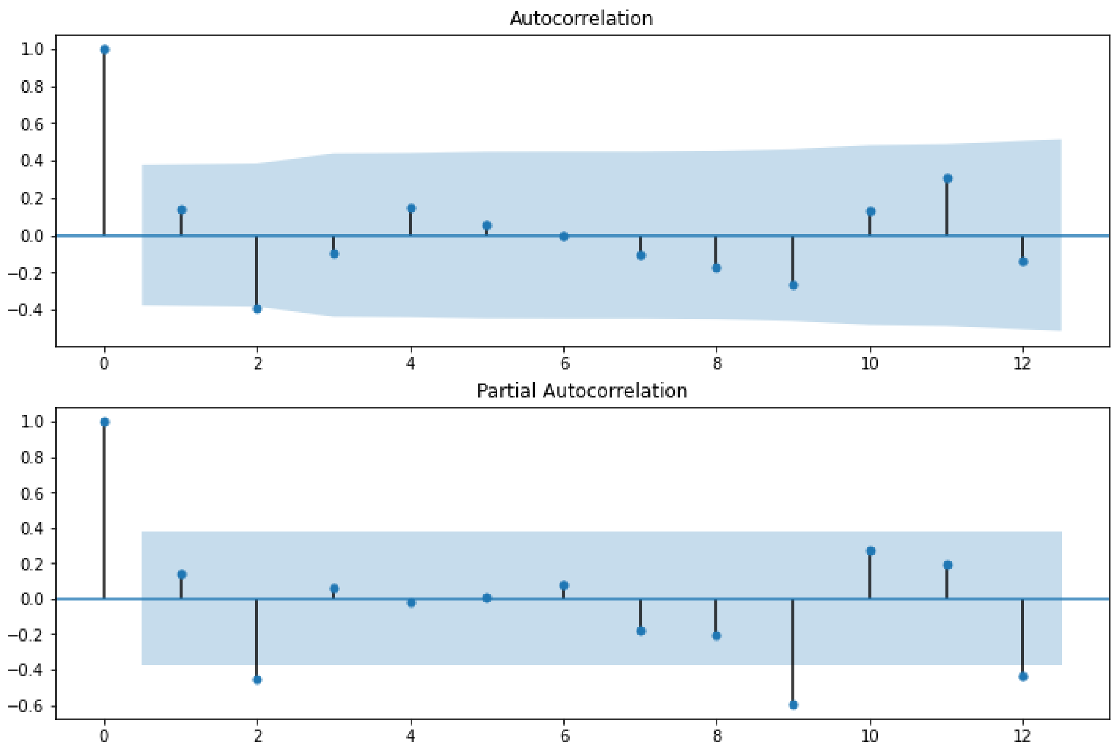
```
[ ]: fig = plt.figure()
     plt.plot(df1diff)
     #plt.savefig('import_1diff.png')
```

d=1

Correlogram

```
[ ]: fig = plt.figure(figsize=(12,8))
     ax1 = fig.add_subplot(211)
     fig = sm.graphics.tsa.plot_acf(df1diff.values.squeeze(),lags=12, ax=ax1)
     ax2 = fig.add_subplot(212)
```

```
fig = sm.graphics.tsa.plot_pacf(df1diff, lags=12, ax=ax2)
#plt.savefig('correlogram.png')
```



q=1 p=1

3 Build the model

```
[ ]: model = sm.tsa.ARIMA(df, order=(1,1,1)).fit(full_output=False, disp=0)
     print(model.summary())
```

```
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency AS-
JAN will be used.
  warnings.warn('No frequency information was'
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency AS-
JAN will be used.
  warnings.warn('No frequency information was'
                             ARIMA Model Results
==============================================================================
Dep. Variable:               D.Import   No. Observations:                   27
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -224.759
```

```
Method:                       css-mle   S.D. of innovations            994.493
Date:                Wed, 08 Feb 2023   AIC                            457.517
Time:                        09:33:21   BIC                            462.701
Sample:                    01-01-1996   HQIC                           459.059
                         - 01-01-2022
================================================================================
==
                      coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const             492.9886    242.839      2.030      0.042      17.034
968.943
ar.L1.D.Import     -0.2448      0.340     -0.721      0.471      -0.911
0.421
ma.L1.D.Import      0.5870      0.244      2.402      0.016       0.108
1.066
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -4.0848           +0.0000j            4.0848            0.5000
MA.1           -1.7036           +0.0000j            1.7036            0.5000
-----------------------------------------------------------------------------
```

q=1 p=0

```
[ ]: model = sm.tsa.ARIMA(df, order=(0,1,1)).fit(full_output=False, disp=0)
     print(model.summary())
```

```
                              ARIMA Model Results
==============================================================================
Dep. Variable:               D.Import   No. Observations:                   27
Model:                 ARIMA(0, 1, 1)   Log Likelihood                -225.023
Method:                       css-mle   S.D. of innovations           1003.745
Date:                Wed, 08 Feb 2023   AIC                            456.045
Time:                        09:34:17   BIC                            459.933
Sample:                    01-01-1996   HQIC                           457.201
                         - 01-01-2022
================================================================================
==
                      coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const             506.7109    272.765      1.858      0.063     -27.898
1041.320
ma.L1.D.Import      0.4275      0.185      2.309      0.021       0.065
```

```
0.790
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus          Frequency
-----------------------------------------------------------------------------
MA.1           -2.3391           +0.0000j            2.3391             0.5000
-----------------------------------------------------------------------------

/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency AS-
JAN will be used.
  warnings.warn('No frequency information was'
/usr/local/lib/python3.8/dist-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency AS-
JAN will be used.
  warnings.warn('No frequency information was'
```

The Ljung–Box Q test to test whether any of a group of autocorrelations of a time series are different from zero.

[ ]:
```
q_test = sm.tsa.stattools.acf(model.resid, qstat=True)
print(pd.DataFrame({'Q-stat':q_test[1], 'p-value':q_test[2]}))
```

```
       Q-stat   p-value
0    0.123226  0.725562
1    2.350165  0.308794
2    2.649561  0.448867
3    3.534987  0.472579
4    3.535054  0.618090
5    3.590275  0.731923
6    3.955340  0.784907
7    4.231127  0.835691
8    6.823001  0.655542
9    7.071294  0.718698
10  12.371430  0.336381
11  13.521947  0.332271
12  15.490780  0.277724
13  15.650352  0.335190
14  16.460932  0.352096
15  16.660380  0.407896
16  16.907736  0.460634
17  17.006190  0.522680
18  17.006270  0.589443
19  17.185768  0.640880
20  17.199787  0.698926
21  17.317525  0.745599
22  18.040265  0.755302
23  18.401786  0.783094
24  18.815868  0.805863
```

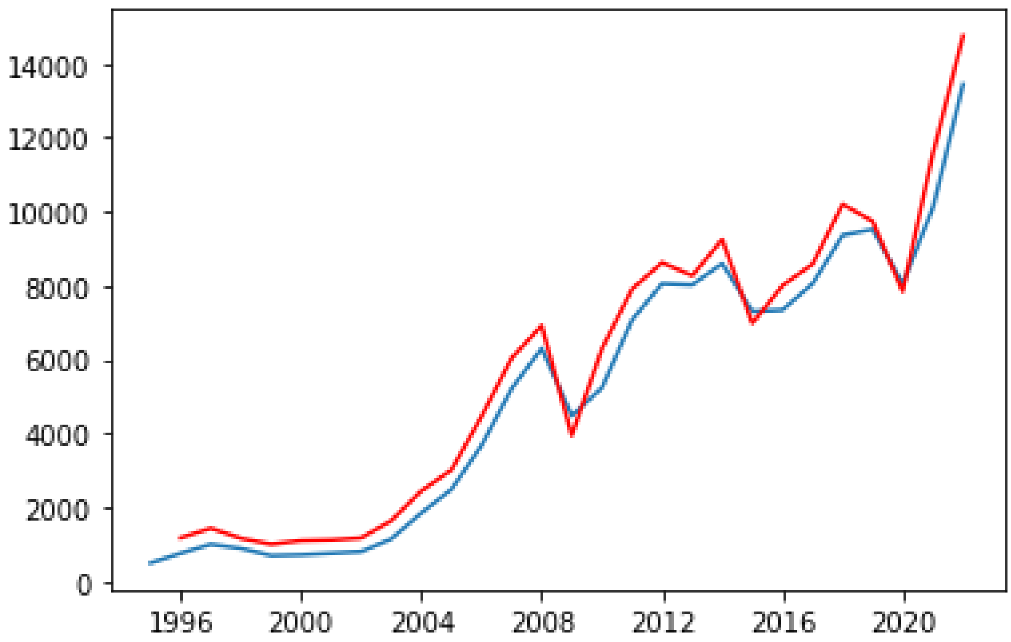
```
25  19.040902  0.834698
```

The value of this statistic and p-values indicate that the hypothesis of the randomness of the residuals is not rejected, and most likely this process is "white noise".

4 Predict and Metrics

[ ]:
```python
from sklearn.metrics import r2_score
```

[ ]:
```python
pred = model.predict(1, 28, typ='levels', dynamic=False).shift(-1)[:-1]
trn = df
fig = plt.figure()
plt.plot(trn)
plt.plot(pred, color='red')
#plt.savefig('metrics.png')
```



[ ]:
```python
def forecast_accuracy(forecast, actual):
    r2 = r2_score(actual, forecast)
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))
    me = np.mean(forecast - actual)
    mae = np.mean(np.abs(forecast - actual))
    mpe = np.mean((forecast - actual)/actual)
    rmse = np.mean((forecast - actual)**2)**.5
    corr = np.corrcoef(forecast, actual)[0,1]
    mins = np.amin(np.hstack([forecast[:,None], actual[:,None]]), axis=1)
    maxs = np.amax(np.hstack([forecast[:,None], actual[:,None]]), axis=1)
```

```
    minmax = 1 - np.mean(mins/maxs)
    return({'r2':r2, 'mape':mape, 'me':me, 'mae': mae,
            'mpe': mpe, 'rmse':rmse,'corr':corr, 'minmax':minmax})
metrics = pd.DataFrame(list(forecast_accuracy(pred.values, trn[1:].values.
 ↪flatten()).items()), columns=['Metrics', 'Value'])
print(metrics)
#metrics.to_excel('metrics.xls', index=False)
```

```
  Metrics       Value
0      r2    0.966789
1    mape    0.217741
2      me  508.140366
3     mae  587.823846
4     mpe    0.203581
5    rmse  665.033009
6    corr    0.994246
7  minmax    0.163774
```

5 Forecasting

[ ]:
```
fc, se, conf = model.forecast(3, alpha=0.05)
```

[ ]:
```
fi = ['2022', '2023', '2024']
fc_series = pd.Series(fc)
lower_series = pd.Series(conf[:, 0])
upper_series = pd.Series(conf[:, 1])

fc_series.index = pd.to_datetime(fi, format='%Y')
lower_series.index = pd.to_datetime(fi, format='%Y')
upper_series.index = pd.to_datetime(fi, format='%Y')
```

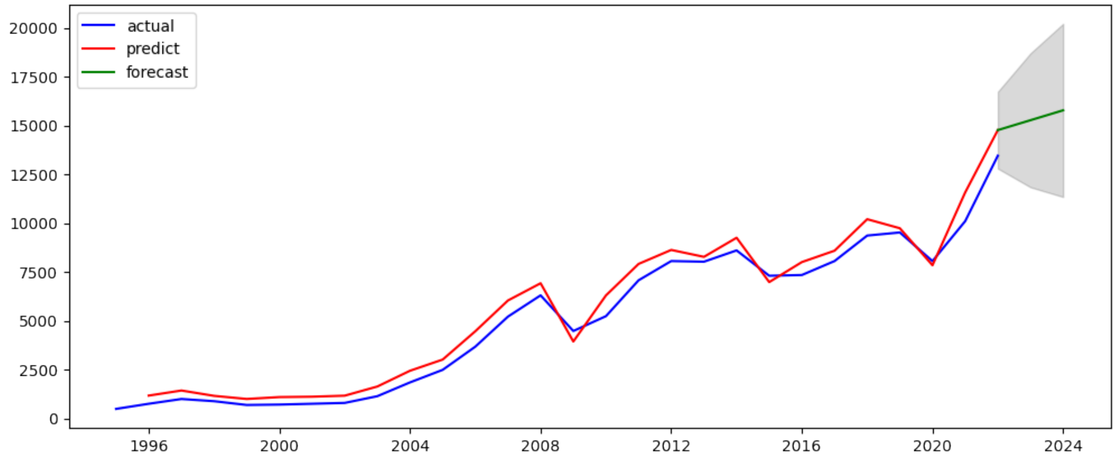
[ ]:
```
fc_df = pd.DataFrame({'Date':['2022', '2023', '2024'], 'forecast':fc_series.
 ↪values, 'conf. intervals lower':lower_series.values, 'conf. intervals upper':
 ↪upper_series.values})
fc_df[1:]
```

[ ]:

	Date	forecast	conf. intervals lower	conf. intervals upper
1	2023	15266.449464	11837.592383	18695.306544
2	2024	15773.160370	11341.023363	20205.297378

[ ]:
```
fig = plt.figure(figsize=(12,5), dpi=100)
plt.plot(df, color='blue', label='actual')
plt.plot(pred, color='red', label='predict')
plt.plot(fc_series, color='green', label='forecast')
plt.fill_between(lower_series.index, lower_series, upper_series, color='k',␣
 ↪alpha=.15)
```

```
plt.legend(loc="upper left")
#plt.savefig('result.png')
```



lmoakfw2u

March 10, 2023

1 Singular Spectrum Analysis for Export Time Series Forecasting

Import the main libraries

```
[ ]: import os
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
```

```
[ ]: from sklearn.metrics import r2_score
```

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

```
[ ]: # upload library file - mySSA.py
     from google.colab import files
     uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving mySSA.py to mySSA.py
```

```
[ ]: from mySSA import mySSA
```

Read in the file as an example:

```
[ ]: from google.colab import files
     uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving export_by_years_95_22.csv to export_by_years_95_22.csv
```

```
[ ]: ts = pd.read_csv('export_by_years_95_22.csv', parse_dates=True,␣
      ↪index_col='Date')
```

```
[ ]: ts
```

[ ]:

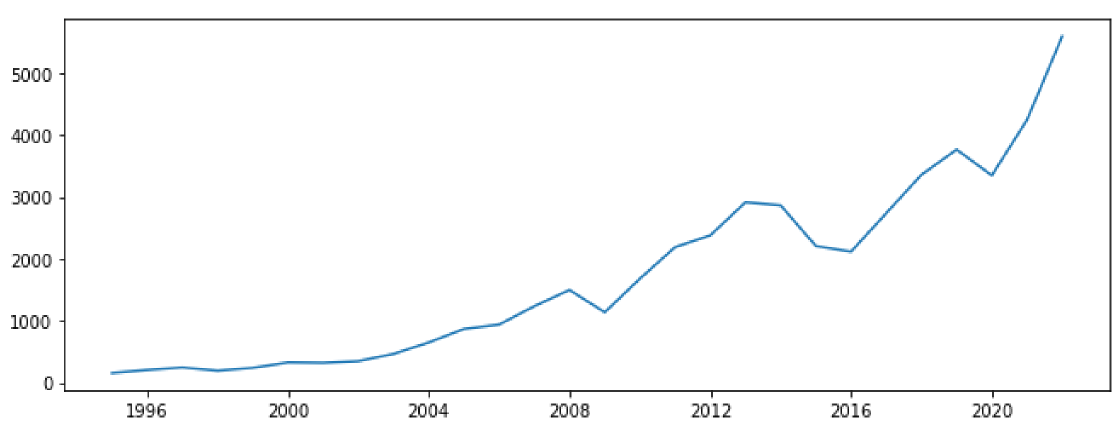
```
            Export
Date
1995-01-01   155.2
1996-01-01   203.0
1997-01-01   244.2
1998-01-01   191.3
1999-01-01   238.0
2000-01-01   323.9
2001-01-01   317.2
2002-01-01   345.7
2003-01-01   461.3
2004-01-01   646.8
2005-01-01   865.5
2006-01-01   936.5
2007-01-01  1232.1
2008-01-01  1495.2
2009-01-01  1133.5
2010-01-01  1677.4
2011-01-01  2186.3
2012-01-01  2376.5
2013-01-01  2910.4
2014-01-01  2860.8
2015-01-01  2204.3
2016-01-01  2112.9
2017-01-01  2735.8
2018-01-01  3355.8
2019-01-01  3761.7
2020-01-01  3344.6
2021-01-01  4242.8
2022-01-01  5592.9
```

[ ]:

```
fig = plt.figure()
plt.plot(ts)
```

[ ]:

```
[<matplotlib.lines.Line2D at 0x7fba69dc7640>]
```



Instantiate the ssa object with the time series

[ ]:
```
ssa = mySSA(ts)
```

The methods and attributes currently associated with the object

[ ]:
```
[x for x in dir(ssa) if '__' not in x and x[0]!='_']
```

[ ]:
```
['decompose',
 'diagonal_averaging',
 'embed',
 'forecast_recurrent',
 'freq',
 'get_contributions',
 'ts',
 'ts_N',
 'ts_name',
 'ts_v',
 'view_reconstruction',
 'view_s_contributions',
 'view_time_series']
```

The general procedure in SSA is as follows: 1. **Embed** the time series by forming a Hankel matrix of lagged window (length K) vectors. 2. **Decompose** the embedded time series via Singular Value Decomposition 3. **Eigentripple Grouping** is the process of identifying eigenvalue-eigenvector pairs as *trend*, *seasonal* and *noise* 4. **Reconstruct** the time series from the eigenvalue-eigenvector pairs identified as *trend* and *seasonal*. This is done through a process called *diagonal averaging*.

[ ]:
```
K = 14
suspected_seasonality = 0
```

[ ]:
```
ssa.embed(embedding_dimension=K, suspected_frequency=suspected_seasonality,␣
 ↪verbose=True)
```

```
----------------------------------------
EMBEDDING SUMMARY:
Embedding dimension     :  14
Trajectory dimensions   : (14, 15)
Complete dimension      : (14, 15)
Missing dimension       : (14, 0)
```

[ ]:
```
ssa.decompose(verbose=True)
```

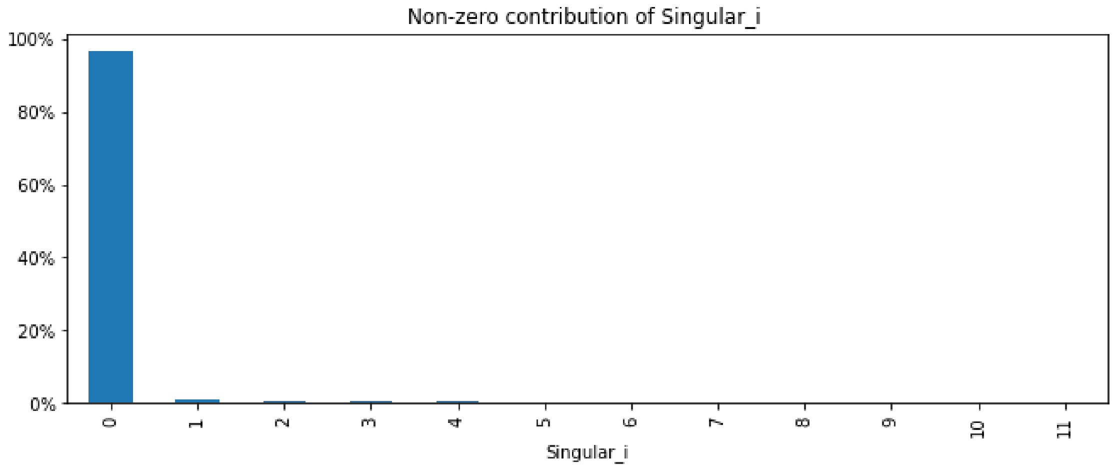
```
----------------------------------------
DECOMPOSITION SUMMARY:
Rank of trajectory              : 14
```

```
Dimension of projection space   : 12
Characteristic of projection    : 0.9999
```

Contribution of each of the signals

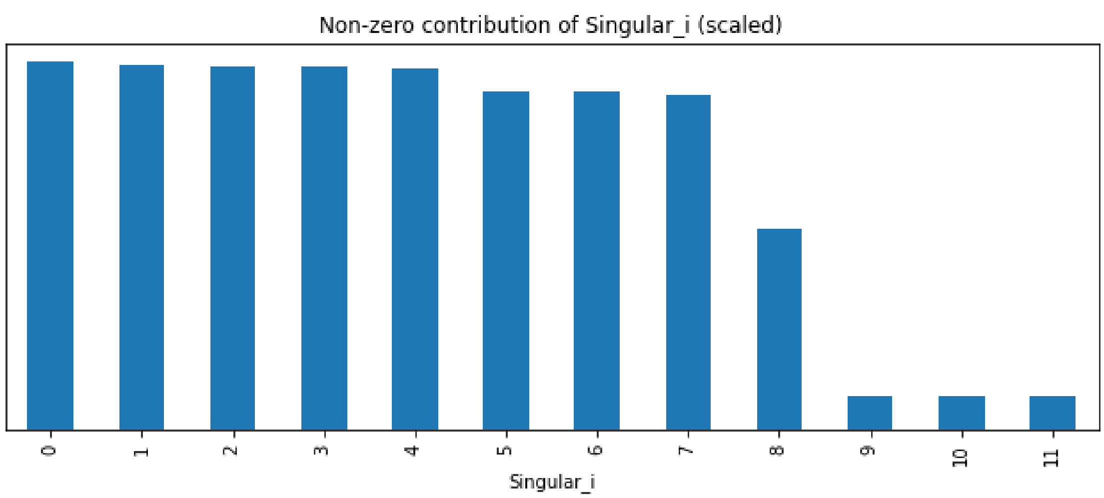
[ ]:
```python
# First enable display of graphs in the notebook
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 11, 4

ssa.view_s_contributions()
```



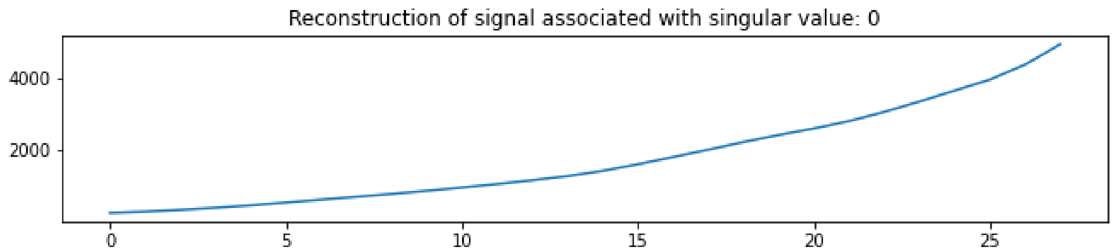
Eigenvalue groupings more clearly.

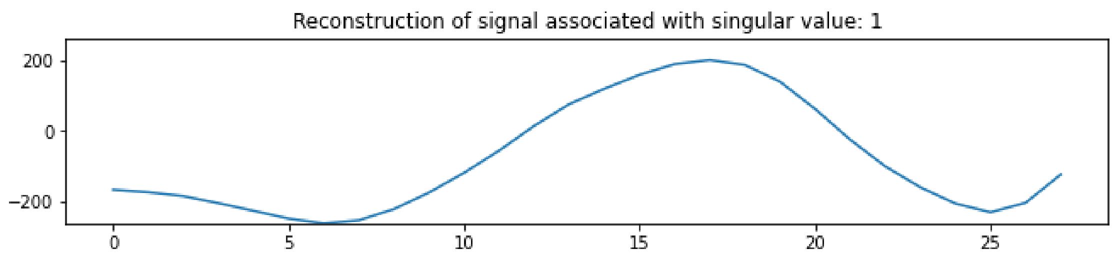
[ ]:
```python
ssa.view_s_contributions(adjust_scale=True)
```

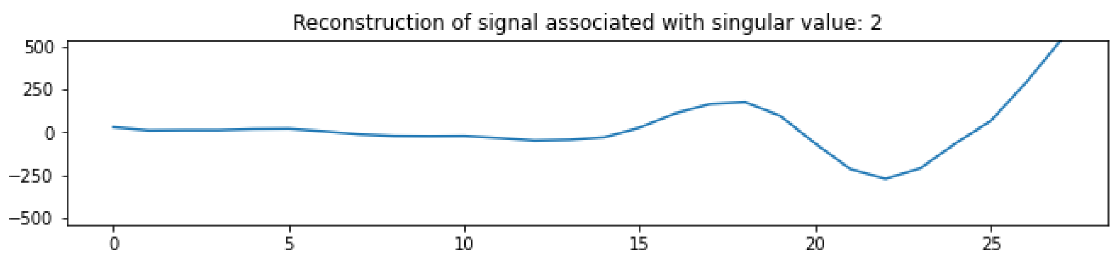


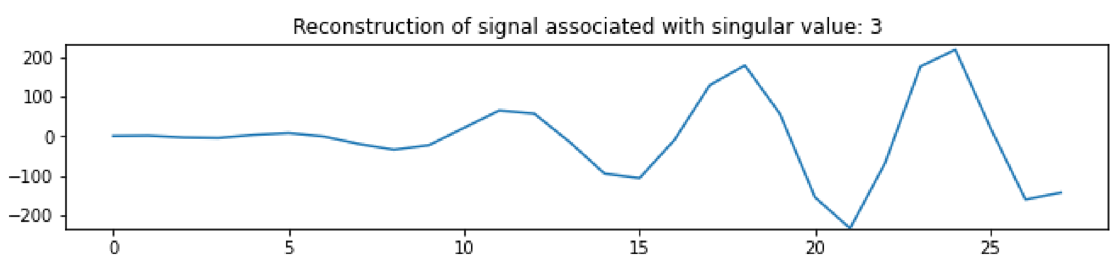
The additive signal elements are stored in the `object.Xs` *dictionary.*

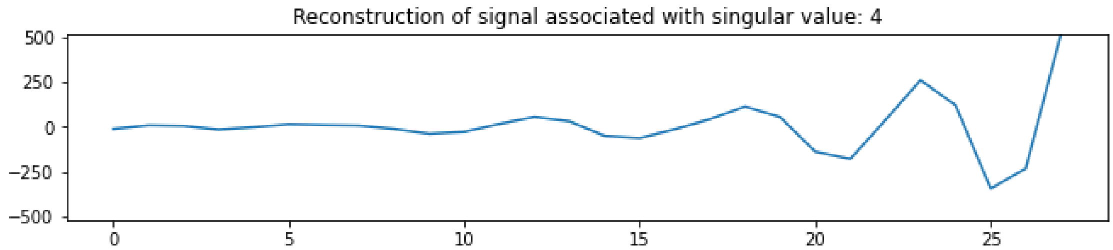
```
[ ]: rcParams['figure.figsize'] = 11, 2
     for i in range(5):
         ssa.view_reconstruction(ssa.Xs[i], names=i, symmetric_plots=i!=0)
     rcParams['figure.figsize'] = 11, 4
```



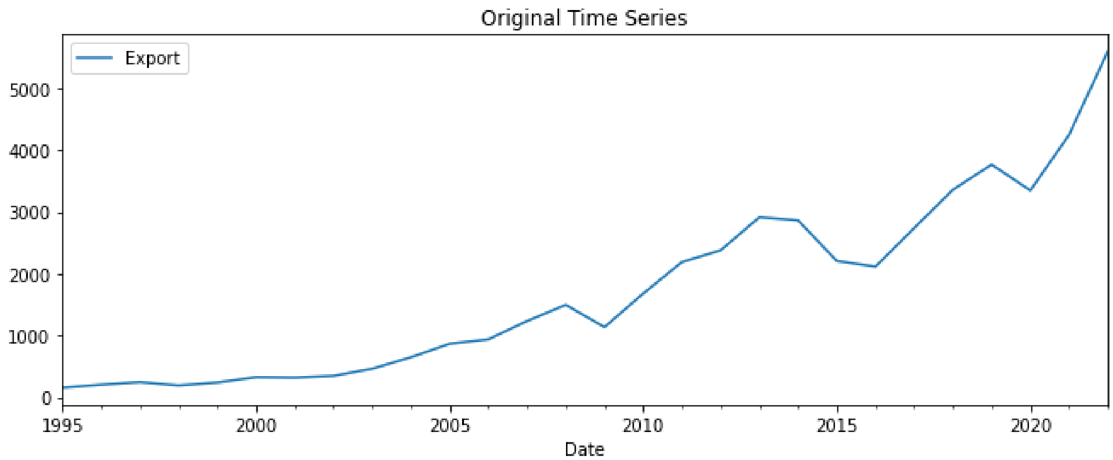


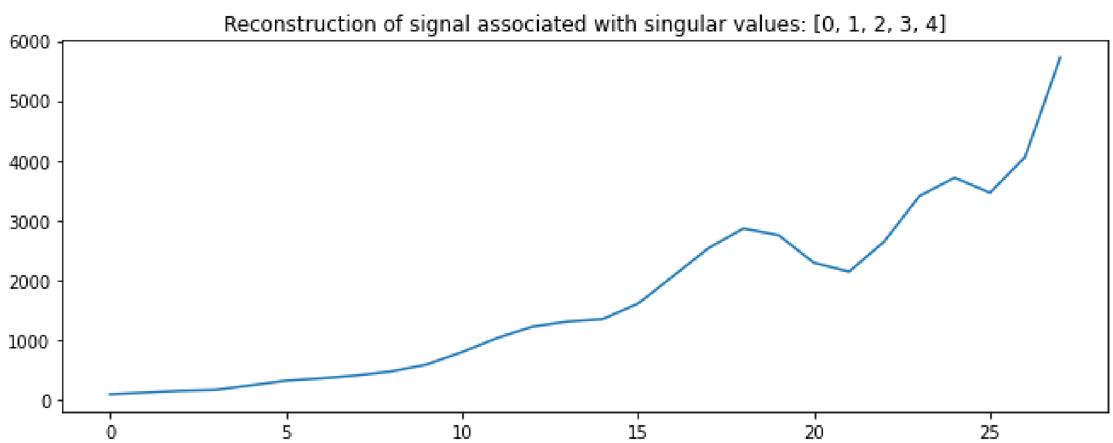




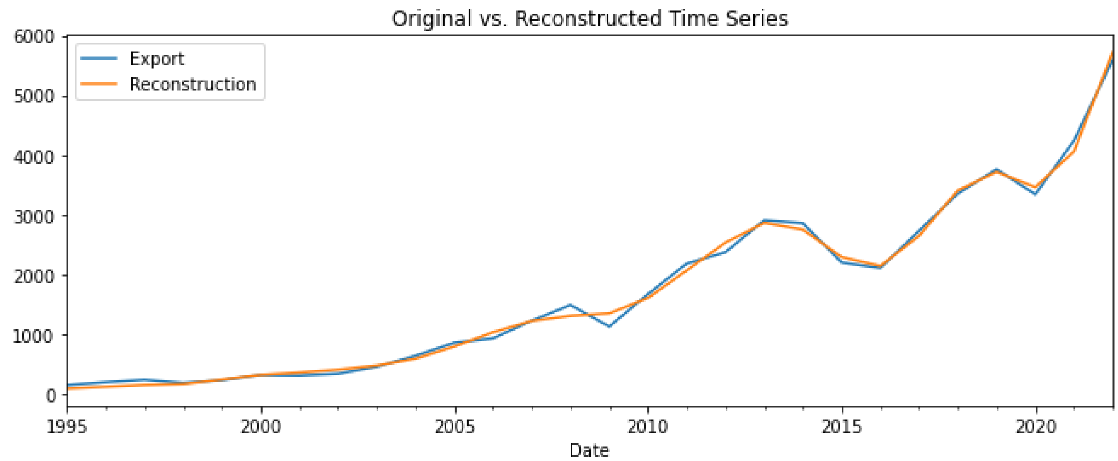


[ ]:
```
ssa.ts.plot(title='Original Time Series')
streams5 = [i for i in range(5)]
reconstructed5 = ssa.view_reconstruction(*[ssa.Xs[i] for i in streams5],␣
    ↪names=streams5, return_df=True)
```





```
[ ]: ts_copy5 = ssa.ts.copy()
     ts_copy5['Reconstruction'] = reconstructed5.Reconstruction.values
     ts_copy5.plot(title='Original vs. Reconstructed Time Series');
```



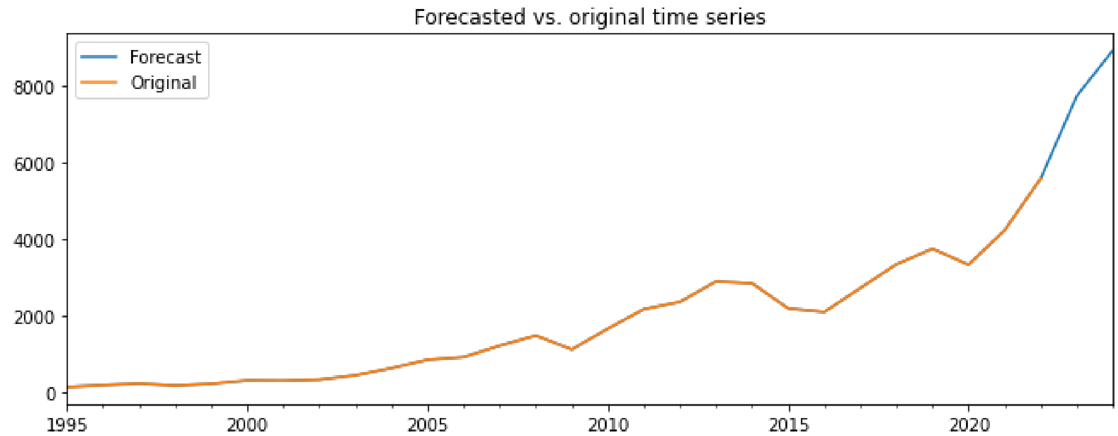

```
[ ]: def forecast_accuracy(forecast, actual):
         r2 = r2_score(actual, forecast)
         mape = np.mean(np.abs(forecast - actual)/np.abs(actual))
         me = np.mean(forecast - actual)
         mae = np.mean(np.abs(forecast - actual))
         mpe = np.mean((forecast - actual)/actual)
         rmse = np.mean((forecast - actual)**2)**.5
         corr = np.corrcoef(forecast, actual)[0,1]
         return({'r2':r2, 'mape':mape, 'me':me, 'mae': mae,
                 'mpe': mpe, 'rmse':rmse,'corr':corr})
```

```
[ ]: actual = np.array(ts_copy5['Export'])
     forecast = np.array(ts_copy5['Reconstruction'])
     metrics = forecast_accuracy(forecast, actual)
     print(pd.DataFrame(metrics, index=[0]))
```

```
         r2      mape        me        mae       mpe       rmse      corr
0  0.995376  0.094581 -4.342984  80.096864 -0.027305  97.544342  0.997721
```

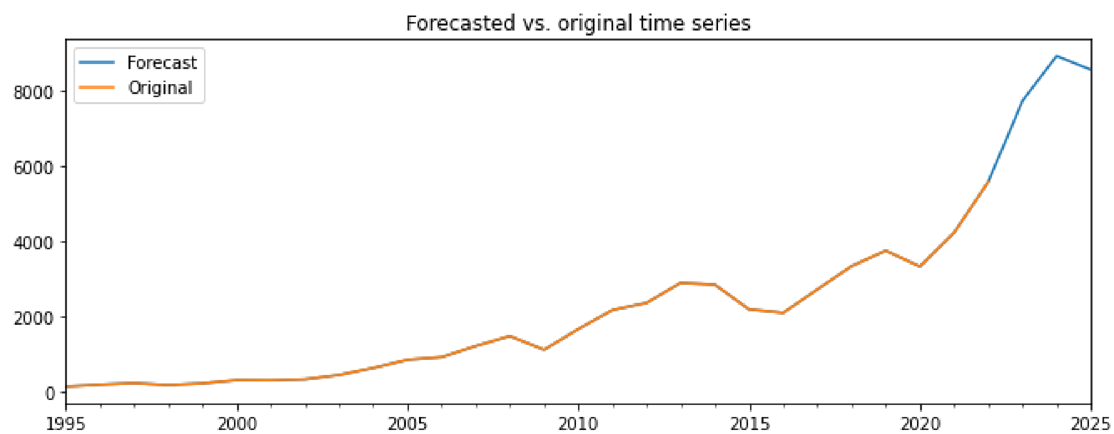
```
[ ]: fc = ssa.forecast_recurrent(steps_ahead=2, singular_values=streams5, plot=True,␣
       ↪return_df=True)
     print(fc['Forecast'][-2:])
```

```
2023-01-01    7747.056485
2024-01-01    8939.839917
Freq: AS-JAN, Name: Forecast, dtype: float64
```




```
[ ]: fc = ssa.forecast_recurrent(steps_ahead=3, singular_values=streams5, plot=True,␣
     ↪return_df=True)
     print(fc['Forecast'][-3:])
```

```
2023-01-01    7747.056485
2024-01-01    8939.839917
2025-01-01    8575.422340
Freq: AS-JAN, Name: Forecast, dtype: float64
```



osnj6hyao

March 10, 2023

1 Singular Spectrum Analysis for Export Time Series Forecasting

Import the main libraries

[ ]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
from sklearn.metrics import r2_score
```

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

[ ]:
```
# upload library file - mySSA.py
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving mySSA.py to mySSA.py
```

[ ]:
```
from mySSA import mySSA
```

Read in the file as an example:

[ ]:
```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving import_by_years_95_22.csv to import_by_years_95_22.csv
```

[ ]:
```
ts = pd.read_csv('import_by_years_95_22.csv', parse_dates=True,␣
↪index_col='Date')
```

[ ]:
```
ts
```

[ ]:

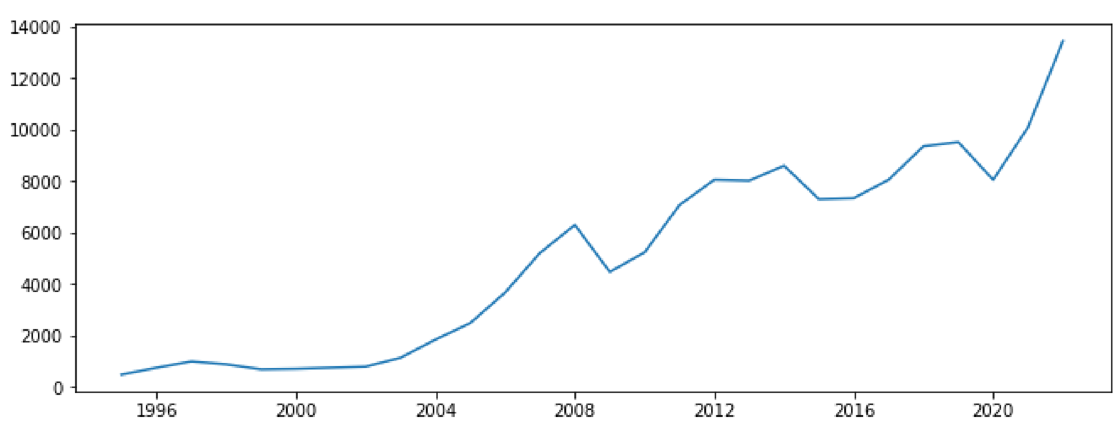
Date	Import
1995-01-01	488.7
1996-01-01	751.2
1997-01-01	995.3
1998-01-01	882.5
1999-01-01	689.6
2000-01-01	709.5
2001-01-01	752.0
2002-01-01	794.7
2003-01-01	1139.0
2004-01-01	1844.3
2005-01-01	2487.5
2006-01-01	3674.8
2007-01-01	5212.2
2008-01-01	6301.5
2009-01-01	4475.7
2010-01-01	5236.0
2011-01-01	7072.3
2012-01-01	8056.4
2013-01-01	8022.7
2014-01-01	8601.8
2015-01-01	7304.2
2016-01-01	7341.7
2017-01-01	8057.1
2018-01-01	9361.4
2019-01-01	9519.5
2020-01-01	8053.8
2021-01-01	10099.8
2022-01-01	13450.1

[ ]:

```
fig = plt.figure()
plt.plot(ts)
```

[ ]:

```
[<matplotlib.lines.Line2D at 0x7f1e5d20c1f0>]
```



Instantiate the ssa object with the time series

[ ]:
```
ssa = mySSA(ts)
```

The methods and attributes currently associated with the object

[ ]:
```
[x for x in dir(ssa) if '__' not in x and x[0]!='_']
```

[ ]:
```
['decompose',
 'diagonal_averaging',
 'embed',
 'forecast_recurrent',
 'freq',
 'get_contributions',
 'ts',
 'ts_N',
 'ts_name',
 'ts_v',
 'view_reconstruction',
 'view_s_contributions',
 'view_time_series']
```

The general procedure in SSA is as follows: 1. **Embed** the time series by forming a Hankel matrix of lagged window (length K) vectors. 2. **Decompose** the embedded time series via Singular Value Decomposition 3. **Eigentripple Grouping** is the process of identifying eigenvalue-eigenvector pairs as *trend*, *seasonal* and *noise* 4. **Reconstruct** the time series from the eigenvalue-eigenvector pairs identified as *trend* and *seasonal*. This is done through a process called *diagonal averaging*.

[ ]:
```
K = 14
suspected_seasonality = 0
```

[ ]:
```
ssa.embed(embedding_dimension=K, suspected_frequency=suspected_seasonality,␣
 ↪verbose=True)
```

```
----------------------------------------
EMBEDDING SUMMARY:
Embedding dimension     :  14
Trajectory dimensions   : (14, 15)
Complete dimension      : (14, 15)
Missing dimension       : (14, 0)
```

[ ]:
```
ssa.decompose(verbose=True)
```

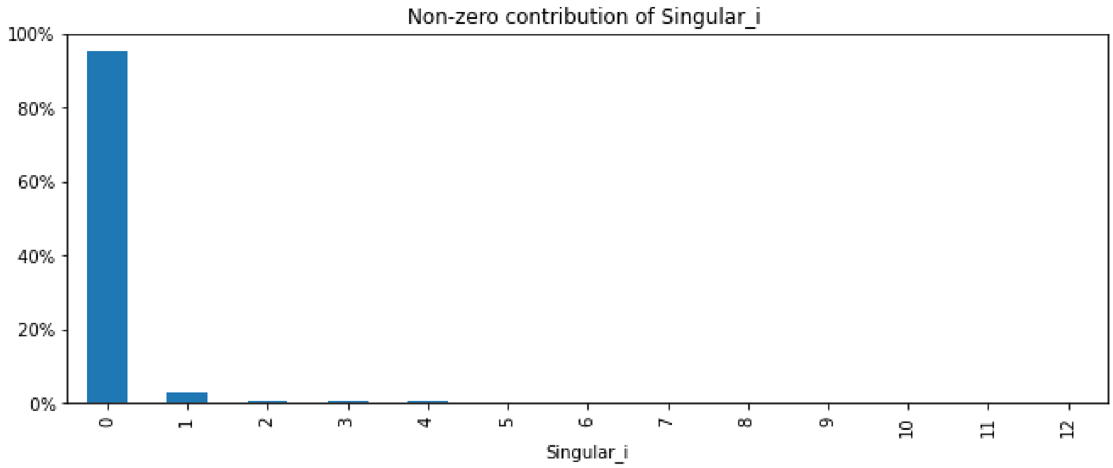
```
----------------------------------------
DECOMPOSITION SUMMARY:
Rank of trajectory              : 14
```

```
Dimension of projection space   : 13
Characteristic of projection    : 1.0
```

Contribution of each of the signals

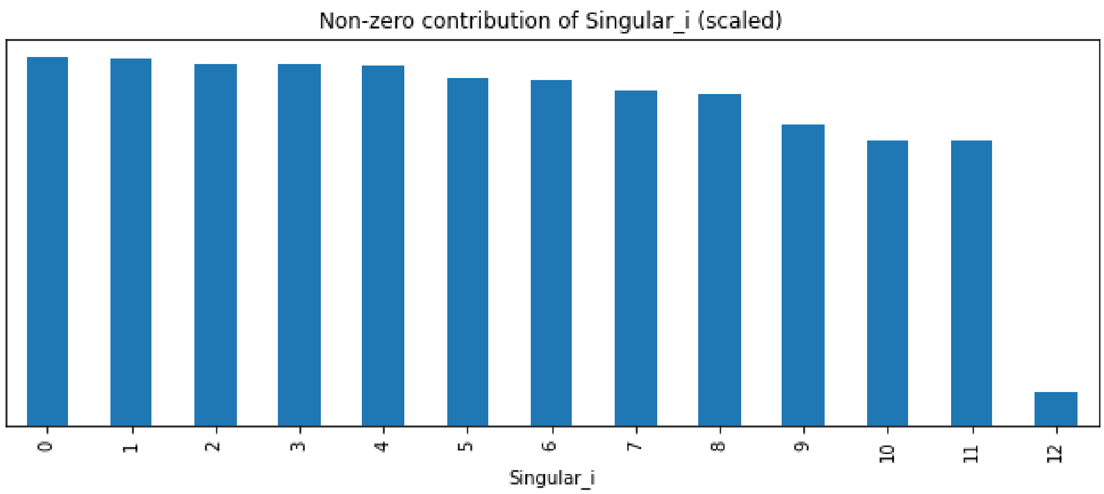
```
[ ]: # First enable display of graphs in the notebook
     from matplotlib.pylab import rcParams
     rcParams['figure.figsize'] = 11, 4

     ssa.view_s_contributions()
```



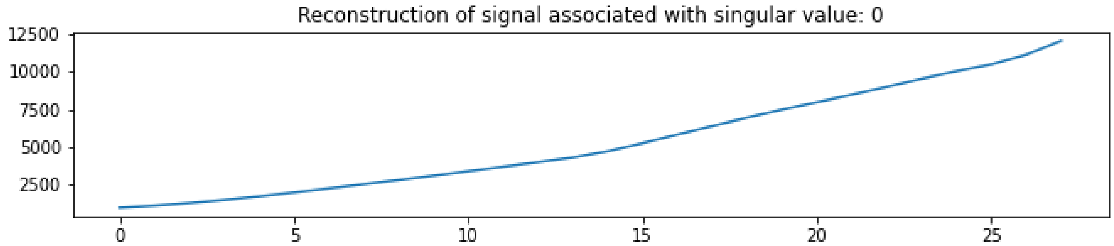
Eigenvalue groupings more clearly.

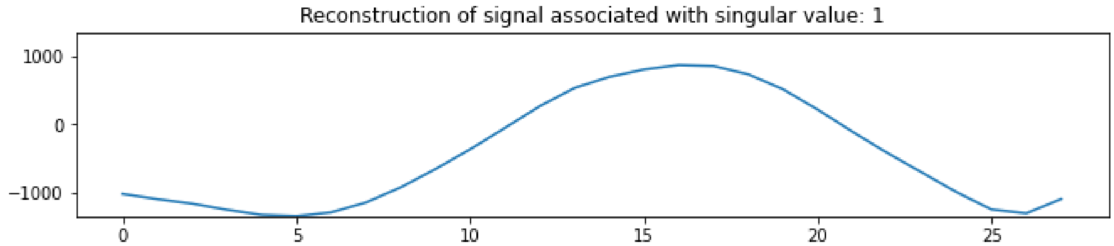
```
[ ]: ssa.view_s_contributions(adjust_scale=True)
```

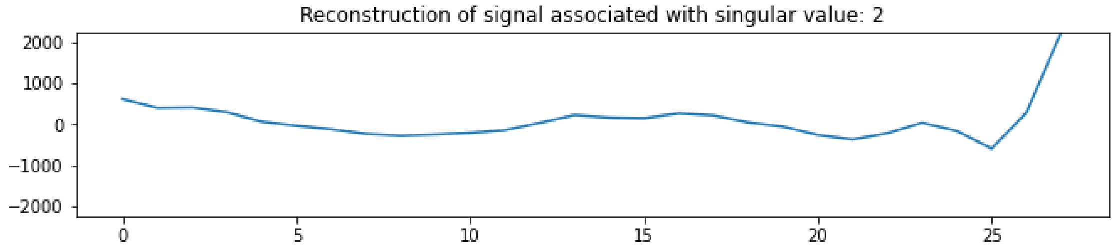


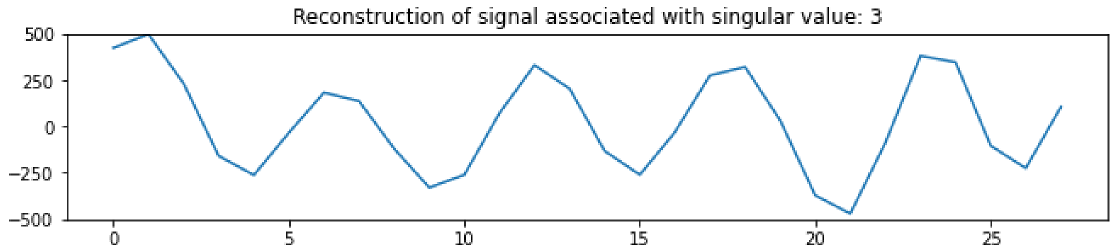
The additive signal elements are stored in the `object.Xs` *dictionary.*

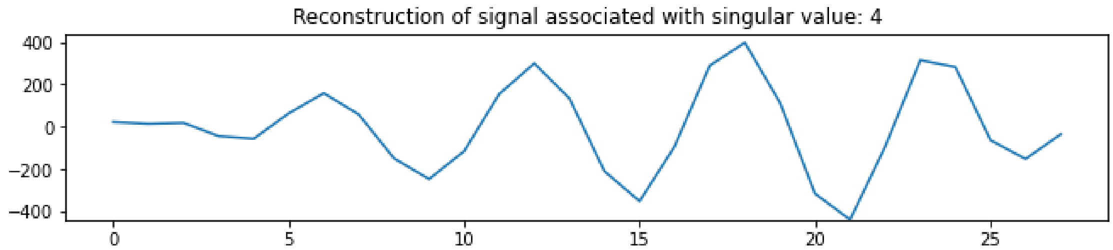
```
[ ]: rcParams['figure.figsize'] = 11, 2
     for i in range(5):
         ssa.view_reconstruction(ssa.Xs[i], names=i, symmetric_plots=i!=0)
     rcParams['figure.figsize'] = 11, 4
```



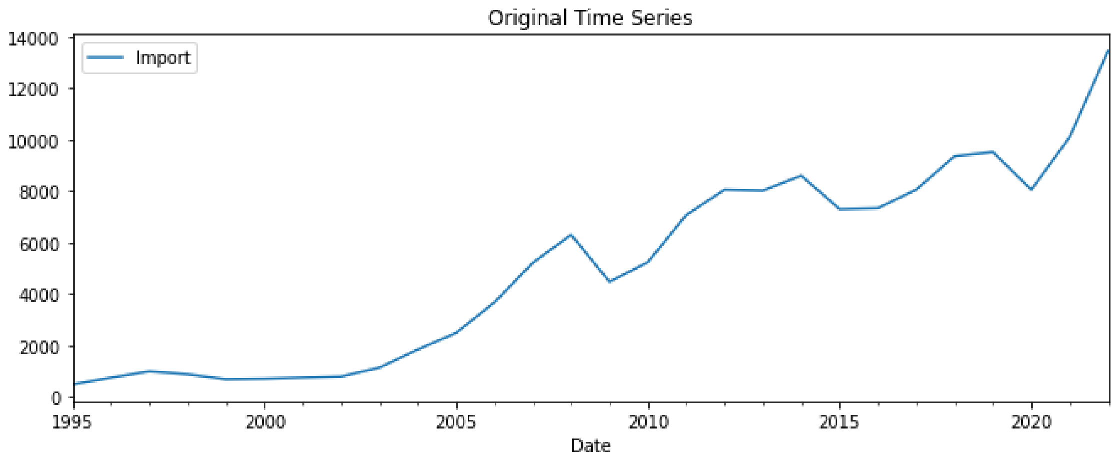


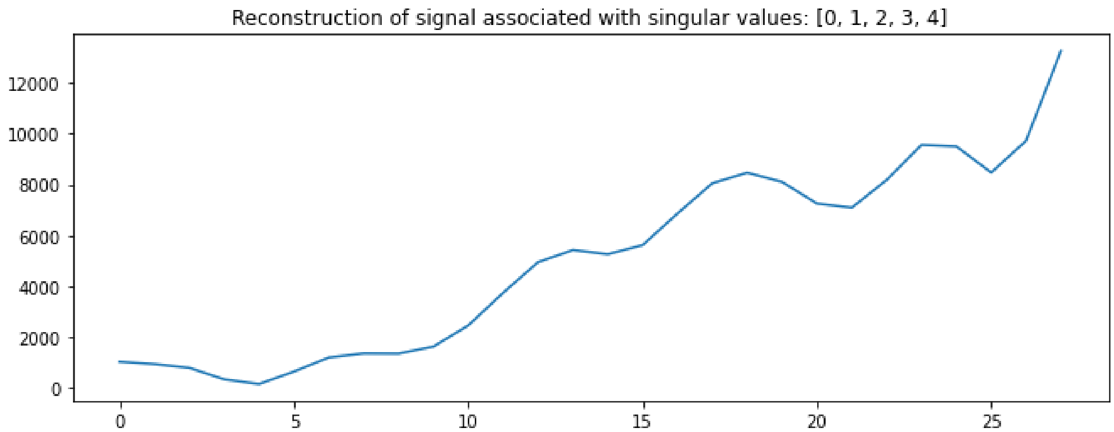




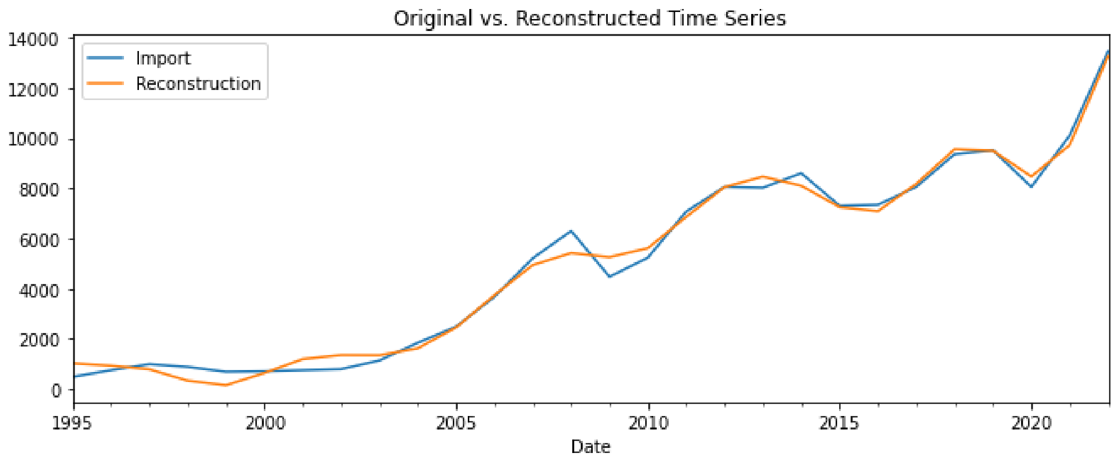


[ ]:
```
ssa.ts.plot(title='Original Time Series')
streams5 = [i for i in range(5)]
reconstructed5 = ssa.view_reconstruction(*[ssa.Xs[i] for i in streams5],␣
↪names=streams5, return_df=True)
```





```
[ ]: ts_copy5 = ssa.ts.copy()
     ts_copy5['Reconstruction'] = reconstructed5.Reconstruction.values
     ts_copy5.plot(title='Original vs. Reconstructed Time Series');
```



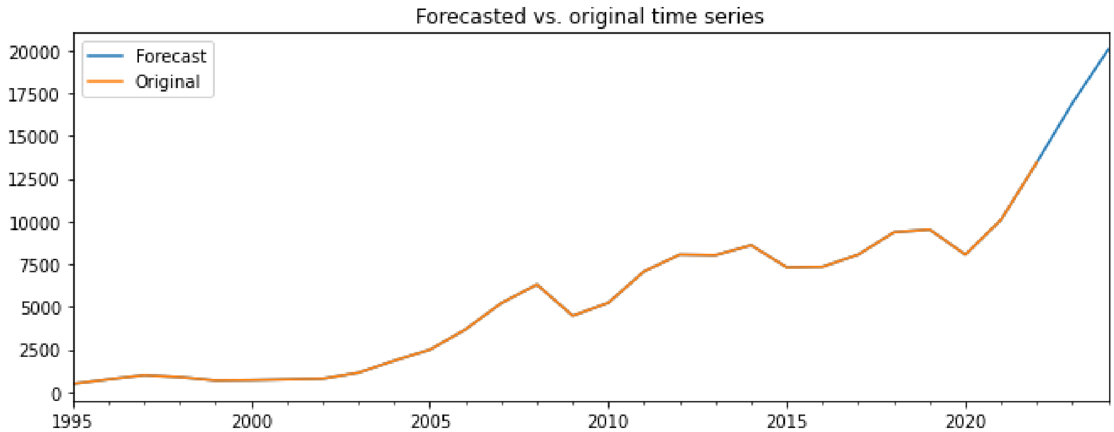

```
[ ]: def forecast_accuracy(forecast, actual):
         r2 = r2_score(actual, forecast)
         mape = np.mean(np.abs(forecast - actual)/np.abs(actual))
         me = np.mean(forecast - actual)
         mae = np.mean(np.abs(forecast - actual))
         mpe = np.mean((forecast - actual)/actual)
         rmse = np.mean((forecast - actual)**2)**.5
         corr = np.corrcoef(forecast, actual)[0,1]
         return({'r2':r2, 'mape':mape, 'me':me, 'mae': mae,
                 'mpe': mpe, 'rmse':rmse,'corr':corr})
```

```
[ ]: actual = np.array(ts_copy5['Import'])
     forecast = np.array(ts_copy5['Reconstruction'])
     metrics = forecast_accuracy(forecast, actual)
     print(pd.DataFrame(metrics, index=[0]))
```

```
         r2      mape        me         mae       mpe       rmse      corr
0  0.989148  0.194436 -2.109558  311.390787  0.036206  384.33361  0.994574
```

```
[ ]: fc = ssa.forecast_recurrent(steps_ahead=2, singular_values=streams5, plot=True,␣
      ↪return_df=True)
     print(fc['Forecast'][-2:])
```

```
2023-01-01    16931.646846
2024-01-01    20059.661628
Freq: AS-JAN, Name: Forecast, dtype: float64
```


Forecasted vs. original time series
Forecast
Original
20000
17500
15000
12500
10000
7500
5000
2500
0
1995
2000
2005
2010
2015
2020