PV degradation rate estimation using seasonal decomposition

Marios Theristis and Joshua S. Stein, Sandia National Laboratories

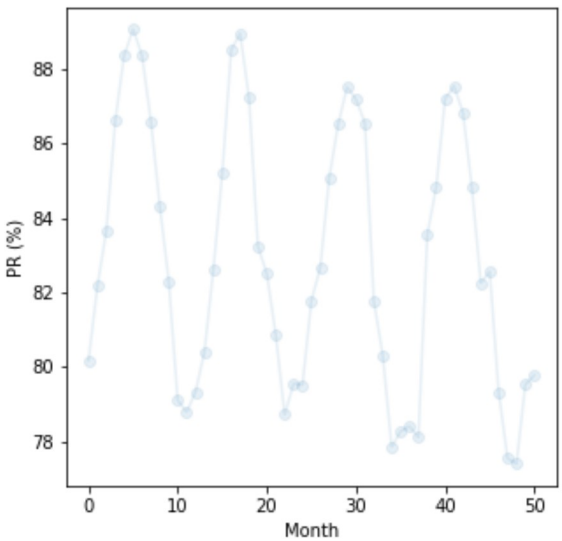
This notebook applies additive seasonal decomposition to the PV performance timeseries and OLS is applied on the trend in order to calculate the degradation rate (DR). More specifically, the time series are decomposed to the trend, seasonal and residual components (i.e. Y[t] = T[t] + S[t] + e[t]) using StatsModels.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import statsmodels.api as sm
from matplotlib import pyplot as plt
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

In [2]:
```
#reading the csv file that contains timeseries with operational and irradiance dat
a. In this example, we import the monthly performance ratio.
df = pd.read_csv(r'C:\...\Sample_data.csv', delimiter = ',' , parse_dates= ['Timest
amp'], dayfirst = False)
```

In [3]:
```
#plot the monthly PRs
fig, axs = plt.subplots(figsize=(5,5))
axs.plot(df.index, df.PR, 'o-', alpha = 0.1)
axs.set_ylabel('PR (%)');
axs.set_xlabel('Month')
```

Out[3]: Text(0.5, 0, 'Month')



Seasonal decomposition is applied using StatsModels. An additive model is used with a specific frequency (e.g. freq = 12 when using monthly data). Once the timeseries decomposition is done, a new dataframe with the trend values is created where the NaNs values caused by the moving average are removed. Finally, OLS is applied on the trend in order to calculate the absolute and relative DR as follows:

DR_abs = resolution * slope

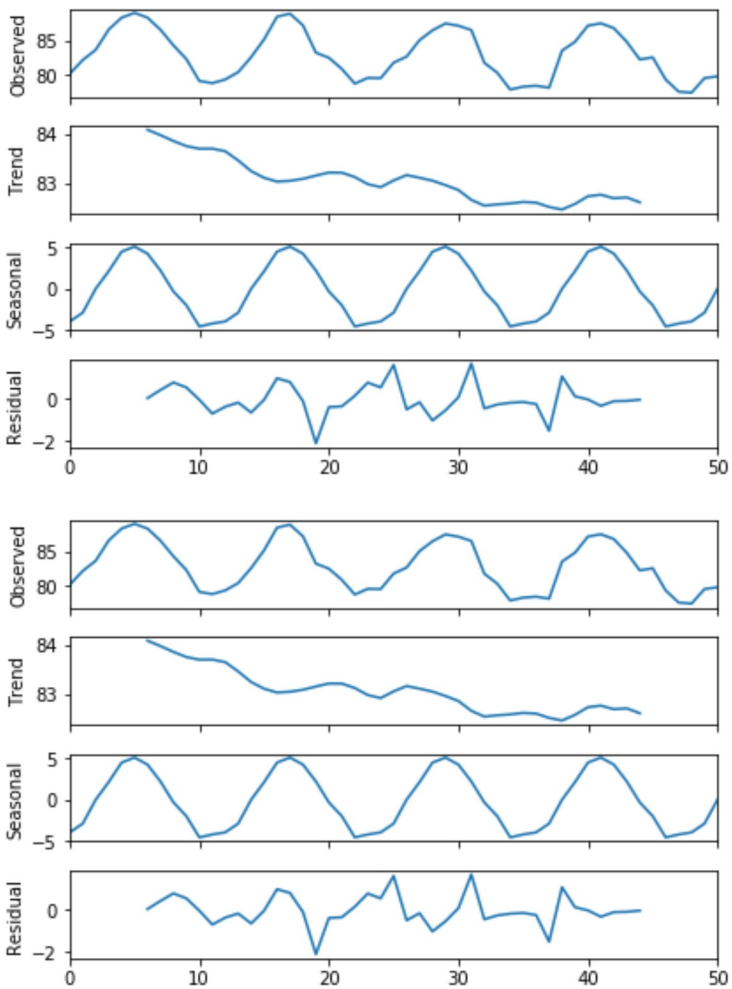
DR_rel = 100 *resolution* slope/intercept

Lower and upper confidence intervals are calculated for a confidence level of 95% (i.e. significance level, alpha = 0.05)

In [4]:
```
#daily 365, monthly 12 etc.
resolution = 12
```

In [5]:
```
s=sm.tsa.seasonal_decompose(df.PR, model = 'additive', freq = resolution)
s.plot()
```

Out[5]:



In [6]:
```
trend_df = pd.DataFrame(list(s.trend), columns = ['Trend'])
```

In [7]:
```
trend_df.insert(loc=0, column = 'Month', value=np.arange(len(trend_df)))
```

In [8]:
```
trend_df=trend_df[np.isfinite(trend_df['Trend'])]
```

Applying OLS on the trend:

In [9]:
```
y = trend_df.Trend
x = trend_df.Month
x, y = np.array(x), np.array(y)
```

In [10]:
```
x = sm.add_constant(x)
```

In [11]:
```
model = sm.OLS(y,x)
```

In [12]:
```
results = model.fit()
```

In [13]:
```
results.summary()
```

Out[13]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.819
Model:	OLS	**Adj. R-squared:**	0.814
Method:	Least Squares	**F-statistic:**	167.0
Date:	Mon, 16 Dec 2019	**Prob (F-statistic):**	2.76e-15
Time:	09:47:40	**Log-Likelihood:**	10.655
No. Observations:	39	**AIC:**	-17.31
Df Residuals:	37	**BIC:**	-13.98
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	83.9292	0.074	1138.209	0.000	83.780	84.079
x1	-0.0348	0.003	-12.921	0.000	-0.040	-0.029

Omnibus:	4.232	**Durbin-Watson:**	0.232
Prob(Omnibus):	0.121	**Jarque-Bera (JB):**	1.759
Skew:	0.062	**Prob(JB):**	0.415
Kurtosis:	1.967	**Cond. No.**	66.9

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [14]:
```
intercept, slope = results.params
```

In [15]:
```
#confidence level 95%
CI_abs = resolution*results.conf_int(alpha  = 0.05)[1]
CIL_abs = CI_abs[1]
CIH_abs = CI_abs[0]
round(CIL_abs,2),round(CIH_abs,2)
```

Out[15]: (-0.35, -0.48)

In [16]:
```
DR_abs = round(resolution*slope, 2)
DR_abs
```

Out[16]: -0.42

In [17]:
```
#confidence level 95%
CI_rel = 100*resolution*results.conf_int(alpha  = 0.05)[1]/intercept
CIL_rel = CI_rel[1]
CIH_rel = CI_rel[0]
round(CIL_rel,2),round(CIH_rel,2)
```

Out[17]: (-0.42, -0.57)

In [18]:
```
DR_rel = round(100*resolution*slope/intercept,2)
DR_rel
```

Out[18]: -0.5

In [19]:
```
#using seaborn to plot the monthly PR and linear regression model fit
sns.regplot(y = trend_df['Trend'], x = trend_df['Month'], data = df).set_ylabel("PR
(%)")
plt.xlabel("Month")
plt.ylim(80, 85)
plt.xlim(0, 50)
plt.title("Relative degradation rate of -0.50%/year on CSD trend")
plt.show(fig)
```

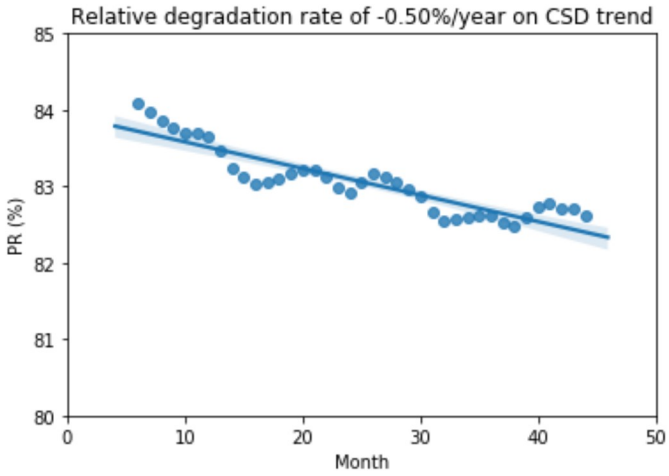


Table of results

In [20]:
```
Results = [['Relative DR', round(DR_rel,2) ], ['Relative CIL', round(CIL_rel,2)],
['Relative CIH', round(CIH_rel,2)], ['Absolute DR', round(DR_abs,2)], ['Absolute CI
L', round(CIL_abs,2)], ['Absolute CIH', round(CIH_abs,2)]]

# Create the pandas DataFrame
Results = pd.DataFrame(Results, columns = ['Parameter', 'Value (%/year)'])

Results
```

Out[20]:

	Parameter	Value (%/year)
0	Relative DR	-0.50
1	Relative CIL	-0.42
2	Relative CIH	-0.57
3	Absolute DR	-0.42
4	Absolute CIL	-0.35
5	Absolute CIH	-0.48

More information about the classical decomposition method can be found in:

Hyndman, Rob J., and George Athanasopoulos. Forecasting: principles and practice. OTexts, 2014.

https://otexts.com/fpp2/classical-decomposition.html (https://otexts.com/fpp2/classical-decomposition.html)