PV degradation rate estimation using Holt-Winters seasonal exponential smoothing

Marios Theristis and Joshua S. Stein, Sandia National Laboratories

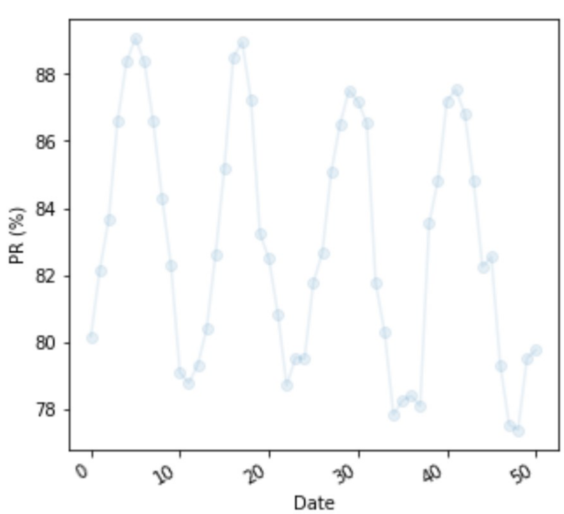
This notebook applies the Holt-Winters seasonal exponential smoothing on PV performance timeseries. This model is commonly used for forecasting timeseries that exhibit trend and seasonality and therefore, it is a good candidate for PV degradation studies. In addition to Holt's linear method equations for level (i.e. overall smoothing), trend smoothing and forecast, the Holt-Winters additive seasonality model also computes a seasonal smoothing equation. The degradation rate (DR) is then estimated by applying OLS on the overall smoothing.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
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]:
```
df.drop(columns=['Month', 'Timestamp'], inplace=True)
```

In [4]:
```
#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('Date')
fig.autofmt_xdate()
```



Triple exponential smoothing, i.e. the Holt-Winters method, is applied using the *statsmodels.tsa.holtwinters.ExponentialSmoothing*. Here we have four equations: level, trend, seasonal and forecast. Since we use monthly data, the seasonal periods are set to 12 months whereas the trend and seasonal components are set to additive. Once the timeseries are smoothed, a new dataframe with the level smoothing is created and OLS is applied 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 [5]:

```
#daily 365, monthly 12 etc.
resolution = 12
```

In [6]:

```
model = ExponentialSmoothing(df, trend='add', seasonal='add', seasonal_periods= resolution, damped = False).fit(optimized=True, use_boxcox=False, remove_bias=False)
results=pd.DataFrame(index=[r"$\alpha$",r"$\beta$",r"$\phi$",r"$\gamma$",r"$l_0$","$b_0$","SSE"])
params = ['smoothing_level', 'smoothing_slope', 'damping_slope', 'smoothing_seasonal', 'initial_level', 'initial_slope']
results["Additive"] = [model.params[p] for p in params] + [model.sse]
results
```

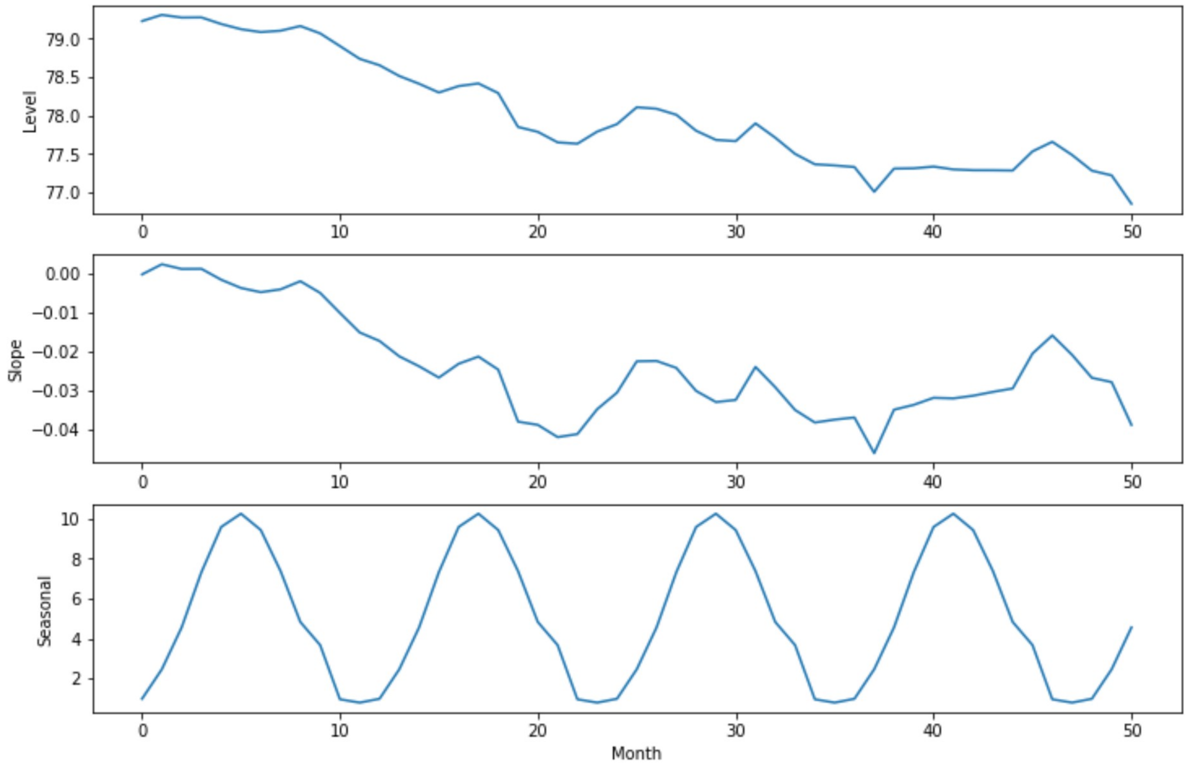
Out[6]:

	Additive
α	0.172210
β	0.032038
ϕ	NaN
γ	0.000000
l_0	79.239639
b_0	0.000000
SSE	37.678795

In [7]:

```
df = pd.DataFrame(np.c_[df, model.level, model.slope, model.season, model.fittedvalues],
                    columns=['original','level','slope','seasonal','fitted values'],index=df.index)
```

In [8]:
```
fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(12,8))
ax1.plot(df.index, df.level)
ax1.set_ylabel('Level')
ax2.plot(df.index, df.slope)
ax2.set_ylabel('Slope')
ax3.plot(df.index, df.seasonal)
ax3.set_ylabel('Seasonal')
ax3.set_xlabel('Month')
plt.show()
```



In [9]:
```
model.summary()
```

Out[9]:

ExponentialSmoothing Model Results

Dep. Variable:	endog	**No. Observations:**	51
Model:	ExponentialSmoothing	**SSE**	37.679
Optimized:	True	**AIC**	16.561
Trend:	Additive	**BIC**	47.470
Seasonal:	Additive	**AICC**	37.936
Seasonal Periods:	12	**Date:**	Mon, 16 Dec 2019
Box-Cox:	False	**Time:**	09:51:01
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.1722104	alpha	True
smoothing_slope	0.0320378	beta	True
smoothing_seasonal	0.000000	gamma	True
initial_level	79.239639	l.0	True
initial_slope	0.000000	b.0	True
initial_seasons.0	0.9871744	s.0	True
initial_seasons.1	2.4655410	s.1	True
initial_seasons.2	4.5605158	s.2	True
initial_seasons.3	7.3278315	s.3	True
initial_seasons.4	9.5915435	s.4	True
initial_seasons.5	10.255753	s.5	True
initial_seasons.6	9.4366507	s.6	True
initial_seasons.7	7.3812038	s.7	True
initial_seasons.8	4.8318440	s.8	True
initial_seasons.9	3.6767799	s.9	True
initial_seasons.10	0.9615431	s.10	True
initial_seasons.11	0.7963274	s.11	True

In [10]:
```
df.drop(columns=['slope', 'seasonal', 'fitted values'], inplace=True)
```

In [11]:
```
df.insert(loc=0, column = 'Month', value=np.arange(len(df)))
```

Applying OLS on level smoothing:

In [12]:
```
y = df.level
x = df.Month
x, y = np.array(x), np.array(y)
```

In [13]:
```
x = sm.add_constant(x)
```

In [14]:
```
model = sm.OLS(y,x)
```

In [15]:
```
results = model.fit()
```

In [16]:
```
results.summary()
```

Out[16]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.877
Model:	OLS	**Adj. R-squared:**	0.874
Method:	Least Squares	**F-statistic:**	348.7
Date:	Mon, 16 Dec 2019	**Prob (F-statistic):**	6.33e-24
Time:	09:51:13	**Log-Likelihood:**	-2.1799
No. Observations:	51	**AIC:**	8.360
Df Residuals:	49	**BIC:**	12.22
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	79.1781	0.071	1113.532	0.000	79.035	79.321
x1	-0.0458	0.002	-18.675	0.000	-0.051	-0.041

Omnibus:	0.616	**Durbin-Watson:**	0.323
Prob(Omnibus):	0.735	**Jarque-Bera (JB):**	0.463
Skew:	-0.229	**Prob(JB):**	0.793
Kurtosis:	2.913	**Cond. No.**	57.2

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

In [17]:
```
intercept, slope = results.params
```

In [18]:
```
#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[18]: (-0.49, -0.61)

In [19]:
```
DR_abs = round(resolution*slope, 2)
DR_abs
```

Out[19]: -0.55

In [20]:
```
#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[20]: (-0.62, -0.77)

In [21]:
```
DR_rel = round(100*resolution*slope/intercept,2)
DR_rel
```

Out[21]: -0.69

In [22]:
```
fig = sns.regplot(y=df.level, x=df.Month, data=df).set_ylabel("PR Level (%)")
plt.title("Relative degradation rate of -0.69%/year using the HW model")
plt.ylim(75, 80)
plt.show(fig)
```

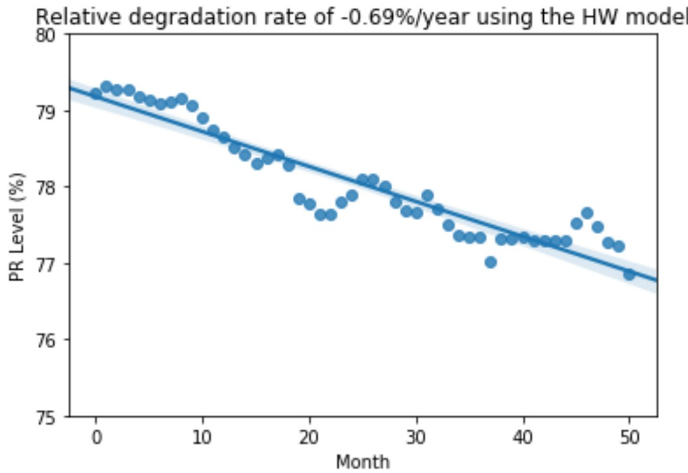


Table of results

In [23]:
```
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[23]:

	Parameter	Value (%/year)
0	Relative DR	-0.69
1	Relative CIL	-0.62
2	Relative CIH	-0.77
3	Absolute DR	-0.55
4	Absolute CIL	-0.49
5	Absolute CIH	-0.61

More information about the Holt-Winters method can be found in:

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

[https://otexts.com/fpp2/holt-winters.html (https://otexts.com/fpp2/holt-winters.html)](https://otexts.com/fpp2/holt-winters.html)