Kalman Filter Linear Regression

Prashant Tak
(2018B4A81050P)

Objective

Energy Prediction via DLMs

- Choose a Dynamic Linear Model suitable for our data
- Find optimal parameters for the same
- Analyse forecast results
- [Optional] Comparative study of DLMs and WS-ARIMA, if time permits

Why use Dynamic Linear Models?

Statistical analysis of time series data is usually faced with the problem that we have only one realization of a process whose properties we might not fully understand.

In linear trend analysis, we assume that there is an underlying change in the background that stays approximately constant over time.

In dynamic regression systems, by explicitly allowing for variability in the regression coefficients we let the system properties change in time. Also, unlike ARMA models, they can be applied to non-stationary data without transformation.

Furthermore, the use of unobservable state variables allows direct modelling of the processes that are driving the observed variability, such as seasonality or external forcing, and we can explicitly allow for some modelling error.

Ref: [DLM Tutorial - MJ Laine]

How do DLMs work? What is a Kalman Filter?

DLM (Gaussian Linear State Space Model)

State space models consider a time series as the output of a dynamic system perturbed by random disturbances. They allow a natural interpretation of a time series as combination of trend, seasonal or regressive components. In a state space model we assume that there is an unobservable Markov chain (x_t), called the *state process*, and that y_t is an *imprecise measurement* of x_t. A trivial DLM consists of two sets of equations:

- $y_t = F_t x_t + v_t$ (Observation Equation)
- $x_t = G_t x_{t-1} + w_t$ (Model Equation)

Here y_t represents the observation at time t, v_t and w_t are sequences of independent gaussian random errors (*observation error and evolution error*) and x_t corresponds to the unobserved state of the system having a *prior distribution* for $x_0 \sim N(m_0, C_0)$. F_t and G_t are the *observation* and *system matrices*. A linear regression model (with lagged values of observation as regression variable) would look like:

- $y_t = (y_{t-1} x_t) + v_t$ *where* $v_t \sim N(0, V_t)$
- $x_t = G_t x_{t-1} + w_t$ *where* $w_t \sim N(0, W_t)$

V_t and W_t are variance matrices.

Ref: DLM with R, Petris et al.

Kalman Filter

Model building can be a major difficulty: there might be no clear identification of physically interpretable states, or the state space representation could be non unique, or unsuitable choice of parameters could result in an inadequate model.

To estimate the state vector we compute the *conditional densities* $\pi(x_s|y_{1:t})$. We distinguish between problems of *filtering* (when s = t), *state prediction* (s > t) and *smoothing* (s < t).

In a DLM, the Kalman filter provides the formula for updating our current inference on the state vector as new data become available, that is, for passing from the *filtering density* $\pi(x_t|y_{1:t})$ to $\pi(x_{t+1}|y_{1:t+1})$.

It allows us to compute the *predictive* and *filtering distributions* recursively, starting from $x_0 \sim N(m_0, c_0)$ then computing $\pi(x_1|y_1)$, and proceeding recursively as new data becomes available. This is the usual Bayesian sequential updating, in which the *posterior at time t* takes the role of a *prior distribution* for what concerns the observations after time t.

*Posterior = (Likelihood*Prior)/Evidence*
Prior : Prob. of event without observation
Evidence : The observation
Likelihood : Prob. of observation given event
Posterior : Updated belief considering both prior & obs.

Ref: DLM with R, Petris et al.

How does the filter work?

Filtering

Taking the vector of observations $y_{1:t}$, the *filtering distribution* $\pi(x_t|y_{1:t})$ is computed recursively as

1. Start with $x_0 \sim N(m_0, C_0)$
2. One step forecast for the *state*:

$$x_t|y_{1:t-1} \sim N(a_t, R_t) \text{ where } a_t = G_t m_{t-1} \text{ and } R_t = (G_t C_{t-1} G'_t) + W_t$$

3. One step forecast for the *observation*:

$$y_t|y_{1:t-1} \sim N(f_t, Q_t) \text{ where } f_t = F_t a_t \text{ and } Q_t = (F_t R_{t-1} F'_t) + V_t$$

4. Compute the *posterior* at time t:

$$x_t|y_{1:t} \sim N(m_t, C_t) \text{ where } m_t = a_t + R_t f'_t Q_t^{-1}(y_t - f_t) \text{ and } C_t = R_t - (R_t F'_t Q_t^{-1} F_t R_t)$$

Ref: [Lalas : KF]

Smoothing and Forecasting

Smoothing deals with estimating the state sequence $x_{1:t}$ given data $y_{1:t}$. Backward recursive algorithm can be used to obtain $\pi(x_t|y_{1:T})$ for fixed T and $t = 0{:}T$

1. Start with $x_T|y_T \sim N(m_T, C_T)$ at time $t = T$
2. For t in T-2 to 0:

$$x_t|y_{1:T} \sim N(s_t, S_t)$$

where $s_t = m_t + C_t G'_{t+1} R^{-1}{}_{t+1}(s_{t+1} - a_{t+1})$

and

$$S_t = C_t - (C_t G'_{t+1} R^{-1}{}_{t+1}(R_{t+1} - S_{t+1}) R^{-1}{}_{t+1} G'_{t+1} C_t)$$

To calculate *forecast distributions*, for $k = 1, 2, \ldots$

1. Start with a sample from $x_T|y_T \sim N(m_T, C_T)$
2. Forecast the *state*:

$$x_{T+k}|y_{1:T} \sim N(a_k{}^T, R_k{}^T)$$

where $a_t{}^k = G_{T+k} a^T{}_{k-1}$

and $R_k{}^T = G_{T+k} R^T{}_{k-1} G'_{T+k} + W_{T+k}$

3. Forecast the *observation*:

$$y_{T+k}|y_{1:T} \sim N(f_k{}^T, Q_k{}^T)$$

where $f_t{}^k = F_{T+k} a^T{}_{k-1}$

and $Q_k{}^T = F_{T+k} R^T{}_{k-1} F'_{T+k} + V_{T+k}$

Ref: [Lalas : KF]

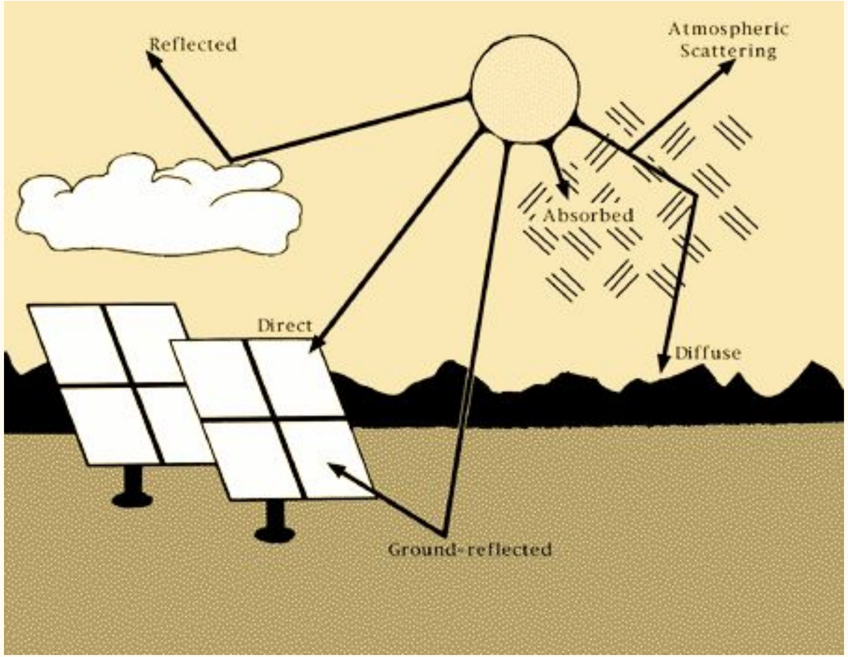
So what are we trying to estimate?

GHI of Bhadla Solar Park *[Clearsky GHI and Wind Speed]*

GHI *(Global Horizontal Irradiance)* refers to the amount of radiant power from sunlight received by a particular surface, perpendicular to the sun's rays.

It is useful when monitoring a solar power plant, finding optimal placement location of solar plants and for assessing solar power plant feasibility.

The higher the GHI value, the more power a system will produce. Its measurement unit is Watts per Sq Meter (W/m^2).



Source: [What is GHI - Solar Powered Blog]

Implementation

[Github : DLM]

```python
### TODO
# 1. Tuning parameters
# 2. Numerical analysis of prediction
# 3. Connect used functions with the math
###

# Read and extract needed data
df = pd.read_csv('Bhadla new.csv')
sort_cols = ["Year", "Month", "Day", "Hour"]
df.sort_values(by=sort_cols, inplace=True)
df = df[(df["Hour"]> 7) & (df["Hour"] < 17)]
df = df[["Year", "Month", "Day", "Hour", "GHI",
"Clearsky GHI"]]
ghi = df[["GHI"]]

# Split data 80 training - 20 testing
train = ghi.head(int(len(ghi)*0.8))
test = ghi.tail(int(len(ghi)*0.2))
regressor = ghi.tail(int(len(ghi)-1))
ghi = ghi.head(int(len(ghi)-1))
```

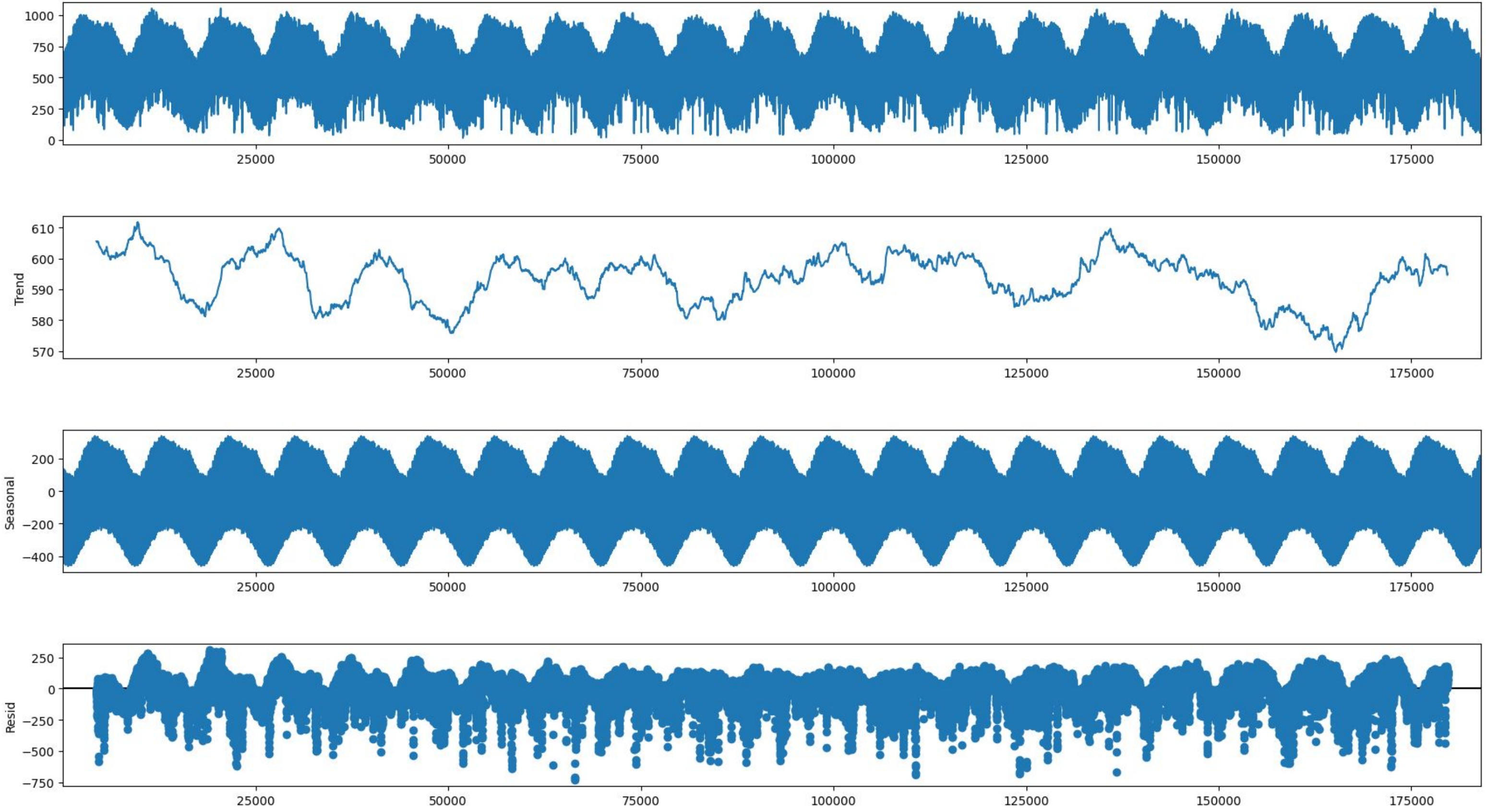
```python
# TS Decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(ghi, model='additive',
period=12*30*9)
result.plot()
plt.show()

# DLM Design, Filtering and Smoothening
dyn = dynamic(features = regressor, discount = 1, name
= 'b', w=10)
longSeas = longSeason(period=12, data=ghi, name="Yearly
Seasonality") # stay=30*9
ghiDLM = dlm(ghi) + dyn + longSeas
ghiDLM.fit()
ghiDLM.plot()

# Prediction
ghiDLM.plotPredictN(N=len(test)-1, date = len(train)-1)
(predictMean, predictVar) =
ghiDLM.predictN(N=len(test)-1, date=len(train)-1)
ghiDLM.getMSE()
residual = ghiDLM.getResidual(filterType='predict')
plt.plot(residual[-len(test)+1:])
```

What did we get?

Decomposition

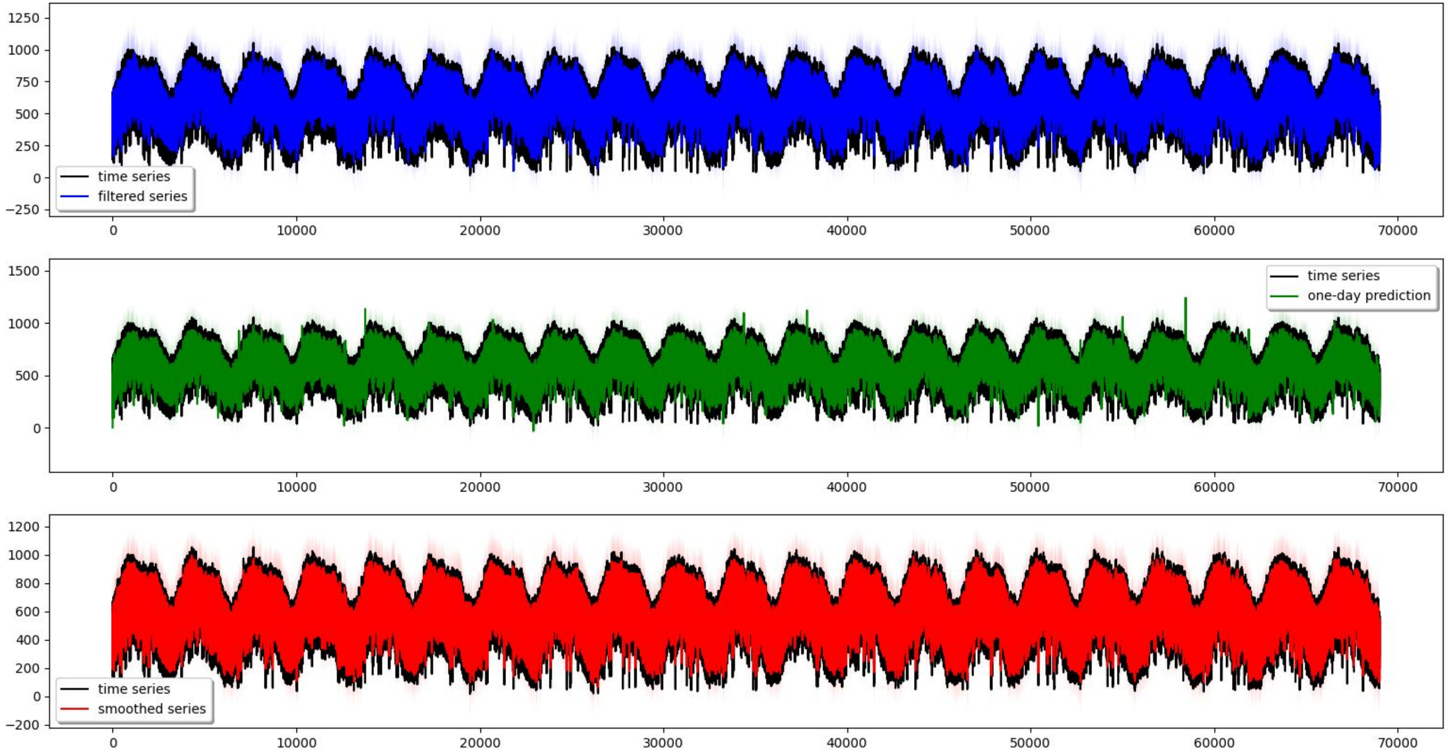


ADF Statistic:
-8.77047

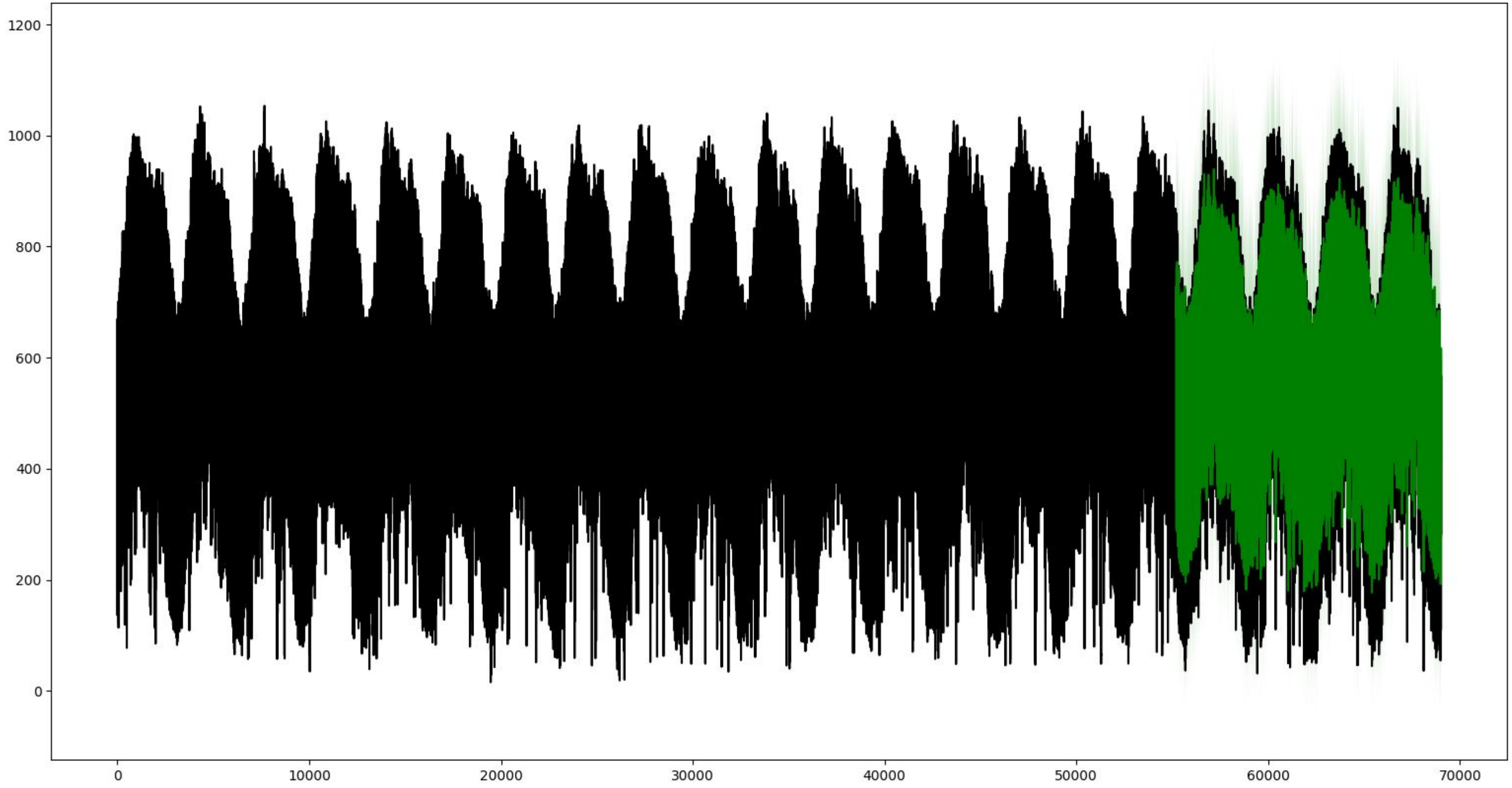
p-value:
$2.529\text{x}10^{-14}$

The series has no unit root, it is stationary.

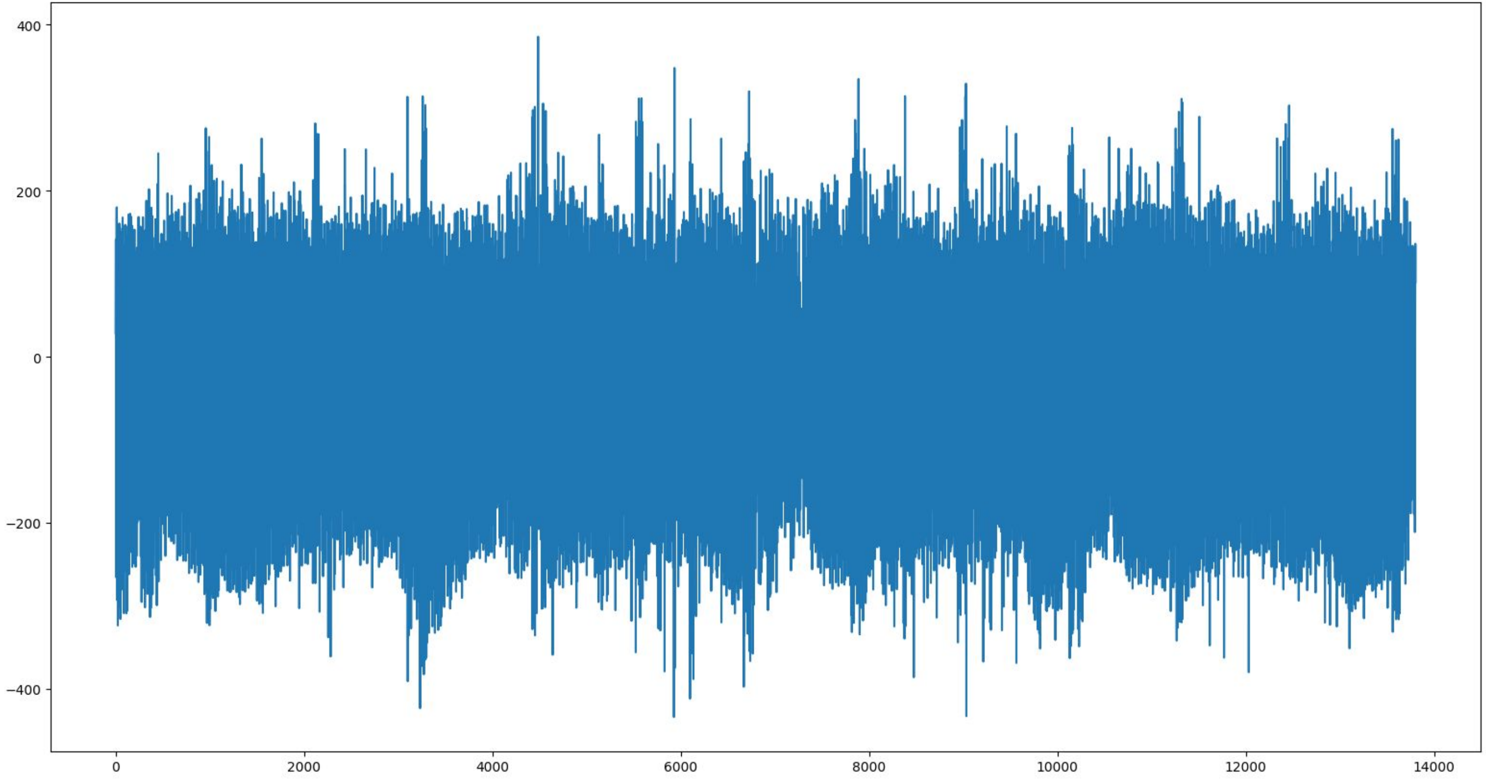
Filtering and Smoothing



Prediction


1200
1000
800
600
400
200
0
0
10000
20000
30000
40000
50000
60000
70000

Residuals *(for predicted values)*



The residuals of a forecast model should exhibit Gaussian distribution with zero mean and a constant variance.

Challenges (so far) and Future Tasks

- Lack of literature tackling actual implementation
- Switching between R and python
- Long period of no tangible progress due to broken pyDLM library
- Data not being in standard form (9 hours/day)
- Get concrete, numerical inferences
- Fix decomposition model, resample to monthly & weekly and analyse
- Get Clearsky GHI and Wind Speed Estimates

Thank You