Dynamic Exponential Smoothing

Rojs Aktumanis

February 4, 2024

Abstract

Often time-based sensors, such as, temperature sensors, are inaccurate and add some random noise to the true data. The goal of the paper is to apply real-time data smoothing of such data by generating a smooth point as a weighted mean of the point and the previous smooth point. This weight is made dynamic by making it dependent on the distribution of errors in a certain parameterized time window. Moreover, we introduce a combination of criteria for model selection that using the error distribution and second order derivative approximation.

1 Proposed trend extraction theory

1.1 Method description and assumptions

Let $(a_t)_{t=1}^m$ be a time series sequence in $\mathbb{R}$ with some noise, such that, $m \in \mathbb{N}$ is the length of our data set. Our goal is to remove the noise from the time series sequence.

We use a time **window** of length $w \in \mathbb{N}$ to help us find a smooth point. For an observation a_n the time window consists of data points at indices $t \in \{n-w, n-w+1, ..., n-1\}$.

The observed time series sequence $(a_t)_{t=1}^m$ is assumed to be made from 2 components. The **random noise** component $(n_t)_{t=1}^m$, such that,

$$n_t \sim Normal(0, \sigma_t^2)$$

for some $\sigma_t^2 \in \mathbb{R}_{\geq 0}$ that is dependent on the data in the time window. We assume that the variance changes over time.
The other component is what we wish to extract from the noisy data $(a_t)_{t=1}^m$. This is the underlying function whose values are described in the sequence $(f_t)_{t=1}^m$. The **trend** can be expressed as

$$f_t = a_t - n_t$$

Our goal is to produce an estimation $\hat{f}_t$ of f_t by using a Dynamic exponential filtering method. To do this we use the errors of our previous estimates $\hat{f}_t$ and our assumption that the true errors n_t are normally distributed. If $\hat{f}_t$ approximates f_t well, then the errors should have a mean value near 0, however, if this is not the case, then the mean value will be smaller or larger, hence we can adjust our exponential filtering parameter k accordingly for our next data point.

Since we use the exponential filter as our basis for the model, we calculate our approximation of f_t as follows:

$$\hat{f}_t = \hat{f}_{t-1}(1-k_t) + a_t k_t, \text{ for } t > 1$$

Where k_t is the exponential filter coefficient at t. We are not able to use our algorithm for data sets smaller than the time window, therefore, we define the parameter **default k** k_d, which is the weight coefficient for the initial data set, such that $k_t = k_d$ for $t \leq w$. This will produce an incorrect approximation of f_t, however, in most cases with a reasonable k_d the long term effect of this parameter will be next to none since the weight coefficient will dynamically adjust to the data. Now, the starting smooth data set $(\hat{f}_t)_{t=1}^w$ will be as follows

$$\hat{f}_1 = a_1$$

$$\hat{f}_t = \hat{f}_{t-1}(1-k_d) + a_t k_d, \text{ for } 1 < t \leq w$$

For future smooth points, we will use a different weight coefficient for each point. Now, we define the rest of the parameters. **Rate of change**

$$r \in [0, 1]$$

This defines the absolute amount by which the weight parameter k_t can change from t to $t + 1$, i.e.,

$$k_{t+1} \in \{\max(k_t - r, 0), \min(k_t + r, 1)\}$$

A larger r will adapt to changes in data quicker than a smaller one. Additionally, we define the **significance level**

$$\alpha \in (0, 1)$$

The goal of the model will be to find the right values of k_t, such that the error distribution stays centered at 0. The significance level will help us define *how close* to 0 does the mean error in the time window needs to be for us to say our model is working properly and not make changes to the coefficient.

Let

$$e_t := \frac{1}{w}\Sigma_{i=t-w}^{t-1}(\hat{f}_i - a_i)$$

Be our approximation of the error in the time window ending before the point t. Since this is just an approximate based on our data (hence might not be exactly 0), there exists a true value of the mean error, call it μ_t, such that $e_t \sim \text{Normal}(\mu_t, \sigma_t^2)$ for some variance σ_t^2. This is our approximation for n_t. We want to test whether there is enough evidence that $\mu_t = 0$ or $\mu_t \neq 0$ to determine whether our current smooth points approximate f_t well (mean error close to 0) or not. If our current smooth points do not approximate f_t well, then it will become apparent by observing the mean error, thus we will be able to make appropriate changes to our $\hat{f}_t$ to make it a better approximation. Then we define the following hypothesis:

$$H_0 : \mu_t = 0$$

$$H_A : \mu_t \neq 0$$

The goal of the significance level is to determine how close does e_t needs to be to 0 for us to say the mean error (μ_t) is likely 0. A larger significance level will generally produce a smaller interval around 0, so we will be less likely to say that the true mean error is 0. A smaller significance level will allow larger $|e_t|$ values to classify the true mean error as 0. The recommended value for α is $\alpha \leq 0.05$, however, it should still be tested through experiments.

Now that the model is initialized, we take the next point after initialization (a_{w+1}) to demonstrate how the model works. Since we used the weight coefficient k_d for all of the initial data set, $k_w = k_d$ describes the previous weight coefficient used.

Now, we take the regular and smooth data from the previous time window and calculate the approximation of the mean error

$$e_{w+1} = \frac{1}{w}\Sigma_{i=1}^{w}(\hat{f}_i - a_i)$$

Now we preform the statistics test to classify whether our observation of e_{w+1} is consistent with the underlying μ_{w+1} being 0 or not.

In the case that μ is likely 0 (H_0 true), we do not change k, since we have found a good optimum. So,

$$k_{w+1} = k_w$$

Now, if the error mean is unlikely 0 (H_1 true), we choose the new k_{w+1}.
We have 2 choices - either to decrease or increase k, so:

$$k_{w+1,\text{low}} = max(k_w - r, 0)$$
$$k_{w+1,\text{high}} = min(k_w + r, 1)$$

Then, we calculate the potential smooth data points:

$$\hat{f}_{w+1,\text{low}} = k_{w+1,\text{low}} a_{w+1} + (1 - k_{w+1,\text{low}})\hat{f}_w$$

$$\hat{f}_{w+1,\text{high}} = k_{w+1,\text{high}} a_{w+1} + (1 - k_{w+1,\text{high}})\hat{f}_w$$

If $\mu_{w+1} > 0$, then we want to take the value that makes our mean error closer to 0.

$$k_{w+1} = \begin{cases} k_{w+1,\text{low}}, & : \hat{f}_{w+1,\text{low}} \leq \hat{f}_{w+1,\text{high}} \\ k_{w+1,\text{high}}, & : \text{otherwise} \end{cases} \tag{1}$$

And if $\mu_{w+1} < 0$, then we want to take the value that increases our mean error to get it closer to 0.

$$k_{w+1} = \begin{cases} k_{w+1,\text{low}}, & : \hat{f}_{w+1,\text{low}} \geq \hat{f}_{w+1,\text{high}} \\ k_{w+1,\text{high}}, & : \text{otherwise} \end{cases} \tag{2}$$

Hence, we calculate the next smooth point

$$\hat{f}_{w+1} = k_{w+1} a_{w+1} + (1 - k_{w+1})\hat{f}_w$$

We repeat this process for every consecutive point by moving the time window each time.

1.2 Parameters

To summarize the parameters required for building the model we present a table.

Parameter	Name	Range	Description
w	Window size	$\mathbb{N}$	Size of window for calculating errors
d_k	Default k	[0, 1]	Default coefficient applied to values in first time window
r	Rate of change	[0, 1]	Absolute maximum value for changing coefficient k between points
α	Significance level	(0, 1)	Sensitivity measure of testing for mean errors

1.3 Parameter fitting

We propose an approach of fitting hyper-parameters $\theta = (w, d_k, r, \alpha)$ for any data smoothing/filtering model. Our goal is to limit the noise of the smooth line and ensure that the error stays evenly distributed on both sides (mean error is close to 0).
Assume, we have our original data set as $(a_t)_{t=1}^n$, model M_θ with hyper-parameters θ, and our smooth data set $(s_t)_{t=1}^n$, such that the smooth data set is produced as follows:

$$s_m := M_\theta((a_t)_{t=1}^{m-1})$$

Where the smooth data point at time m is produced based on the hyper-parameters θ and all the previous regular points.
We want to find the set of parameters that acquire the following simultaneously:

- $\hat{\theta}_1 = \underset{\theta}{argmin}(\Sigma_{t=2}^n |a_t - M_\theta((a_d)_{d=1}^{t-1})|) = \underset{\theta}{argmin}(\Sigma_{t=2}^n |a_t - s_t|)$
- $\hat{\theta}_2 = \underset{\theta}{argmin}(\Sigma_{t=2}^n |\text{diff}_2[M_\theta((a_d)_{d=1}^{t-1})]|) = \underset{\theta}{argmin}(\Sigma_{t=2}^n |\text{diff}_2[s_t]|)$

To understand why these 2 equations are important we can define our requirements for the *perfect* model. Firstly, We want the model to have a low absolute error. The alternative option - high absolute error - is not great since by increasing the absolute error, we diverge from our true data. This makes our smooth data a very poor approximation of the actual trend. Therefore we want to limit the absolute error of the model.
However, if the absolute error is equal to 0, our smooth data set will be equivalent to the original data. (Minimizing the absolute error would equate to the set of parameters that predict s_t to be as close as a_t as possible). This will make our model trivial and useless! Therefore, we need to introduce a mechanism that ensures the smooth data does not actually converge to the real data,

so we look at the second derivative.

Here diff_2 is defined as an approximation of the second order derivative (second order difference in discrete terms), since we cannot calculate the second derivative in itself using a discrete data set.
To ensure the model *looks smooth* we require its second derivative to be as close to 0 as possible. For a continuous function, if the second derivative equals 0 (or is close to 0) it indicates local linearity of the function (try integrating 0 twice and you will arrive at a linear function). In our interpretation if a function is locally linear, it appears smooth. Therefore we add this requirement to our model.
In addition to this, this requirement penalizes large values of the coefficient heavily. If the coefficient is large, then the smooth data set will change drastically between points, therefore, increasing the second derivative.
However, minimizing just the second order derivative will lead to a set of parameters that draw a straight line. This is again redundant, since the smoothing is too extreme to be of any use.

The combination of both criteria is critical to ensure we pick the optimal parameters to generate a smooth data set (second derivative as small as possible) and to ensure it represents the observed original data accurately (mean error as small as possible).
Normally, $\hat{\theta}_1$ and $\hat{\theta}_2$ will not equate to the same set of parameters since there is a trade-off between minimizing the absolute error and minimizing the second order derivative. The difficulty here lies in combining both of these criteria into one, which we have not done rigorously in this paper. This is still an open question as far as we know, however, we propose using the *Elbow Method* to pick near-optimal parameters.

1.4 Elbow method

The elbow method is a heuristic method of weighing the trade off between 2 criteria. It is often used in clustering methods, such as, K-means clustering to fit the parameters of number of clusters and the distortion within the clusters (within cluster variance).
To apply the elbow method we produce a graph containing a curve defined by the values of 2 criteria of interest. So, criteria 1 would be on X-axis and criteria 2 on Y-axis. This will allow to see the trade off between the 2 criteria and determine the near-optimal trade off of both criteria located on the *elbow* of the curve.
In our case, we propose using the elbow method to compare between the 2 criteria:

1. Sum of errors
2. Sum of second derivatives

To do this, we evaluate our model using both criteria at different parameter values. Then we plot the criteria one against another and pick the model that has the best trade off.

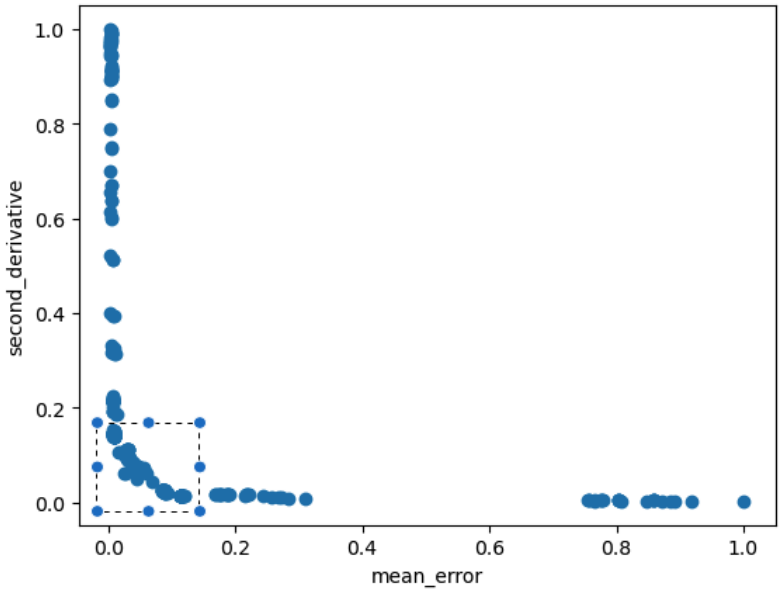

Figure 1: Example of criteria plot

Since we want to minimize both criteria (error and second derivative), we inspect the points (corresponding to models) that are close to 0 for both. I.e., in this case we could look into models represented by bounding the plot for values less than 0.1 or 0.2 in both criteria. This already reduces our search space, however, the final decision is up to the user.

Moreover, it is important to note how crucial the choice of models to test on is for this method. If we do not pick enough parameter combinations, even the elbow of the plot might not represent valid models. In addition to this, the model might not fit the data well, so even by trying all possible combinations, none of them will be good.
It is also important to note that it could be the case that both criteria are weighed differently and, for example, even models with *relatively* high second derivative are fine to use. Then we would only try to minimize the error. This could be the case when the selected parameter search space only contains combinations of parameters that produce smooth curves on the data set. The appropriate follow up to this would be to increase the parameter search space or inspect the resulting models.

1.5 Method variations

So far we have looked at a method that:

- Does not change the coefficient if the error distribution is centered at 0
- Increases the coefficient if the error distribution is not centered at 0 and increasing the coefficient will result in an error distribution closer to 0 (compared to decreasing the coefficient)
- And, alternatively, decreases the coefficient if the error distribution is not centered at 0 and decreasing the coefficient will result in an error distribution closer to 0 (compared to increasing the coefficient)

However, we can add an extra requirement to the method to ensure that the value of the coefficient k is as small as possible for the error distribution to still be centered. A smaller value of k will produce smaller second derivatives, therefore, the curve will seem smoother. This criteria can be enforced by rather than not changing the coefficient if the error distribution is centered at 0, but decreasing it.
Formally, in the case that μ_t is likely 0 (H_0 true), we decrease k (instead of "do not change k").

$$k_{t+1} = max(k_t - r, 0)$$

Now, if the error mean is unlikely 0 (H_1 true), we choose the new k_{t+1}. The choice of this is the same as in the original method. In our experiments we notice drastic changes in the output using the different methods, which we demonstrate in further chapters.
To summarize the difference between methods:

- **Method 1**: do not change k, when error distribution centered at 0. (H_0 true)
- **Method 2**: decrease k ($k_{t+1} = max(k_t - r, 0)$), when error distribution centered at 0. (H_0 true)

2 Method example

In this section we go through an example of applying this method using Python. All of the code for the following is available on GitHub here in the Jupyter Notebooks.

First, we generate random data with the trend $f_t = t$ and noise $n_t \sim Normal(f_t, 50)$ for $t = 1, 2, ..., 500$.

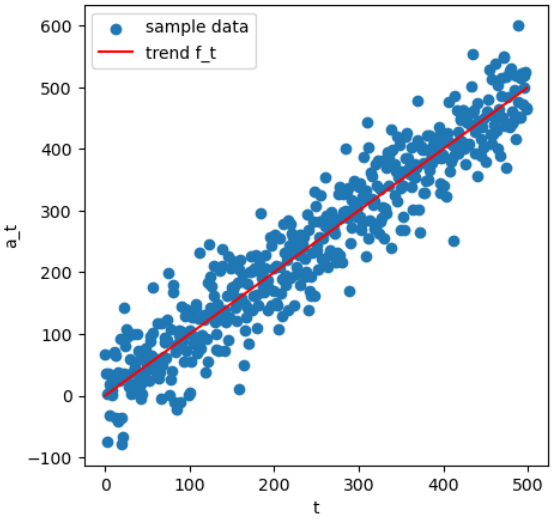

Figure 2: Generated data

The goal of the method is to extract f_t from the data. We define a set of candidate parameters to test our model on. We use the following set of potential parameter values and perform grid search on them (try every combination).

- w = 5, 20, 50, 200
- d_k = 0, 0.001, 0.01, 0.1, 0.2
- r = 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1
- α = 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1

It is important to note that *grid search* is a highly inefficient method for tuning parameters since it is equivalent to checking every combination. There exist other more efficient methods, such as, random search, however, we stick to grid search in this paper since results are still generated reasonably quickly. In total we check $4 * 5 * 6 * 6 = 720$ combinations of parameters using grid search.
We run grid search using all the parameters and Method 1 (k stays constant if error is likely 0) to produce the following criteria trade off curve.

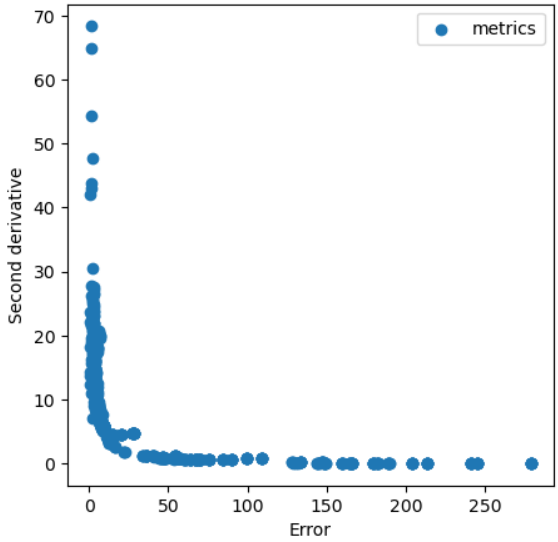

Figure 3: Trade off curve between criteria

To reiterate, each point in this curve represents a model and the criteria values it achieved. From the curve we see that best models are likely to have an error below 20 and second derivative below 10. We inspect all the models that have an error below 20 and order them by the lowest second derivative (and limit the number of models to 5). The numbers and order of operations that we have chosen to limit the number of models is arbitrary and any other would also suffice (for example, looking at models that have the second derivative below 10 and sorting by the error). This has to be determined experimentally. We acquire the following models:

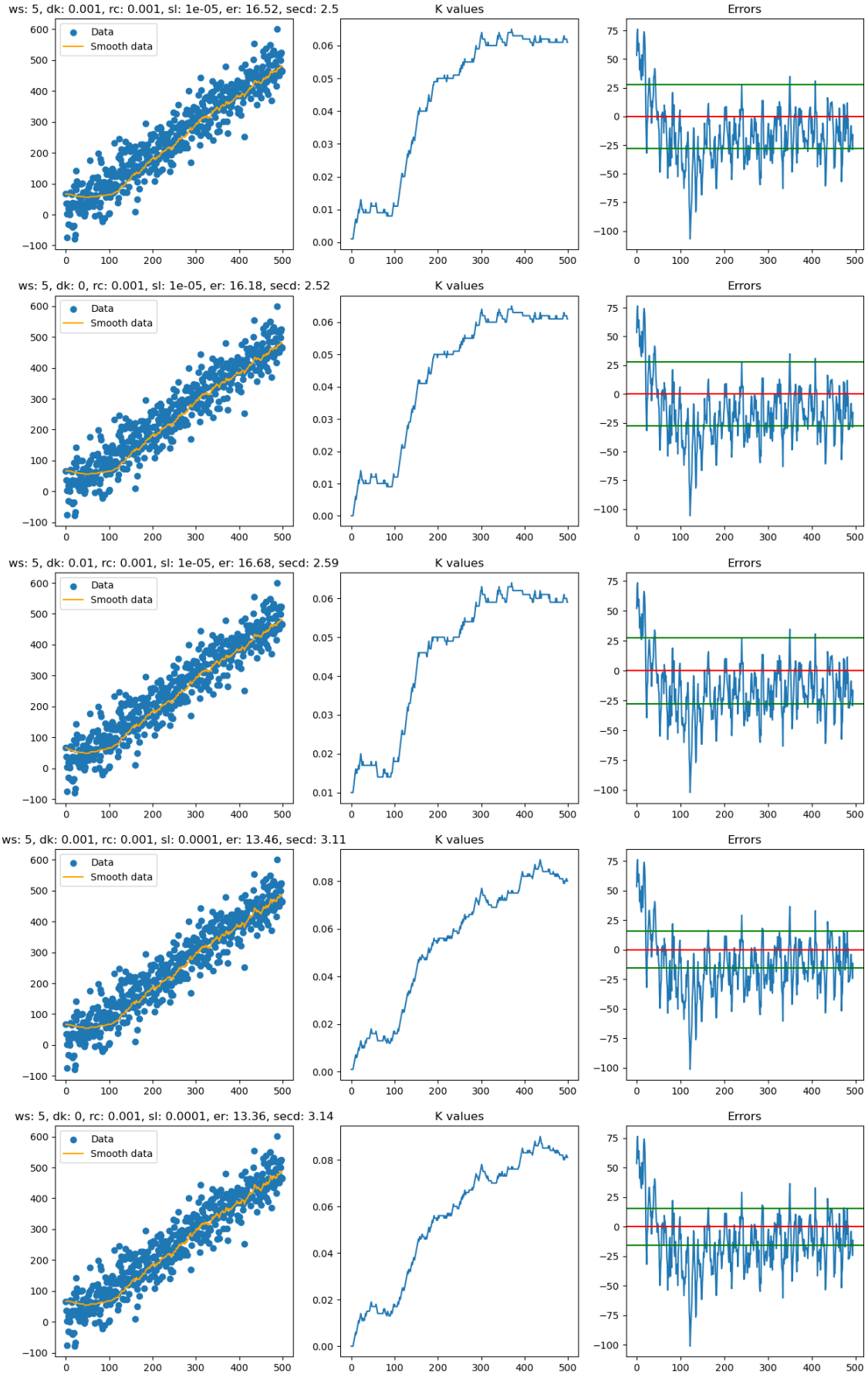

Figure 4: Produced models using Method 1

Each row in the output image corresponds to a model. The left-most graph shows the *smooth* line that has been drawn overlaying the original data. The title of the left-most graph contains information about the parameters and criterion values (ws - window size w, dk - default k d_k, rc - rate of change r, sl - significance level α, er - error criterion value, secd - second derivative criterion value). The middle plot displays the evolution of k_t values over time. The right-most plot displays the mean errors acquired at each of the points. That is, the mean error over the previous time window at that point. The red line is drawn to indicate where 0 is located; and the green lines show boundaries for the significance level. If the error is outside of the green boundaries, the coefficient k gets modified; if not - the coefficient k stays constant (since we are using Method 1). From the plots, we see that all of the models look acceptable, so we choose any parameters for the model, e.g, $\theta = (w = 5, d_k = 0, r_c = 1e-3, \alpha = 1e-5)$.

Now, we consider the same data under Method 2 (decrease k if error is likely 0). We acquire the following plot of trade off between criteria when applying Method 2 and using the same parameter search space.

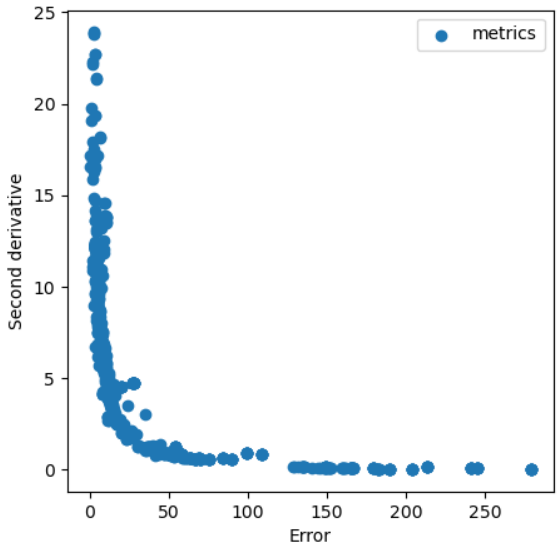

Figure 5: Trade off curve between criteria for Method 2

By similar reasoning to Method 1, we inspect all the models that have an error below 20, order them by the lowest second derivative and filter on the top 5 models to produce the following graphs.

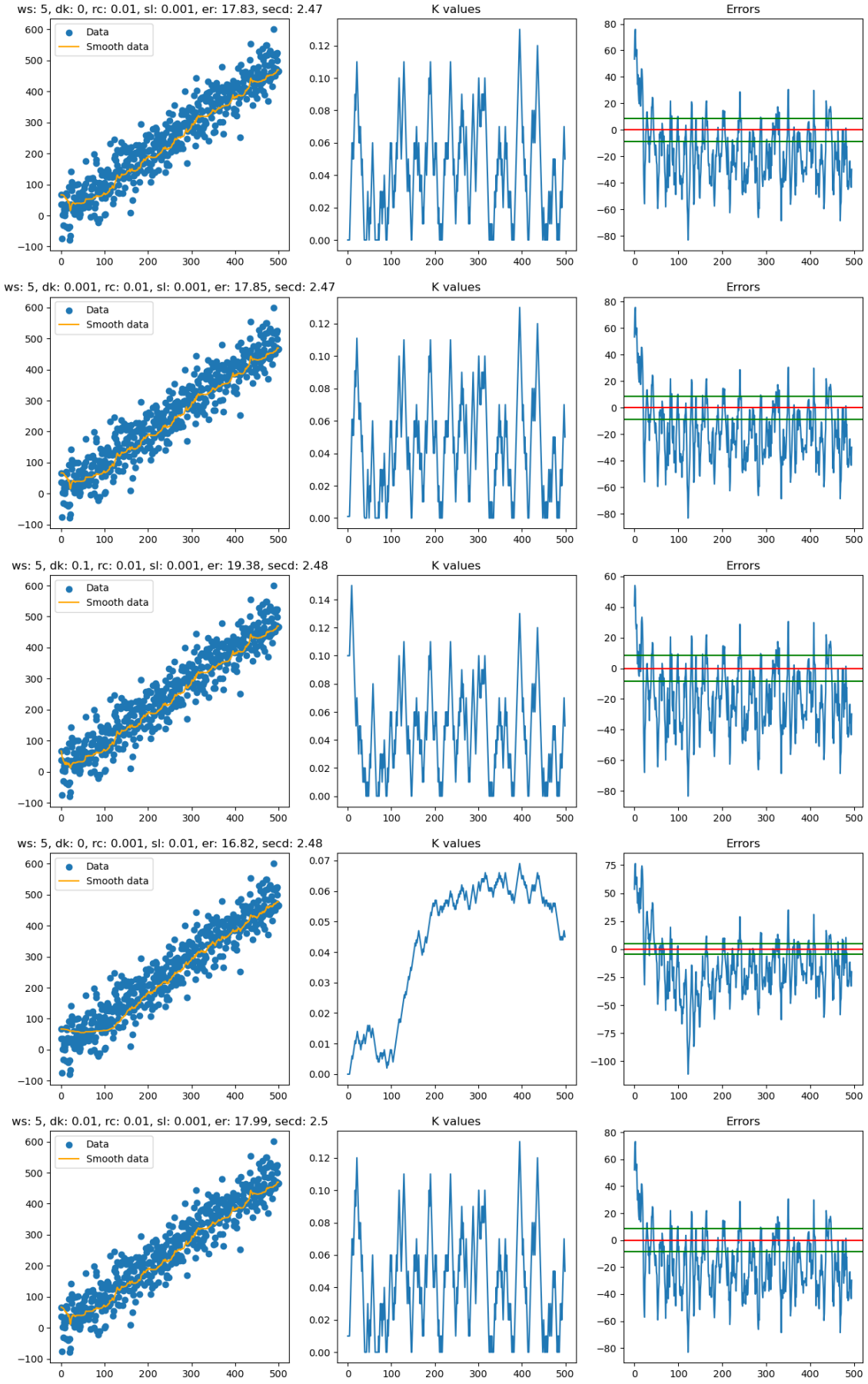

Figure 6: Produced models using Method 2

Notice how the *K value* plots are different from Method 1. In this scenario, k values decrease more often. This seems to be compensated by the increased rate of change and significance levels in the best models.
The differences between these two methods are further illustrated by looking at a different data set.

2.1 Real data set

In this example we consider a real data set of the following form:

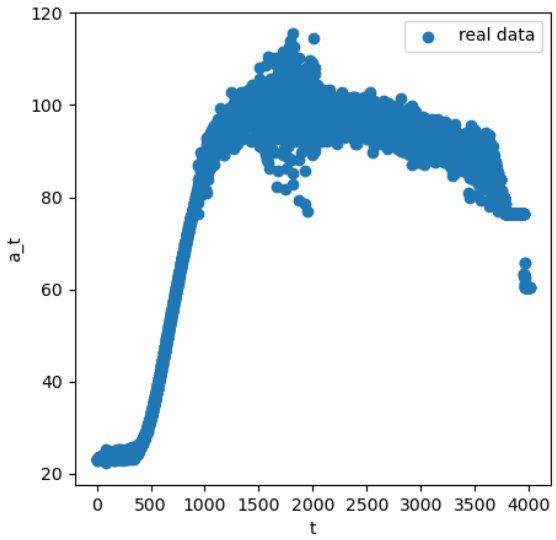

Figure 7: Real data

For this illustration we will use a larger parameter space of the form:

- w = 5, 20, 50, 200
- d_k = 0, 0.00001, 0.0001, 0.001, 0.01, 0.1, 0.2
- r = 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1
- α = 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1

We run grid search using all the 1372 parameter combinations and Method 1 (k stays constant if error is likely 0) to produce the following trade off curve between criteria.

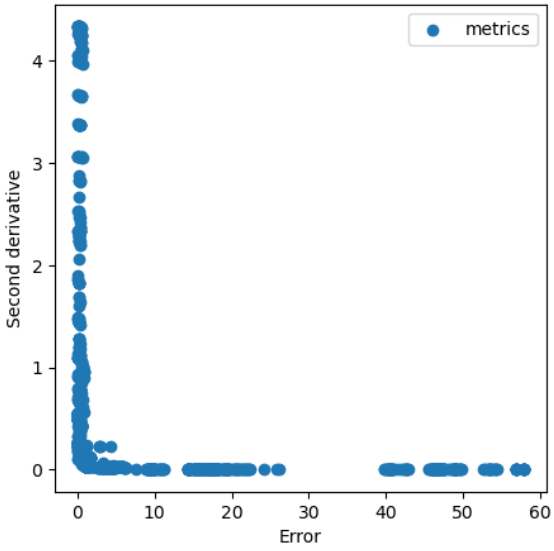

Figure 8: Trade off curve between criteria for Method 1

Further, we inspect all the models that have an error below 1 and order them by the lowest second derivative to produce the following top 5 models.

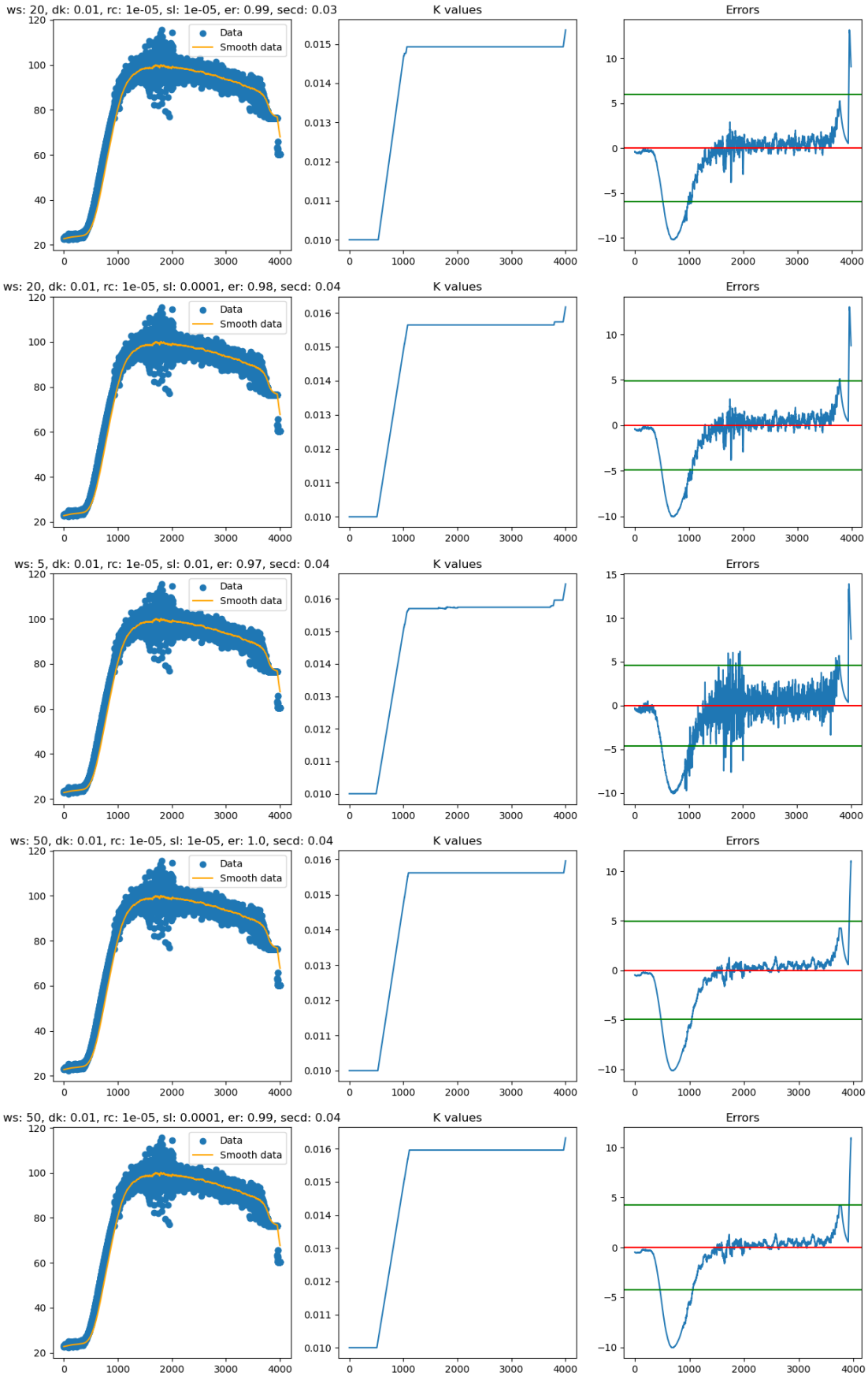

Figure 9: Produced models using Method 1

Now, we can use any of the parameters of the displayed models, since all of them seem reasonable.
Moreover, we execute the same pipeline using Method 2 and use the same parameter search space to produce the following criteria trade off curve and models.

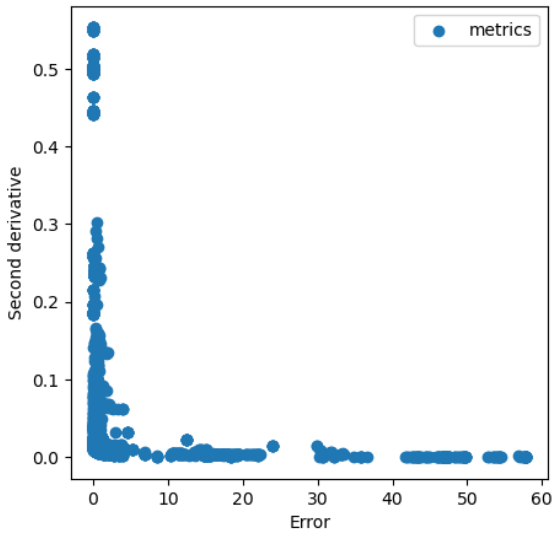

Figure 10: Trade off curve between criteria for Method 2

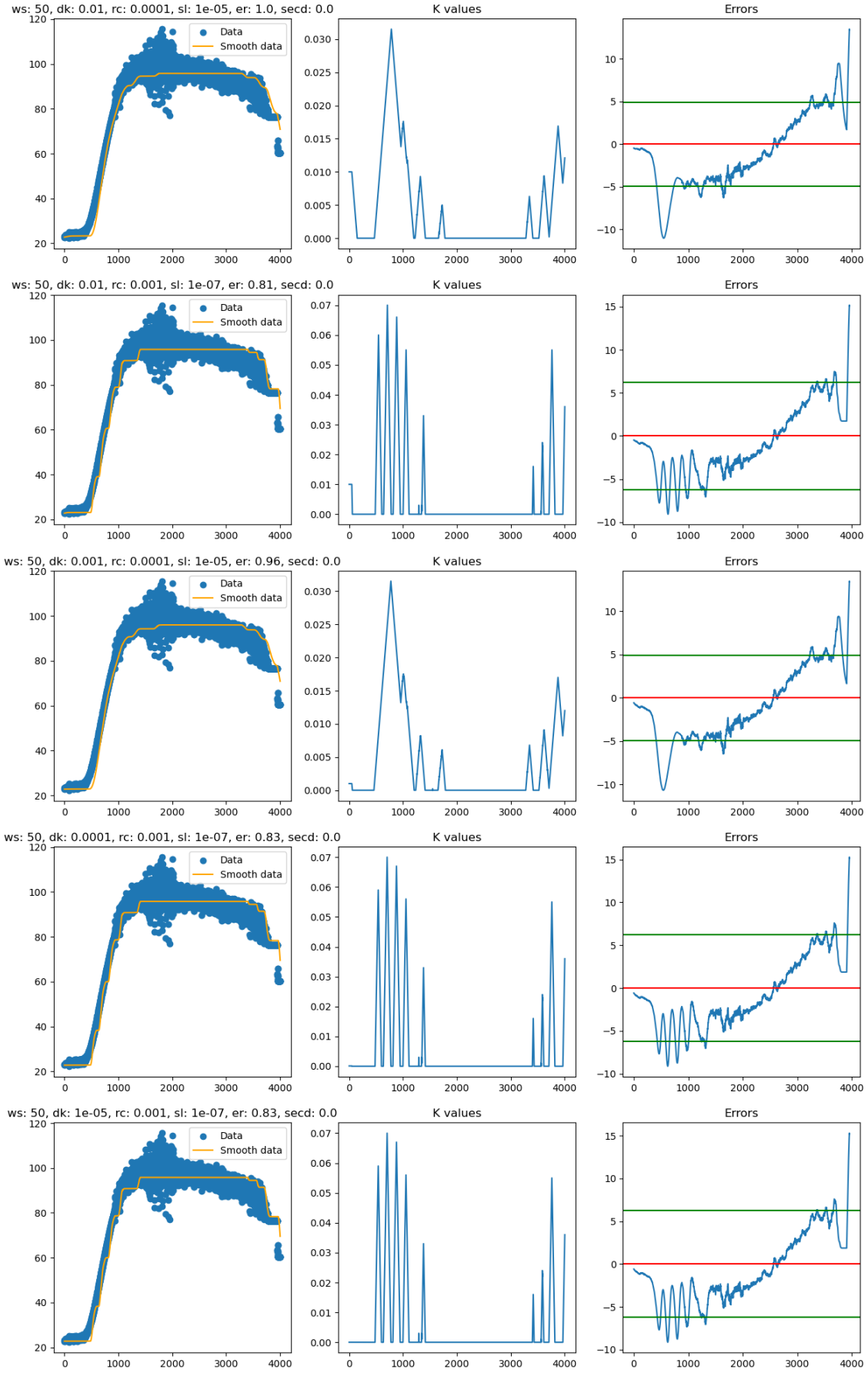

Figure 11: Produced models using Method 2

Now, we are further able to compare the methods. Method 1 seems to smooth the data generally and extract the trend better than Method 2. However, Method 2 extracts *change points* in data better. If the application required the trend to change only if we are absolutely *certain* that the trend has changed, Method 2 is more applicable. This is due to the fact Method 2 always tends to $k = 0$ as its ground/default state, therefore, produces a smoother curve that only reacts when it is absolutely necessary. Whereas, Method 1 has a ground state of fixed k whatever that k might be, therefore it allows more fluctuation when the model is confident that the error is near 0.

3 Comparison to *non-dynamic* Exponential Smoothing

The Dynamic Exponential Smoothing family of models also contains the non-dynamic variant - Exponential Smoothing. An Exponential Smoothing model defined in the terms of our model is any model with the parameter $r = 0$. This is obvious since rate of change determines the dynamic part of the model, therefore, if we do not allow change in the coefficient k, we are just using a regular Exponential Smoothing model with the parameter d_k as the coefficient.
In this section we compare the results from Dynamic Exponential Smoothing models with the results from Exponential Smoothing using the same criterion, and demonstrate the usefulness of the criteria and Elbow Method when using the Exponential Smoothing model.
We run the model search algorithm using $d_k = \{0.0, 0.01, 0.02, ..., 0.99, 1\}$, $r = 0$, and any value of the other parameters since that will not impact the model. We produce models that have an error below 20 and sort them by second derivative. Moreover, we try this method on generated and real data.
First, we use the generated data and produce the trade off curve and select top 5 models. Similarly, we filter the models on error below 20 and sort by smallest second derivative.

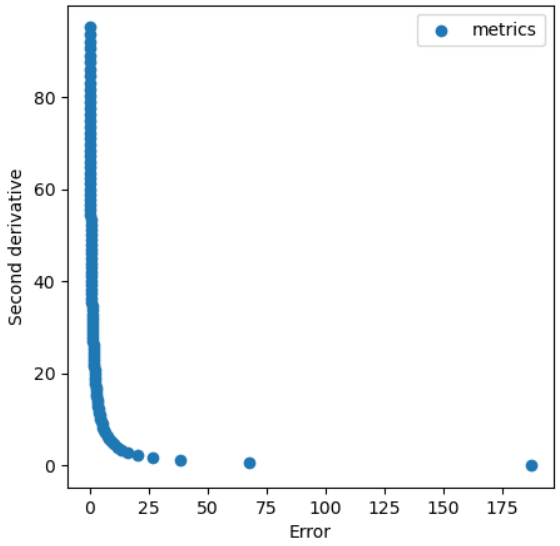

Figure 12: Trade off curve between criteria for regular Exponential Smoothing

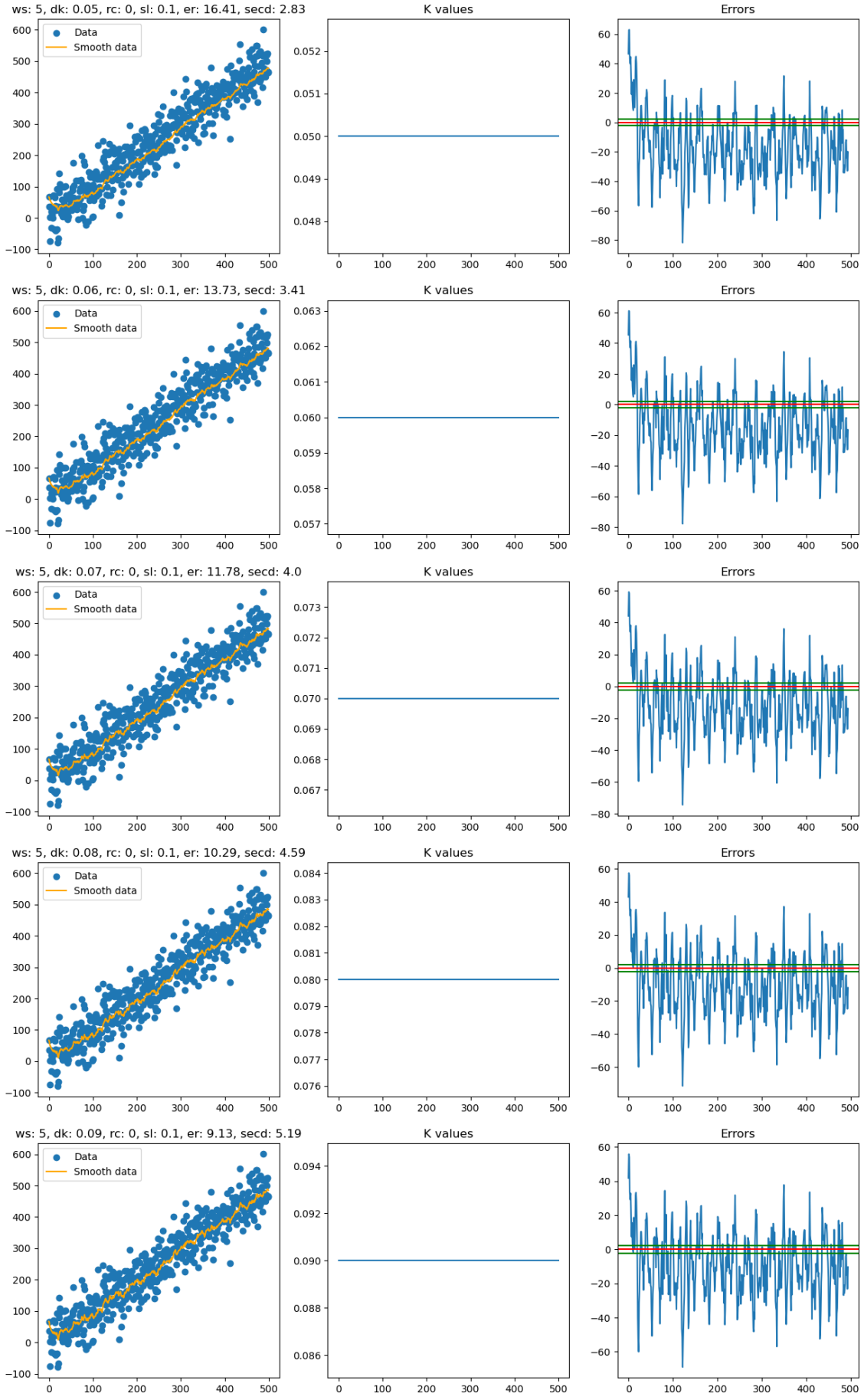

Figure 13: Produced models

Second, we use the real data and produce the trade off curve and select top 5 models. Similarly, we filter the models on error below 1 and sort by smallest second derivative.

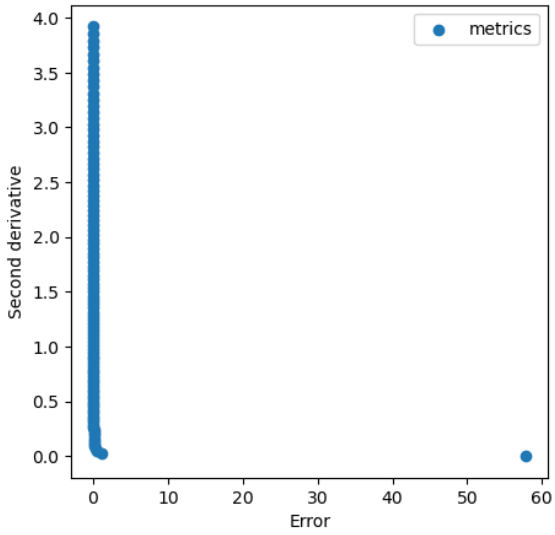

Figure 14: Trade off curve between criteria for regular Exponential Smoothing

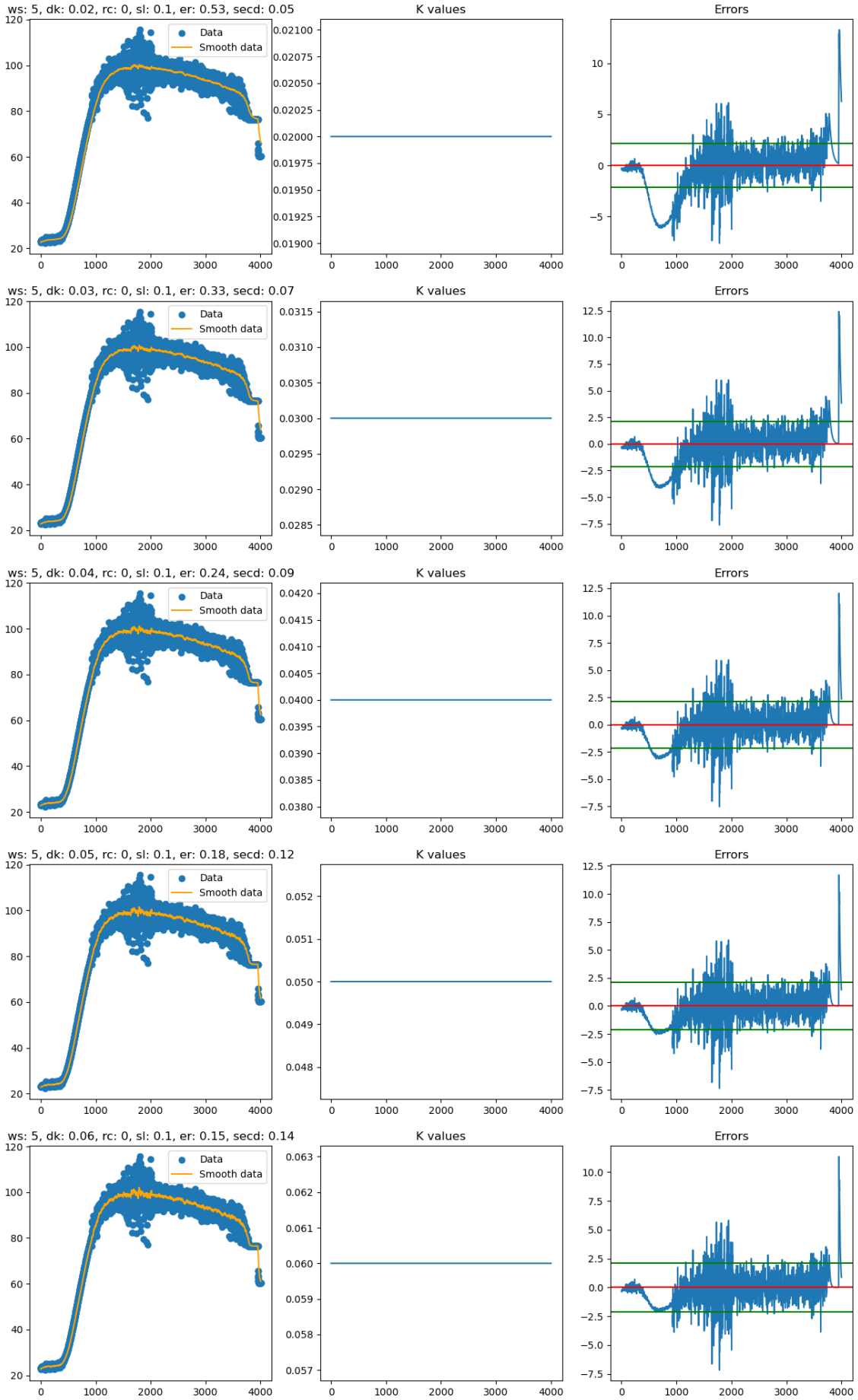

Figure 15: Produced models

As results show, the evaluation criteria also performs well on the regular Exponential Smoothing model. By Occam's razor, fewer parameters are always better since it will be a simpler model, if they yield the same results.

4 Applications and conclusion

The Dynamic Exponential Smoothing model can be applied not only to smooth an existing data set, but to filter data as well. Filtering involves appending new data points to the data set and allowing the model to smooth them. If the resulting model eventually does not perform well enough, it can be re-trained using the full data set at that time. However, caution must be taken since the model might not be able to handle data sets that go through many phases (e.g., variance keeps drastically changing every 100 data points). A different approach or data processing might be more applicable in that scenario.

We have developed a Python library for the execution, fitting and filtering of time series points using either Dynamic Exponential Smoothing or Exponential Smoothing methods. A link to a GitHub page is found here.