`pybats-detection`: A python package for outlier and structural changes detection in time series analysis

André Menezes and Eduardo Gabriel

Last compiled on July 29, 2022

Contents

1 Introduction

Dynamic Models and Bayesian Forecasting were introduced in the seminal work by Harrison (1976). Since then, many scientists do seem to have an interest in the theoretical innovations and practical applications of this technique.

Two major works in the field of dynamic models are West and Harrison (1986) and West and Harrison (1989), in which the authors introduced Bayesian subjective intervention, automatic monitoring, and adaptation in the class of Dynamic Linear Model (DLM).

To our knowledge, there is no open source tool for Bayesian monitoring and intervention despite the software literature offering a variety of programs to work with state space models. To fill this gap, we introduce `pybats-detection` package, an effective **python** package for the identification of structural changes and outliers in DLM. The package's current version includes smoothing for univariate DLM, automatic monitoring, and subjective intervention.

2 Smoothing

To demonstrate the use of the Smoothing class we will start with a simulated example in which a sequence of observations $y_1, \ldots, y_t$ were generated following DLM evolution structure given by

$$
\begin{aligned}
y_t &= \mathbf{F}\boldsymbol{\theta}_t + \epsilon_t, \quad \epsilon_t \sim N[0, V_t], \\
\boldsymbol{\theta}_t &= \mathbf{G}\boldsymbol{\theta}_{t-1} + \omega_t, \quad \omega_t \sim N[0, W_t]
\end{aligned}
$$

This can be done using the `RandomDLM` class which has the arguments (n, V, W): the number of observations, observational variance and state vector variance. This class has three methods that simulate data using different mechanisms:

- `.level`: dynamic level model;
- `.growth`: dynamic growth model;
- `.level_with_covariates`: dynamic level model where Y is simulated given Y, a matrix of fixed covariates.

For now, we stick with `.level`, simulating $n = 100$ observations with both observational and state vector variance equals to one 1, the starting level is set to 100. The simulated data is plotted in Figure Figure 1.

```
>>> # Generating level data model
>>> np.random.seed(66)
>>> rdlm = RandomDLM(n=100, V=1, W=1)
>>> df_simulated = rdlm.level(start_level=100, dict_shift={})
>>> y = df_simulated["y"]
```

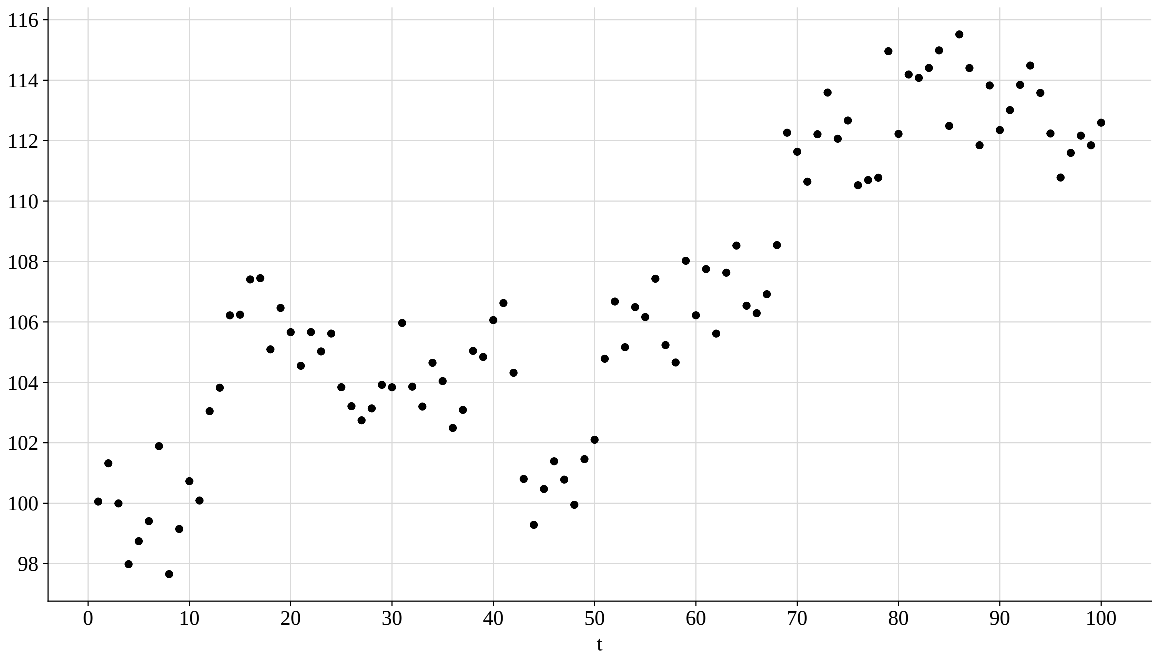


Figure 1: Simulated data

The Smoothing class allows you to perform a retrospective analysis for $\mathbf{Y}$, obtaining the distribution of $(\boldsymbol{\theta}_{T-k}|D_T)$, for $k \geq 1$, the k-step smoothed distribution for the state vector at time T, which is analogous to the k-step ahead forecast distribution $(\boldsymbol{\theta}_{t+k}|D_t)$.

To use Smoothing, first it is necessary to define the model components with prior values, which is done with the `dlm` class available in the `pybats` package. In this case, it was considered a DLM with level and growth. The prior vector and covariances are defined by $\mathbf{a}$ and $\mathbf{R}$. Lastly, the discount factor denoted by `deltrend` is a constant in the interval [0, 1], which is used to coordinate the adaptive capacity of predictions with increasing variance of model components.

```
>>> # Define model components
>>> a = np.array([100, 0])
>>> R = np.eye(2)
>>> np.fill_diagonal(R, val=1)
>>> mod = dlm(a, R, ntrend=2, deltrend=.95)
```

Given this, the method `.fit` will initialize the model and the loop forecast, observe and update begin. The prior and posterior moments $(\mathbf{a}_t, \mathbf{m}_t, \mathbf{C}_t, \mathbf{R}_t)$ will be computed for all t and saved. Subsequently, these moments will be used to obtain the moments for $(\boldsymbol{\theta}_{T-k}|D_T)$, recursively with $k \geq 1$, and denoted by $(\mathbf{a}_T(-k), \mathbf{m}_T(-k), \mathbf{C}_T(-k), \mathbf{R}_T(-k))$.

```
>>> # Fit with monitoring
>>> smooth = Smoothing(mod=mod)
>>> smooth_fit = smooth.fit(y=y)
```

This returns a dictionary with the following keys:

- `model`: the updated `pybats.dglm.dlm` object.
- `filter`: a dictionary with:
  - `posterior`: `pandas.DataFrame` with the filtering posterior moments.
  - `predictive`: `pandas.DataFrame` with the one-step ahead predictive moments.

If `smooth` is `True`, then also contains:

- `smooth`: a dictionary with:
  - `posterior`: `pandas.DataFrame` with the smooth posterior moments.
  - `predictive`: `pandas.DataFrame` with the smooth one-step ahead predictive moments.

2.1 smoothed predictive

The results for the smoothed predictive distribution consists of: $f_T(-k), q_T(-k)$ and the bounds for the credible interval (`ci_lower`, `ci_upper`). Given by

$$f_T(-k) = \mathbf{F}^{'}\mathbf{a}_T(-k), \qquad q_T(-k) = \mathbf{F}^{'}\mathbf{R}_T(-k)\mathbf{F}$$

The credible interval is is obtained from the corresponding smoothed distributions for the mean response of the series. Since V is considered unknown, then

$$(\mu_T(-k)|D_T) \sim T_{n_T}[f_T(-k), q_T(-k)]$$

For this simulated example, the results for the smoothed predictive distribution for the mean response are

```
>>> smooth_fit.get('smooth').get('predictive').round(2).head(5)
```

Table 1: Smothed predictive distribution results

qk	fk	t	df	ci_lower	ci_upper
0.31	99.97	1	1	98.85	101.1
0.27	100.07	2	2	99.05	101.1

qk	fk	t	df	ci_lower	ci_upper
0.24	100.12	3	3	99.14	101.1
0.23	100.20	4	4	99.24	101.2
0.22	100.39	5	5	99.47	101.3

as for the filtered distribution

```
>>> smooth_fit.get('smooth').get('predictive').round(2).head(5)
```

Table 2: Filtered predictive distribution results

t	parameter	mean	variance	df	ci_lower	ci_upper
1	Intercept	99.97	0.31	1	98.85	101.1
2	Intercept	100.07	0.27	2	99.05	101.1
3	Intercept	100.12	0.24	3	99.14	101.1
4	Intercept	100.20	0.23	4	99.24	101.2
5	Intercept	100.39	0.22	5	99.47	101.3

Plotting the filtered vs smoothed predictive distributions results is possible to see difference, primarily in the length of the credible interval (see Figure Figure 2).

2.2 smoothed posterior

The results for the posterior distributions are analogous, where

- parameter: Indicator for the respective state space parameter in $\boldsymbol{\theta}$;
- mean: The smoothed posterior distribution mean for time $t = T - k$ ($\mathbf{m}(-k)$);
- variance: The smoothed posterior distribution variance for time t ($\mathbf{C}(-k)$).
- credible interval (`ci_lower`, `ci_upper`): The credible interval obtained from the corresponding smoothed posterior distributions. Since V is considered unknown, then

$$(\boldsymbol{\theta}_{T-k}|D_T) \sim T_{n_T}[\mathbf{a}_T(-k), \mathbf{R}_T(-k)].$$

```
>>> smooth_fit.get('smooth').get('posterior').round(2).head(5)
```

Table 3: Smothed posterior distribution results

t	parameter	mean	variance	df	ci_lower	ci_upper
1	Intercept	99.97	0.31	1	98.85	101.1
2	Intercept	100.07	0.27	2	99.05	101.1
3	Intercept	100.12	0.24	3	99.14	101.1
4	Intercept	100.20	0.23	4	99.24	101.2
5	Intercept	100.39	0.22	5	99.47	101.3

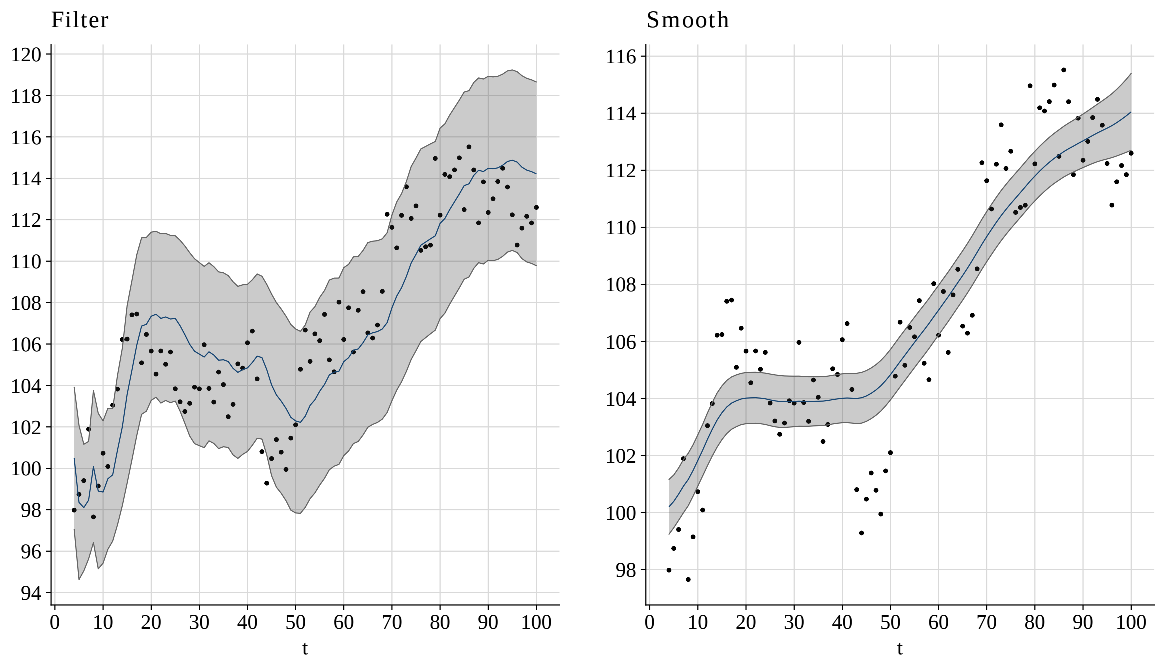


Figure 2: Mean response for Filtered and Smoothed predictive distributions for each model component with 95% credible intervals.

In Figure Figure 3 we plot the results for filtered and smoothed distributions, in this case for each state space parameter. As expected, the smoothed posterior distributions show a less erratic behavior with shorter credible intervals.

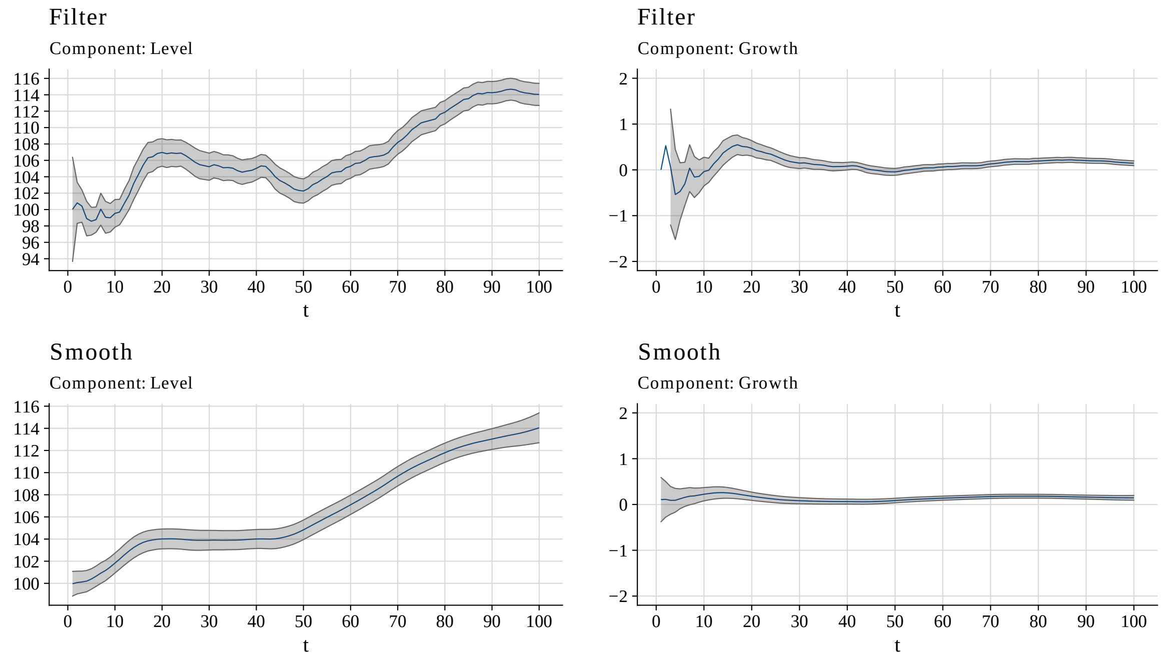


Figure 3: Mean response for Filtered and Smoothed posterior distributions for each model component with 95% credible intervals.

2.3 Aplication: AirPassangers dataset

Now we'll demonstrate the `Smoothing` class with the classic Box & Jenkins airline data, Monthly totals of international airline passengers (1949 to 1960). The time series is plotted in Figure Figure 4. This data has a clear multiplicative seasonality, using a linear model (with additive seasonality) may be a naive approximation for this data. But, just for the sake of comparison between filtered and smoothing we stick with the linear model.

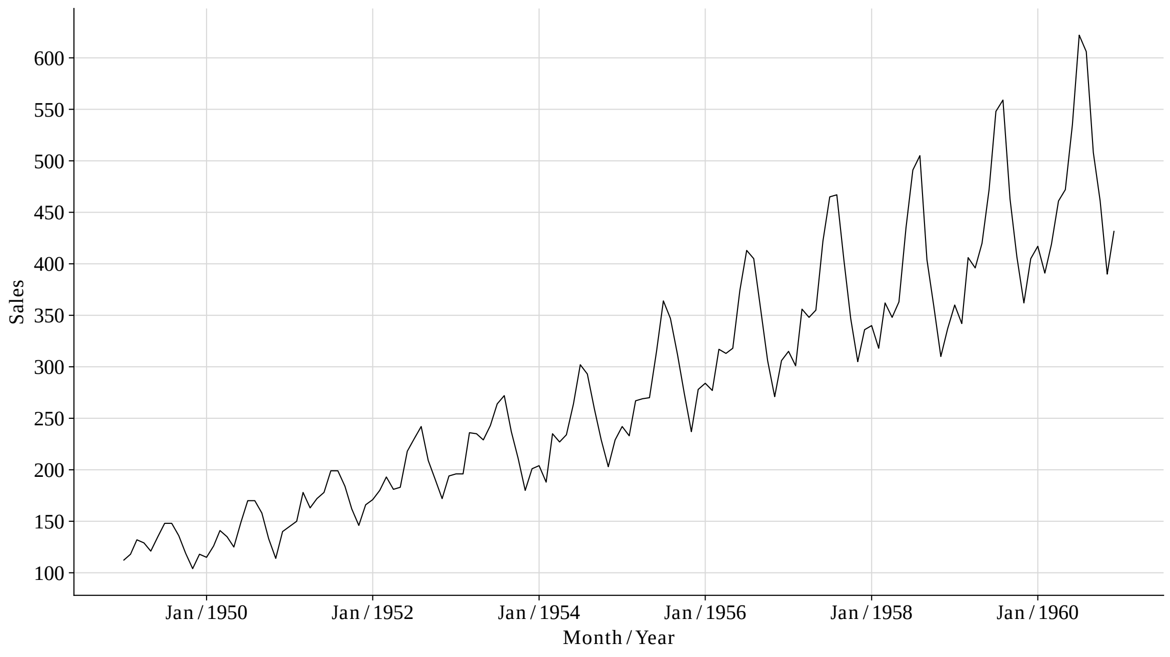


Figure 4: Monthly totals of international airline passengers, 1949 to 1960.

Using a normal DLM with three main components: Trend, Growth and Seasonality. The seasonality is modeled using the Fourier form representation, which depends on the parity of a period p and the number of harmonics components. Formally, the $\mathbf{r}^{th}$ harmonic component is given by

$$S_r(.) = a_r \cos(\alpha r) + b_r \sin(\alpha r), \quad r = 1, \ldots, h, \quad a_r = 2\pi/p, \quad h <= p/2$$

Here it was specified a yearly seasonal effect with period $p = 12$ and the first two harmonics. The discount factor for the level and growth components is set to 0.95, and 0.98 for the seasonal components. The results are plotted below.

```
>>> a = np.array([112, 0, 0, 0, 0, 0])
>>> R = np.eye(6)
>>> np.fill_diagonal(R, val=100)
>>> mod = dlm(a, R, ntrend=2, deltrend=.95, delseas=.98,
>>>           seasPeriods=[12], seasHarmComponents=[[1, 2]])
```

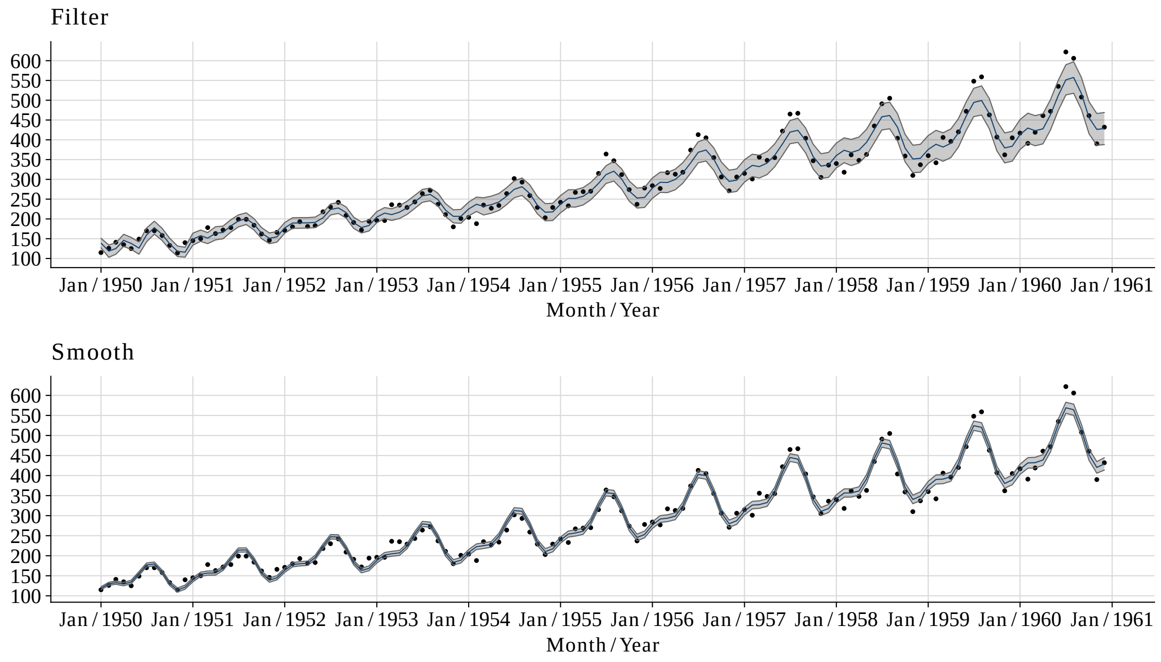


Figure 5: Mean response for Filtered and Smoothed predictive distributions with 95% credible intervals.

Since the seasonality was modeled using harmonic components, the model has a total of six parameters: level, growth and four for seasonality (a_1, b_1, a_2, b_2). For simplicity, the results for de posterior distributions considered the sum of the harmonic components, whose moments are given by

$$\mu_{seas} = \mathbf{F}'_{seas}\mathbf{a}_T(-k), \qquad \sigma^2_{seas} = \mathbf{F}'_{seas}\mathbf{R}_T(-k)\mathbf{F}_{seas}$$

where $\mathbf{F}'_{seas} = [0, 0, 1, 0, 1, 0]$. The results are illustrated below.

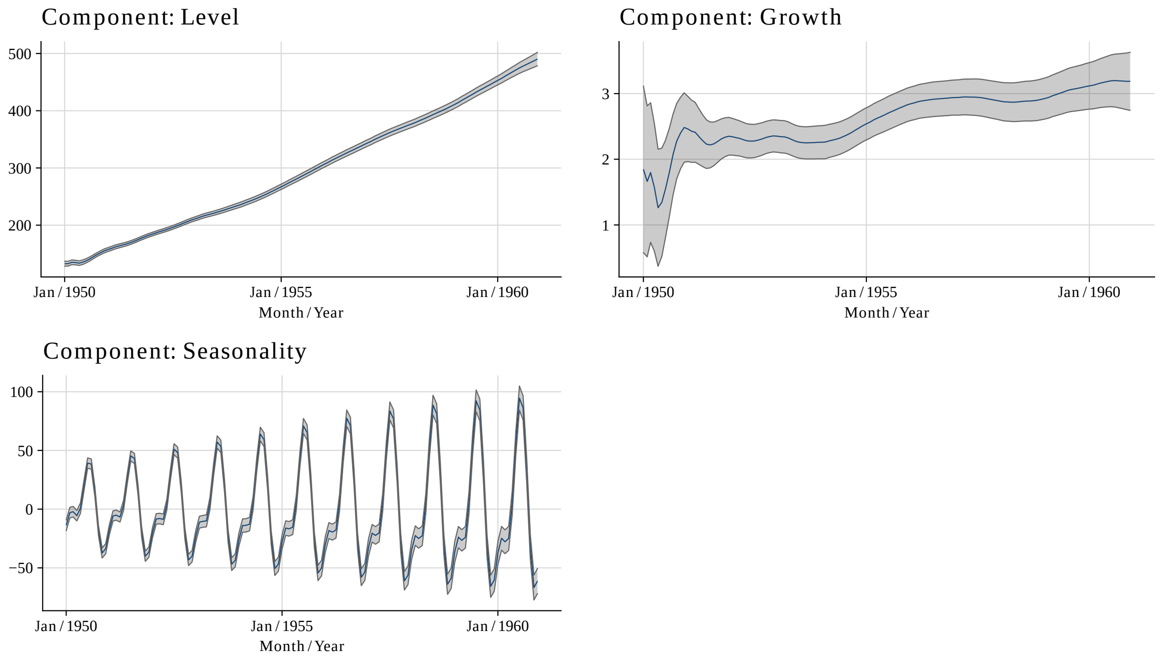


Figure 6: Mean response for Filtered and Smoothed posterior distributions for each model component with 95% credible intervals.

3 Manual Intervention

3.1 CP6

To illustrate the subjective intervention class we use the CP6 data graphed below. This time series runs from January 1955 to December 1959, providing monthly total sales, in monetary terms on a standard scale, of a product by a major company in UK. Note that the use of standard time series models may not wield satisfactory results as there a some points that need some attention:

1. During 1955 the market grows fast at a fast but steady rate,
2. A jump in December 1955.
3. The sales flattens off for 1956.
4. There is a major jump in the sales level in early 1957.
5. Another jump in early 1958.
6. Throughout the final two years, there is a steady decline back to late 1957.

```
>>> cp6 = load_cp6()
```

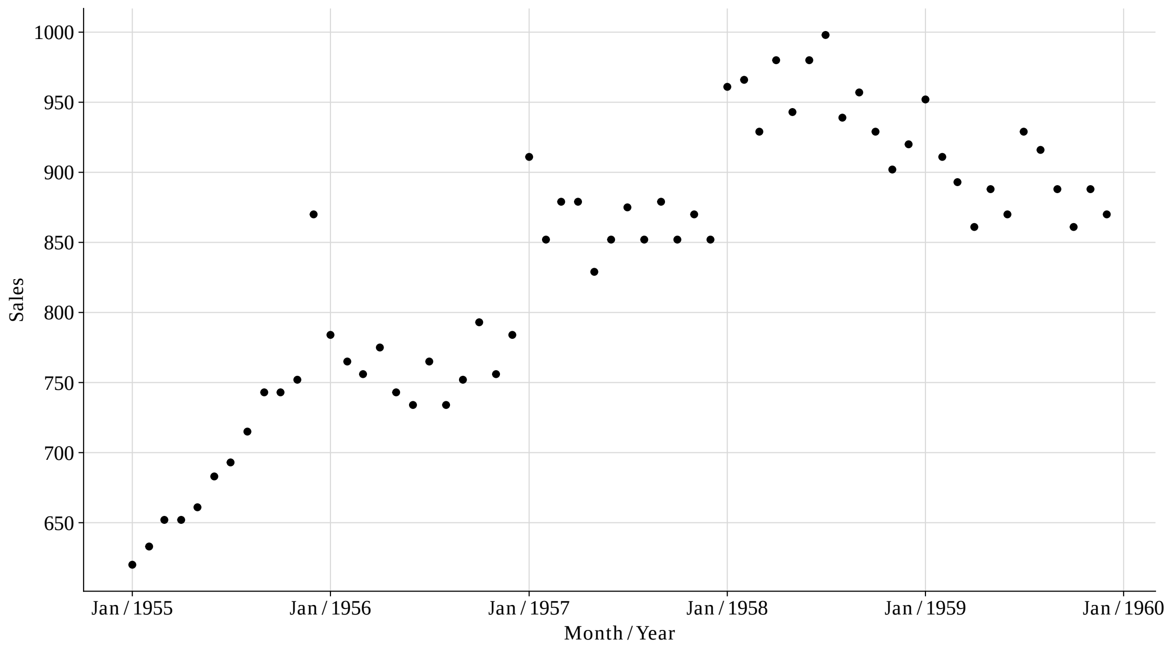


Figure 7: CP6 sales series

3.2 Fit Without Intervention

Given this, let's see how a standard dlm performs. The model used is defined below.

```
>>> # Define the growth model
>>> a = np.array([600, 1])
>>> R = np.array([[100, 0], [0, 25]])
>>> mod = dlm(a, R, ntrend=2, deltrend=[0.90, 0.98])
```

```
>>> # Filter and Smooth without intervention
>>> smooth = Smoothing(mod=mod)
>>> out_no_int = smooth.fit(y=cp6["sales"])
>>> dict_filter_no_int = out_no_int.get("filter").get("predictive")
```

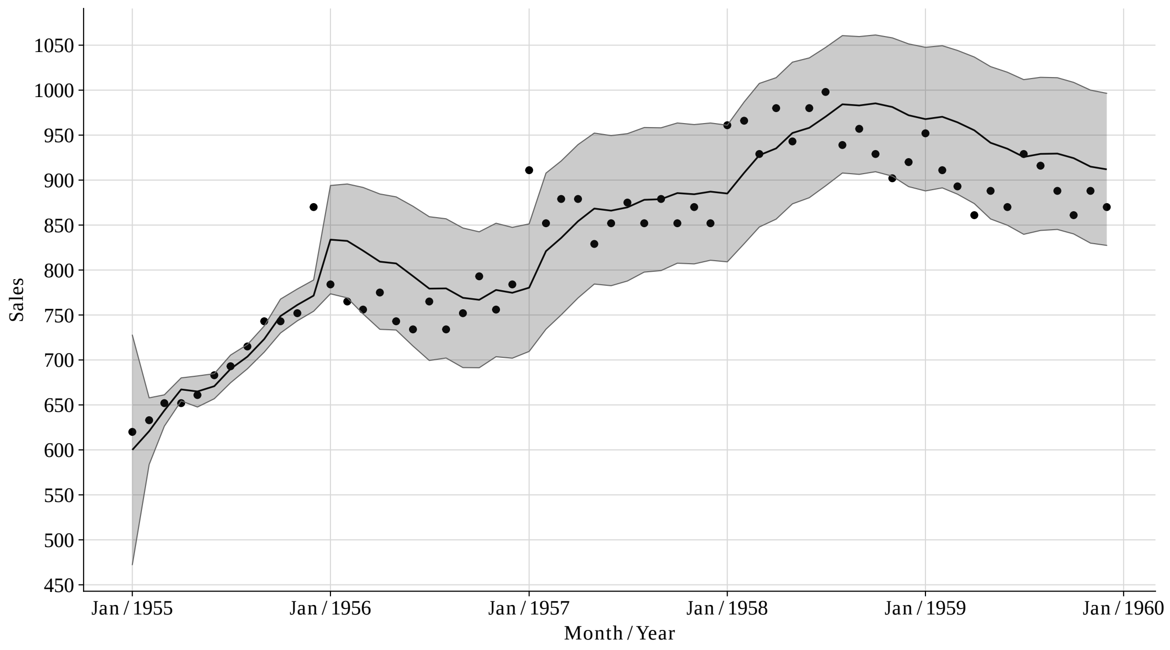


Figure 8: Mean response for Filtered predictive distribution with 95% credible interval

Note that until November 1955 the forecast distribution was quite acceptable, the credible interval was relatively small and the errors was were distributed around zero and inside the interval. But with the jump in December 1955 the level rises dramatically, the biggest problem is not the model's inability to efficiently predict this point, but the influence it has on future predictions. Note that for most of the year 1956 the predicted sales overestimation the real sales, giving a cluster of negative errors $(y_t - f_t)$. In early 1957 another jump was observed, but in this case, it was accompanied by a regime change. And this has great impact in the amplitude of the credible intervals. In early 1958 another regime change, followed by a change in grow, that is not properly modeled since from August 1958 to January 1960 all errors were negative with the exception of July 1959.

3.3 Fit With Intervention

With the intervention class it is possible to consider outside information to define the prior distribution at the time t. This can be done in two ways: noise or subjective. Which must be provided in a list of dictionaries containing the time the intervention will be carried out and the type. Lets start with a empty list

```
>>> intervention_list = []
```

3.3.1 Noise Intervention in Prior Variance

In our example, suppose that a change in growth for the year 1956 was anticipated. An increase in uncertainty about level and growth can be done by the addition of a matrix H_t to R_t at time

$t = 12$ given by

$$H_t = \begin{bmatrix} 100 & 25 \\ 25 & 25 \end{bmatrix}$$

Thus, there is an increase (a positive shift) in the prior variance of the components. In our list of interventions we have

```
>>> intervention_list = [{
>>>   "time_index": 12, "which": ["noise"],
>>>   "parameters": [{
>>>     "h_shift": np.array([0, 0]),
>>>     "H_shift": np.array([[100, 25], [25, 25]])}]
>>> }]
```

where

- `time_index`: time of intervention;
- `which`: type of intervention (in this case, a noise intervention);
- `parameters`: the values for the intervention.
  - `h_shit`: Shift in mean (we'll see more about that later).
  - `H_shift`: Shift in variance.

3.3.2 Noise Intervention in Prior Mean and Variance

It is also possible to intervene in the prior mean. Suppose an increase in the market level is expected for the year 1957, we can add a change in level of 80 units and increase the variance by 100 at January ($t = 25$)

$$\mathbf{h}_{25} = \begin{bmatrix} 80 \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{H}_{25} = \begin{bmatrix} 100 & 0 \\ 0 & 0 \end{bmatrix}$$

now, updating our intervention list

```
>>> intervention_list = [{
>>>   "time_index": 12, "which": ["noise"],
>>>   "parameters": [{
>>>     "h_shift": np.array([0, 0]),
>>>     "H_shift": np.array([[100, 25], [25, 25]])}],
>>>
>>>   "time_index": 25, "which": ["noise"],
>>>   "parameters": [{
```

```
>>>     "h_shift": np.array([80, 0]),
>>>     "H_shift": np.array([[100, 0], [0, 0]])}]
>>> }]
```

In January 1958 ($t = 37$) another jump in level is anticipated, this time of about 100 units with a feeling of increased certainly about the new level and also a anticipated uncertainly for the growth. At this time, the prior mean and variance given by

$$\mathbf{a}_{37} = \begin{bmatrix} 864.5 \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{R}_{37} = \begin{bmatrix} 91.7 & 9.2 \\ 9.2 & 1.56 \end{bmatrix}$$

are simply altered to

$$\mathbf{a}^*_{37} = \begin{bmatrix} 970 \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{R}^*_{37} = \begin{bmatrix} 50 & 0 \\ 0 & 5 \end{bmatrix}$$

3.3.3 Observational Variance Intervention

It is also possible to perform interventions on observational variance. This can be useful for outlier anticipation.

Suppose that at the end of 1955 there will be an announcement of future price increases which will result in forward-buying. So, a intervention at December 1955 ($t = 12$) will allow for an anticipated outlier. In late 1956, there is a a view that the marked change in the new year will begin with a maverick value, as the product that are to be discontinued are sold cheaply.

This interventions can be done by including a variance intervention in our list of interventions for the respective time:

```
>>> list_interventions = [
>>>     {"time_index": 12, "which": ["variance", "noise"],
>>>      "parameters": [{"v_shift": "ignore"},
>>>                     {"h_shift": np.array([0, 0]),
>>>                      "H_shift": np.array([[1000, 25], [25, 25]])}]
>>>      },
>>>     {"time_index": 25, "which": ["noise", "variance"],
>>>      "parameters": [{"h_shift": np.array([80, 0]),
>>>                      "H_shift": np.array([[100, 0], [0, 0]])},
>>>                     {"v_shift": "ignore"}]},
>>>     {"time_index": 37, "which": ["subjective"],
>>>      "parameters": [{"a_star": np.array([970, 0]),
>>>                      "R_star": np.array([[50, 0], [0, 5]])}]}
>>> ]
```

3.3.4 Performing the fit (filter and smoothing) with interventions

Finally, the fit with intervention can be done using the `Intervention` class. In the `.fit` method we will initialize the model and the loop forecast, observe and update, this time with the interventions given in `list_interventions`, begin. This will return a dictionary with the same structure as presented in the smoothing section.

```
>>> manual_interventions = Intervention(mod=mod)
>>> out_int = manual_interventions.fit(
>>>     y=cp6["sales"], interventions=list_interventions)
>>> dict_filter_int = out_int.get("filter").get("predictive")
```

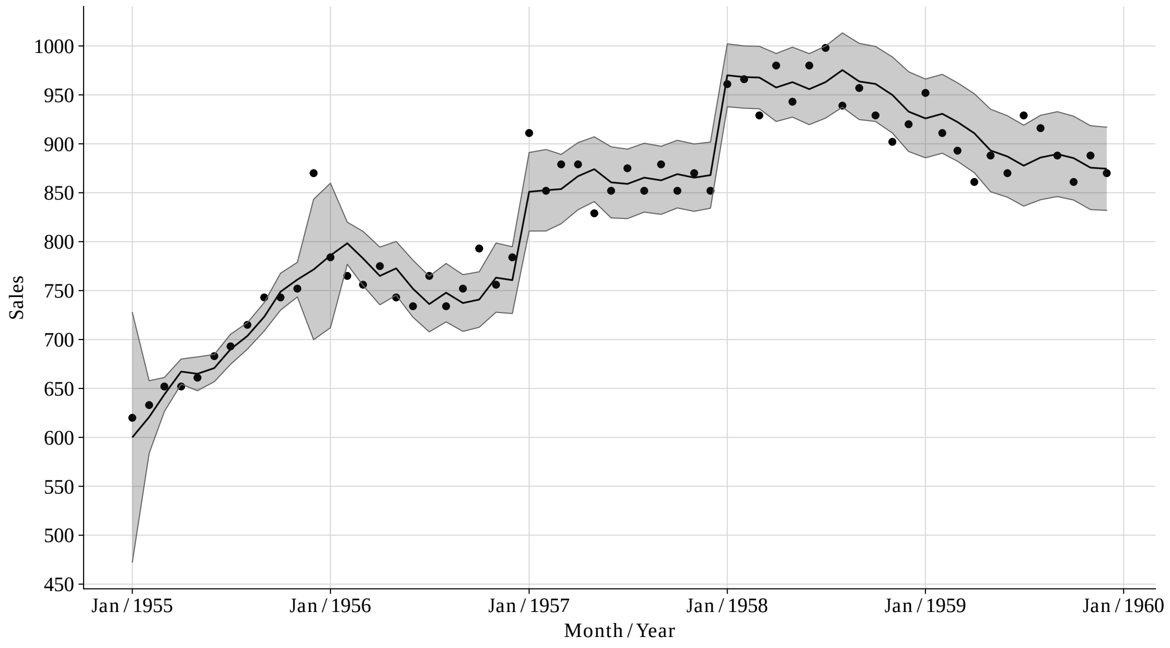


Figure 9: Mean response for filtered predictive distribution with 95% credible interval and ideal interventions

4 Monitoring

The automatic monitoring method sequentially evaluate the forecasting activity to detect breakdowns, based on Bayes factor for two models M_0 versus M_1 with same mathematical structure, differing only through the values for $\boldsymbol{\theta}_t$ or simply the discount factors. Let M_0 be a standard DLM without intervention and M_1 and alternative model that is introduced to provide assessment of M_0 by comparison. The Bayes' factor for the observed value y_t is given by

$$H_t = \frac{p_0(y_t \mid D_{t-1})}{p_1(y_t \mid D_{t-1})},$$

where p_0 and p_1 are the predictive densities at time t for M_0 and M_1.If H_t is small then the M_1 model is preferred. For $k = 1, \ldots, t$ last consecutive observations y_t, y_{t-1}, y_{t-k+1} the local Bayes factor is given by

$$H_t(k) = \prod_{r=t-k+1}^{t} H_r = \frac{p_0(y_t, y_{t-1}, \ldots, y_{t-k+1})}{p_1(y_t, y_{t-1}, \ldots, y_{t-k+1})}.$$

and the cumulative Bayes factor (L_t) is

$$L_t = \min_{1 \le k \le t} H_t(k),$$

the minimum at time t is taken at $k = l_t$, with $L_t = H_t(l_t)$ and l_t being a integer given by

$$l_t = (1 + l_{t-1})I(L_{t-1} < 1) + I(L_{t-1} \ge 1),$$

where $I(\cdot)$ is a indicator function.

Basically, H_t is initially used to indicate if y_t is a outlier when $H_t < \tau$ (which represent preference for M_1). However a small Bayes factor may indicate the start of a regime change, in this case we need to accumulate evidences. For this L_t and l_t are used. The automatic detection is done following the steps

- If $H_t \le \tau$, then y_t is a outlier and is omitted from the analysis.
- If $H_t > \tau$, we must look at L_t for cumulative evidence against M_0.
  - If $L_t < \tau$ or $l_t > 2$ then a parametric chance is detected M_1 is adopted.

It is also possible to consider two alternative models M_1 and M_2, this is useful for identification of outliers/regime change in two directions.

4.1 The `Monitoring` class

The `Monitoring` class implements automatic methods of sequentially monitoring the forecasting activity of DLM in order to detect breakdowns. The model performance is purely based on statistical measures related to model.

An instance of `Monitoring` class can be initialized as follows:

```
>>> from pybats_detection.monitor import Monitoring
>>> monitoring_learning = Monitoring(
>>>     mod: pybats.dglm.dlm, prior_length: int = 10, bilateral: bool = False,
>>>     smooth: bool = True, interval: bool = True, level: float = 0.05)
```

where `mod`, `interval`, `level`, and `smooth` have the same meaning as in `Intervention` class, `prior_length` is an integer that indicates the number of prior observations with the monitor off, and `bilateral` is a Boolean that performs bilateral monitoring if `True`, otherwise unilateral monitoring.

The fit method of `Monitoring` has the following arguments:

```
>>> monitoring_res = monitoring_learning.fit(
>>>      y: pandas.Series, X: pandas.DataFrame = None,
>>>      h: int = 4, tau: float = 0.135,
>>>      discount_factors: dict = {"trend": 0.10, "seasonal": 0.90,
>>>                                 "reg": 0.98},
>>>      distr: str = "normal", type: str = "location", verbose: bool = True)
```

where

- `h` is the location or scale shift for alternative distribution.
- `tau` is the threshold for Bayes' factors, indicating the lower limit on acceptability of L_t. `tau` lies on $(0, 1)$, values between 0.1 and 0.2 being most appropriate.
- `discount_factors` is a dictionary with exceptional discount factors values to increase the uncertainty about state space parameter, when the monitor detects points of intervention. The dictionary should contain values with the following keys representing the model blocks: `trend`: level and growth; `seasonal`: seasonality; and `reg`: regressors.
- `dist` is the Bayes' factors distribution family. It could be `"normal"` or `"tstudent"`.
- `type` is the alternative model use to compute the Bayes' factors. It could be `"location"` to detect change in the location of the distribution or `"scale"` to detect changes in the scale/dispersion of the predictive distribution.
- `verbose` is a Boolean value that if `True` prints the detection of monitor.

As in the `Intervention` class the output object `monitoring_res` has the same similar dictionary structure.

4.2 Examples

The effectiveness of the `Monitoring` class is demonstrated in this section using time series with irregular changes and outliers. Smaller discount factor values are used to increase the state parameter uncertainty when a change is regarded as exceptional, which accelerates model adaption.

4.2.1 Simulate example

For the first 40 observations, the following 20, and the last 40, respectively, this simulated data was generated using a normal distribution, $N[\mu, \sigma^2]$, with $\mu = 100, 104$ and 98 and $\sigma^2 = 0.8, 0.5$ and 0.5. The model was defined and simulated using the code:

```
>>> np.random.seed(66)
>>> y1 = np.random.normal(loc=100, scale=0.8, size=40)
>>> y2 = np.random.normal(loc=104, scale=0.5, size=20)
>>> y3 = np.random.normal(loc=98, scale=0.5, size=20)
>>> y = np.concatenate([y1, y2, y3])
>>> t = np.arange(0, len(y)) + 1
>>> df_simulated = pd.DataFrame({"t": t, "y": y})
```

```
>>> a = np.array([100])
>>> R = np.eye(1)
>>> R[[0]] = 100
>>> mod = dlm(a, R, ntrend=1, deltrend=0.95)
```

The sequential learning with and without the monitor is performed as follows:

```
>>> monitor = Monitoring(mod=mod)
>>> fit_monitor = monitor.fit(y=df_simulated["y"],
>>>                           bilateral=True, h=4, tau=0.135,
>>>                           discount_factors={"trend": 0.10},
>>>                           verbose=True)
```

```
## Upper potential outlier detected at time 41 with H=3.4763e-05, L=3.4763e-05 and l=1
## Upper potential outlier detected at time 42 with H=5.7640e-02, L=5.7640e-02 and l=1
## Lower potential outlier detected at time 61 with H=1.6672e-10, L=1.6672e-10 and l=1
## Lower potential outlier detected at time 62 with H=6.8162e-06, L=6.8162e-06 and l=1
```

Evidence was found against model M_0 at $t = 41$ and $t = 61$, with $L_{41} = 6.85\,e^{-6}$, $L_61 = 2.23\,e^{-4}$, and $L_{61} = 2.23\,e^{-4}$, both with $l_t = 1$, indicating a possible *outlier*. With the arrival of the following observations, a regime change is recognized by the monitor. The interventions performed can be observed in Figure below, where we can see that the model with monitoring quickly adapts to the regime changes (B), compared to the one without monitoring (A).

4.2.2 Telephone Calls

The telephone calls data set concerns the monthly average number of phone calls in Cincinnati, USA. This data features three levels of modifications. The first was in early 1968, with three months of impact; the second was in the middle of 1973, less significant; and the third was in early 1974, more lasting. This data set is available in `pybats-detection` and can be loaded as follows:

```
>>> telephone_calls = load_telephone_calls()
```

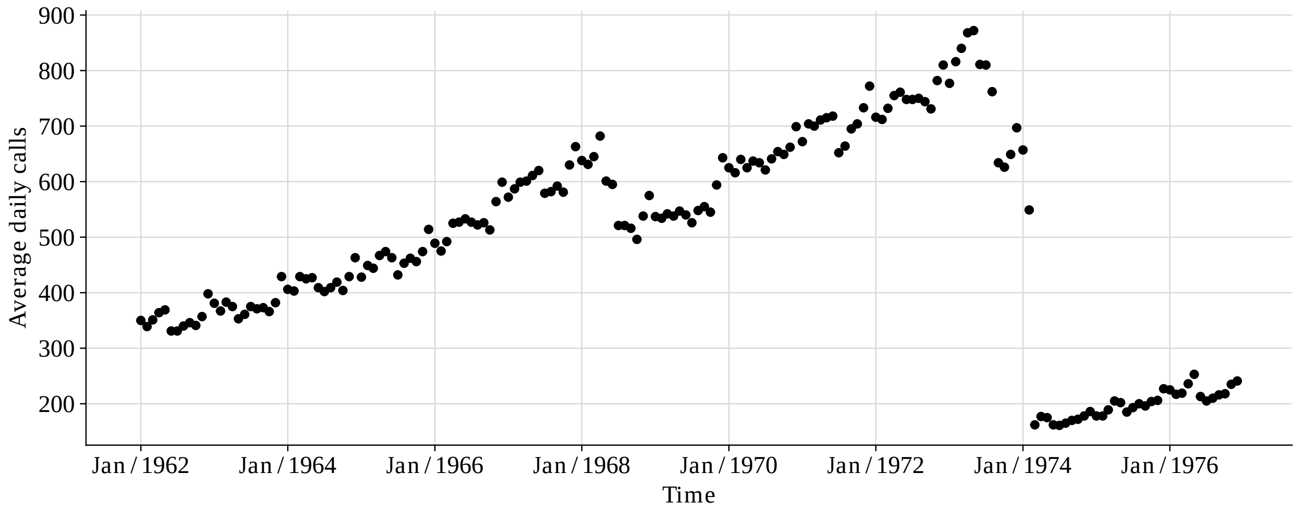


Figure 10: Average daily telephone calls

It is decided to use the linear growth model to explain this phenomenon. We set, as usual, vague prior distributions for the level and growth parameters.

```
>>> a = np.array([350, 0])
>>> R = np.eye(2)
>>> np.fill_diagonal(R, val=[100])
>>> mod = dlm(a, R, ntrend=2, deltrend=0.90)
```

Then, the Monitor class is initialized and the `fit` method is used to update the model sequentially, taking into account the automatic monitoring. Note that the discount factor for the level component has dropped from 0.90 to 0.20, while that for the growth remains the same.

```
>>> monitor = Monitoring(mod=mod)
>>> fit_monitor = monitor.fit(y=telephone_calls["average_daily_calls"], h=4,
>>>                           tau=0.135,
>>>                           discount_factors={"trend": [0.20, 0.90]},
>>>                           bilateral=True, prior_length=40)
```

```
## Upper potential outlier detected at time 48 with H=1.3042e-01, L=1.3042e-01 and l=1
## Upper potential outlier detected at time 60 with H=5.3347e-03, L=5.3347e-03 and l=1
## Upper potential outlier detected at time 72 with H=7.2439e-02, L=7.2439e-02 and l=1
## Lower parametric change detected at time 82 with H=2.2808e-01, L=4.1043e-14 and l=6
## Upper parametric change detected at time 97 with H=3.6864e+00, L=2.3943e-03 and l=3
## Lower potential outlier detected at time 115 with H=1.5791e-02, L=1.5791e-02 and l=1
## Lower potential outlier detected at time 140 with H=1.7892e-03, L=1.7892e-03 and l=1
## Lower potential outlier detected at time 141 with H=3.3177e-08, L=3.3177e-08 and l=1
## Lower potential outlier detected at time 142 with H=2.4040e-03, L=2.4040e-03 and l=1
## Lower potential outlier detected at time 146 with H=2.0036e-06, L=2.0036e-06 and l=1
## Lower potential outlier detected at time 147 with H=2.7869e-21, L=2.7869e-21 and l=1
## Lower potential outlier detected at time 148 with H=2.5817e-09, L=2.5817e-09 and l=1
## Lower potential outlier detected at time 149 with H=6.6363e-03, L=6.6363e-03 and l=1
```

A summary of the changes detection (outlier or parametric change) is printed with the corresponding values for H_t, L_t and l_t. A comparison between the model with and without the automatic monitoring is shown throughout the one-step-ahead forecasts and the corresponding 95% credible interval in Figure below.

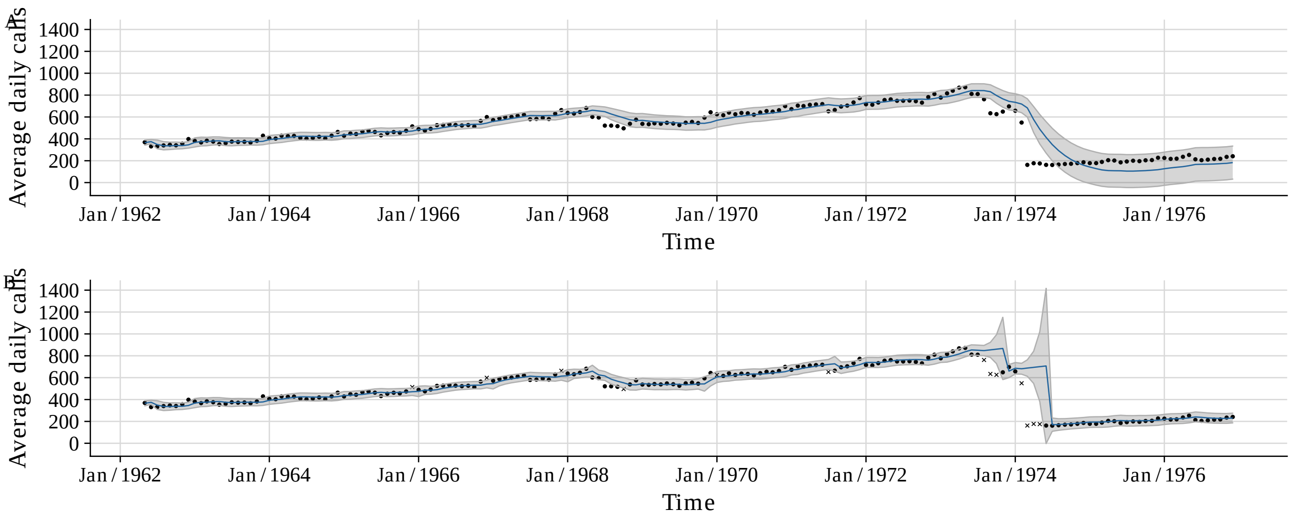


Figure 11: One-step-ahead forecasts with 95% credible interval for the telephone calls data. **A**: without monitoring. **B**: with monitoring. Observations represented by $\times$ indicate instants with intervention.

It is observed that there is a strong difference in the forecasts between the models with and without automatic monitoring. In particular, the adaptation for the future of the model without monitoring is quite poor, which generates large and imprecise credible intervals. Therefore, for forecasting purposes, the model with monitoring is well adapted to the level changes.

4.2.3 Aditional simulation examples

```
>>> np.random.seed(66)
>>> rdlm = RandomDLM(n=50, V=0.1, W=0.005)
>>> df_simulated = rdlm.level(
>>>   start_level=100,
>>>   dict_shift={"t": [40],
>>>     "level_mean_shift": [1],
>>>     "level_var_shift": [1]})
>>> df_simulated.loc[40:50, "y"] = 101 + np.random.normal(0, 0.2, 10)
```

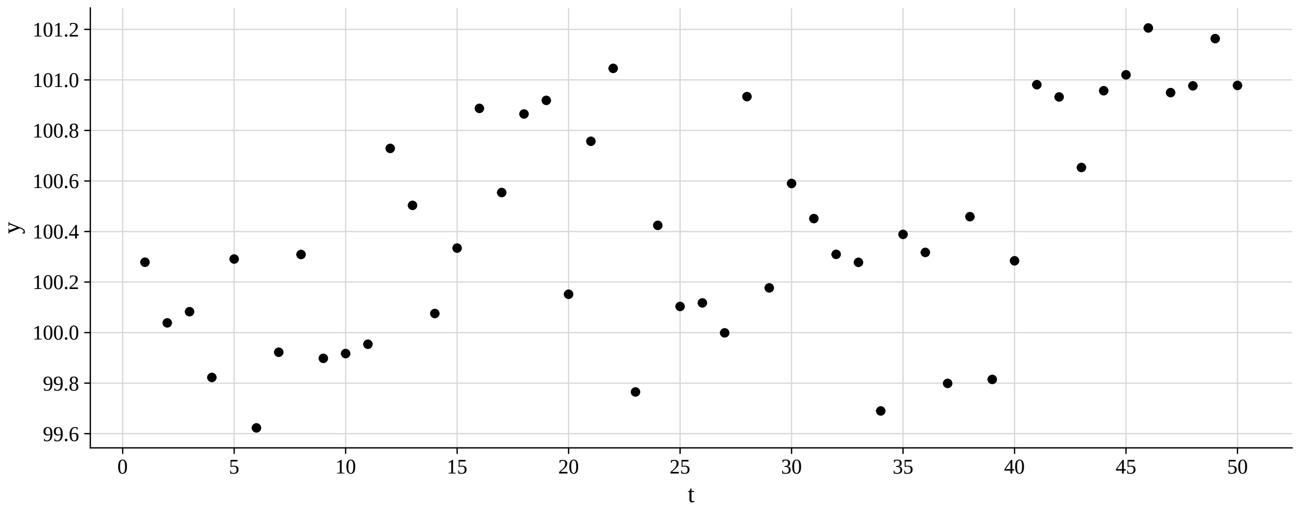


Figure 12: Simulated data with level change

```
>>> a = np.array([100])
>>> R = np.eye(1)
>>> R[[0]] = 100
>>> mod = dlm(a, R, ntrend=1, deltrend=0.9)
>>>
>>> # Fit without monitoring
>>> fit_without_monitor = Smoothing(mod=mod).fit(y=df_simulated["y"])
>>> df_res = fit_without_monitor.get("filter").get("predictive")
>>>
>>> # Fit with monitoring
>>> monitor = Monitoring(mod=mod)
>>> fit_monitor = monitor.fit(y=df_simulated["y"], h=3, tau=0.135,
>>>                           discount_factors={"trend": 0.10})
```

4.2.3.1 Level change

```
## Parametric change detected at time 43 with H=1.2090e+01, L=3.7693e+00 and l=3
```

```
>>> df_tmp = fit_monitor.get("filter").get("predictive")
>>> df_res["monitor"] = False
>>> df_tmp["monitor"] = True
>>> cols_ord = ["t", "y", "f", "q", "ci_lower", "ci_upper", "monitor", "e",
>>>             "H", "L", "l"]
>>> df_res = pd.concat([df_res, df_tmp[cols_ord]]).reset_index(drop=True)
```

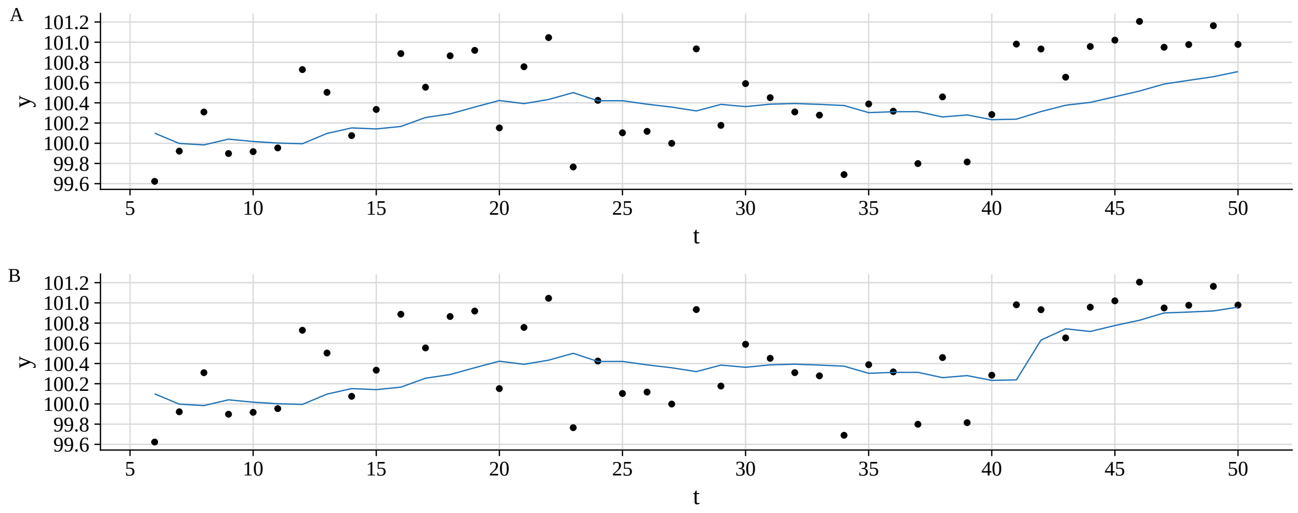
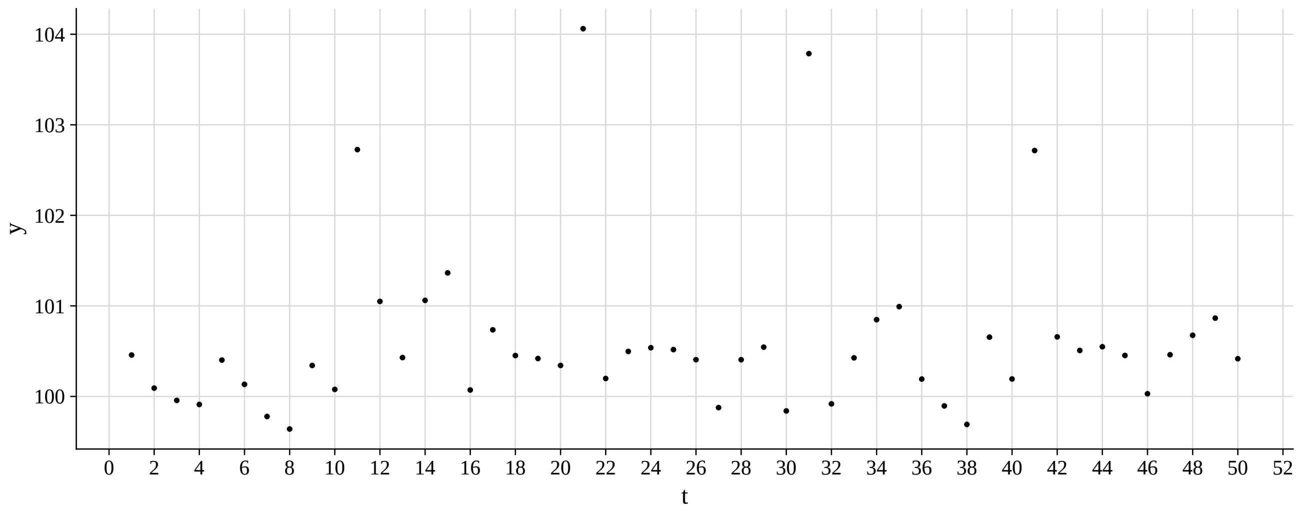


Figure 13: One-step-ahead forecasts for the simulate data with level change. **A**: without monitoring. **B**: with monitoring.

```
>>> np.random.seed(66)
>>> rdlm = RandomDLM(n=50, V=0.1, W=0.01)
>>> df_simulated = rdlm.level(
>>>   start_level=100,
>>>   dict_shift={"t": [10, 11, 20, 21, 30, 31, 40, 41],
>>>     "level_mean_shift": [2, -2, 3, -3, 3.4, -3.4, 3, -3],
>>>     "level_var_shift": [1, 1, 1, 1, 1, 1, 1, 1]})
```

4.2.3.2 Outliers



```
>>> a = np.array([100])
>>> R = np.eye(1)
>>> R[[0]] = 100
>>> mod = dlm(a, R, ntrend=1, deltrend=0.9)
>>>
>>> # Fit without monitoring
>>> fit_without_monitor = Smoothing(mod=mod).fit(y=df_simulated["y"])
>>> df_res = fit_without_monitor.get("filter").get("predictive")
>>>
>>> # Fit with monitoring
>>> monitor = Monitoring(mod=mod)
>>> fit_monitor = monitor.fit(y=df_simulated["y"], h=4, tau=0.135,
>>>                           discount_factors={"trend": 0.10})
```

```
## Potential outlier detected at time 11 with H=2.0200e-08, L=2.0200e-08 and l=1
## Potential outlier detected at time 21 with H=2.9386e-11, L=2.9386e-11 and l=1
## Potential outlier detected at time 31 with H=1.0894e-12, L=1.0894e-12 and l=1
## Potential outlier detected at time 41 with H=1.3753e-07, L=1.3753e-07 and l=1
```

```
>>> df_tmp = fit_monitor.get("filter").get("predictive")
>>> df_res["monitor"] = False
>>> df_tmp["monitor"] = True
>>>
>>> # Append
>>> cols_ord = ["t", "monitor", "y", "f", "q", "ci_lower", "ci_upper"]
>>> df_res = pd.concat([df_res[cols_ord], df_tmp[cols_ord]]).reset_index(drop=True)
```

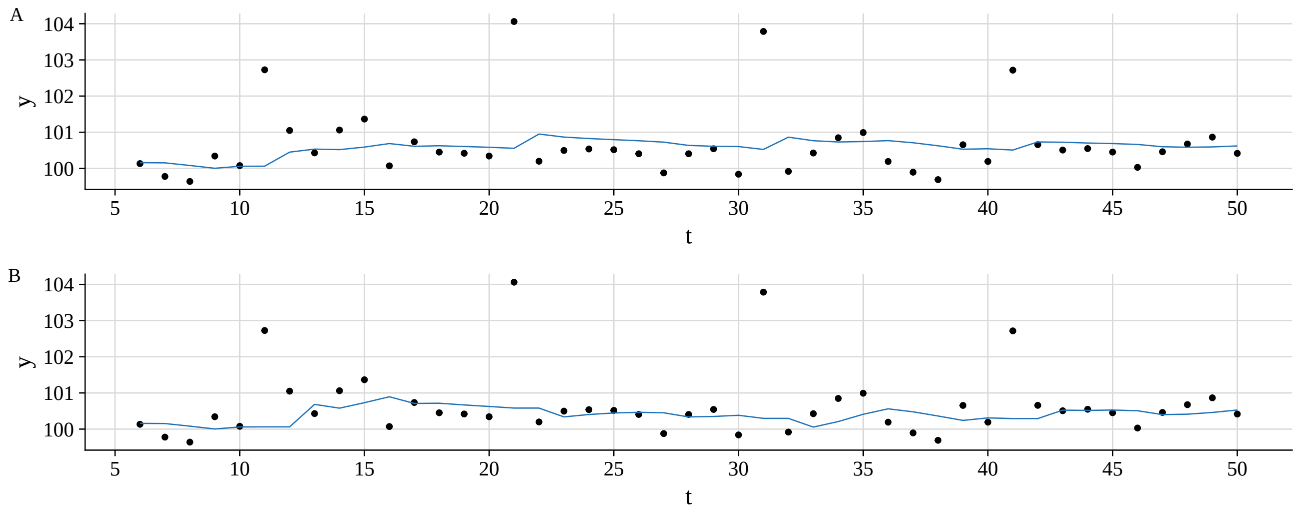
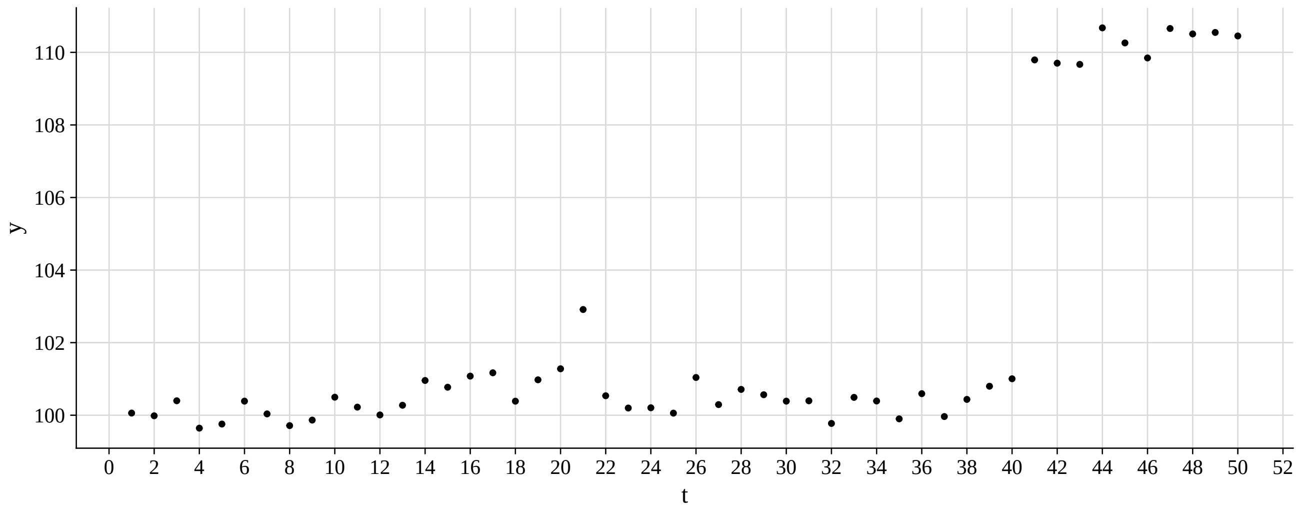


Figure 14: One-step-ahead forecasts for the simulate data with outliers. **A**: without monitoring. **B**: with monitoring.

```
>>> np.random.seed(66)
>>> rdlm = RandomDLM(n=50, V=0.1, W=0.01)
>>> df_simulated = rdlm.level(
>>>   start_level=100,
>>>   dict_shift={"t": [20, 21, 40],
>>>     "level_mean_shift": [3, -3, 10],
>>>     "level_var_shift": [1, 1, 1]})
```

4.2.3.3 Outlier and Level Change



```
>>> a = np.array([100])
>>> R = np.eye(1)
>>> R[[0]] = 100
>>> mod = dlm(a, R, ntrend=1, deltrend=0.9)
>>>
>>> # Fit without monitoring
>>> fit_without_monitor = Smoothing(mod=mod).fit(y=df_simulated["y"])
>>> df_res = fit_without_monitor.get("filter").get("predictive")
>>>
>>> # Fit with monitoring
>>> monitor = Monitoring(mod=mod)
>>> fit_monitor = monitor.fit(y=df_simulated["y"], h=4, tau=0.135,
>>>                           discount_factors={"trend": 0.10})
```

```
## Potential outlier detected at time 21 with H=3.1219e-05, L=3.1219e-05 and l=1
## Potential outlier detected at time 41 with H=3.0603e-34, L=3.0603e-34 and l=1
## Potential outlier detected at time 42 with H=4.4933e-23, L=4.4933e-23 and l=1
## Potential outlier detected at time 43 with H=9.9249e-08, L=9.9249e-08 and l=1
```

```
>>> df_tmp = fit_monitor.get("filter").get("predictive")
>>> df_res["monitor"] = False
>>> df_tmp["monitor"] = True
>>>
>>> # Append
>>> cols_ord = ["t", "monitor", "y", "f", "q", "ci_lower", "ci_upper"]
>>> df_res = pd.concat([df_res[cols_ord], df_tmp[cols_ord]]).reset_index(drop=True)
```

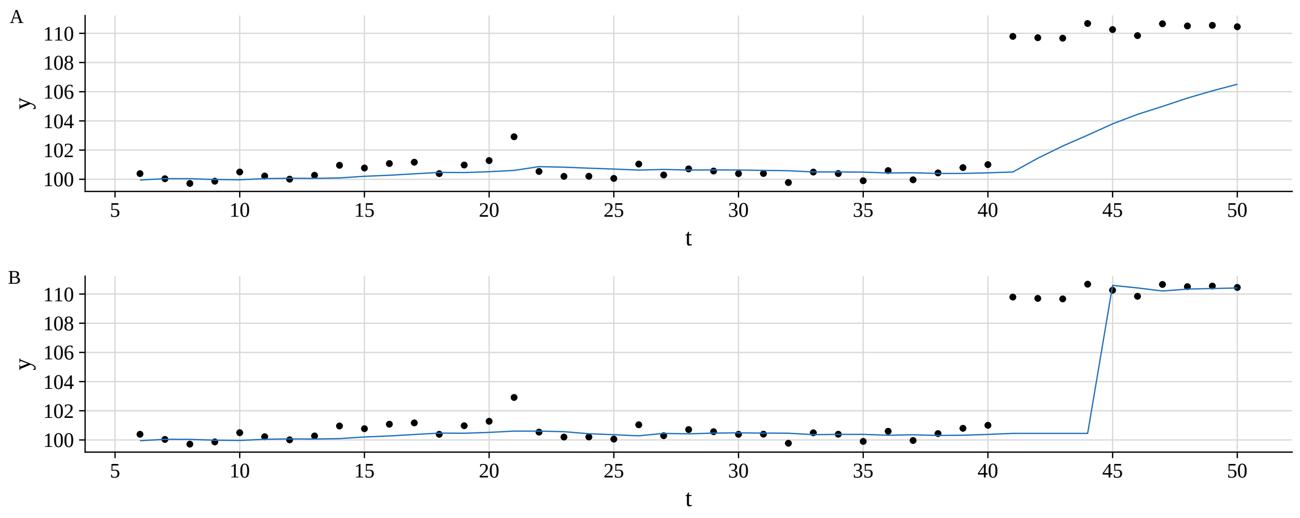
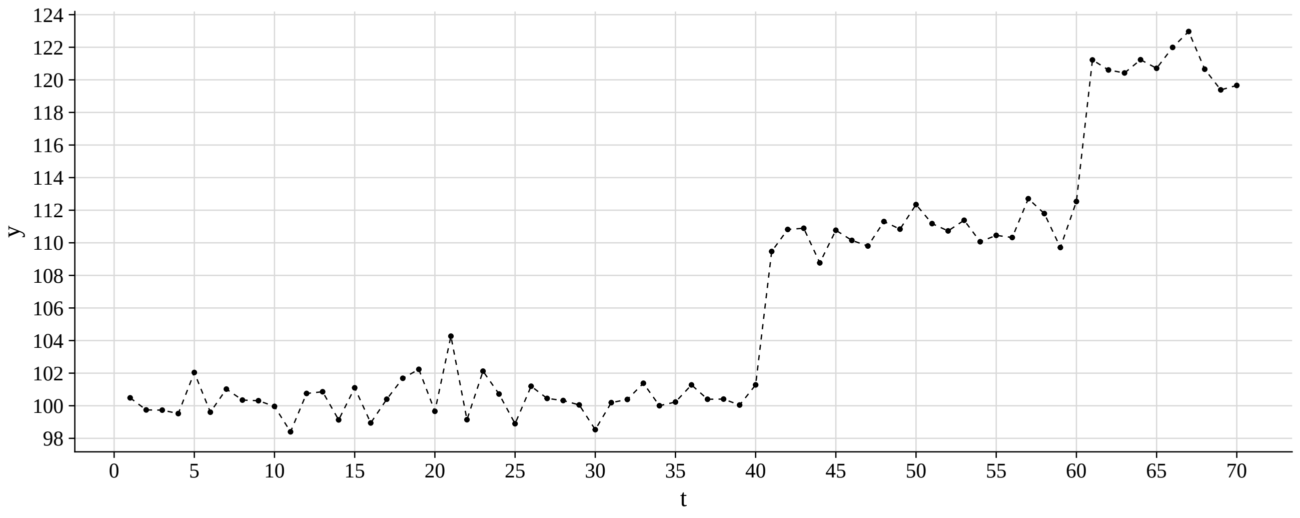


Figure 15: One-step-ahead forecasts for the simulate data with outliers and level change. **A**: without monitoring. **B**: with monitoring.

```
>>> np.random.seed(66)
>>> rdlm = RandomDLM(n=70, V=1, W=0.01)
>>> df_simulated = rdlm.level(
>>>   start_level=100,
>>>   dict_shift={"t": [20, 21, 40, 60],
>>>     "level_mean_shift": [5, -5, 10, 10],
>>>     "level_var_shift": [1, 1, 1, 1]})
```

4.2.3.4 Outlier and Two Level Change



```
>>> a = np.array([100])
>>> R = np.eye(1)
>>> R[[0]] = 100
>>> mod = dlm(a, R, ntrend=1, deltrend=0.9)
>>>
>>> # Fit without monitoring
>>> fit_without_monitor = Smoothing(mod=mod).fit(y=df_simulated["y"])
>>> df_res = fit_without_monitor.get("filter").get("predictive")
>>>
>>> # Fit with monitoring
>>> monitor = Monitoring(mod=mod)
>>> fit_monitor = monitor.fit(y=df_simulated["y"], h=4, tau=0.135,
>>>                           discount_factors={"trend": 0.10})
```

```
## Potential outlier detected at time 21 with H=1.0052e-03, L=1.0052e-03 and l=1
## Potential outlier detected at time 41 with H=2.4362e-13, L=2.4362e-13 and l=1
## Potential outlier detected at time 42 with H=2.7146e-10, L=2.7146e-10 and l=1
## Potential outlier detected at time 43 with H=1.3395e-02, L=1.3395e-02 and l=1
## Potential outlier detected at time 61 with H=8.6538e-15, L=8.6538e-15 and l=1
## Potential outlier detected at time 62 with H=9.8401e-09, L=9.8401e-09 and l=1
## Potential outlier detected at time 63 with H=8.5385e-02, L=8.5385e-02 and l=1
```

```
>>> df_tmp = fit_monitor.get("filter").get("predictive")
>>> df_res["monitor"] = False
>>> df_tmp["monitor"] = True
>>>
>>> # Append
>>> cols_ord = ["t", "monitor", "y", "f", "q", "ci_lower", "ci_upper"]
>>> df_res = pd.concat([df_res, df_tmp[cols_ord]]).reset_index(drop=True)
```

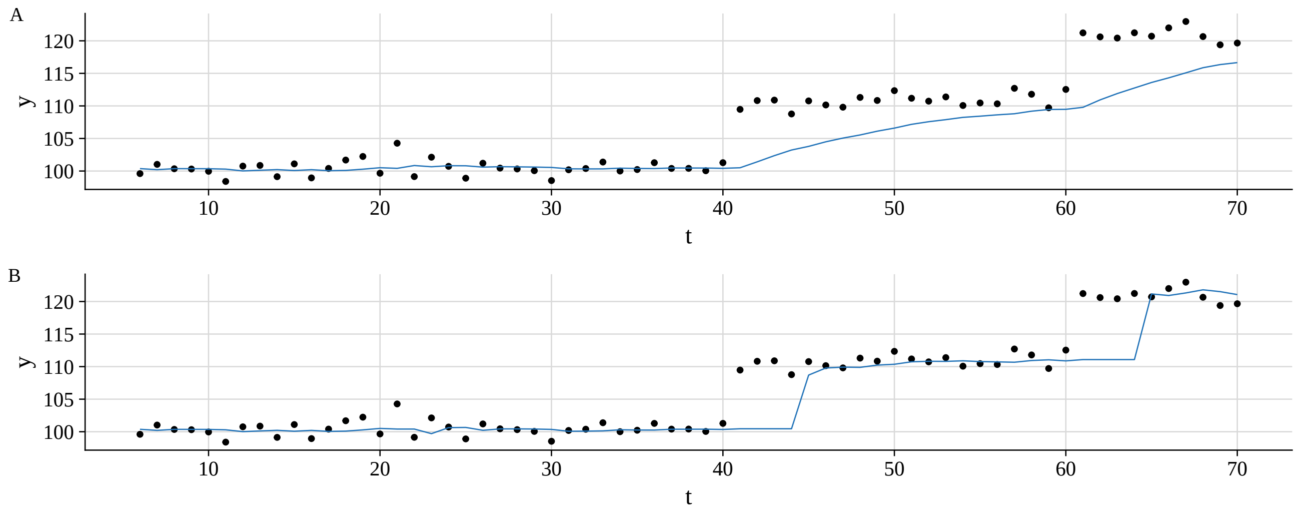


Figure 16: One-step-ahead forecasts for the simulate data with outliers and two level change. **A**: without monitoring. **B**: with monitoring.

5 References

- Harrison, P.J., Stevens, C.F., 1976. Bayesian forecasting. Journal of the Royal Statistical660 Society. Series B (Methodological) 38, 205–247

- West, M., Harrison, J., 1989. Subjective intervention in formal models. Journal of Forecasting 8, 33–53. doi:10.1002/for.3980080104.

- West, M., Harrison, P.J., 1986. Monitoring and adaptation in Bayesian forecasting models. Journal of the American Statistical Association 81, 741–750. doi:10.1080/01621459.1986.10478331.