Project Outside the Course Scope

Algorithms for Change Detection in Satellite Image Time Series

Dmitry Serykh
dmise@di.ku.dk

August 23, 2020

Contents

Chapter 1

Introduction

1.1 Motivation

Climate change is the defining issue of our time, and we are at a defining moment [14]. Without drastic and swift action in the nearest future, adapting to these changes is becoming progressively more challenging. One way, in which we can fight this development is by monitoring and reducing global deforestation. If the areas of deforestation are registered in a timely fashion, targeted countermeasures could be applied, and the tide on this trend could be turned.

Meanwhile, enormous amount of satellite data is becoming available to the earth science community. This image data can, inter alia, be utilized for the time-series based change-detection, which is a promising tool for monitoring and mapping of deforestation and forest degradation. Most of these approaches operate on individual pixels. Therefore, the sheer scale of this problem (being hundreds of billions of pixels) becomes its most significant challenge, since there are petabytes of image data being added on a yearly basis. Without an efficient data-parallel implementation, this approach quickly becomes infeasible in practice, even when equipped with significant computational resources and specialized hardware.

Recently, a novel massively-parallel implementation for a change detection method has been proposed [9]. It combines state-of-the-art methods for the seasonal data change-detection with modern data-parallel programming models and yields an implementation that makes the task feasible for giant sets. Furthermore, this implementation is feasibly executable on consumer-grade graphical processing hardware. The authors of the paper provide a massively-parallel implementation for the BFAST-Monitor unsupervised change detection algorithm. In this project, I will research alternative approaches to BFAST-Monitor.

1.2 BFAST

Remotely sensed long term data is a valuable resource that was shown to be applicable for change detection. There are three types of changes over time that are of interest to the earth science research:

1. **Seasonal change**: changes that happen withing a season (e.g. year)
2. **Gradual change**: changes that are caused by interannual climate variability.
3. **Abrupt change**: rapid changes that are triggered by deforestation, floods, fires and similar.

In the paper from 2010, Verbesselt et al. [20], describe an approach for time series change detection, which involves detection of multiple abrupt changes in the seasonal and trend components of the time series and characterization of such changes by their magnitude and direction. The piecewise linear model is assumed for the trend component and a periodic models is applied to the seasonal component. The coefficients for the linear models are estimated using ordinary least squares (OLS) based linear regression. This report covers the underlying theory and implementation of two algorithms based on this approach (BFAST0n and BFAST).

1.3 Contributions

The principal contributions of this project are:

- I describe the underlying theory and concrete steps of the Seasonal and Trend decomposition using Loess (STL) time series decomposition algorithm, which is an essential part of BFAST. In particular, the description and values for all algorithm parameters are covered (Chapter 2).
- Describe the underlying theory and link it to the concrete steps of the Ordinary Least Squares Moving Sum (OLS-MOSUM) test from the `strucchange` package [24] for the R programming language by Achim Zeileis. I also provide examples to illustrate the operation of the non-trivial parts of the algorithm. OLS-MOSUM test is another important building block of BFAST (Chapter 3).
- Describe the theory and algorithm steps of the multiple breakpoint estimation algorithm by Bai and Perron, based on the paper from 2003 [2] and provide illustrative examples. This dynamic programming-based algorithm is an integral component of BFAST (Chapter 4).
- Describe the theory and steps of the BFAST and BFAST0n structural change detection algorithms by Jan Verbesselt et. al. [20]. In particular, two seasonal models are explained in depth and matrix form of both models is provided (not included in the original paper). (Chapters 6 and 5).
- Provide and describe a prototype implementation of BFAST0n and BFAST in a general-purpose programming language (Python) (Chapter 7).
- Validate the implementation using simulated and real-life small datasets from the original paper by Verbesselt et al. [20]. The results are compared to the original R implementation [17]. (Chapter 8).

Chapter 2

Seasonal Trend Decomposition (STL)

STL, as first described by Cleveland et al. [5], is an algorithm for decomposition of time series, which is a common task in statistics and used by the BFAST algorithm to get the initial estimate of the seasonal componenet $\hat{S}_t$. STL decomposes a time series Y_v into three components:

$$Y_v = T_v + S_v + R_v \text{ for } v \in 1..N$$

where:

- T_v is the trend component, which captures the long-term progression of the time-series
- S_v is the seasonal component, which reflects the seasonal variation of the data
- R_v is the remainder(residual) component, which is the remaining variation of the time series

An example of such decomposition is illustrated on Figure 2.1. STL alone, however, can not be used for break detection because it uses smoothing that would temper with the breakpoint information.

2.1 Locally Weighted Regression (LOESS)

The key component of the STL method is the Locally Weighted Regression (LOESS), that is described in [6] and [5]. LOESS is a robust smoothing method that is commonly used in statistics and operates as follows:

- We have a set of (x_i, y_i) for $1 \leq i \leq n$.
- We wish for all x fit a curve $\hat{g}(x)$ by giving the other points x_i a weight v_i.
- We select a value of $q \in \mathbb{Z}^+$
- Let $\lambda_q(x)$ be the distance from x to q'th farthest x_i. This definition would not, however, work when $q > n$. In that case, we must apply scaling by setting:

  $$\lambda_q(x) = \frac{q \cdot \lambda_n(x)}{n}$$

- We calculate the weights using the tricube weight function:

  $$v_i = \left(1 - \left(\frac{|x_i - x|}{\lambda_q(x)}\right)^3\right)^3$$

  we also add a low shelf, for $|x_i - x| \geq \lambda_q(x)$, we set $v_i = 0$
- We then use locally-linear or locally quadratic fitting with weights $\rho_i v_i$, where ρ_i are the robustness weights, used by the STL. Addition of ρ_v is crucial, since it allows us to ignore the outliers in the dataset and in this way introduce robustness into STL. The value of locally-fitted polynomial at x is $\hat{g}(x)$, and d is the degree of the fitted polynomial.

In this algorithm, q is the smoothing factor. The higher the value of q is, the more the curve resembles a fitted polynomial of degree d for all x. Another important parameter is the value of d. Cleveland et al. [5] suggest using $d \in \{1, 2\}$ for STL. Value of 2 is used when the function of interest has a more pronounced curvature. An example of the usage of LOESS smoothing can be seen on Figure 2.2.

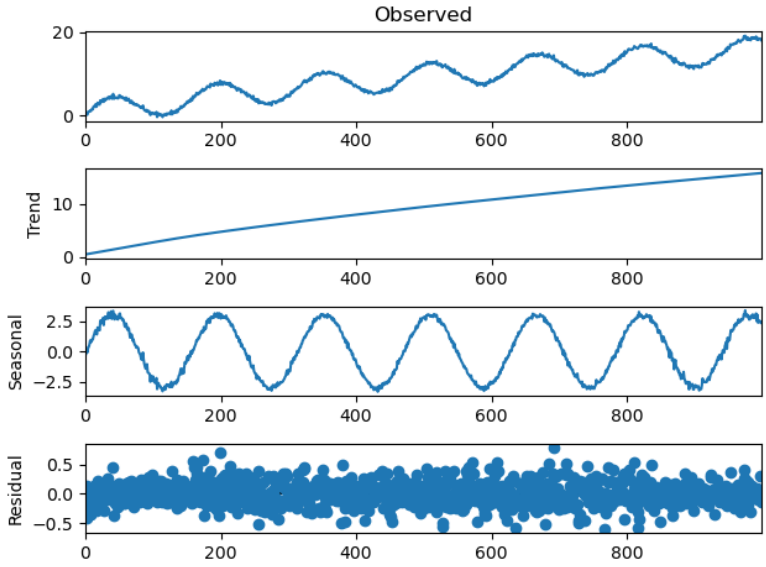
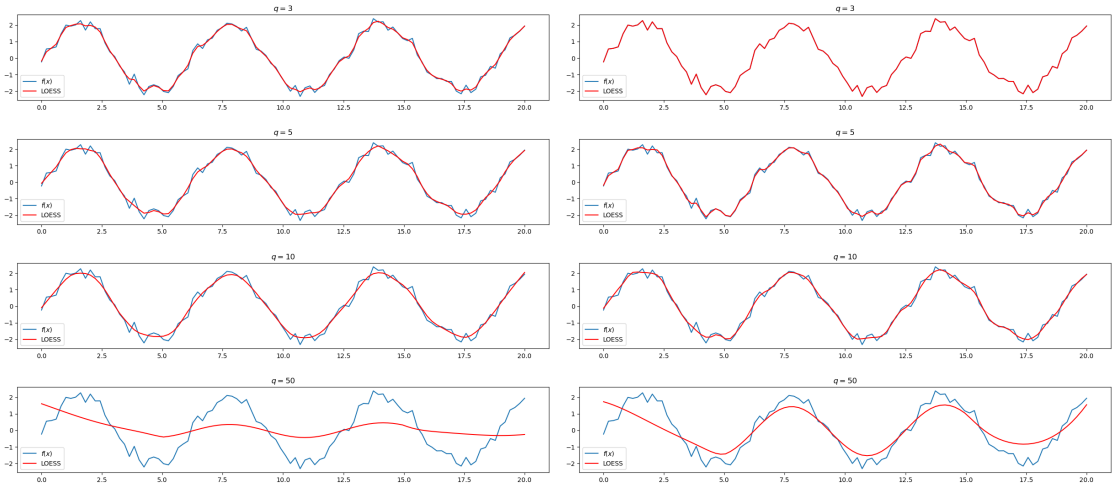


Figure 2.1: STL applied to $f(x) = x^{0.75} + 2\sin(x)$ with added Gaussian noise $\sim \mathcal{N}(0, 0.25)$ and $n = 1000$. It can be seen that $T_v \approx v^{0.75}$, $S_v \approx 2\sin(v)$ and $R_v \sim \mathcal{N}(0, 0.25)$.



(a) $d = 1$ (b) $d = 2$

Figure 2.2: LOESS applied to $f(x) = 2\sin(x)$ with added Gaussian noise and $n = 100$

2.2 The Incremental Algorithm

STL is performed through two nested loops and we begin by setting:

$$T_v^0 = 0$$
$$R_v^0 = 0$$
$$\rho_v = 1$$

2.2.1 Inner loop

The inner loop consists of following steps:

1. **Detrending**: $Y_v - T_v^k$, where k is the iteration number. Note however, that if a value of Y_v is missing, the detrending value must also be missing.
2. **Cycle-Subseries Smoothing**: $Y_v - T_v^k$ is split into cycle-subseries. LOESS with $q = n_s$ and $d = 1$ is applied to each cycle-subseries, resulting in C^{k+1}.
3. **Low-pass Filter of Smoothed Cycle-Subseries**: Apply the low-pass filter to C_{k+1}. This is accomplished by application of two moving averages of lag equal to 3 followed by LOESS smoothing with $q = n_l$ and $d = 1$. The result is saved as L^{k+1}
4. **Detrending of the Smoothed Cycle-Subseries**: $S^{k+1} = C^{k+1} - L^{k+1}$
5. **Deseasoning**: $Y_v - S_v^k$. If a value of Y_v is missing, the detrending value must also be missing.
6. **Trend Smoothing**: Apply LOESS to $Y_v - S_v^k$ with $q = n_t$, resulting in T^{k+1}

2.2.2 Outer loop

The outer loop consists of following steps:

1. Run the inner loop
2. Find the remainder: $R_v = Y_v - T_v - S_v$
3. Calculate the robustness weights from the remainder component:

$$\rho_v = B\left(\frac{|R_v|}{6 \cdot \text{median}(|R_v|)}\right)$$

where B is the bisquare weight function:

$$B(x) = \begin{cases} \left(1 - x^2\right)^2 & \text{for } 0 \leqslant x < 1 \\ 0 & \text{for } x > 1 \end{cases}$$

2.3 Parameters and Their Values

STL has following parameters [5]:

1. n_p : period of the seasonal component in number of observations. Specific for each data set.
2. n_i : number of inner loop iterations. `stlplus`, which is popular STL library that is used in the BFAST R implementation [17], uses a value of $n_i = 2$ [10].
3. d: degree for the polynomials that is being fitted. `stlplus` uses a value of $d = 1$ for all smoothing.

4. n_o : number of the outer loop iterations. The default value in `stlplus` is 0. This can be adjusted if outliers are present in the dataset.

5. n_l : value of the smoothing parameter (q) for the LOESS procedure in the **Low-pass filter** stage of the inner loop. Typically (which includes `stlplus`), n_l is the smallest odd integer greater than the period.

6. n_t : value of q for the trend component, which should be odd. `stlplus` uses following:

$$n_t = \text{nextodd}\left(\left\lceil \frac{1.5 \cdot n_p}{1 - 1.5/n_s} \right\rceil\right)$$

7. n_s : value of q for the seasonal component. According to Cleveland et al. [5], it must be specifically and carefully chosen for each application. `stlplus` allows the user to select $n_s =$ "*periodic*", which is, according to the package documentation [10], equivalent to replacing the smoothing in the seasonal component with taking the mean. It is the value, used by the BFAST R-implementation and consequently for this project.

Chapter 3

OLS-MOSUM Test

Tests from the generalized fluctuation test framework [11] is one of the most important classes of tests on structural change. This class of tests include, in particular, Moving Sum (MOSUM) test, which is the topic of this chapter. In the paper from 2003 [24], Zeileis et al. describe the underlying theory and usage of the `strucchange` package for the R programming language that includes multiple tools for detection of structural changes in linear regression relationships.

3.1 The Model

In `strucchange` [24], a standard linear regression model is considered:

$$y_i = x_i^\top \beta_i + u_i \quad (i = 1, \ldots, n)$$

where:

- i is the observation number
- $x_i = (1, x_{i2}, x_{i3}, ..., x_{ik})^T \in \mathbb{R}^k$, given to us
- $y_i \in \mathbb{R}$, also given
- $\beta_i \in \mathbb{R}^k$ is the regression coefficient vector, that we wish to estimate using linear regression
- $u_i \in \mathbb{R}$ is the residual term that is independently and identically distributed with mean $\mu = 0$ and variance σ^2. This is the difference between the prediction (using β_i) and the actual observation y_i.

We can then test for structural change by testing the null hypothesis:

$$H_0 : \quad \beta_i = \beta_0 \quad (i = 1, \ldots, n)$$

which states that there is no structural change present in the time series. While the alternative is that the coefficient vector β varies over time - hence is are one or more breakpoints.

Let $\hat{\beta}^{(i)}$ be the ordinary least squares (OLS) estimate of the regression coefficients based on all the observations up to i. Therefore, it follows that $\hat{\beta}^{(n)}$ is the common OLS estimate in the regression model.

Let the OLS residuals be defined as:

$$\hat{u}_i = y_i - x^T \hat{\beta}^{(n)} \tag{3.1}$$

the variance estimate would then be:

$$\hat{\sigma}^2 = \frac{1}{n-k} \sum_{i=1}^{n} \hat{u}_i^2 \tag{3.2}$$

3.2 Empirical Fluctuation Process (OLS-MOSUM)

It is possible to detect structural change by analyzing moving sum of residuals [24]. The resulting empirical fluctuation process consists of a sum of a fixed number of residuals in a data interval, which size is determined by the value of the parameter $h \in (0, 1)$ (bandwidth).

The OLS-based MOSUM process is given by [3]:

$$M(t) = \frac{1}{\hat{\sigma}\sqrt{n}} \left(\sum_{i=\lfloor N_n t \rfloor + 1}^{\lfloor N_n t \rfloor + \lfloor nh \rfloor} \hat{u}_i \right) \quad (0 \leq t \leq 1 - h) \tag{3.3}$$

where $N_n = (n - \lfloor nh \rfloor)/(1 - h)$ and $\lfloor nh \rfloor$ is the size of the window.

This expression can be simplified by first re-scaling the expression s.t. $0 \leq t \leq 1$:

$$M(t) = \frac{1}{\hat{\sigma}\sqrt{n}} \left(\sum_{i=\lfloor t(n-\lfloor nh \rfloor) \rfloor + 1}^{\lfloor t(n-\lfloor nh \rfloor) \rfloor + \lfloor nh \rfloor} \right) \quad (0 \leq t \leq 1) \tag{3.4}$$

Then, we can apply a second re-write, s.t. $t \in \mathbb{Z}^+ \wedge t \leq (n - \lfloor nh \rfloor + 1)$:

$$M(t) = \frac{1}{\hat{\sigma}\sqrt{n}} \left(\sum_{i=t}^{t+\lfloor nh \rfloor} \right) \quad (t \in \mathbb{Z}^+ \wedge t \leq (n - \lfloor nh \rfloor + 1)) \tag{3.5}$$

3.3 Significance Testing

A key observation is that if a structural change takes place at t_0, the OLS-MOSUM path would also have a strong shift at t_0. We reject the null hypothesis of there being no structural change, when the fluctuation of the OLS-MOSUM process becomes too large.

In practice, we determine whether the null hypothesis can be rejected using a significance test (also called statistical hypothesis testing). First, we calculate the test statistic. For the residual-based OLS-MOSUM process, it is defined as:

$$S_{\text{MOSUM}} = \max(\|\text{M(t)}\|) \quad \text{for } t \text{ in } 1, 2, .., (n - \lfloor nh \rfloor + 1) \tag{3.6}$$

where $\|.\|$ signifies the absolute value.

We use the value of S_{MOSUM} to calculate the probability of getting such sample, given that the null hypothesis holds, using the critical values approach. We then compare the resulting probability with a chosen confidence interval, which is typically some low value of $\alpha = 0.05$ or $\alpha = 0.01$. If the resulting probability is below the value of α, we reject the null hypothesis. In our case, it would mean that a structural change is detected in the time series.

3.4 Steps of the Algorithm

We determine whether the structural changes are present in the time series by following these steps:

1. **Find the value of $\hat{\beta}^{(n)}$ using OLS-based linear regression using the vectors of observations x and y**
2. **Calculate the OLS residuals $\hat{\mu}$, using** (3.1)

3. **Calculate the standard deviation $\hat{\sigma}$, using** (3.2)

4. **Calculate the residual-based OLS-MOSUM process from** (3.5).
   We need to calculate the values of the $M(t)$ for all positive integer values of $t \leq (n - \lfloor nh \rfloor + 1)$. Hence, we can use the cumulative sum to calculate the process by evaluating following (using zero-indexing):

   ```
   nh = floor(n * h)
   e = concat([0], residuals)
   process = cumsum(e)
   process = process[nh:n] - process[0:(n - nh + 1)]
   process = process / (sigma * sqrt(n))
   ```

5. **Calculate the test statistic, using** (3.6)

6. **Calculate the p-value**:
   The p-value is calculated from the table of simulated asymptotic critical values of the Moving Estimate (ME) tests with the maximum norm, given by Chu et al. [4]. The structure of the table can be seen in Table 3.1 and there is a 4×10 table for each value of $n_{\text{dim}} \in \{1..6\}$, where n_{dim} is the size of the vector x_i, without the added identity row, hence $n_{\text{dim}} = k - 1$. The columns in the table sections correspond to the tail probabilities $p \in \{0.10, 0.05, 0.025, 0.001\}$, while the rows correspond to the values of the bandwidth parameter $h \in \{0.05, 0.10, ..., 0.5\}$. The values in the table are the values of the test statistic S_{MOSUM}.

   We start by selecting the table section that corresponds to our value of $k-1$. Then we select the table row for our value of h. However, when the value of $h \notin \{0.05, 0.10, ..., 0.5\}$ we use one-dimensional piecewise linear interpolation in order to get the four values of S_{MOSUM} that correspond to the tail probabilities $p \in \{0.1, 0.05, 0.025, 0.01\}$.

   Finally, we apply the linear interpolation in order to caclulate the tail probability that corresponds to the value of the test statistic (S_{MOSUM}). This is our p-value.

   Consider this example. Let us assume that we are working with a time series, $x_i = (1, x_{i1})^T$, hence $n_{\text{dim}} = 1$ and we choose the first section of the table. Let the bandwidth parameter $h = 0.12$. This value is not present in the set $\{0.05, 0.10, ..., 0.5\}$. Therefore, we must perform a linear interpolation in order to obtain the four values of the test statistic that correspond to tail probabilities $\{0.1, 0.05, 0.025, 0.01\}$. We calculate following values: $\{1.03698, 1.11134, 1.18094, 1.26396\}$. Let the value of our statistic be $S_{\text{MOSUM}} = 1.1914$. We notice that the value of 1.1914 lies between the last two values that we have just calculated. In order to calculate the tail probability that corresponds to 1.1914, we perform another linear interpolation and calculate the tail probability $p = 0.023$.

7. **Return the result**:
   If $p \leq \alpha$, we reject the null-hypothesis. In other words, we detect structural change in the time-series, otherwise we do not.

   For the example from the previous step, assume the confidence interval $\alpha = 0.05$. Then $p = 0.023 < \alpha$ and we must reject the null-hypothesis of no structural change.

3.5 Parameters and Their Values

The OLS-MOSUM test uses two parameters:

- The bandwidth parameter h. The BFAST R implementation [17], uses 0.15.
- Significance level α. The BFAST implementation [17], uses a ubiquitous value of 0.05

n_{dim}	h	Tail probability			
		0.10	0.05	0.025	0.01
1	0.05	0.7552	0.8017	0.8444	0.8977
	0.10	0.9809	1.0483	1.1119	1.1888
	...	...	...	...	...
	0.50	1.3560	1.4938	1.6166	1.7663
2	0.05	0.7997	0.8431	0.8838	0.9351
	0.10	1.0448	1.1067	1.1634	1.2388
	...	...	...	...	...
	0.50	1.4884	1.6125	1.7266	1.8639
...	...	...	...	...	...

Table 3.1: An excerpt from the table of simulated asymptotic critical values of the ME tests with the maximum norm [4].

Chapter 4

Estimation of Breakpoints

In the paper from 2003 [2], Jushan Bai and Pierre Perron provide an efficient algorithm for time series multiple breakpoint estimation that uses a dynamic-programming approach and requires $O(T^2)$ least-squares operations for any number of breakpoints.

4.1 The Model

We assume a pure structural change model for m breaks ($m+1$ segments):

$$y_t = x_t^\top \beta_j + u_t \quad t = T_{j-1}+1, \ldots, T_j$$

for $j = 1, ..., m+1$, and where:

- $x_t \in \mathbb{R}^q$ is the value of the independent variable at time $t = 1..T$, known
- $y_t \in \mathbb{R}$ is the observation at time t, known
- β_j $(j = 1, ..., m+1)$ is the vector of coefficients that we wish to estimate using linear regression
- $u_t \in \mathbb{R}$ is the disturbance (error) at time t, unknown
- $(T_1, .., T_m)$ are the indices of the m breakpoints and are unknown. We additionally set $T_0 = 0$ and $T_{m+1} = T$

This linear regression system can also be expressed in matrix form:

$$Y = \overline{X}\beta + U$$

where:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \end{bmatrix} \quad \beta = \begin{bmatrix} | & | & & | \\ \beta_1 & \beta_2 & \cdots & \beta_{m+1} \\ | & | & & | \end{bmatrix} \quad U = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_T \end{bmatrix}$$

and $\overline{X}$ is a block matrix that diagonally partitions X at breaking points $(T_1, ..., T_m)$:

$$\overline{X} = \begin{bmatrix} X_1 & 0 & \ldots & 0 \\ 0 & X_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & X_{m+1} \end{bmatrix} \quad X_i = \begin{bmatrix} — & x_{T_{i-1}+1}^\top & — \\ — & x_{T_{i-1}+2}^\top & — \\ & \vdots & \\ — & x_{T_i}^\top & — \end{bmatrix}$$

In other words, we split the time series into $m+1$ segments (of potentially different sizes) and perform linear regression independently for each segment, where for segment i, we estimate a coefficient vector β_i. Hence, we find the estimate of the coefficient vector $(\hat{\beta})$ for each m-partition $(T_1, ..., T_m)$ by minimizing the sum of squared residuals:

$$(Y - \overline{X}^\top \beta)^\top (Y - \overline{X}^\top \beta) = \sum_{i=1}^{m+1} \sum_{i=T_{i-1}+1}^{T_i} \left[y_t - x_t^\top \beta_i \right]^2$$

starting date \ Stopping date	1	2	3	4	5	6	7	8	9
1	nf_1	nf_1	nf_1	f	f	f	nf_2	nf_2	nf_2
2		nf_1	nf_1	nf_1	nf_3	nf_3	nf_2	nf_2	nf_2
3			nf_1	nf_1	nf_1	nf_3	nf_2	nf_2	nf_2
4				nf_1	nf_1	nf_1	f	f	f
5					nf_1	nf_1	nf_1	f	f
6						nf_1	nf_1	nf_1	f
7							nf_1	nf_1	nf_1
8								nf_1	nf_1
9									nf_1

Table 4.1: An example of an upper-triangular matrix of squared residuals for $T = 9$, $h = 3$, $m = 2$. We must compute sum of squared residuals for all feasible segments (f).

Let $\{T_j\}$ denote an m-partition $(T_1, ..., T_m)$ and $S_T(T_1, ..., T_m)$ denote the resulting sum of squared residuals. Since we can estimate the coefficient vector for each partition, we can minimize the sum of squared residuals by finding the optimal position for the breakpoints

$$(\hat{T}_1, ..., \hat{T}_m) = \text{argmin}_{T_1,...,T_m} S_T(T_1, \ldots, T_m)$$

There is a finite number of possible breakpoints, and only a subset of possible partitions is feasible. There are $T(T+1)/2$ (sum of integers from 1 to T) possible segments that can be chosen. This can be demonstrated by building a matrix of possible segments, with starting date on the y-axis and terminal date on the y-axis. A length of a segment is positive, hence we can eliminate one half of the potential segments. There are further reductions that can be made that reduce the number of feasible segments to $T(T+1)/2 - \big(T(h-1) - mh(h-1) - (h-1)^2 - h(h-1)/2\big)$, where h is the minimal segment length. Those are covered in-depth in the paper by Bai and Perron [2].

An example of an upper triangular matrix of sums of squared residuals with $T = 9$, $m = 2$ and $h = 3$ can be seen on Table 4.1 and uses following notation:

- f denotes a feasible segment, for which we would compute the sum of squared residuals
- nf_1 means that the segment is not feasible since all segments must be at least have 3 observations,
- nf_2 means that the segment is not feasible since it would not leave enough place for another segment of length 3
- nf_3 means that the segment is not feasible, since its establishment would inadvertently create an illegal segment (with length less than 3) before it

A key observation is that only a small fraction of possible segments is feasible when the value of h or m is set high relative to T.

4.2 Recursive Residuals

In order to calculate the sum of squared residuals for the upper-triangular matrix, me must use the recursive residuals of a linear regression model that were first described by Brown et al. [18]. We calculate the recursive residuals for each of $m+1$ segments individually, hence we can assume a simple linear regression model:

$$y_i = x_i^\top \beta + u_i$$

or in marix form:

$$Y = X\beta + U$$

Let $\hat{\beta}^{(i)}$ be the OLS-estimate of the regression coefficient vector based on all observations up to i, and $X^{(i)}$ be the subset of the regressor X with with rows from q to i. Then the recursive residuals can be calculated using a recursive procedure:

$$\bar{u}_q(i) = \frac{y_i - x_i^\top \hat{\beta}^{(i-1)}}{\sqrt{1 + x_i^\top \left(X^{(i-1)\top} X^{(i-1)}\right)^{-1} x_i}} \quad (i = q+1, \ldots, n)$$

We initialize the recursion by taking $X^{(q-1)}$, y_q and x_q in order to use OLS-based regression to estimate β^q. The coefficient vector can then be used to compute the next iteration. This procedure continues until we reach iteration i. Since we are calculating the X and β for each step of calculation of $\bar{u}_i$, we can, without adding any significant cost, calculate $\bar{u}_j \quad$ for $j \in (q...i)$. This will become useful for the calculation of the triangular matrix of sums of squared residuals. Let $\hat{u_i}(j)$ return the sequence of $\bar{u}$:

$$\hat{u}_i(j) = (\bar{u}_i, \bar{u}_{i+1}, ..., \bar{u}_j)$$

4.3 Calculation of the Triangular Matrix of Sums of Squared Residuals

Before we apply the dynamic programming approach to find the optimal partition, we must construct a table of squared residuals for all possible segments. Since we are considering a pure structural change model, there are no constraints between the segments and we can therefore apply OLS one by one. Let $u_i(j)$ be the recursive residual at time j that we received from a sample starting at date i. Let $\mathrm{SSR}(i, j)$ be the sum of squared residuals, acquired from application of OLS to segment that starts at i and ends at j. Please note the difference between the squared residuals, described in Section 4.1 and recursive residuals, described in Section 4.2. It was shown by Brown et al. [18], that we can calculate the values in the table using a recursive relation:

$$\mathrm{SSR}(i, j) = \mathrm{SSR}(i, j-1) + \bar{u}_i(j)^2$$

We can utilize the recursive quality of the residual calculation and calculate one row from the upper triangular matrix of squared residuals at a time:

$$\mathrm{SSR}(i..j) = \mathrm{cumsum}(\hat{u}_i(j)^2)$$

4.4 The dynamic programming algorithm

After the table of squared residuals is constructed, we can apply the dynamic programming algorithm in order to find the optimal partition that solves the following recursive problem:

$$\mathrm{SSR}\left(\{T_{m,T}\}\right) = \min_{mh \le j \le T-h} \left[\mathrm{SSR}\left(\{T_{m-1,j}\}\right) + \mathrm{SSR}(j+1, T)\right]$$

where m is the number of breakpoints, T is the number of observations, h is the minimal segment length (distance between two breakpoints) and $\mathrm{SSR}\left(\{T_{r,n}\}\right)$ be the sum of squared residuals associated with the optimal partition that contains r breaks, using the first n observations.

The idea behind this algorithm is conceptually similar to the bottom-up solution to the rod cutting problem [7]. We can unroll the recursive calls in order to gain more insight into the workings of this method and get a following expression:

$$\begin{gathered} SSR\left(\{T_{m,T}\}\right) = \\ \min_{mh \le j_1 \le T-h} \left[SSR\left(j_1+1, T\right) + \right. \\ \min_{(m-1)h \le j_2 \le j_1-h} \left[SSR\left(j_2+1, j_1\right) + \right. \\ \vdots \\ \left.\left.\min_{h \le j_m \le j_{m-1}-h} \left[\mathrm{SSR}\left(1, j_m\right) + \mathrm{SSR}\left(j_m+1, j_{m-1}\right)\right] \ldots\right]\right] \end{gathered}$$

In order to find the solution to the minimization problem in an efficient manner, we will utilize the bottom-up approach, which is a common technique in dynamic programming.

We start the procedure by solving the minimization problem from the last line of the expression. We evaluate the optimal one-break partition for all sub-samples that allow a possible break ranging from observations h to $T-mh$ (since we still have to fit m segments of minimal size h after the first breakpoint). We then store $T-(m+1)h+1$ optimal breakpoints along with the corresponding sums of squared residuals. These one-break partitions would have an endpoint between $2h$ and $T-(m-1)h$.

Then, we continue to 2-partitions that can have and endpoint between $3h$ and $T-(m-2)h$. For each of $T-(m+1)h+1$ endpoints, we select an 1-partition that we have calculated earlier that would minimize the sum of squared residuals. We then store the 2-partitions and corresponding sums of squared residuals.

We iterate until we have computed the $T-(m+1)h+1$ $(m-1)$-partitions with endpoints between $(m-1)h$ and $T-2h$. We finish by finding the optimal $(m-1)$ partition that gives the overall minimal sum of squared residuals, when the last segment is appended to it.

Consider this example. Let $T=20$, $h=4$ and $m=3$. The allowed placements and the possible endpoints of the 1-partition can bee seen on Figure 4.1 (in green) and Figure 4.2 (in orange) respectively.

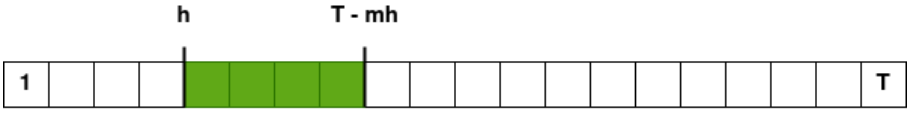


Figure 4.1: Allowed placements of j_m

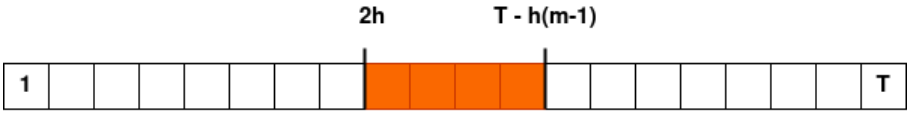


Figure 4.2: Possible endpoints of 1-partion

Figure 4.3: Possible start points of the last segment

We have obtained the optimal $(m-1)$-partitions with endpoints between $(m-1)h$ and $T-2h$. We find the optimal partition by calculating the sum of squared residuals for all possible segments that startpoints between $3h$ and $T-h$ (Figure 4.3 in blue) and finding one that yields the minimal sum of squared residuals.

This approach scales very efficiently with the number of breakpoints, and the time it takes to find the optimal m-partition is negligible, compared to the time it takes to compute the upper-triangular matrix of squared residuals.

4.5 Bayesian Information Criterion

The dynamic programming algorithm from the previous section is only applicable when we know the number of breakpoints. However, in most cases, this information is not available to us. Hence,

in order to determine the optimal number of breakpoints, we must utilize one of the existing criterions for model selection. Bai and Perron [2] discuss the strengths and weaknesses of multiple approaches, one of which being the Bayesian Information Criterion (BIC) [19]. Bai and Perron state that BIC works well when breaks are present in time series and poorly when it is not the case. However, this would not constitute a issue for the BFAST algorithm, because the breakpoint by Bai and Perron is only applied if the OLS-MOSUM test signifies that there are structural change present in the time series. Therefore, BIC constitutes a viable criterion for selecting the optimal number of breakpoints m_{best} in the context of this project.

For each possible number of breakpoints, up to some limit $m_{\max}$, we find the sum of squared residuals for all breakpoints and then calculate the Bayesian Information Criterion:

$$\text{BIC} = k \ln n - 2\hat{L}$$

where:

- k: the number of degrees of freedom for our model, in our case $k = (q+1)(m+1)$, where q is the dimensionality of the vector x and m is the number of breakpoints.
- n is the number of observations
- $\hat{L}$ is the maximized value of the log-likelihood of the model. In our case, the value is given by [22]:

  $$\hat{L} = -\frac{n}{2}\left(\ln\left(S_T(\hat{T}_1, .., \hat{T}_m)\right) + \ln\frac{2\pi}{n} + 1\right)$$

  where $S_T(\hat{T}_1, .., \hat{T}_m)$ is the sum of squared residuals for the m-partition of the dataset.

The optimal number of breakpoints would be the number of breakpoints for which the value of BIC is lowest.

4.6 Steps of the Algorithm

The algorithm by Bai and Perron operates in following steps:

1. **Calculate the upper-triangular matrix of sums of squared residuals**
   We calculate the matrix, by following the guidelines, described in Section 4.3.
2. **Find the largest number of allowed breakpoints**
   Given a value of $0 \leq h' \leq 1$, which signifies the smallest allowed size of a segment, and m_{user} which is the maximum number of breakpoints, determined by the user, we can then find the largest number of allowed breakpoints $m_{\max} = \min(\lfloor \frac{1}{h'} \rfloor + 1, m_{\text{user}})$.
3. **Create a table `BIC` of size $m_{\max} + 1$**
4. **For $m \in (m_{\max}, ...0)$, do following:**
   1) Find the optimal position of the m breakpoints, using algorithm from Section 4.4.
   2) Calculate the sum of squared residuals, and then use it to calculate the Bayesian Information Criterion BIC_m of this partition by following the guidelines from Section 4.5.
   3) Add BIC_m to the table
5. **Find the value of m_{best} that corresponds to the minimal value in the `BIC`**
6. **Use the algorithm from Section 4.4 to find the m_{best} breakpoints**

4.7 Parameters and Their Values

The algorithm takes two parameters:

- $0 \leq h' \leq 1$, which can be used to calculate the minimal allowed size of the segment $h = h'n$. The default value is $h' = 0.15$.
- `max_breaks`, which can be used to calculate the maximum number of allowed segments m_{user}. This parameter is optional, and in that case, m_{user} would be calculated from the value of h'.

Chapter 5

BFAST0n

BFAST0N is the light-weight variation on the BFAST approach (Chapter 6). The idea behind this method is to apply the breakpoint estimation algorithm by Bai and Perron [2], which is described in Chapter 4, to a pre-processed dataset,

5.1 Dataset Pre-processing

Let Y_t be the observed data for t in $(1, \ldots, n)$.

5.1.1 STL

Before any other pre-processing takes place, STL time series decomposition algorithm is used for trend- and/or season-adjustment. For more information about the STL algorithm, please refer to Chapter 2.

5.1.2 Harmonic model

The second step of dataset pre-processing is the application of the harmonic model, that was described in paper by Verbesselt et. al from 2010. [21].

$$S_t = \sum_{k=1}^{K} \left[\gamma_{j,k} \sin\left(\frac{2\pi kt}{f}\right) + \theta_{j,k} \cos\left(\frac{2\pi kt}{f}\right) \right]$$

where

- K is the harmonic term, i.e. how many of pairs of sin + cos terms we use,
- $\gamma_{j,k}$ and $\theta_{j,k}$ are the seasonal coefficients to be estimated by linear regression in the breakpoint estimation algorithm.
- t is the observation time.
- f frequency of the dataset

In order to apply the model to the time-series s.t. it could be used by the breakpoint estimation algorithm, we need to compute the matrix form of the harmonic model (for $K = 3$):

$$X_h = \begin{bmatrix} \sin {}^{2\pi}/_{f} & \cos {}^{2\pi}/_{f} & \sin {}^{4\pi}/_{f} & \cos {}^{4\pi}/_{f} & \sin {}^{6\pi}/_{f} & \cos {}^{6\pi}/_{f} \\ \sin {}^{4\pi}/_{f} & \cos {}^{4\pi}/_{f} & \sin {}^{8\pi}/_{f} & \cos {}^{8\pi}/_{f} & \sin {}^{12\pi}/_{f} & \cos {}^{12\pi}/_{f} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sin {}^{2\pi n}/_{f} & \cos {}^{2\pi n}/_{f} & \sin {}^{4\pi n}/_{f} & \cos {}^{4\pi n}/_{f} & \sin {}^{6\pi n}/_{f} & \cos {}^{6\pi n}/_{f} \end{bmatrix}$$

X_h has n rows and $2K$ columns.

5.2 Steps of the Algorithm

BFAST0n consists of following steps:

1. **Apply STL to the observation matrix** Y in order to get the adjusted time series $\tilde{Y}$

2. **Apply the harmonic model to $\tilde{Y}$**
3. **Use the breakpoint estimation algorithm** (Chapter 4) to $x = X_h$ and $y = \tilde{Y}$
4. **Return the number and placement of the breakpoints**

5.3 Parameters and Their Values

BFAST0n has following parameters:

- f. Frequency of the time series. Specific for each dataset.
- `stl`. Which type of STL-based pre-processing to use. Allowed values:
  - "none" - the default value. No STL adjustment would be performed
  - "trend" - Trend component is removed from the input time series
  - "seasonal" - Seasonal component is removed from the input time series
  - "both" - Seasonal component is removed from the input time series
- $0 \leq K \leq 3$. The integer value of the harmonic term, if no value provided, a value of 3 is used.

Chapter 6

BFAST

In paper from 2010, Verbesselt et. al [20] outline a generic change detection approach that combines iterative decomposition into trend, seasonal and remainder components and detection and characterizing of breakpoints within a time series.

6.1 The model

For BFAST, we consider a following data model:

$$Y_t = T_t + S_t + e_t, \quad t = 1, ..., n$$

where:

- $Y_t \in \mathbb{R}$ is the observation at time t
- $T_t \in \mathbb{R}$ is the trend component at time t
- $S_t \in \mathbb{R}$ is the seasonal component at time t
- $e_t \in \mathbb{R}$ is the remainder component at time t
- n is the number of observations in time series

For more information about the time series components, please refer to Chapter 2.

6.1.1 The Trend Component

We assume that T_t is piecewise linear and has breakpoints $t_1^*, \ldots, t_m^*$, where m is the number of breakpoints in the trend component and set $t_0^* = 0$. Then, the trend component can be described as

$$T_t = \alpha_j + \beta_j t \quad \text{for} \quad t_{j-1}^* < t \leq t_j^*$$

where j is the number of the next breakpoint, i.e. $j = 1, ..., m$. One way, in which we can characterize the abrupt changes in the trend component is by calculating the magnitude of the change:

$$\text{Magnitude}(j) = T_{j-1} - T_j = \alpha_{j-1} + \beta_{j-1}t - (\alpha_j + \beta_j t) = (\alpha_{j-1} - \alpha_j) + (\beta_{j-1} - \beta_j)t$$

where $j = 1, ..., m$.

6.1.2 The Seasonal Component - Dummy Model

The breakpoints in the seasonal component can occur at different times than the breaks in the trend component. Let $t_1^\#, \ldots, t_p^\#$ be the breakpoints in the seasonal component, where p is the number of breakpoints and $t_0^\# = 0$.

For $t_{j-1}^\# < t \leq t_j^\#$, the seasonal term can be expressed as:

$$S_t = \sum_{i=1}^{s-1} \gamma_{i,j}(d_{t,i} - d_{t,0})$$

where

- s is the period of seasonality
- $\gamma_{i,j}$ is the influence of season i that we are going to estimate with linear regression.
- $d_{t,i}$ is the dummy variable [12], for which it holds that $d_{t,i} = 1$ when t is in season i and 0 otherwise. Then for season 0, we have that $d_{t,i} - d_{t,0} = -1$ and $d_{t,i} - d_{t,0} = 1$ otherwise.

An important observation is that the sum of S_t across $s-1$ successive elements is zero. The model is build this way in order to avoid breakpoints in trend being caused by seasonal breaks.

Breakpoint Detection and Estimation

When detecting breakpoints in the seasonal component, we apply the breakpoint detection algorithm to the model (in matrix form):

$$X_d = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -1 & -1 & -1 & \dots & -1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -1 & -1 & -1 & \dots & -1 \\ 1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

where X_{d} has $s-1$ columns and n rows.

We then use the same model for the breakpoint estimation algorithm by Bai and Perron. The estimated breakpoints $t_1^{\#}, \dots, t_p^{\#}$ can then be used to apply the linear regression to estimate the vector of seasonal coefficients γ ($\gamma_{i,j}$ for $j = 1, \dots, p$ and $i = 1, \dots s-1$).

$$Y - T = X_{part}\gamma$$

where Y is the vector of observations, T is the trend component and X_{part} is matrix X_d that was partitioned according to the estimated breakpoints $t_1^{\#}, \dots, t_p^{\#}$. Consider an example with a single breakpoint, where $n = 12$, $s = 4$ and $t_1^{\#} = 5$. Then:

$$X_{part} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 & -1 \end{bmatrix}$$

Linear regression is applied to estimate the values: $\hat{\gamma}_{1,1}, \hat{\gamma}_{1,2}, \hat{\gamma}_{1,3}, \hat{\gamma}_{2,1}, \hat{\gamma}_{2,2}$ and $\hat{\gamma}_{2,3}$.

The seasonal component can then be estimated: $\hat{S}_t = \sum_{i=1}^{s-1} \hat{\gamma}_{i,j}(d_{t,i} - d_{t,0})$, or in matrix form $\hat{S} = X_{\text{part}}\gamma$

6.1.3 The Seasonal Component - Harmonic Model

Alternatively, we can use the Harmonic model that is also used by the BFAST0n algorithm, for $K = 3$. For more information, please refer to Chapter 5. We have:

$$X_h = \begin{bmatrix} \sin {}^{2\pi}/_{f} & \cos {}^{2\pi}/_{f} & \sin {}^{4\pi}/_{f} & \cos {}^{4\pi}/_{f} & \sin {}^{6\pi}/_{f} & \cos {}^{6\pi}/_{f} \\ \sin {}^{4\pi}/_{f} & \cos {}^{4\pi}/_{f} & \sin {}^{8\pi}/_{f} & \cos {}^{8\pi}/_{f} & \sin {}^{12\pi}/_{f} & \cos {}^{12\pi}/_{f} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sin {}^{2\pi n}/_{f} & \cos {}^{2\pi n}/_{f} & \sin {}^{4\pi n}/_{f} & \cos {}^{4\pi n}/_{f} & \sin {}^{6\pi n}/_{f} & \cos {}^{6\pi n}/_{f} \end{bmatrix}$$

For the application of linear regression X_h would be partitioned the same way as X_d (dummy model), resulting in matrix X_{part} with n rows and $6p$ columns, where p is the number of breakpoints in the seasonal component. The estimate for the seasonal component can be rebuilt using:

$$\hat{S} = X_{\text{part}}\omega$$

where ω is a $(1 \times 6p)$ vector $(\gamma_{1,1},\ \theta_{1,1},\ \gamma_{1,2},\ \theta_{1,2},\ \ldots,\ \gamma_{p,3},\ \theta_{p,3})^\top$

6.2 Steps of the Algorithm

BFAST operates in a following way:

1. **Estimate $\hat{S}_t$ using STL** and finding the average of all seasons, by setting $n_s =$ "harmonic". For more information about STL parameters, please refer to Chapter 2.
2. Then iterate:
   1) **Calculate the deasonalized time series:** $V_t = Y_t - \hat{S}_t$
   2) **Apply the OLS-MOSUM test** (Chapter 3) to $x =$ ti and $y = V_t$. Where ti denotes the values of the time interval of Y_t. If the returned p-value is lower than the significance level α, **estimate the number and position of the trend components** using the breakpoint estimation algorithm by Bai and Perron (Chapter 4) from ti and V_t.
   3) **Compute the trend coefficients** α_j and β_j for $j = 1, \ldots, m$ using linear regression. Set the trend estimate $\hat{T}_t = \hat{\alpha}_j + \hat{\beta}_j t$ for $t = t^*_{j-1} + 1, \ldots, t^*_j$.
   4) **Calculate the detrended time series**: $W_t = Y_t - \hat{T}_t$
   5) If the OLS-MOSUM test, applied to $y = W_t$ and the chosen seasonal model signifies that the breakpoints are present in the seasonal data, **estimate the number and position of the breakpoints in the trend component** using the breakpoint estimation algorithm, as described in Section 6.1.2 and 6.1.3.
   6) **Compute the coefficients for the seasonal component and reconstruct** $\hat{S}_t$ from the chosen seasonal model and seasonal coefficient, as described in Section 6.1.2 and 6.1.3.

   The iteration is stopped when the number and position of the breakpoints do not change during the iteration, or when we reach the maximum allowed number of iterations.
3. **Return** $\hat{T}_t$, $\hat{S}_t$ the number and position of trend and seasonal breakpoints and the maximum breakpoint magnitude.

6.3 Parameters and Their Values

BFAST algorithm has following paramers

- s. Period of seasonality. Specific for each dataset.
- $0 < h < 1$. Minimal segment length for the Breakpoint estimation and bandwidth parameter for the OLS-MOSUM test. For more information please refer to Chapters 3 and 4 respectively. The default value is 0.15.
- `seasonal`. Type of model to be fitted. Should be either "harmonic", "dummy" or "none". The later means that no seasonal model would be fitted, i.e. $S_t = 0$.
- `max_iter`. Maximal number of iterations of the main loop. The default value is 10.
- `max_breaks`. Upper limit on the number of breakpoints that could be computed by the breakpoint estimation algorithm. The default value is "None", i.e. no upper limit. For more information, please refer to Chapter 4.
- α. The significance level for the OLS-MOSUM test. If the returned p value is below that threshold - we consider that there is structural change and proceed with the linear regression. Higher values of α make BFAST more sensitive to structural change. For more information about the confidence intervals, please refer to the Chapter 4. The default value is 0.05.

Chapter 7

Implementation

Both BFAST0n and BFAST algorithms were implemented in Python 3.8.3 using `numpy` [15],`statsmodels` [16] and `pandas`[13] for all the computation and `matplotlib` for all the plotting. The implementation is based on the original R implementation of BFAST and BFAST0n by Verbesselt et al. [17] and the `strucchange` library by Zeileis [23].

7.1 Linear Regression

Linear regression is the cornerstone of BFAST and all its components such as OLS-MOSUM test and the breakpoint estimation algorithm. Therefore it is essential to choose a suitable implementation for this operation. There exist a plethora of linear regression implementations for Python, but I have decided to use the `statsmodels.regression.linear_model.OLS` function from the `statsmodels` library [16]. It was chosen for its stability, flexibility and detailed documentation.

7.2 STL

For the STL decomposition algorithm, I have chosen the `tsa.seasonal.STL` function from the `statsmodels` library. Source file `stl.py` includes a wrapper around the `STL` method that also replaces the missing values (NaNs) with interpolated values. It also includes a function `seasonal_average` that emulates the functionality of setting the n_s parameter to "periodic" by taking the seasonal average of the seasonal component of the decomposition, since `statsmodels` implementation does not have a support for this option. For more information about the STL parameters, please refer to Chapter 2.

7.3 OLS-MOSUM Test

OLS-MOSUM test is implemented in source file `efp.py`. Instance of a class `EFP()` is created, given matrix X, vector y and value of the bandwidth parameter h and calculates the empirical fluctuation process of type OLS-MOSUM. Then class method `sctest` can be called in order to calculate the value of the statistic and the p value. P value is calculated using linear interpolation from the table of critical values in the `utils.py` by following the algorithms steps, described in Chapter 3.

7.4 Breakpoint Estimation

The breakpoint estimation algorithm by Bai and Perron (Chapter 4) is the most complicated step of the BFAST algorithm and its implementation is structured in a following manner:

- Breakpoints are caclulated by creating an instance of a class `Breakpoints`, given matrix X, vector y, minimal segment length h and maximum number of breaks `breaks`.
- The upper-triangular matrix is computed in the source file `ssr_triang`. In order to speed up this step, an option for multi-processing is enabled and can be activated by setting `use_mp = True`.

- Method `breakpoints_for_m` estimates the number of breakpoints using Bayesian Information Criterion and then their location by calling another method `summary` that calculates the table of BIC for $m \in 1..m_{max}$, where m_{max} is the maximum number of allowed breaks.

7.5 BFAST0n and BFAST

The implementation of BFAST0n and BFAST is very straight-forward and closely follows the algorithm steps, outlined in Chapter 6 and 5 respectively. An instance of class `BFAST` calculates the time series decomposition and the breakpoints, given matrix X, vector y, value of h, significance level α and maximum number of breaks `breaks`. The resulting decomposition and breaks can be accessed by calling obtaining the object field `output`.

Chapter 8

Validation

In order to validate the implementation of BFAST and BFAST0n, four datasets were chosen. Each of the datasets was chosen to test a specific property of the algorithm. Three of the datasets were taken from the paper by Verbesselt et al. [20] and one from the standard dataset library for the R programming language [8]. The results were compared to the same data applied to the original R-implementation of BFAST and BFAST0n [17]. Both mine and R implementation find exactly same breakpoints for the same input and parameter values. There are however some small differences in the decomposition due to different implementations of STL and linear regression. Chapter 7 elaborates on both of these points. This chapter only covers the validation of the BFAST algorithm, but the BFAST0n implementation was tested in a similar manner.

8.1 `nile`

Measurements of the annual flow of the river Nile at Aswan (formerly Assuan), 1871–1970. With apparent changepoint near 1898. From [8]. There is no seasonal component, since there is only a single observation each year. This tests the breakpoint estimating capacity without STL and seasonal model. Following parameter values were used:

- `season`= "none"
- $\alpha = 0.05$
- $h = 0.15$

The results can be seen on Figure 8.1.

8.2 `harvest`

16-day NDVI Time Series for a Pinus Radiata Plantation, with approximately 23 observation per year from [20]. There are 4 breakpoints in the trend component exclusively. The results can be seen on Figure 8.2 Following parameter values were used:

- `season`= "harmonic"
- $\alpha = 0.05$
- $h = 0.15$

8.3 `simts`

Simulated seasonal 16-day NDVI time series from [20]. The seasonal component has a single breakpoint that is difficult to detect. Hence, following parameter values were used:

- `season`= "harmonic"
- $\alpha = 0.35$
- $h = 0.3$
- `max_iter` = 2

The results can be seen on Figure 8.3.

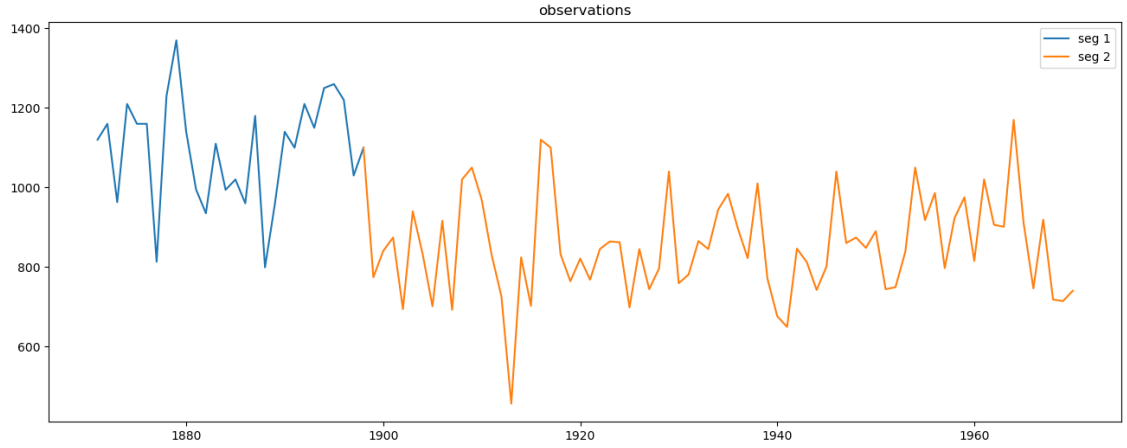


Figure 8.1: Result of applying BFAST to the `nile` dataset. `season` = "none" had been chosen, hence no decomposition was done

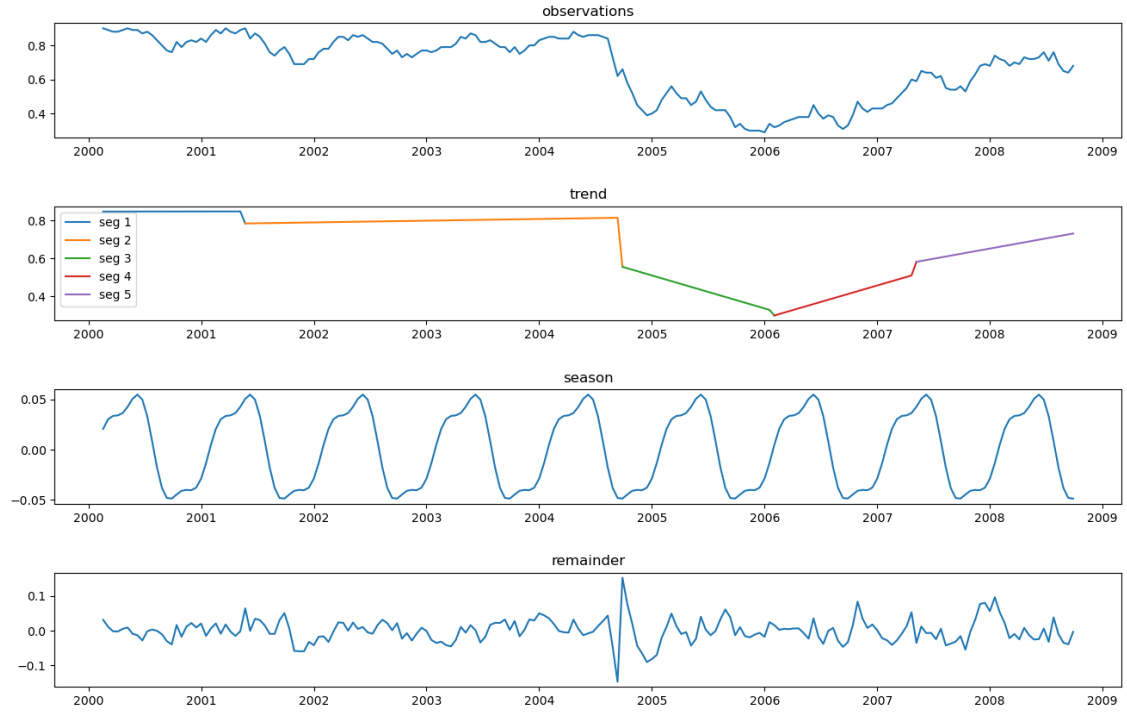


Figure 8.2: Result of applying BFAST to the `harvest` dataset

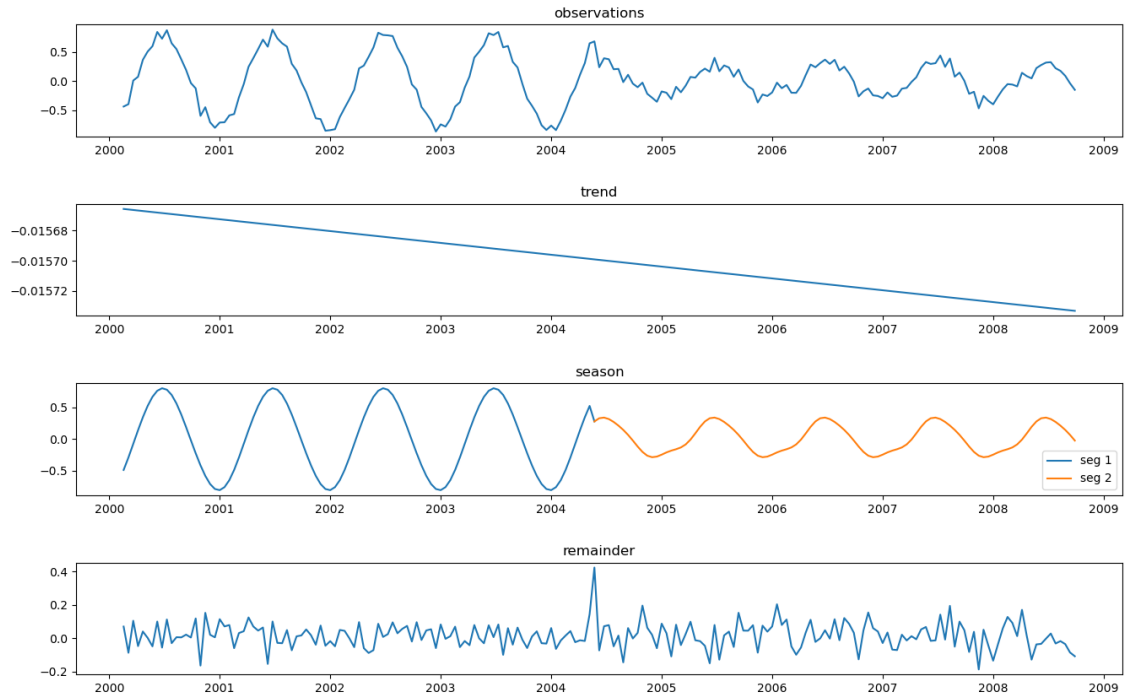


Figure 8.3: Result of applying BFAST to the `simts` dataset

8.4 `ndvi`

A random NDVI time series with missing values. BFAST should pass the missing values into the seasonal, trend and remainder components, and my implementation accomplishes that. Frequency is set to 24. There is a single breakpoint in the trend component. Source: [20]. Following parameter values were used:

- `season`= "dummy"
- $\alpha = 0.05$
- $h = 0.15$

The results can be seen on Figure 8.4.

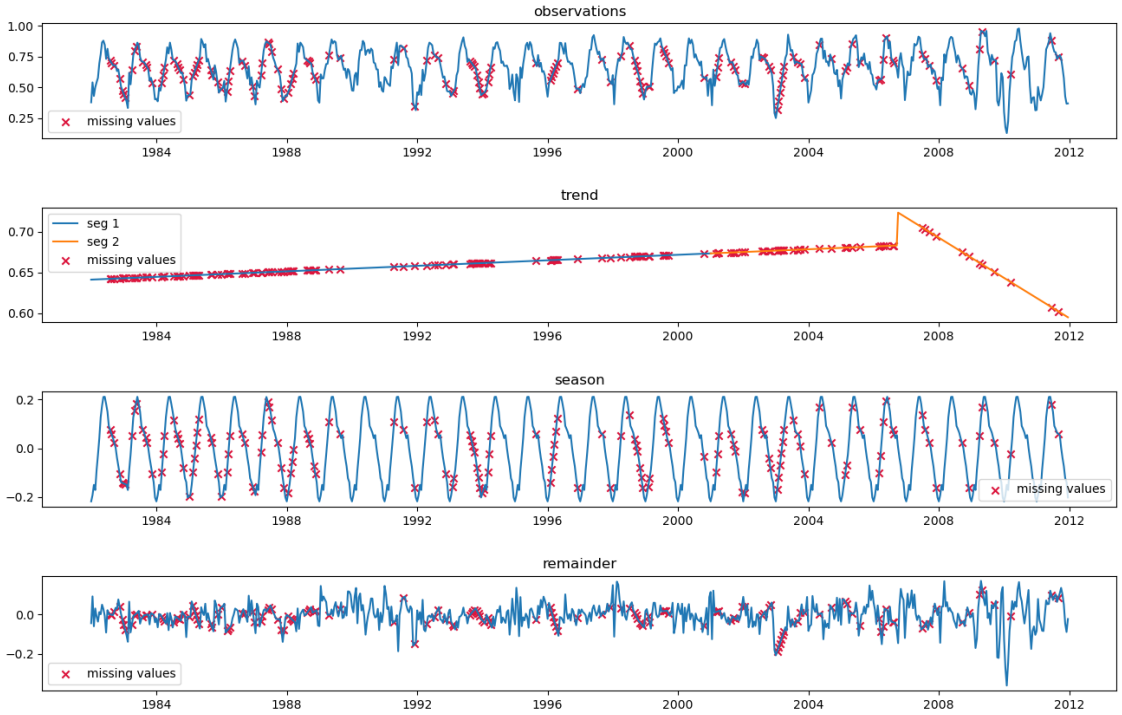


Figure 8.4: Result of applying BFAST to the `NDVI` dataset. Red crosses denote the missing values

Chapter 9

Conclusion and Future Work

I have completed a working prototype implementation of BFAST and BFAST0n structural change detection algorithms by Verbesselt et al. [20] in Python. My implementation was validated using the datasets from the original paper and results were compared to the original R implementation [17].

This report chronicles my implementation of the BFAST and BFAST0n and covers the underlying theory of both algorithms. In particular, I describe in detail the concrete steps of each component of BFAST. This includes the STL decomposition algorithm [5], OLS-MOSUM test by Achim Zeileis [24], the breakpoint estimation algorithm by Bai and Perron [2] and BFAST itself. The description of the concrete algorithm steps can be helpful to a potential programmer that would like to implement one of those algorithms without diving too deep into the statistical theory.

However, some parts of the algorithm have been omitted and should be added in the future. This includes the computation of the confidence intervals for the breakpoints in the algorithm by Bai and Perron. The underlying theory was too overwhelming for a person without a background in either statistics or econometrics and therefore was given the lowest priority, since both BFAST and BFAST0n can function without it.

Moreover, the differences between the BFAST and BFAST Monitor [9] approach are not covered in this report either, since it would be a crucial part of my MSc thesis, and this project is focused around the theory of the BFAST and its components.

Since performance was not the aim of this particular project, this implementation is rather slow and is comparable to the original R implementation. It struggles with any time series with more than 500 observation (as demonstrated by the `NDVI` dataset. In addition, the Python implementation is about 30% slower than the original implementation, since the original uses C++ for some of its most resource-demanding components.

The result of this project would form a solid foundation for the future work, where a parallelized, GPU-based implementation would be developed and could potentially serve as an alternative to the parallelized implementation of BFAST-Monitor.

References

[1] `bfast-py`, python implementation of the bfast and bfast0n structural change detection algorithms. `https://github.com/mortvest/bfast-py`. [Accessed 23-August-2020].

[2] Jushan Bai and Pierre Perron. Computation and analysis of multiple structural change models. *Journal of Applied Econometrics*, 18(1):1–22, 2003.

[3] Chia-Shang James Chu, Kurt Hornik, and Chung-Ming Kuan. Mosum tests for parameter constancy. *Biometrika*, (82):603–617, 1995.

[4] Chia-Shang James Chu, Kurt Hornik, and Chung-Ming Kuan. The moving-estimates test for parameter stability. *Econometric Theory*, 11(4):708, 1995.

[5] R. B. Cleveland, W. S. Cleaveland, J. E. McRae, and I. Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. In *Journal of Official Statistics*, volume 6, pages 3–33, 1990.

[6] William S. Cleveland and Susan J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cifford Stein. *Introduction to Algorithms*, pages 360–370. MIT Press, third edition, 2009.

[8] The R Foundation. R: The r datasets package. `https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html`. [Accessed 23-Augugst-2020].

[9] F. Gieseke, S. Rosca, T. Henriksen, J. Verbesselt, and C. E. Oancea. Massively-parallel change detection for satellite time series data with missing values. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 385–396, 2020.

[10] Ryan Hafen. `stlplus` - seasonal decomposition of time series by loess - v0.5.1 documentation. `https://www.rdocumentation.org/packages/stlplus/versions/0.5.1/topics/stlplus`. [Accessed 23-August-2020].

[11] Chung-Ming Kuan and Kurt Hornik. The generalized fluctuation test: A unifying view. *Econometric Reviews*, 14(2):135–161, 1995.

[12] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE*, 13(3):1–26, 03 2018.

[13] Wes McKinney and pandas comminity. `pandas` - python data analysis library. `https://pandas.pydata.org/`. [Accessed 23-Augugst-2020].

[14] United Nations. Climate change, united nations. `https://www.un.org/en/sections/issues-depth/climate-change/`, 2020. [Accessed 23-August-2020].

[15] numpy community. `numpy`. `https://numpy.org/`. [Accessed 23-Augugst-2020].

[16] Josef Perktold, Skipper Seabold, Jonathan Taylor, and statmodels developers. `statsmodels`: statistical models, hypothesis tests, and data exploration. `https://statsmodels.org`. [Accessed 23-Augugst-2020].

[17] BFAST Project. `bfast2/bfast` breaks for additive season and trend source code. `https://github.com/bfast2/bfast/blob/master/R/bfast.R`. [Accessed 23-August-2020].

[18] Brown R.L., Durbin J., and Evans J.M. Techniques for testing constancy of regression relationships over time. *Journal of the Royal Statistal Society*, 37:149–163, 1975.

[19] Schwarz and Gideon. Estimating the dimension of a model. *Ann. Statist.*, 6(2):461–464, 03 1978.

[20] J. Verbesselt, R. Hyndman, G. Newnham, and D. Culvenor. Detecting trend and seasonal changes in satellite image time series. In *Remote Sensing of Environment*, volume 114, page 106–115, 2010.

[21] Jan Verbesselt, Rob Hyndman, Achim Zeileis, and Darius Culvenor. Phenological change detection while accounting for abrupt and gradual trends in satellite image time series. *Remote Sensing of Environment*, 114(12):2970 – 2980, 2010.

[22] Yi-Ching Yao. Estimating the number of change-points via schwarz' criterion. *Statistics and Probability Letters*, 6(3):181–189, 1988.

[23] A. Zeileis, F. Leisch, K. Hornik, and C. Kleiber. strucchange: Testing, monitoring, and dating structural changes, source code. `https://cran.r-project.org/web/packages/strucchange/index.html`. [Accessed 23-August-2020].

[24] A. Zeileis, F. Leisch, K. Hornik, and C. Kleiber. strucchange: An r package for testing for structural change in linear regression models. In *Journal of Statistical Software*, volume 7, 2002.

Appendix A

Practical Information

Source Files

The source code for the implementation, described in this report, can be found on GitHub [1] and is structured in a following manner:

- `bfast0n.py`
  Implementation of the BFAST0N algorithm, described in Chapter 5
- `bfast.py`
  Implementation of the BFAST algorithm, described in Chapter 6
- `breakpoints.py`
  Implementation of the breakpoint estimation algorithm, described in Chapter 4
- `converter.py`
  Script that converts the datasets for the R programming language (`.rda`) into numpy data files (`.npy`). It requires a working R installation.
- `datasets.py`
  Collection of datasets for testing. They are taken from the R programming language standard library [8] and the `strucchange` package for the R-language [23].
- `loess.py`
  Implementation of the LOESS algorithm (Chapter [**?**]). Used only for demonstration.
- `efp.py`
  Implementation of the empirical fluctuation processes, in particular the OLS-MOSUM test that is described in Chapter 3.
- `plots.py`
  Script that produces all the plots from the validation chapter (Chapter 8).
- `recresid.py`
  Implementation of the recursive residuals for the linear regression relationships, as described in Chapter 4.
- `ssr_triang.py`
  Calculation of the upper-triangular matrix of sums of squared residuals for the breakpoint estimation algorithm as described in Chapter 4.
- `stl.py`
  A wrapper for the `statsmodels` [16] implementation of the STL decomposition as described in Chapter 2.
- `utils.py`
  Helper functions and lookup tables. Used by other modules.

Dependencies

The implementation was tested using Python 3.8.3. In order to install all the necessary libraries, start the virtual environment and import the requirements:

```
pip install -f requirements.txt
```

How to Run the Tests

Tests for each file in the *src* directory are contained withing that source file. In order to run test use following:

```
python file.py
```

In order to get a more verbose output, run:

```
python file.py --log=INFO
```

In order to see the debug information, run:

```
python file.py --log=DEBUG
```

In order to reproduce the plots, run:

```
python plots.py
```