Fast RobustSTL: Efficient and Robust Seasonal-Trend Decomposition for Time Series with Complex Patterns

Conference Paper · August 2020

DOI: 10.1145/3394486.3403271

Qingsong Wen
Alibaba Group (U.S.) Inc.
60 PUBLICATIONS **627** CITATIONS
SEE PROFILE

Fast RobustSTL: Efficient and Robust Seasonal-Trend Decomposition for Time Series with Complex Patterns

Qingsong Wen
DAMO Academy, Alibaba Group
Bellevue, WA, USA
qingsong.wen@alibaba-inc.com

Zhe Zhang
Georgia Institute of Technology
Atlanta, GA, USA
zzhang724@gatech.edu

Yan Li, Liang Sun
DAMO Academy, Alibaba Group
Bellevue, WA, USA
{yl.yy6993,liang.sun}@alibaba-inc.com

ABSTRACT

Many real-world time series data exhibit complex patterns with trend, seasonality, outlier and noise. Robustly and accurately decomposing these components would greatly facilitate time series tasks including anomaly detection, forecasting and classification. RobustSTL is an effective seasonal-trend decomposition for time series data with complicated patterns. However, it cannot handle multiple seasonal components properly. Also it suffers from its high computational complexity, which limits its usage in practice. In this paper, we extend RobustSTL to handle multiple seasonality. To speed up the computation, we propose a special generalized ADMM algorithm to perform the decomposition efficiently. We rigorously prove that the proposed algorithm converges approximately as standard ADMM while reducing the complexity from $O(N^2)$ to $O(N \log N)$ for each iteration. We empirically study our proposed algorithm with other state-of-the-art seasonal-trend decomposition methods, including MSTL, STR, TBATS, on both synthetic and real-world datasets with single and multiple seasonality. The experimental results demonstrate the superior performance of our decomposition algorithm in terms of both effectiveness and efficiency.

CCS CONCEPTS

• **Mathematics of computing** → Time series analysis; • **Information system** → Data mining.

KEYWORDS

time series; seasonal-trend decomposition; multiple seasonality; generalized ADMM

ACM Reference Format:
Qingsong Wen, Zhe Zhang, and Yan Li, Liang Sun. 2020. Fast RobustSTL: Efficient and Robust Seasonal-Trend Decomposition for Time Series with Complex Patterns. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403271

1 INTRODUCTION

One of the fundamental problems in time series analysis is how to deal with the seasonality as it is commonly observed in many real-world data [2, 13, 22]. Seasonality generally refers to the repeated pattern which affects the data. However, many time series data from the real-world applications are affected by several repeated activities or schedules, which introduce multiple seasonality. For example, the gross merchandise value (GMV) of an online retailer depends on both the time of day and the day of week. In other words, the time series of GMV has two seasonal periods – one day and one week. Multiple seasonal components are usually interlacing together, making the seasonality adjustment more challenging for time series with multiple seasonality.

The seasonal-trend decomposition can deconstruct a time series into several components including trend, seasonality, and remainder. It plays an important role in many time series applications, including forecasting and anomaly detection [10, 14, 17, 26]. It can reveal the insights of time series from different perspectives and facilitate further analysis of the time series data. Among many seasonal-trend decomposition methods, the most classical one is the STL (Seasonal-Trend decomposition using Loess) [5]. Recently, a decomposition algorithm called RobustSTL (Robust Seasonal-Trend decomposition for Long time series) [24] has been proposed. Due to its robustness to outliers, noises, and abrupt trend changes, RobustSTL gains popularity especially in anomaly detection and forecasting scenarios where noises and outliers are abundant. Unfortunately, it cannot process time series which exhibit multiple seasonality properly. Also its high computational complexity makes it not ideal for many real-world online applications where fast computation is required.

In this paper, we first extend RobustSTL to handle multiple seasonality. Specifically, we extract the trend by a robust sparse model. After the trend extraction, we extract the sum of multiple seasonal components by applying the non-local seasonal filter. And then we recover each seasonal component through multiple designed ℓ_1-norm based regularizations. In order to speed up the computation, we further propose an algorithm based on the generalized alternating direction method of multipliers (GADMM) [7] for efficient decomposition. Note that the most computationally expensive part in our decomposition is the trend extraction, which solves a minimization problem containing three ℓ_1-norms. The minimization is costly due to the high dimensionality of the problem. Popular algorithms like primal-dual hybrid gradient method (PDHG) [4] requires a large number of iterations to converge. The standard ADMM algorithm [3] converges with less iterations, but each iteration is relatively slow since it involves solving a large linear system. By utilizing the partial circulant structure of the linear system,

Table 1: Comparison of different seasonal-trend decomposition algorithms

Algorithm	Support Single Seasonality	Support Multiple Seasonality	Capture Abrupt Trend Change	Robust to Outliers
STL	Yes	No	No	No
RobustSTL	Yes	No	Yes	Yes
MSTL	Yes	Yes	No	No
STR	Yes	Yes	No	Yes
TBATS	Yes	Yes	Yes	Yes
Proposed Fast RobustSTL	Yes	Yes	Yes	Yes

we expand it to a standard circulant linear system and apply fast Fourier transform to solve it efficiently. The proposed generalized ADMM can be applied for both trend extraction and multiple seasonal component extraction. We rigorously prove that the proposed GADMM algorithm converges approximately as standard ADMM while reducing the complexity from $O(N^2)$ to $O(N \log N)$ for each iteration. Our empirical studies on both synthetic and real-world datasets demonstrate that our proposed Fast RobustSTL decomposition performs significantly better than other state-of-the-art seasonal-trend decomposition methods.

The rest of this paper is organized as follows. In Section 2 we briefly introduce the related work. In Section 3 we present our seasonal-trend decomposition method to handle multiple seasonality. We introduce our special generalized ADMM algorithm for both trend extraction and multiple seasonal component extraction, and its convergence analysis in Section 4. We summarize our empirical studies in Section 5, and conclude in Section 6.

2 RELATED WORK

Seasonal-trend decomposition is an effective method to adjust seasonality and widely applied in time series analysis [5, 6, 9, 14, 17, 19, 27]. The most classical and widely used decomposition method is the STL (Seasonal-Trend decomposition using Loess) [5] decomposition. In STL, the trend and seasonal components are estimated by using an alternating algorithm where the trend component is computed using LOESS (LOcal regrESSion) smoothing. It has been widely used in many applications while still suffers from the inability to handle abrupt trend changes and less flexibility when data has long period and high noise. To handle multiple seasonality, an extension of STL called MSTL [15] has been proposed. Its framework is similar to STL, and the difference lies in how to estimate multiple seasonal components, which are also estimated iteratively by fixing trend and rest seasonal components in MSTL. However, STL and MSTL fail to extract the seasonal component accurately when abrupt trend change and outliers exist.

Other decomposition algorithms like STR and TBATS are proposed recently. STR (Seasonal-Trend decomposition based on Regression) [9] is proposed to handle multiple seasonality and seasonal shifts by jointly extracting trend, seasonal and remainder components without iterations. In STR, a two-dimensional representation of the seasonal component is utilized to learn the slowly changing seasonal component. A robust version called RSTR is proposed to handle non-Gaussian data by utilizing the ℓ_1-norm regularization. Nevertheless, it cannot follow abrupt changes in trend and shows high computational cost. Also it cannot scale to time series with long seasonality period due to the cost to learn the huge two-dimensional seasonal structure. In TBATS (Trigonometric Exponential Smoothing State Space model with Box-Cox transformation, ARMA errors, Trend and Seasonal Components) [6], a framework based on state space model is introduced for complex seasonal time series with multiple seasonal periods, high frequency seasonality and non-integer seasonality. It incorporates Box-Cox transformations, Fourier representations, and ARMA error correlation. Although an efficient algorithm is proposed to estimate the parameters of the state space model, parameter estimation is still a burden especially when the seasonality period is long. Also it cannot handle abrupt change of trend.

Recently, a novel seasonal-trend decomposition called RobustSTL [24] has been proposed. The RobustSTL is robust to noises and outliers, but it cannot handle multiple seasonality. In addition, its high computational cost on long time series has limited its application especially in the setting of real-world streaming data.

Table 1 summarizes the comparison of popular seasonal-trend decomposition methods, including our new method Fast RobustSTL.

3 PROPOSED APPROACH

3.1 Model Overview

We consider the following complex time series model with trend and multiple seasonality/periodicity [6, 9] as

$$y_t = \tau_t + \sum_{i=1}^{m} s_{i,t} + r_t, \quad t = 0, 1, \cdots, N-1 \tag{1}$$

where y_t represents the observed time series at time t, τ_t denotes the trend component, m is the number of different seasonal components, $s_{i,t}$ is the i-th seasonal component at time t, $s_t = \sum_{i=1}^{m} s_{i,t}$ is the sum of multiple seasonal components, and $r_t = a_t + n_t$ denotes the remainder part which contains the noise n_t and possible outlier a_t. In this paper, we assume that the number of seasonal components m and their seasonal/period lengths $T_i, i = 1, \cdots, m$ are known, which can be obtained by using multiple periodicities detection algorithms like [21, 25]. Without loss of generality, we assume that the maximum seasonal period length is multiple times of other seasonal period length, which is reasonable in most practical scenarios. Note that for non-periodic case, we can just perform trend filtering [16, 23] to decompose time series into trend and remainder components.

For seasonal-trend decomposition in this paper, we aim to decouple each component in Eq. (1) accurately and efficiently. This decomposition is widely adopted in time series analysis, since each component can bring us insights and help to facilitate time series analysis like anomaly detection and forecasting [10, 17, 19]. The main challenges is how to robustly perform seasonal-trend decomposition in the case of outliers and abrupt trend changes, as well as how to separate each seasonal component under multiple seasonality when it is needed. We would deal with the aforementioned challenges in the following sections.

Briefly, our proposed robust seasonal-trend decomposition algorithm consists of four main components: 1) time series denoising; 2) trend extraction; 3) extraction of the sum of multiple seasonal components; and 4) further decomposition of each seasonal component. In the following, we will introduce each component in detail.

3.2 Signal Denoising by Bilateral Filter

Removing noise before the seasonal-trend decomposition can lead to more accurate decomposition. Common existing methods like low-pass filter and Gaussian filter [1] cannot preserve abrupt trend change and hence destruct the underlying seasonal-trend structure. Here we adopt the bilateral filter [18, 20] to denoise time series due to its edge-preserving characteristics and simplicity. The output of the bilateral filter for input time series y_t is given as

$$y'_t = \sum_{j \in J} w^t_j y_j, \quad J = t, t \pm 1, \cdots, t \pm H \tag{2}$$

where the filter window is denoted by J with length $2H + 1$, and the filter weights are calculated by

$$w^t_j = \frac{1}{z} e^{-\frac{|j-t|^2}{2\delta_d^2} - \frac{|y_j - y_t|^2}{2\delta_i^2}}, \tag{3}$$

where $\frac{1}{z}$ is used for normalization, δ_d^2 and δ_i^2 are used to control the smoothness and edge-preserving characteristics. The bilateral filter combines two Gaussian functions for space distance and value difference, respectively; therefore, it achieves edge-preserving and noise removal.

3.3 Trend Extraction by Robust Sparse Model

After removing noise, there are still the interferences from seasonal components and outliers when we aim to extract the trend component. We adopt the seasonal difference operation to roughly remove the seasonal components. Note that different from RobustSTL, in this paper, we need to handle multiple seasonality; therefore, we perform the seasonal difference based on the maximum seasonal period length, i.e.,

$$g_t = \nabla_{\tilde{T}} y'_t = y'_t - y'_{t-\tilde{T}} = \sum_{i=0}^{\tilde{T}-1} \nabla \tau_{t-i} + (\nabla_{\tilde{T}} s_t + \nabla_{\tilde{T}} r'_t),$$

where $\tilde{T} = \max T_i, i = 1, 2, \cdots, m$ is the maximum seasonal period length. After this process, all seasonal components have been roughly removed, and then we propose to extract the trend component by modeling the trend difference with least absolute deviations (LAD) objective function and two ℓ_1-norm regularizations.

Specifically, the trend difference is computed by solving the following optimization problem:

$$\min_{\nabla \tau_t} \sum_{t=\tilde{T}+1}^{N} |g_t - \sum_{i=0}^{\tilde{T}-1} \nabla \tau_{t-i}| + \lambda_1 \sum_{t=2}^{N} |\nabla \tau_t| + \lambda_2 \sum_{t=3}^{N} |\nabla^2 \tau_t|, \tag{4}$$

where $\nabla \tau_t = \tau_t - \tau_{t-1}$ is the first order difference operation, and $\nabla^2 \tau_t = \nabla(\nabla \tau_t) = \tau_t - 2\tau_{t-1} + \tau_{t-2}$ is the second order difference operation. The LAD loss function in Eq. (4) makes the extracted trend robustness to outliers. The two ℓ_1-norm regularizations in the second and third terms of Eq. (4) are used to capture abrupt trend change and slow trend change, respectively. To simplify notation, we can rewrite Eq. (4) in an equivalent matrix form as

$$\min_{\nabla \boldsymbol{\tau}} ||\mathbf{g} - \mathbf{M} \nabla \boldsymbol{\tau}||_1 + \lambda_1 ||\nabla \boldsymbol{\tau}||_1 + \lambda_2 ||\mathbf{D} \nabla \boldsymbol{\tau}||_1, \tag{5}$$

where $\mathbf{g}$ and $\nabla \boldsymbol{\tau}$ are the vector forms as

$$\mathbf{g} = [g_{\tilde{T}+1}, g_{\tilde{T}+2}, \cdots, g_N]^T, \; \nabla \boldsymbol{\tau} = [\nabla \tau_2, \nabla \tau_3, \cdots, \nabla \tau_N]^T, \tag{6}$$

while $\mathbf{M}$ and $\mathbf{D}$ are the $(N - \tilde{T}) \times (N - 1)$ and $(N - 2) \times (N - 1)$ Toeplitz matrix, respectively,

$$\mathbf{M} = \begin{bmatrix} \overbrace{1 \;\; \cdots \;\; 1}^{\tilde{T} \text{ ones}} & & & \\ & 1 \;\; \cdots \;\; 1 & & \\ & & \ddots & \\ & & & 1 \;\; \cdots \;\; 1 \end{bmatrix}, \; \mathbf{D} = \begin{bmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & & \ddots & \\ & & & 1 & -1 \end{bmatrix}. \tag{7}$$

The optimization problem in Eq. (5) can be solved by linear programming (LP) as in original RobustSTL algorithm [24]. However, it is not efficient for very long time series due to the high dimensionality of direct matrix operation. In Section 4, we will introduce our novel GADMM algorithm to efficiently solve the problem in Eq. (5).

Once we obtain the solution of trend difference $\nabla \tilde{\tau}_t$ in Eq. (4) (equivalently Eq. (5)), the relative trend can be calculated by cumulative summation as

$$\tilde{\tau}^r_t = \begin{cases} 0, & t = 1 \\ \sum_{i=2}^{t} \nabla \tilde{\tau}_i, & t \geq 2 \end{cases}. \tag{8}$$

Based on the relative trend from the denoised time series, we can obtain the detrended time series y''_t as follows

$$y''_t = y'_t - \tilde{\tau}^r_t. \tag{9}$$

3.4 Seasonal Component Extraction by Non-local Seasonal Filter

After noise removal and trend extraction, in this step we extract the sum of multiple seasonal component from y''. We achieve this by extending the non-local seasonal filter from RobustSTL [24] with the following two improvements. First, we consider multiple seasonality by weighting multiple non-local seasonal filters, where each non-local seasonal filter corresponds to a seasonal component. Second, the filtering process utilizes both previous and next seasonal neighborhoods instead of only previous seasonal neighborhoods as adopted in RobustSTL. As a result, the first seasonal periods for multiple seasonal components can be processed properly as the outliers in the first seasonal period can be removed using the data happened after that. However, this is not true for the original RobustSTL.

To extract the multiple seasonal components, we extend the non-local seasonal filter to the multiple seasonality scenario. Specifically, we define the weighted non-local seasonal filters by considering multiple seasonality together as follows:

$$\tilde{s}_t = \frac{1}{z}\sum_{i=1}^{m}\alpha_i \sum_{(t'_i,j)\in\Omega} w^t_{(t'_i,j)} y''_j \tag{10}$$

where $\frac{1}{z}$ is the normalization factor, $\alpha_i > 0$ is the weighting factor for each non-local seasonal filter, $w^t_{(t'_i,j)}$ is the weights of each non-local seasonal filter, and Ω denotes previous and next seasonal neighborhoods. The $w^t_{(t'_i,j)}$ and Ω in each non-local seasonal filter are defined as

$$w^t_{(t'_i,j)} = e^{-\frac{|j-t'_i|^2}{2\delta_d^2}-\frac{|y''_j-y''_{t_i}|^2}{2\delta_i^2}} \tag{11}$$

$$\Omega = \{(t'_i,j)|(t'_i = t \pm k \times T_i, j = t'_i \pm h)\} \tag{12}$$
$$k = 1, 2, \cdots, K;\ h = 0, 1, \cdots, H$$

by considering previous K seasonal neighborhoods and next K seasonal neighborhoods where each neighborhood contains $2H + 1$ points.

As a common treatment to make seasonal-trend decomposition unique, we remove the mean value of the multiple seasonal component from Eq. (10) such that the sum of all seasonal components in a period is approximately zero [9]. This mean value is then added to the relative trend extracted in Section 3.3 to get the final trend component. Therefore, the final trend component and multiple seasonal component are obtained by

$$\hat{\tau}_t = \tilde{\tau}^r_t + \mu_{\tilde{s}_t},\ \tilde{\tilde{s}}_t = \tilde{s}_t - \mu_{\tilde{s}_t}, \tag{13}$$

where $\mu_{\tilde{s}_t}$ is the mean value of multiple seasonal component $\tilde{s}_t$ in Eq. (10). And the estimate of remainder can be obtained as

$$\hat{r}_t = y_t - \hat{\tau}_t - \tilde{\tilde{s}}_t. \tag{14}$$

3.5 Further Decomposition of Multiple Seasonal Components

In this section, we propose a sparse modeling based algorithm to further decompose the sum of multiple seasonal component $\tilde{\tilde{\mathbf{s}}}$ extracted from Section 3.4. Mathematically, the sum of m multiple seasonal component, i.e., $\tilde{\tilde{\mathbf{s}}}$, can be formulated as:

$$\tilde{\tilde{\mathbf{s}}} = \sum_{i=1}^{m}\mathbf{s}_i, \quad \text{and} \quad \sum_{t=j}^{t=j+T_i-1} s_{i,t} = 0,\ \forall i = 1, 2, \cdots, m \tag{15}$$

where $\mathbf{s}_i$ is the i-th seasonal component with period T_i, without loss of generality, we assume $T_1 < T_2 < \cdots < T_m$.

Our goal is to recover $\mathbf{s}_i$ as accurate as possible, and the corresponding optimization problem can be formulated as:

$$\underset{\{\mathbf{s}_i|i=1,2,\cdots,m\}}{\arg\min}\ \mathcal{L}(\tilde{\tilde{\mathbf{s}}} - \sum_{i=1}^{m}\mathbf{s}_i). \tag{16}$$

However, the aforementioned formulation has an infinite number of feasible solutions; therefore, in order to properly recover all latent seasonal patterns, one has to make some further proper assumptions:

- Each seasonal component $\mathbf{s}_i$ has a repeated pattern that changes slowly or stays constant over time;
- A seasonal component with a shorter period usually has more volatility than a seasonal component with a longer period;
- The smoothness of different seasonal component is also different.

In our model, these assumptions are encoded via multiple ℓ_1-norm regularization terms. Specifically, the seasonal-wise stability can be measured via seasonal-wise second-order difference, i.e, $\nabla^2_T x_t = x_t - 2x_{t-T} + x_{t-2T}$, the volatility of a signal can be described by its first-order difference, i.e., $\nabla x_t = x_t - x_{t-1}$, and the smoothness of a time series data can be quantified via its second-order difference, i.e., $\nabla^2 x_t = x_t - 2x_{t-1} + x_{t-2}$. The ℓ_1-norm has been employed on each of these measurements to encode our aforementioned assumptions, and the property of sparseness of the ℓ_1-norm also enables our model to handle the potential non-smooth patterns in some seasonal components. Accordingly, we can further formulate the multiple seasonal time series decomposition as:

$$\underset{\{\mathbf{s}_i|i=1,2,\cdots,m\}}{\arg\min}\ \frac{1}{2}||\tilde{\tilde{\mathbf{s}}} - \sum_{i=1}^{m}\mathbf{s}_i||_2^2 + \sum_{i=1}^{m}\lambda_{1,i}||\mathbf{D}\mathbf{s}_i||_1 + \sum_{i=1}^{m}\lambda_{2,i}||\mathbf{D}^2\mathbf{s}_i||_1 + \sum_{i=1}^{m}\lambda_{3,i}||\mathbf{D}^2_{\mathbf{T_i}}\mathbf{s}_i||_1, \tag{17}$$

where $\mathbf{D}$ is the first-order difference matrix as in Eq.(7), $\mathbf{D}^2 \in \mathbb{R}^{(N-2)\times N}$ and $\mathbf{D}^2_{\mathbf{T_i}} \in \mathbb{R}^{(N-2T_i)\times N}$ are the second-order difference matrix and the seasonal-wise second-order difference matrix, respectively, with the following forms

$$\mathbf{D}^2 = \begin{bmatrix} 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & & \ddots & & \\ & & & 1 & -2 & 1 \end{bmatrix},$$

$$\mathbf{D}^2_{T_i} = \begin{bmatrix} 1 & \overbrace{0 \ \cdots \ 0}^{T_i - 1\ zeros} & -2 & \overbrace{0 \ \cdots \ 0}^{T_i - 1\ zeros} & 1 & \\ & 1 \ \ 0 \ \cdots & 0 & -2 \ \ 0 \ \cdots & 0 & 1 \\ & & \ddots & & & \\ & & 1 \ \ 0 \ \cdots & 0 & -2 \ \ 0 \ \cdots & \end{bmatrix}.$$

Note that the multiple seasonal decomposition in Eq. (17) has similar formulation as the trend extraction in Eq. (5). One difference is that the sum-of-square loss is adopted in multiple seasonal decomposition since the outliers have already been removed by the proposed non-local seasonal filter as discussed in Section 3.4. We will show that the optimization problem in Eq. (17) can also be efficiently solved by our designed GADMM algorithm in Section 4.

4 OPTIMIZATION METHODOLOGY

In this section, we develop a specialized GADMM algorithm to solve the trend extraction in Eq. (5) in Section 3.3 and the multiple seasonal decomposition in Eq. (17) in Section 3.5.

Note that both Eq. (5) and Eq. (17) can be formulated in a common compact form as

$$\min_{\mathbf{x}} f(\mathbf{A}\mathbf{x}), \tag{18}$$

where $\mathbf{A} := [\mathbf{A}_1; \mathbf{A}_2; \cdots; \mathbf{A}_k]$ is a structured filter matrix with $\mathbf{A}_i \in \mathbb{R}^{n_i \times N}$, and $f(\mathbf{y}) := f_1(\mathbf{y}_{[1:n_1]}) + \cdots + f_k(\mathbf{y}_{[\sum_{i=1}^{k-1} n_i+1:\sum_{i=1}^{k} n_i]})$

is the sum of *simple functions* $f_i(\cdot)$, e.g., the LAD loss function or the sum-of-squares loss function.

In the following, we will first discuss some limitations of current approaches for solving the optimization problem in Eq. (18), which motivates us to propose a novel GADMM algorithm. Then, we present the detailed procedure of our GADMM algorithm with its convergence analysis.

Note that the computation efficiency of iterative methods for solving Eq. (18) can be analyzed from two dimensions:

- Cheap Iteration: the algorithm should be able to exploit the filter structure of the matrix $\mathbf{A}$ to reduce the number of algebraic operations required in each iteration to $O(N)$.
- Small Iteration Number: the algorithm should be able to exploit the geometric properties of Eq. (18), i.e., the objective function "operates" in the linear space defined by $\mathbf{y} := \mathbf{Ax}$ rather than directly in $\mathbf{x}$.

However, two state-of-the-art methods, namely the ADMM [3] and the PDHG [4] algorithm, can only do well in one of those two dimensions. More specifically, in the PDHG algorithm with primal prox-update in $\mathbf{x}$ using gradient $\mathbf{A}^T\mathbf{u}_t$ and dual prox-update $\mathbf{u}$ using gradient $\mathbf{Ax}_t$, the filter structure of $\mathbf{A}$ can be exploited to compute these gradients efficiently, thus achieving cheap iterations. However, since the primal updates are in $\mathbf{x}$, the geometric properties of Eq. (18) are not utilized, resulting in a large iteration complexity with a constant of $\|\mathbf{A}\|\,\|\mathbf{x}_0 - \mathbf{x}_*\|$ as opposed to $\|\mathbf{A}(\mathbf{x}_0 - \mathbf{x}_*)\|$. On the other hand, the ADMM method essentially performs primal and dual prox update in $\mathbf{y} := \mathbf{Ax}$ and $\mathbf{u}$, so it can save the iteration number. But projection onto the $\mathbf{Ax}$ space is expensive because we need to perform a matrix vector multiplication with a possibly dense matrix $(\mathbf{A}^T\mathbf{A})^{-1}$.

To excel in both dimensions, our proposed GADMM generalizes the ADMM algorithm by appending a stability term to the x-prox update step such that the difficult quadratic term $\langle \mathbf{A}^T\mathbf{Ax}, \mathbf{x}\rangle$ can be approximated by $\langle \mathbf{Gx}, \mathbf{x}\rangle$ with $\mathbf{G} := \mathbf{A}^T\mathbf{A} + \hat{\mathbf{H}}$ and $\hat{\mathbf{H}} \succeq 0$. By picking a $\mathbf{G}$ which both closely approximates $\mathbf{A}^T\mathbf{A}$ and is easily invertible (i.e. matrix vector multiplication involving $\mathbf{G}^{-1}$ can be carried out efficiently), we are able to achieve both a small iteration number and cheap iterations, as shown in Table 2. The detailed steps of our proposed GADMM is summarized in Algorithm 1, where the appended stability term is highlighted by red color.

We remark here that our proposed GADMM algorithm is similar to the generalized ADMM algorithm in [8]. Note that [8] uses the extra stability term to ensure the convergence of multi-block ADMM, thus the convergence proofs are completely different.

Table 2: Theoretical computational cost of the proposed GADMM, PDHG and ADMM algorithms.

Algorithm	Iter Number Complexity[1]	Per Iter Cost
PDHG	$O(\|\mathbf{A}\|\,\|\mathbf{x}_0 - \mathbf{x}^*\|\,\|\mathbf{u}_0 - \mathbf{u}^*\|/\epsilon)$	$O(N)$
ADMM	$O(\|\mathbf{A}(\mathbf{x}_0 - \mathbf{x}^*)\|\,\|\mathbf{u}_0 - \mathbf{u}^*\|/\epsilon)$	$O(N^2)$
GADMM	$O(\|\sqrt{\mathbf{G}}(\mathbf{x}_0 - \mathbf{x}^*)\|\,\|\mathbf{u}_0 - \mathbf{u}^*\|/\epsilon)$	[2]usually $O^\#(N)$

[1] Iteration number complexity is an upper bound for number of iterations to find ϵ-suboptimal solution, i.e., $f(\mathbf{A}\bar{\mathbf{x}}_\mathcal{T}) - f(\mathbf{Ax}_*) \le \epsilon$
[2] $O^\#(\cdot)$ means $O(\cdot)$ algebraic operations except for some $\log(N)$.

Algorithm 1: The proposed GADMM Algorithm

```
Input $\mathbf{x}_0 \in X$, $\mathbf{u}_0$ and stepsizes $\rho > 0$;
Initialization set $\mathbf{y}_0 = \mathbf{Ax}_0$;
for $t = 0, 1, 2, 3 \ldots \mathcal{T} - 1$ do
    set $\mathbf{x}_{t+1} = \arg\min_{\mathbf{x}\in X} \langle \mathbf{Ax}, \mathbf{u}_t\rangle + \frac{\rho}{2}\|\mathbf{Ax} - \mathbf{y}_t\|^2 + \frac{\rho}{2}\langle \hat{\mathbf{H}}(\mathbf{x} - \mathbf{x}_t), \mathbf{x} - \mathbf{x}_t\rangle$;
    set $\mathbf{y}_{t+1} = \arg\min_{\mathbf{y}} f(\mathbf{y}) - \langle \mathbf{y}, \mathbf{u}_t\rangle + \frac{\rho}{2}\|\mathbf{Ax}_{t+1} - \mathbf{y}\|^2$;
    set $\mathbf{u}_{t+1} = \mathbf{u}_t + \rho(\mathbf{Ax}_{t+1} - \mathbf{y}_{t+1})$;
end
Return the ergodic solution $\bar{\mathbf{x}}_\mathcal{T} = \sum_{t=1}^{\mathcal{T}} \frac{\mathbf{x}_t}{\mathcal{T}}, \bar{\mathbf{u}}_\mathcal{T} = \sum_{t=1}^{\mathcal{T}} \frac{\mathbf{u}_t}{\mathcal{T}}$;
```

4.1 Implementation Details

When applying the proposed GADMM method to our problems, we need to specify the choice of $\mathbf{G}$. We mention a few commonly used easily invertible matrices, which will be used as building blocks for constructing specific $\mathbf{G}$s for Eq. (5) and Eq. (17) later.

Scaled Identity Matrix $\alpha\mathbf{I}$ for some $\alpha > 0$. For example, given any matrix $\mathbf{B}$, the minimal scaled identity matrix dominating it is $\|\mathbf{B}\|_{2,2}\,\mathbf{I}$.

Circulant Matrix C , in which the first row is repeated N times with an consecutive offsets from 0 to $N - 1$:

$$\mathbf{C} := \begin{bmatrix} \mathbf{c}_{[1]}, & \mathbf{c}_{[2:N]} \\ \vdots & \vdots \\ \mathbf{c}_{[N-i+1:N]}, & \mathbf{c}_{[1:N-i]} \\ \vdots & \vdots \\ \mathbf{c}_{[2:N]}, & \mathbf{c}_{[1]} \end{bmatrix}.$$

As shown in [11], such a matrix admits a eigenvalue decomposition $\mathbf{C} = \mathbf{W}^{-1}\mathrm{diag}(\hat{\mathbf{c}})\mathbf{W}$, where $\mathbf{W}$ is the discrete Fourier transform matrix and $\hat{\mathbf{c}} := \mathrm{DFT}(\mathbf{c})$. So $\mathbf{C}^{-1}\mathbf{v}$ could be carried out in $O(N\log N)$ algebraic operations by $\mathrm{IDFT}(\mathrm{DFT}(\mathbf{v})./\hat{\mathbf{c}})$, where ./ denotes the component-wise division.

Moreover, the class of circulant matrices is closed under transpose, addition and composition. So when designing a circulant matrix $\mathbf{G}$ for $\mathbf{A}^T\mathbf{A} = \sum_{i=1}^{n} \mathbf{A}_i^T\mathbf{A}_i$, it is useful to build a dominating circulant approximation matrix $\mathbf{G}_i$ for each $\mathbf{A}_i^T\mathbf{A}_i$ and then combine them to get an overall $\mathbf{G}$, $\mathbf{G} = \sum_{i=1}^{k}\mathbf{G}_i$.

In fact, circulant matrix is especially well suited for filter matrices $\mathbf{A}_i$, which is almost circulant except for the bottom boundary, as shown below.

$$\mathbf{A}_i = \begin{bmatrix} \mathrm{xxxxx} & \ldots & \ldots \\ & \ddots & \\ \ldots & \mathrm{xxxxx} & \ldots \\ & \ddots & \\ \ldots & \ldots & \mathrm{xxxxx} \end{bmatrix}$$

$$\hat{\mathbf{A}}_i = \begin{bmatrix} \mathrm{x} & \ldots & \mathrm{xxxx} \\ & \ddots & \\ \mathrm{xxxx} & \ldots & \mathrm{x} \end{bmatrix}$$

We can append a small $\hat{\mathbf{A}}_i$ to the bottom of $\mathbf{A}_i$ to make $\mathbf{A}_i^{\text{aug}} := [\mathbf{A}_i; \hat{\mathbf{A}}_i]$ a circulant matrix. Then the circulant matrix $\mathbf{G}_i := (\mathbf{A}_i^{\text{aug}})^T \mathbf{A}_i^{\text{aug}}$dominates $\mathbf{A}_i^T \mathbf{A}_i$ because $\hat{\mathbf{A}}_i^T \hat{\mathbf{A}}_i \succeq 0$. In latter applications, we will use $\mathbf{C}(\mathbf{A}^T\mathbf{A}) := \sum_{i=1}^k (\mathbf{A}_i^{\text{aug}})^T \mathbf{A}_i^{\text{aug}}$ to denote such a dominating circulant matrix.

Block Diagonal Matrix Given that $\mathbf{G} := \text{diag}(\mathbf{G}_1, \mathbf{G}_2, \ldots \mathbf{G}_n)$, its inverse is calculated by $\mathbf{G}^{-1} := \text{diag}(\mathbf{G}_1^{-1}, \mathbf{G}_2^{-1}, \ldots \mathbf{G}_n^{-1})$ as

$$\mathbf{G} := \begin{bmatrix} \mathbf{G}_1 & \cdots & \cdots & \cdots \\ \cdots & \mathbf{G}_2 & \cdots & \cdots \\ \ddots & \ddots & \ddots & \ddots \\ \cdots & \cdots & \cdots & \mathbf{G}_n \end{bmatrix} \quad \mathbf{G}^{-1} := \begin{bmatrix} \mathbf{G}_1^{-1} & \cdots & \cdots & \cdots \\ \cdots & \mathbf{G}_2^{-1} & \cdots & \cdots \\ \ddots & \ddots & \ddots & \ddots \\ \cdots & \cdots & \cdots & \mathbf{G}_n^{-1} \end{bmatrix}$$

So if every block of $\mathbf{G}$ is easy invertible, $\mathbf{G}$ is also easily invertible.

Now we are ready to construct specific $\mathbf{G}$s for our problems.

Trend Extraction : the Eq. (5) can be formulated to Eq. (18) as $f_1(\cdot) = \|\cdot - \mathbf{g}\|_1, f_2(\cdot) = f_3(\cdot) = \|\cdot\|_1$ and $\mathbf{A}_1 = \mathbf{M}, \mathbf{A}_2 = \lambda_1 \mathbf{I}, \mathbf{A}_3 = \lambda_2 \mathbf{D}$. Observe that all $\mathbf{A}_i$s are nearly circulant, so we can pick $\mathbf{G} = \mathbf{C}([\mathbf{A}_1; \mathbf{A}_2; \mathbf{A}_3]) = \mathbf{C}(\mathbf{M}^T\mathbf{M}) + \lambda_1^2 \mathbf{I} + \lambda_2^2 \mathbf{C}(\mathbf{D}^T\mathbf{D})$.

Seasonal Component Extraction : for Eq. (17), we can concatenate seasonal components into $\mathbf{s} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m]$, and rewrite it more concisely as

$$\min_{\mathbf{s}} \frac{1}{2} \left\|\tilde{\tilde{\mathbf{s}}} - \tilde{\mathbf{I}}\mathbf{s}\right\|_2^2 + \left\|\tilde{\mathbf{D}}\mathbf{s}\right\|_1, \tag{19}$$

where $\tilde{\mathbf{I}} := \underbrace{[\mathbf{I}, \mathbf{I}, \ldots, \mathbf{I}]}_{m \text{ Is}}$, $\tilde{\mathbf{D}}_i := \left[\lambda_{1,i}\mathbf{D}; \lambda_{2,i}\mathbf{D}^2; \lambda_{3,i}\mathbf{D}^2_{\mathrm{T_i}}\right]$ and

$$\tilde{\mathbf{D}} := \begin{bmatrix} \tilde{\mathbf{D}}_1 & 0 & \cdots & 0 \\ 0 & \tilde{\mathbf{D}}_2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & \tilde{\mathbf{D}}_m \end{bmatrix}.$$

Now, Eq. (19) can be formulated to Eq. (18) as $f_1(\cdot) = \frac{1}{2}\left\|\cdot - \tilde{\tilde{\mathbf{s}}}\right\|_2$, $f_2(\cdot) = \|\cdot\|_1$ and $\mathbf{A}_1 = \tilde{\mathbf{I}}, \mathbf{A}_2 = \tilde{\mathbf{D}}$. Since $\mathbf{G}$ needs to dominate $\tilde{\mathbf{I}}^T\tilde{\mathbf{I}} + \tilde{\mathbf{D}}^T\tilde{\mathbf{D}}$, we use a scaled identity matrix to dominate $\tilde{\mathbf{I}}^T\tilde{\mathbf{I}}$ and circulant matrices to dominate each block of $\tilde{\mathbf{D}}^T\tilde{\mathbf{D}}$. Then we put those dominating blocks together to obtain a block diagonal $\mathbf{G}$, i.e.,

$$\mathbf{G} := \begin{bmatrix} \mathbf{C}(\tilde{\mathbf{D}}_1^T\tilde{\mathbf{D}}_1) + m\mathbf{I} & 0 & \cdots & 0 \\ 0 & \mathbf{C}(\tilde{\mathbf{D}}_2^T\tilde{\mathbf{D}}_2) + m\mathbf{I} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & \mathbf{C}(\tilde{\mathbf{D}}_m^T\tilde{\mathbf{D}}_m) + m\mathbf{I} \end{bmatrix}.$$

Note that all blocks of such a block-diagonal $\mathbf{G}$ are easily invertible since they are circulant.

4.2 Complexity Analysis

Based on the above analysis, our proposed GADMM can achieve overall $O(N \log N)$ complexity in each iteration/updating step due to the efficient IDFT/DFT computation for the constructed circulant matrix. Next, for the convergence of the optimality gap of the proposed GADMM algorithm, it can be guaranteed theoretically by the following Theorem 4.1

Theorem 4.1. *Let $\bar{\mathbf{x}}_{\mathcal{T}}$ be generated by Algorithm 1 with* $\mathbf{G} \succeq \mathbf{A}^T\mathbf{A}$. *Let $\mathbf{x}^*$ be the solution to* (18), $\mathbf{u}^* \in \partial f(\mathbf{A}\bar{\mathbf{x}}_{\mathcal{T}})$ *and* $\mathbf{z}^* := (\mathbf{x}^*, \mathbf{u}^*)$. *If* $\rho = \|\mathbf{u}_0 - \mathbf{u}^*\|_2 \Big/ \left\|\sqrt{\mathbf{G}}(\mathbf{x}_0 - \mathbf{x}^*)\right\|_2$, *then we have*

$$f(\mathbf{A}\bar{\mathbf{x}}_{\mathcal{T}}) - f(\mathbf{A}\mathbf{x}^*) \leq \frac{\|\mathbf{u}_0 - \mathbf{u}^*\|_2 \left\|\sqrt{\mathbf{G}}(\mathbf{x}_0 - \mathbf{x}^*)\right\|_2}{\mathcal{T}}, \tag{20}$$

The proof is provided in the Appendix due to the space limit. The whole complexity comparisons with other optimization solutions are summarized in Table 2.

5 EXPERIMENTS AND DISCUSSION

In this section, we study the proposed Fast RobustSTL algorithm empirically in comparison with other state-of-the-art seasonal-trend decomposition algorithms on both synthetic and real-world datasets.

5.1 Baseline Algorithms

For the comparison of seasonal-trend decomposition, we use the following Four state-of-the-art baseline algorithms:

- *RobustSTL [24]*: It performs seasonal-trend decomposition using robust loss function and sparse regularization. However, it can only handle time series with a single seasonality.
- *MSTL [15]*: It performs seasonal-trend decomposition based on Loess. The multiple seasonality is supported and computed in an iterative way.
- *TBATS [6]*: It decomposes the input time series into trend, level, seasonality, and remainder. The trend and level are jointly together to represent the real trend. The multiple seasonality is decomposed based on their proposed trigonometric formulation.
- *STR [9]*: It assumes the continuity in both trend and seasonal components using regression based on ℓ_2-norm. The multiple seasonality is decomposed based on the regularizations of second difference along the time, time-season and season dimensions.

We implemented both RobustSTL and Fast RobustSTL in Python. To test other decomposition algorithms, we use their implementations in R: stl, tbats in forecast package [1] and AutoSTR in stR package [2].

5.2 Datasets

5.2.1 Synthetic Datasets. We generate two synthetic datasets based on sine wave and square wave. For each wave pattern, we use three seasonal components with period lengths as $T_1 = 24, T_2 = 7 \times T_1 = 168, T_3 = 4 \times T_2 = 672$, and generate sine or square wave according to these periods with amplitude of 1, 1.5 and 2 respectively. After that, we add Gaussian noise with 0.2 standard deviation, randomly add 10 spikes and 10 dips outliers with amplitude as 10, and add trend with both slow and abrupt trend changes to the original signal, which gives our final synthetic datasets. Figure 1 demonstrates the generation of sine wave dataset with 5376 observations, and similar procedure is used to generate square wave dataset.

[1]https://cran.r-project.org/web/packages/forecast/index.html
[2]https://cran.r-project.org/web/packages/stR/index.html

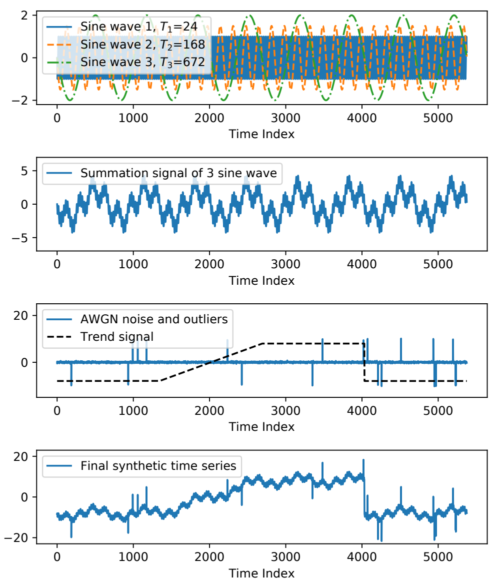

Figure 1: From top to bottom: the three sine waves with different seasonal periods; the sum of the three sine waves; simulated noise, 20 spike & dip outliers, and trend with both slow and abrupt changes (orange line); the final synthetic data.

5.2.2 *Real-world Datasets.* For time series with single seasonality, we adopt the file exchange count dataset from RobustSTL [24], which has a total of 4320 observations with the period being 288. For time series with multiple seasonality, we collect some operation count from one of largest clouding companies. This dataset contains the records of 11 weeks operation counts with hourly resolution, and it clearly shows daily ($T_1 = 24$) and weekly ($T_2 = 168$) seasonal patterns. To better demonstrate whether different decomposition algorithms are robust to sudden trend changes and spikes/dips, we generated a perturbed data. we add a abrupt trend, a spike, and a dip to the original data. Furthermore, we multiply one day's data by 3 times to evaluate whether different decomposition algorithms are also robust to pattern outlier (not just point outliers like spikes/dips). The original and final signal are shown in Figure 2.

5.3 Comparisons on Time Series with Single Seasonality

Since the proposed Fast RobustSTL algorithm is the generalization of RobustSTL with much faster computation speed, we omit the demonstration of seasonal-trend decomposition for time series with single seasonality due to that both have very similar decomposition performance. Instead, here we compare their computational speed based on the real-world single-seasonal time series of file exchange count in Figure 2. In order to investigate the performance under different lengths of time series, we use upsampling and downsampling to get 4 different time series with length from 1080 to 8640. Note that the most computationally expensive part in RobustSTL is the extraction of the trend component, here we compare the CPU execution

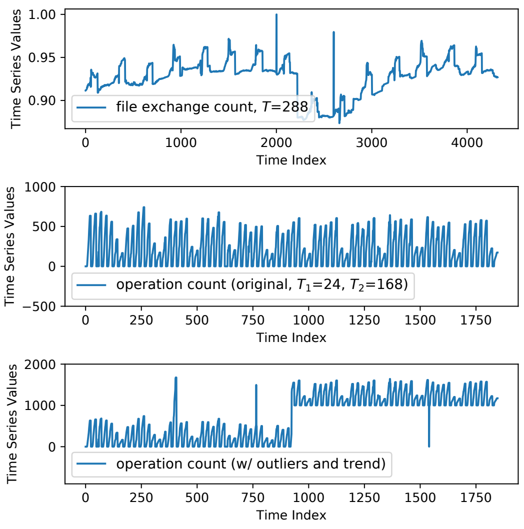

Figure 2: Top figure: real-world time series of file exchange count with single seasonality; middle figure: real-world time series of operation count with two seasonal patterns (daily and weekly seasons); bottom: the perturbed operation count time series by adding a abrupt trend change, one spike outlier, one dip outlier, and pattern outlier spanning one day.

Table 3: Computational speed comparison of trend extraction in RobustSTL-ADMM, RobustSTL-PDHG, and the proposed Fast RobustSTL with time series of increased length on N. The speedup is based on the computational time of RobustSTL-ADMM.

N	Metrics	RobustSTL-ADMM	RobustSTL-PDHG	**Fast RobustSTL**
N=1080	Iter	41	310	37
	Time(Sec)	0.142	0.0319	0.0109
	SpeedUp		4.45x	**13.0x**
N=2160	Iter	48	319	40
	Time(Sec)	1.11	0.0571	0.0295
	SpeedUp		19.4x	**37.6x**
N=4320	Iter	68	602	37
	Time(Sec)	5.98	0.191	0.0988
	SpeedUp		31.3x	**60.5x**
N=8640	Iter	92	1377	53
	Time(Sec)	36.7	0.62	0.254
	SpeedUp		59.1x	**144x**

time of this step. Since the trend extraction in the original RobustSTL with LP for the optimization solver has much slower speed than RobustSTL with ADMM solver (denote as RobustSTL-ADMM) or PDHG solver (denote as RobustSTL-PDHG), here we only summarize the speed comparisons of RobustSTL-ADMM, RobustSTL-PD, and the proposed Fast RobustSTL algorithm. The experiments are conducted on a MacBook Pro with a 2.3GHz Intel i5 CPU and 8GB RAM, and the results are summarized in Table 3. It can be observed that the proposed Fast RobustSTL has the fastest speed, and achieves 10x to 144x speedup over the RobustSTL-ADMM across different N's.

To further demonstrate why the proposed GADMM in our Fast RobustSTL leads to the fast computation, we also plot the convergence and computational time of the ADMM, PDHG, and the

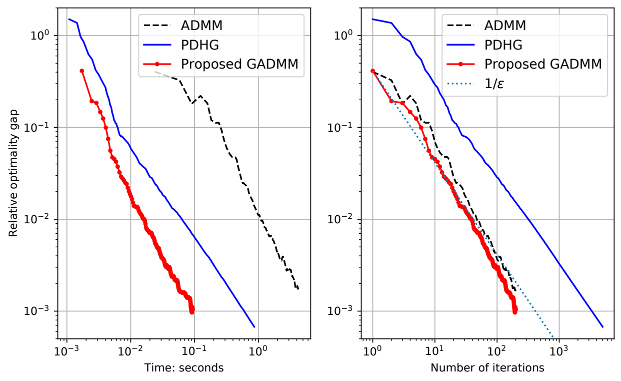

Figure 3: Convergence and computational time comparison of the ADMM, PDHG, and the proposed GADMM optimization methods.

proposed GADMM in the step of trend extraction when the time series length is $N = 2160$ in Figure 3. We can observe that all algorithms converges with the same rate as $O(1/\epsilon)$. But our proposed GADMM has less iteration complexity than ADMM and fewer iteration number than PDHG, leading to a mush faster speed, which is also consistent with our theoretical results in Table 2.

5.4 Comparisons on Time Series with Multiple Seasonalities

For synthetic datasets, we apply all seasonal-trend decomposition algorithms on both sine and square wave datasets to evaluate the decomposition performance. For synthetic data, we know the true seasonal and trend components, so we calculate the mean square error (MSE) of the extracted trend and 3 seasonal components of different algorithms to compare their performance. The experimental results are summarized in Table 4, where the best MSE's are highlighted in bold. Overall, it can be observed that our algorithm outperforms other decomposition algorithms significantly.

To illustrate the decomposition difference of different algorithms, we provide case studies on both synthetic sine wave time series in Figure 1 and real-world perturbed operation count time series in Figure 2. The detailed decomposition results on synthetic datasets and real-world datasets are summarized in Figures 4 and Figures 5, respectively. It can be observed that the MSTL method fails to capture the abrupt drop of the trend, causing the remainder exhibits a sudden increase/decrease around the location of abrupt change point of the trend component. The TBATS method can obtain relative good seasonal components, while the trend component contains unsatisfying noises and outliers. For AutoSTR, it can obtain relative good trend component but the seasonal and remainder components are affected by outliers and trend changes. In contrast, our Fast RobustSTL algorithm recovers all components accurately. For example, the three sine waves in synthetic dataset are very close to the "ground truth" as in Figure 1. Overall, our proposed Fast RobustSTL method achieves the best performance.

Table 4: Comparison of MSE of the seasonal and trend components of the MSTL, TBATS, AutoSTR, and the proposed Fast RobustSTL algorithms on synthetic data.

Data Type	Algorithm	MSE$_{T_1}$	MSE$_{T_2}$	MSE$_{T_3}$	MSE$_{trend}$
Sine	MSTL	0.0601	0.1527	0.4240	2.8479
	TBATS	**0.0018**	0.0409	0.2781	0.4289
	AutoSTR	0.0330	4.5909	4.7550	2.4615
	Fast RobustSTL	0.0284	**0.0047**	**0.0178**	**0.0330**
Square	MSTL	0.1018	0.2086	0.4349	2.8480
	TBATS	0.0721	0.1909	0.8456	0.9843
	AutoSTR	N/A	N/A	N/A	N/A
	Fast RobustSTL	**0.0630**	**0.0386**	**0.0451**	**0.0331**

6 CONCLUSIONS

Time series with multiple seasonality is widely observed in real-world applications, which calls for effective and efficient seasonal-trend decomposition. However, due to the existence of multiple seasonal components, data non-stationarity, noise, outliers and abrupt trend change, current state-of-the-art decomposition algorithms cannot perform the decomposition well. In this paper, we propose a new decomposition algorithm called Fast RobustSTL, which can handle all these challenges. In order to speed up the computation of the proposed decomposition, we further propose a special generalized ADMM algorithm. We rigorously prove that the proposed algorithm converges approximately as standard ADMM while reducing the complexity from $O(N^2)$ to $O(N \log N)$ for each iteration. Empirical studies on time series data with multiple seasonality in comparison with other state-of-the-art algorithms including MSTL, STR, TBATS show the superior performance of our Fast RobustSTL, in terms of both effectiveness and computing efficiency. In the future, we plan to explore multi-resolution seasonal-trend decomposition algorithm for further improved efficiency.

REFERENCES

[1] Herman Blinchikoff and Helen Krause. 2001. *Filtering in the time and frequency domains.* Noble Pub, Atlanta, GA.
[2] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control.* John Wiley & Sons.
[3] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine learning* 3, 1 (2011), 1–122.
[4] Antonin Chambolle and Thomas Pock. 2011. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision* 40, 1 (2011), 120–145.
[5] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. 1990. STL: A Seasonal-Trend Decomposition Procedure Based on Loess. *Journal of Official Statistics* 6, 1 (1990), 3–73.
[6] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. 2011. Forecasting Time Series with Complex Seasonal Patterns using Exponential Smoothing. *J. Amer. Statist. Assoc.* 106, 496 (2011), 1513–1527.
[7] Wei Deng, Ming-Jun Lai, Zhimin Peng, and Wotao Yin. 2017. Parallel multi-block ADMM with O (1/k) convergence. *Journal of Scientific Computing* 71, 2 (2017), 712–736.
[8] Wei Deng and Wotao Yin. 2016. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing* 66, 3 (2016), 889–916.
[9] Alexander Dokumentov, Rob J Hyndman, et al. 2015. *STR: A Seasonal-Trend Decomposition Procedure Based on Regression.* Technical Report. Monash University, Department of Econometrics and Business Statistics.
[10] Jingkun Gao, Xiaomin Song, Qingsong Wen, Pichao Wang, Liang Sun, and Huan Xu. 2020. RobustTAD: Robust Time Series Anomaly Detection via Decomposition and Convolutional Neural Networks. *arXiv preprint arXiv:2002.09535* (2020).

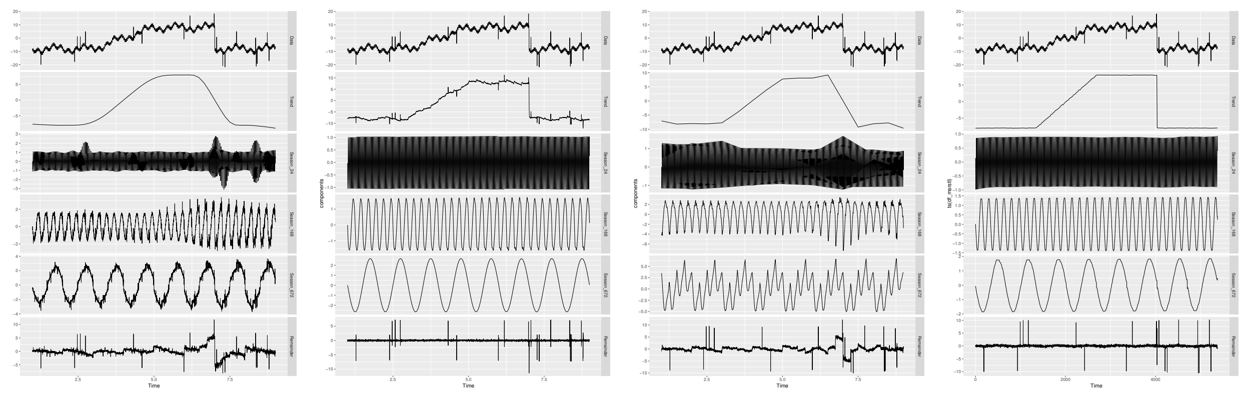
(a) MSTL (b) TBATS (c) AutoSTR (d) Fast RobustSTL

Figure 4: Comparison of different seasonal-trend decomposition algorithms on synthetic dataset.

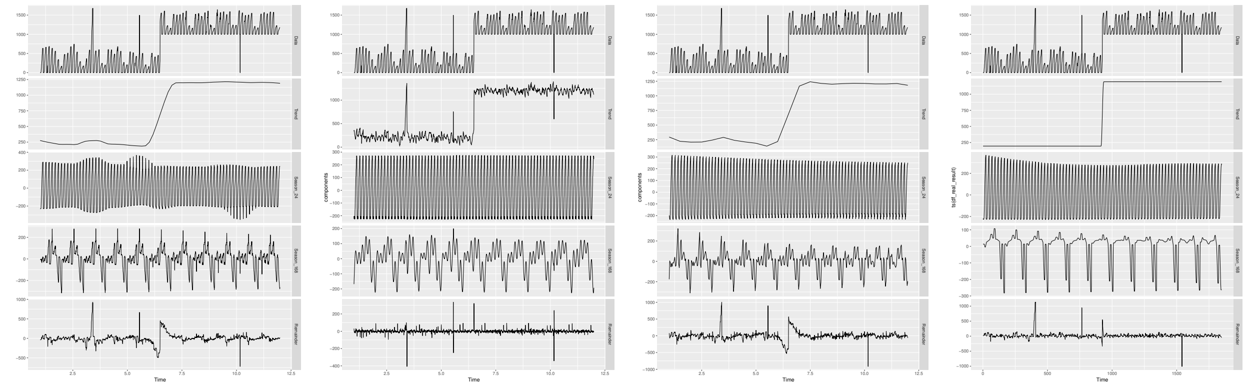
(a) MSTL (b) TBATS (c) AutoSTR (d) Fast RobustSTL

Figure 5: Comparison of different seasonal-trend decomposition algorithms on perturbed real-world dataset.

[11] Robert M. Gray. 2006. Toeplitz and Circulant Matrices: A Review. *Foundations and Trends® in Communications and Information Theory* 2, 3 (2006), 155–239.

[12] Lan Guanghui. 2019. *Lectures on Optimization Methods for Machine Learning.*

[13] David Hallac, Sagar Vare, Stephen Boyd, and Jure Leskovec. 2017. Toeplitz inverse covariance-based clustering of multivariate time series data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 215–223.

[14] Jordan Hochenbaum, Owen S. Vallis, and Arun Kejariwal. 2017. Automatic Anomaly Detection in the Cloud Via Statistical Learning. (2017). arXiv:1704.07706

[15] Rob J Hyndman, George Athanasopoulos, Christoph Bergmeir, Gabriel Caceres, Leanne Chhay, Mitchell O'Hara-Wild, Fotios Petropoulos, Slava Razbash, Earo Wang, and Farah Yasmeen. 2018. Package 'forecast'. *[Online] https://cran. r-project. org/web/packages/forecast/forecast. pdf* (2018).

[16] Seung-Jean Kim, Kwangmoo Koh, Stephen Boyd, and Dimitry Gorinevsky. 2009. ℓ_1 Trend Filtering. *SIAM Rev.* 51, 2 (2009), 339–360.

[17] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. 2015. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1939–1947.

[18] Sylvain Paris, Pierre Kornprobst, and Jack Tumblin. 2009. *Bilateral Filtering*. Now Publishers Inc., Hanover, MA.

[19] Marina Theodosiou. 2011. Forecasting monthly and quarterly time series using STL decomposition. *International Journal of Forecasting* 27, 4 (2011), 1178–1195.

[20] JL Thompson. 2014. *An Empirical Evaluation of Denoising Techniques for Streaming Data.* Technical Report. Report LLNL-TR-659435. Lawrence Livermore National Laboratory, Livermore, Calif.

[21] Michail Vlachos, Philip Yu, and Vittorio Castelli. 2005. On periodicity detection and structural periodic similarity. In *Proceedings of the 2005 SIAM international conference on data mining*. SIAM, 449–460.

[22] Jingyuan Wang, Ze Wang, Jianfeng Li, and Junjie Wu. 2018. Multilevel wavelet decomposition network for interpretable time series analysis. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2437–2446.

[23] Qingsong Wen, Jingkun Gao, Xiaomin Song, Liang Sun, and Jian Tan. 2019. RobustTrend: A Huber Loss with a Combined First and Second Order Difference Regularization for Time Series Trend Filtering. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI '19)*. 3856–3862.

[24] Qingsong Wen, Jingkun Gao, Xiaomin Song, Liang Sun, Huan Xu, and Shenghuo Zhu. 2019. RobustSTL: A Robust Seasonal-Trend Decomposition Algorithm for Long Time Series. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '19)*, Vol. 33. 5409–5416.

[25] Qingsong Wen, Kai He, Liang Sun, Yingying Zhang, Min Ke, and Huan Xu. 2020. RobustPeriod: Time-Frequency Mining for Robust Multiple Periodicities Detection. *arXiv preprint arXiv:3056069* (2020).

[26] Bo Wu, Tao Mei, Wen-Huang Cheng, and Yongdong Zhang. 2016. Unfolding Temporal Dynamics: Predicting Social Media Popularity Using Multi-scale Temporal Decomposition *(AAAI '16)*. AAAI Press, 272–278. http://dl.acm.org/citation.cfm?id=3015812.3015852

[27] Dazhi Yang, Vishal Sharma, Zhen Ye, Lihong Idris Lim, Lu Zhao, and Aloysius W Aryaputera. 2015. Forecasting of global horizontal irradiance by exponential smoothing, using decompositions. *Energy* 81 (2015), 111–119.

A APPENDIX

A.1 Some Implementations in Algorithm 1

.

Matrix Vector Multiplication

For all filter matrices used in this paper, it is unnecessary either to store them in memory (even in sparse form) or to perform the usual matrix multiplication operation. A more efficient alternative is to program the actual operations involved in such matrices. More specifically, all the matrix-vector multiplications are summarized in Python-like pseudocode as shown in following Table 5 and 6.

Table 5: Matrix Vector Multiplication of Filter Matrices

$\mathbf{Ax}$	code in Python
$\mathbf{D}_1\mathbf{x}$	`x[:-1] - x[1:]`
$\mathbf{D}_2\mathbf{x}$	`x[:-2] - 2*x[1:-1] - x[2:]`
$\mathbf{D}_T\mathbf{x}$	`x[:-T] - x[T:]`
$\mathbf{D}_T^2\mathbf{x}$	`x[:-2T] - 2*x[T:-T] - x[2T:]`
$\mathbf{M}_T\mathbf{x}$	`let z=[0, cumsum(x)]` `return z[T:] - z[:-T]`

Table 6: Matrix Vector Multiplication for Transposed Filter Matrices

$\mathbf{A}^t\mathbf{y}$	code in Python
$\mathbf{D}_1^T\mathbf{y}$	`[y[0], y[1:] - y[:-1]]`
$\mathbf{D}_2^T\mathbf{y}$	`[y[0], -2*y[0] + y[1],` `y[:-2] - 2*y[1:-1]+y[2:]]`
$\mathbf{D}_T^T\mathbf{y}$	`[y[:T], y[T:] - y[:-T]]`
$(\mathbf{D}_T^2)^T\mathbf{y}$	`[y[:T], -2*y[:T]+y[T:2T],` `y[:-2T]-2*y[T:-T]+y[2T:]]`
$\mathbf{M}_T^T\mathbf{y}$	`let z=[0, cumsum(y)]` `return [cumsum(x[:T]);z[T:]-z[:-T]]`

Closed-form Solution for y Update in Line 5

Since $f(y) := \sum_{i=1}^k f_i(y_i)$ and $\rho/2\,\|\mathbf{Ax}_{t+1} - \mathbf{y}\|_2^2 :=$ $\rho/2\sum_{i=1}^k \|\mathbf{A}_i\mathbf{x}_{t+1} - \mathbf{y}_i\|_2^2$, we can obtain $\mathbf{y}_{t+1}$ by updating those separable blocks $\{\mathbf{y}_i\}$ in parallel. More specifically, if $f_i(\mathbf{y}_i) = 1/2\,\|y_i - \mathbf{b}_i\|_2^2$, then the closed-form solution is

$$\mathbf{y}_{t+1,i} = \frac{1}{1+\rho}(\mathbf{b}_i + \mathbf{u}_t + \rho\mathbf{A}_i\mathbf{x}_{t+1}).$$

As for l_1 loss function, $f_i(\mathbf{y}_i) = \|\mathbf{y}_i - \mathbf{b}_i\|_1$, the closed form solution is slightly more complicated. To avoid double sub-indexing, we drop the i block index when describing the solution,

$$\mathbf{y}_{t+1,j} = \begin{cases} \mathbf{b}_j, & \text{if } |(\frac{1}{\rho}\mathbf{u}_t + \mathbf{Ax}_{t+1} - \mathbf{b})_j| < \frac{1}{\rho} \\ (\frac{1}{\rho}\mathbf{u}_t + \mathbf{Ax}_{t+1})_j - \frac{1}{\rho}, & \text{if } (\frac{1}{\rho}\mathbf{u}_t + \mathbf{Ax}_{t+1} - \mathbf{b})_j \geq \frac{1}{\rho} \\ (\frac{1}{\rho}\mathbf{u}_t + \mathbf{Ax}_{t+1})_j + \frac{1}{\rho}, & \text{if } (\frac{1}{\rho}\mathbf{u}_t + \mathbf{Ax}_{t+1} - \mathbf{b})_j \leq -\frac{1}{\rho}, \end{cases}$$

where j element index for vector $\mathbf{y}_{t+1,i}$.

A.2 Proof of Theorem 4.1

Proof. For convenience, we consider a saddle point reformulation of Eq. (18) as

$$\min_{\mathbf{x}\in X}\max_{\mathbf{u}}\{\mathcal{L}(\mathbf{x},\mathbf{u}) := \langle \mathbf{Ax},\mathbf{u}\rangle - f^*(\mathbf{u})\}, \tag{21}$$

where $f^*(\mathbf{u})$ is the Fenchel conjugate to f. As is standard in saddle point algorithm analysis, we will show convergence of the gap function $Q(\bar{\mathbf{z}}_\mathcal{T}, z)$ for some feasible $\mathbf{z} := (\mathbf{x},\mathbf{u})$ in the following proposition A.1, where

$$Q(\bar{\mathbf{z}},\mathbf{z}) := (\langle \mathbf{A}\bar{\mathbf{x}},\mathbf{u}\rangle - f^*(\mathbf{u})) - (\langle A\mathbf{x},\bar{\mathbf{u}}\rangle - f^*(\bar{\mathbf{u}})).$$

Notice that $\bar{\mathbf{z}}$ is saddle point to Eq. (21) if and only if $Q(\bar{\mathbf{z}},\mathbf{z}) \leq 0$ for any feasible $\mathbf{z}$.

Proposition A.1. *Let $\bar{\mathbf{z}}_\mathcal{T} = (\bar{\mathbf{x}}_\mathcal{T}, \bar{\mathbf{u}}_\mathcal{T})$ be generated by Algorithm 1 with some initial $\mathbf{u}_0$, $\mathbf{x}_0$ and $\mathbf{G} \geq \mathbf{A}^T\mathbf{A}$. Then for any feasible $\mathbf{z} := (\mathbf{x},\mathbf{u})$, we have*

$$Q(\bar{\mathbf{z}}_\mathcal{T},\mathbf{z}) \leq \frac{1}{\mathcal{T}}(\frac{1}{2\rho}\|\mathbf{u}_0 - \mathbf{u}\|^2 + \frac{\rho}{2}\left\|\sqrt{\mathbf{G}}\mathbf{x}_0 - \mathbf{x}\right\|^2). \tag{22}$$

First, let's fix a $t \geq 0$ and show a bound for $Q(\mathbf{z}_t,\mathbf{z})$ in each iteration. Define a extrapolated dual variable $\tilde{\mathbf{u}}_t := \mathbf{u}_t + (\mathbf{u}_t - \mathbf{u}_{t-1})$, which satisfies

$$\tilde{\mathbf{u}}_t - \rho\mathbf{Ax}_t \overset{(a)}{=} \mathbf{u}_t + \rho(\mathbf{Ax}_t - \mathbf{y}_t) - \rho\mathbf{Ax}_t = \mathbf{u}_t - \rho\mathbf{y}_t, \tag{23}$$

where (a) follows from Line 6 in Algorithm 1. Next, from the first order optimality condition of Line 4 in Algorithm 1, we have

$$\begin{aligned} &\mathbf{A}^T\mathbf{u}_t + \rho\mathbf{A}^T\mathbf{Ax}_{t+1} - \rho\mathbf{A}^T\mathbf{y}_t + \rho\hat{\mathbf{H}}(\mathbf{x}_{t+1} - \mathbf{x}_t) = 0 \\ \overset{(b)}{\Longleftrightarrow}\ &\mathbf{A}^T\tilde{\mathbf{u}}_t + \rho\mathbf{A}^T\mathbf{A}(\mathbf{x}_{t+1} - \mathbf{x}_t) + \rho\hat{\mathbf{H}}(\mathbf{x}_{t+1} - \mathbf{x}_t) = 0 \\ \overset{(c)}{\Longleftrightarrow}\ &\mathbf{A}^T\tilde{\mathbf{u}}_t + \rho\mathbf{G}(\mathbf{x}_{t+1} - \mathbf{x}_t) = 0 \\ \overset{(d)}{\Longrightarrow}\ &\langle \mathbf{Ax}_{t+1} - \mathbf{x}, \tilde{\mathbf{u}}_t\rangle \\ &\leq \frac{\rho}{2}\|\mathbf{x}_t - \mathbf{x}\|_\mathbf{G}^2 - \frac{\rho}{2}\|\mathbf{x}_{t+1} - \mathbf{x}\|_\mathbf{G}^2 - \frac{\rho}{2}\|\mathbf{x}_{t+1} - \mathbf{x}_t\|_\mathbf{G}^2, \end{aligned} \tag{24}$$

where (b) follows from (23), (c) follows from the definition of $\mathbf{G} := \hat{\mathbf{H}} + \mathbf{A}^T\mathbf{A}$ and (d) follows from multiplying both sides with $\mathbf{x}_{t+1} - \mathbf{x}$ and some simple algebraic manipulation of $\langle \mathbf{G}(\mathbf{x}_{t+1} - \mathbf{x}_t), \mathbf{x}_{t+1} - \mathbf{x}\rangle$ into quadratic terms.

For the $\mathbf{y}$ prox update step in Line 5, again it follows from the first order optimality condition that

$$\begin{aligned} &\mathbf{u}_t + \rho(\mathbf{Ax}_{t+1} - \mathbf{y}_{t+1}) \in \partial f(\mathbf{y}_{t+1}) \\ \overset{(e)}{\Longleftrightarrow}\ &\mathbf{u}_{t+1} \in \partial f(\mathbf{y}_{t+1}) \overset{(f)}{\Longleftrightarrow} \mathbf{y}_{t+1} \in \partial f^*(\mathbf{u}_{t+1}) \\ \overset{(g)}{\Longleftrightarrow}\ &\mathbf{Ax}_{t+1} - \frac{1}{\rho}(\mathbf{u}_t - \mathbf{u}_{t+1}) \in \partial f^*(\mathbf{u}_{t+1}) \\ \overset{(h)}{\Longrightarrow}\ &\langle \mathbf{Ax}_{t+1}, \mathbf{u} - \mathbf{u}_{t+1}\rangle + f^*(\mathbf{u}_{t+1}) - f^*(\mathbf{u}) \\ &\leq \frac{1}{2\rho}\|\mathbf{u} - \mathbf{u}_t\|_2^2 - \frac{1}{2\rho}\|\mathbf{u}_{t+1} - \mathbf{u}_t\|_2^2 - \frac{1}{2\rho}\|\mathbf{u} - \mathbf{u}_{t+1}\|_2^2, \end{aligned} \tag{25}$$

where (e) follows form Line 6 of Algorithm 1, (f) follows from the properties of sub-differential of conjugate function, (g) follows from

Line 6 again, and (h) follows from the definition of subdifferential and a similar algebraic manipulation of $\langle \mathbf{u}_t - \mathbf{u}_{t+1}, \mathbf{u}_{t+1} - \mathbf{u} \rangle$.

Now summing up (24) and (25), we get

$$\begin{aligned} Q(\mathbf{z}_{t+1}, \mathbf{z}) \leq & \langle \mathbf{A}(\mathbf{x}_{t+1} - \mathbf{x}), \mathbf{u}_{t+1} - \mathbf{u}_t - (\mathbf{u}_t - \mathbf{u}_{t-1}) \rangle \\ & + \frac{\rho}{2} \|\mathbf{x}_t - \mathbf{x}\|_{\mathbf{G}}^2 - \frac{\rho}{2} \|\mathbf{x}_{t+1} - \mathbf{x}\|_{\mathbf{G}}^2 - \frac{\rho}{2} \|\mathbf{x}_{t+1} - \mathbf{x}_t\|_{\mathbf{G}}^2 \\ & + \frac{1}{2\rho} \|\mathbf{u} - \mathbf{u}_t\|_2^2 - \frac{1}{2\rho} \|\mathbf{u}_{t+1} - \mathbf{u}_t\|_2^2 - \frac{1}{2\rho} \|\mathbf{u} - \mathbf{u}_{t+1}\|_2^2 . \end{aligned} \tag{26}$$

Finally, by noting that $-\frac{1}{2\rho} \|\mathbf{u}_t - \mathbf{u}_{t-1}\|_2^2 - \frac{\rho}{2} \|\mathbf{x}_{t+1} - \mathbf{x}_t\|_{\mathbf{G}}^2 + \langle A(\mathbf{x}_{t+1} - \mathbf{x}_t), \mathbf{u}_t - \mathbf{u}_{t-1} \rangle \leq 0$ and following primal dual method's analysis in [12], we can use telescoping cancellation $\sum_{t=0}^{\mathcal{T}-1} Q(\mathbf{z}_{t+1}, \mathbf{z})$ to obtain (22).

Next, we are ready to show that the convergence of the gap function could be converted to the optimality gap of (18) as the Theorem 4.1. Since f^* is the conjugate function of f, we have the inequality $f(\mathbf{A}\mathbf{x}) \geq \langle \mathbf{A}\mathbf{x}, \mathbf{u} \rangle - f^*(\mathbf{u})$, where the equality holds iff $\mathbf{u} \in \partial f(\mathbf{A}\mathbf{x})$. Thus with the choice of $\mathbf{u}^*$, we have

$$\begin{aligned} f(\mathbf{A}\bar{\mathbf{x}}_{\mathcal{T}}) &= \langle \mathbf{A}\bar{\mathbf{x}}_{\mathcal{T}}, \mathbf{u}^* \rangle - f^*(\mathbf{u}^*), \\ f(\mathbf{A}\mathbf{x}^*) &\geq \langle \mathbf{A}\mathbf{x}^*, \bar{\mathbf{u}}_{\mathcal{T}} \rangle - f^*(\bar{\mathbf{u}}_{\mathcal{T}}). \end{aligned}$$

Then, taking the difference of the previous two inequalities and referring to Proposition A.1, we get

$$f(\mathbf{A}\bar{\mathbf{x}}_{\mathcal{T}}) - f(\mathbf{x}^*) \leq Q(\bar{\mathbf{z}}_{\mathcal{T}}, \mathbf{z}^*) \leq \frac{1}{\mathcal{T}}\Big(\frac{1}{2\rho} \|\mathbf{u}_0 - \mathbf{u}\|_2^2 + \frac{\rho}{2} \left\|\sqrt{\mathbf{G}}\mathbf{x}_0 - \mathbf{x}\right\|_2^2\Big).$$

Finally, plugging in $\rho = \|\mathbf{u}_0 - \mathbf{u}^*\|_2 \Big/ \left\|\sqrt{\mathbf{G}}(\mathbf{x}_0 - \mathbf{x}^*)\right\|_2$, we get the convergence rate as in Theorem 4.1. □