Signal Decomposition via Quadratic-Separable Optimization

Luke Volpatti Bennet E. Meyers Stephen P. Boyd

December 27, 2022

Abstract

We consider the problem of decomposing a signal with missing entries into a sum of components, using the optimization formulation described by Meyers and Boyd [MB22]. We propose a general form for the component loss functions, and an alternative algorithm based on quadratic-separable optimization. This yields a flexible and extensible method for formulating and solving signal decomposition problems.

1 Introduction

1.1 Signal decomposition via optimization

In signal decomposition we decompose a given signal (possibly with some missing entries) into a sum of components, each with specific characteristics. We follow the formulation of [MB22], where the components are specified via loss functions (possibly including constraints) and the signal decomposition is carried out by minimizing the sum of the losses, subject to the constraint that the sum of the components equals the given signal on the known entries.

We adopt the notation of [MB22], which we review here. We consider a scalar signal $y \in (\mathbf{R} \cup \{?\})^T$ with some entries possibly missing. We use the value ? to denote missing entries, and say that entry i is known if $y_t \in \mathbf{R}$. We define $\mathcal{K}$ as the set of indices corresponding to known entries of y, *i.e.*, $\mathcal{K} = \{t \mid y_t \in \mathbf{R}\}$. We denote the K components into which we decompose y as $x^1, \ldots, x^K$, each in $\mathbf{R}^T$ (and with no missing entries). We require that the components sum to y on its known entries, *i.e.*,

$$y_t = x_t^1 + \cdots + x_t^K, \quad t \in \mathcal{K}.$$

The component classes are defined via loss functions $\phi_k : \mathbf{R}^T \to \mathbf{R} \cup \{\infty\}$, where $\phi_k(x^k)$ gives the implausibility or undesirability of the choice x^k (with infinite values corresponding to constraints). Following a long tradition, the signal decomposition is found as a solution or approximate solution of the signal decomposition (SD) optimization problem

$$\begin{array}{ll} \text{minimize} & \phi_1(x^1) + \cdots + \phi_K(x^K) \\ \text{subject to} & y_t = x_t^1 + \cdots + x_t^K, \quad t \in \mathcal{K}, \end{array} \tag{1}$$

with variables $x^1, \ldots, x^K$. When all of the component losses ϕ_k are convex, the SD problem is a convex optimization problem [BV09].

In [MB22] the authors give two methods for solving the SD problem (1) when it is convex, and approximately solving it when it is not. Both methods rely on evaluations of the so-called masked proximal operators of the loss functions. For many simple component classes, these can be worked out analytically or computed efficiently. For such signal decomposition problems (of which there are many), the methods described in that paper are very effective.

For more complex component classes, evaluating the masked proximal operator requires numerically solving an optimization problem, which is inconvenient, and also considerably slows the methods since the optimization problem must be solved in each iteration of the overall algorithm. In addition to slowing down the overall algorithm, it puts the burden on the user to work out and implement the masked proximal operator.

As a very simple example, consider the class of (quadratic) smooth signals bounded by one, with loss function

$$\phi(x) = \begin{cases} \sum_{t=2}^{T-1} (x_{t+1} - 2x_t + x_{t-1})^2 & \|x\|_\infty \leq 1 \\ \infty & \text{otherwise.} \end{cases} \tag{2}$$

Implementing the masked proximal operator requires solving a quadratic program (QP). This QP must be solved in every iteration of the algorithms given in [MB22]. Note that the two parts of this component class, quadratic smoothness and boundedness, are each readily handled, with simple analytic masked proximal operators. The masked proximal operator of the combination, however, is not easy to evaluate.

1.2 This paper

In this paper we address the issue described above, *i.e.*, the implementation challenge and inefficiency when a component class is more complex. We describe a method that solves exactly the same problem (1), but avoids the use of the masked proximal operator. The method makes it straightforward to work with complex component classes, such as the example above. While the methods of [MB22] are extensible in principle, they require an implementation of the masked proximal operator. The method described in this paper avoids this, and is conveniently extensible.

Our method is based on a specific form for the losses, which is partial minimization of a quadratic-separable function (explained in detail below). This form is very general, and covers a very wide variety of useful loss functions. Moreover it is conveniently extensible; we can easily handle new loss functions that include multiple objective terms and constraints.

Using this specific form for the loss functions, we assemble the SD problem into a large optimization problem with a specific form we call quadratic-separable (QS). Such problems have an objective that is the sum of a convex quadratic function and a separable (not necessarily convex) function, and linear equality constraints. We give an operator splitting method for solving such problems when they are convex, or approximately solving such problems, when they are not. Our method requires only the evaluation of the proximal operators of scalar functions, which is straightforward, and can be implemented in a library, making it very easy for a user to specify even a complex component class without low level programming.

We have implemented our method in an open-source software package which supports simple and natural descriptions of the component loss functions, and handles all the rest. It is available at `https://github.com/cvxgrp/signal-decomposition`. Our quadratic-separable solver is included as a separate package called QSS.

1.3 Related work

Using optimization as a framework for solving signal decomposition problems has an extensive history, discussed in detail in [MB22, §3], so we do not repeat it here.

There is a large literature on solution methods for solving quadratic-separable and related problems using operator splitting methods. POGS [FB18] solves graph form problems [PB14], to which quadratic-separable problems can be reduced. Like QSS, POGS implements a library of separable functions and is based on the alternating direction method of multipliers (ADMM) [BPC+11].

Another problem form equivalent to the quadratic-separable problems is studied in the context of portfolio optimization in [MGBK21]. The authors consider separable-affine problems, *i.e.*, problems with a separable objective function and affine equality constraints. They implement their method in a software package called SeparableOptimization.jl, again based on ADMM. The package allows users to specify a piecewise-quadratic separable objective. These piecewise-quadratics are allowed to be nonconvex, in which case ADMM is a heuristic solution algorithm. Quadratic-separable problems include QP as a special case. The popular software package OSQP [SBG+20] uses ADMM to solve QPs. Apart from [MGBK21], the use of ADMM as a heuristic to solve nonconvex problems has been described in a number of other works, see, *e.g.*, [DTB18, KK18]. For optimization via abstract linear operators, see *e.g.*, [DB16].

In some respects, QSS differs from the work above, and in others it draws from and unifies a number of the techniques they consider in order to efficiently solve large-scale QS problems. The split performed in QSS, described below, is ideal for QS problems as it allows for instant one-iteration termination when the problem under consideration is simply an equality-constrained quadratic program. Additionally, QSS allows for the solution to nonconvex problems and problems involving abstract linear operators, both of interest in the context of signal decomposition.

1.4 Outline

In §2 we describe quadratic-separable functions, quadratic-separable problems, our specific general form for component losses, and the SD problem that results. In §3 we give a generic method for solving such problems when they are convex, and approximately solving them when they are not. In §4 we describe the software implementation for the methods described in this paper, focusing on how extensible signal classes are expressed. We give some numerical examples in §5.

2 Quadratic-separable formulation

2.1 Quadratic-separable optimization problems

A convex quadratic function $f : \mathbf{R}^n \to \mathbf{R}$ has the form

$$f(w) = (1/2)w^T P w + q^T w + r,$$

where P is a symmetric positive semidefinite $n \times n$ matrix, $q \in \mathbf{R}^n$, and $r \in \mathbf{R}$. A function $g : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$ is separable if it is a sum of functions of the components, *i.e.*, it has the form

$$g(w) = \sum_{i=1}^{n} g_i(w_i),$$

where $g_i : \mathbf{R} \to \mathbf{R} \cup \{\infty\}$. We do not assume that g_i (and therefore g) are convex. A *quadratic-separable* (QS) function is the sum of a convex quadratic and a separable function, *i.e.*, $f(w) + g(w)$. It is convex if g_i are all convex.

A *quadratic-separable optimization problem* has a QS objective and linear equality constraints, *i.e.*,

$$\begin{array}{ll} \text{minimize} & f(w) + g(w) \\ \text{subject to} & Aw = b, \end{array} \tag{3}$$

with variable $w \in \mathbf{R}^n$. A QS problem is specified by the coefficients of the quadratic (P, q, and r), the component functions $g_1, \ldots, g_n$, the equality constraint coefficient matrix $A \in \mathbf{R}^{p \times n}$, and the equality constraint righthand side $b \in \mathbf{R}^p$. When the functions g_i are convex, the QS problem is convex.

2.2 Quadratic-separable losses with latent variables

We will assume that the component losses ϕ_k in our signal decomposition problem are expressible as the partial minimization of QS functions. This means they have the form

$$\phi_k(x^k) = \inf_{z^k} \left\{ f_k(x^k, z^k) + g_k(x^k, z^k) \mid A_k x^k + B_k z^k = c_k \right\}, \quad k = 1 \ldots, K, \tag{4}$$

where $f_k : \mathbf{R}^T \times \mathbf{R}^{n_k}$ are convex quadratic, and $g_k : \mathbf{R}^T \to \mathbf{R} \cup \{\infty\}$ are separable, $k = 1, \ldots, K$, $A_k \in \mathbf{R}^{p_k \times T}$, $B_k \in \mathbf{R}^{p_k \times n_k}$, and $c_k \in \mathbf{R}^{p_k}$. We refer to $z^k \in \mathbf{R}^{n_k}$ as the latent variable in the loss function. We can have $n_k = 0$, *i.e.*, no latent variables, or $p_k = 0$, *i.e.*, no equality constraints. We observe that ϕ_k is the partial minimization of the QS function $f_k + g_k$ subject to the equality constraints (see [BV09, §3.4.4]). Partial minimization preserves convexity, *i.e.*, if g_k is convex, so is ϕ_k.

Example. In §1.1 we considered the class of bounded quadratic-smooth signals with loss function (2). This loss can be expressed in the form (4) by introducing the latent variable $z = x$ (*i.e.*, $A = I$, $B = -I$, and $c = 0$), and defining

$$f(x,z) = \sum_{t=2}^{T-1} (x_{t+1} - 2x_t + x_{t-1})^2, \qquad g(x,z) = \sum_{t=1}^{T} \mathcal{I}(|z_t| \leq 1),$$

where $\mathcal{I}$ denotes the indicator function, which is 0 when its argument is true and ∞ otherwise. In this example, f does not depend on z, and g does not depend on x. Finally, we note that f and g have simple, closed-form proximal operators, while the original loss function (2) does not.

2.3 Signal decomposition via quadratic-separable optimization

Using the specific loss functions (4), we can express the SD problem (1) as

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^K f_k(x^k, z^k) + \sum_{k=1}^K g_k(x^k, z^k) \\ \text{subject to} & A_k x^k + B_k z^k = c_k, \quad k = 1, \ldots, K, \\ & y_t = x_t^1 + \cdots + x_t^K, \quad t \in \mathcal{K}, \end{array} \tag{5}$$

with variables x^k, z^k, $k = 1, \ldots, K$. The first sum in the objective is convex quadratic, and the second is separable, and the constraints are all linear equality constraints, so this is a QS problem. If the functions g_k are all convex, it is convex.

3 A splitting method for quadratic-separable problems

In this section we describe an algorithm based on the alternating direction method of multipliers (ADMM) [BPC+11] for solving the QS problem (3) when it is convex, and approximately solving it when it is not. An open-source implementation of the method called QSS (QS solver) is available at `https://github.com/cvxgrp/qss`. QSS is used in our implementation of the SD algorithm, but is independent of it.

3.1 Consensus form ADMM

With the introduction of auxiliary variable $\widetilde{w} \in \mathbf{R}^n$, the original problem (3) can be written as

$$\begin{array}{ll} \text{minimize} & f(w) + g(\widetilde{w}) + \mathcal{I}(Aw = b) \\ \text{subject to} & w = \widetilde{w}, \end{array} \tag{6}$$

with variables w and $\widetilde{w}$. The problem above is sometimes referred to as consensus form, in which a variable is replicated, and a constraint added that the two versions of the variable should be equal; the constraint $w = \widetilde{w}$ is called the consensus constraint.

The ADMM updates for this problem are

$$
\begin{aligned}
w^{k+1} &= \operatorname*{argmin}_{w:Aw=b} \left(f(w) + \|(1/2)R_k^{1/2}(w - \widetilde{w}^k + u^k)\|_2^2 \right) && (7)\\
\widetilde{w}^{k+1} &= \operatorname*{argmin}_{\widetilde{w}} \left(g(\widetilde{w}) + \|(1/2)R_k^{1/2}(\alpha w^{k+1} + (1-\alpha)\widetilde{w}^k - w + u^k)\|_2^2 \right) && (8)\\
u^{k+1} &= R_{k-1}R_k^{-1}u^k + \alpha w^{k+1} + (1-\alpha)\widetilde{w}^k - \widetilde{w}^{k+1}, && (9)
\end{aligned}
$$

where $u^k \in \mathbf{R}^n$ is an approximation of the scaled dual variable associated with the consensus constraint $w = \widetilde{w}$, and $R_k = \mathbf{diag}(\rho^k)$ with $\rho^k \in \mathbf{R}^n_{++}$, and $\alpha \in (0,2)$ are algorithm parameters. We can interpret the inverse of R_k as a step size, which can be different for each component. We recover an estimate of the unscaled dual variable associated with the constraint as $y^k = R_{k-1}u^k$. We will explain below how ρ^k is updated; values of α such as 1.4 or 1.9 work well for convex problems, whereas smaller values such as 0.7 or 1.0 can work better for nonconvex problems.

Generic ADMM convergence analysis tells us that the algorithm will converge to an optimal point if the problem is convex and has a solution, and ρ^k is constant [BPC+11]. The same analysis holds if ρ^k is eventually constant, which is the case for our method. If the original problem is not convex, there are no convergence guarantees, but ADMM-based algorithms have been observed to often converge to good points, *i.e.*, ones with small if not optimal objective value. For nonconvex problems a common technique is to run the algorithm from multiple random starting points and take the best approximate solution found among these.

We now give more detail about how to efficiently carry out the w-update step (7) and the $\widetilde{w}$-update step (8); the dual update step (9) is straightforward.

3.2 Updating w

The w update (7) requires the solution of an equality-constrained convex quadratic problem, which can be done by solving the associated optimality (KKT) conditions, which are linear:

$$
\begin{bmatrix} P + R_k & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} w^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} R_k(\widetilde{w}^k - u^k) - q \\ b \end{bmatrix}. \qquad (10)
$$

Here $\nu^{k+1} \in \mathbf{R}^m$ is the dual variable associated with $Aw = b$ (and not used in the ADMM algorithm). We can solve this system using either a direct method, based on factorization of the coefficient matrix, or via an indirect or iterative method. The line between these two approaches is not sharp; for example we can use a factorization of the coefficient matrix, possibly simplified, as a pre-conditioner for an iterative method.

With a direct method, we can use factorization caching (see [BPC+11, §4.2.3]), when ρ^k does not change. In this case only the right hand side of this linear system changes between iterations; the coefficient (KKT) matrix remains the same. So we factor it once, using, say, a sparse LDL^T factorization, and then in each iteration we solve the system using back-solves, which are cheaper [BV09, App. C]. To enhance stability, we add a small regularization term

$-\epsilon I$ to the 2, 2 block of the coefficient matrix, and then use iterative refinement to obtain an accurate solution, as is done in *e.g.*, [MB12, OSB13]. Note that any change in ρ^k will require a re-factorization, increasing the cost of that iteration.

Indirect methods like MINRES [PS75] solve the linear system (10) only accessing methods to multiply the KKT matrix and its transpose by a vector. These methods have three advantages over direct methods. First, there is no additional cost to changing or udpating ρ^k. Second, it can take advantage of warm-start, with initial guesses of w and ν. Third, the method supports problems where P or A are give as abstract linear operators, and not explicit matrices. The main disadvantage of indirect methods is the variability of solution time, which can vary substantially with problem data.

QSS uses a direct method by default, if the data is given as explicit matrices. For larger problems with matrices with dense blocks, which increase the cost of re-factorization, the optional indirect method can be much faster.

3.3 Updating $\widetilde{w}$

The $\widetilde{w}$ update (8) involves evaluating the proximal operator of g at the point $\alpha w^{k+1} + (1 - \alpha)\widetilde{w}^k + u^k$, *i.e.*,

$$\widetilde{w}^{k+1} = \mathbf{prox}_g(\alpha w^{k+1} + (1 - \alpha)\widetilde{w}^k + u^k).$$

The key insight to this work is in choosing g to make the above update easy to evaluate. Indeed with separable g, the update can be calculated in parallel across the components. This involves evaluating the proximal operators of n scalar functions. This is very efficiently done, using a library of proximal operators, as discussed in Appendix A. As we show in §5, many useful SD class costs may be expressed with these simple functions.

3.4 Updating ρ

The key to good performance of ADMM-based algorithms in practice is the choice of the step size parameter ρ^k. While many strategies have been proposed in the literature for doing this, we take advantage of our particular formulation to introduce a new ρ selection scheme that is able to significantly boost performance in practice versus other known schemes.

We update ρ on a block-by-block basis, where a block refers to a contiguous range of indices with the same g_i. Blocks (of length more than one) arise naturally in many applications. We also consider a special block that consists of all indices for which $g_i(x) = 0$. (If needed, we can permute the original variables to create blocks, each associated with the same g_i.) We use the same value of ρ within each block, but the blocks can have different values of ρ. We denote the common value of ρ^k within block b as ρ_b, $b = 1, \ldots, B$, where B is the number of blocks.

We choose each ρ_b^k to attempt to balance the (relative) consensus form primal and dual residuals (see §3.5) associated with each block. In particular, we update as follows

$$\rho_b^{k+1} = \rho_b^k \left(\frac{\|r^k_{\text{p, cons}}[b]\|_\infty / \max\{\|w^k[b]\|_\infty, \|\widetilde{w}^k[b]\|_\infty\}}{\|r^k_{\text{d, cons}}[b]\|_\infty / \|\rho_b^k u^k[b]\|_\infty} \right)^{1/2},$$

where we use $v[b]$ to represent the indices of vector v restricted to the entries associated with block b.

Updates to ρ should be performed parsimoniously, since each update necessitates a refactorization of the KKT matrix when a direct method is used. This can be achieved by not changing ρ^{k+1} unless the new value ρ_b^{k+1} associated with at least one block b is more than, say, 5 times larger or smaller than the block's previous value ρ_b^k, or by ensuring that a minimum number of iterations have passed between factorizations, perhaps on the order of the amount of time required to refactorize. Finally, we can stop updating ρ after some number of iterations. This ensures that convergence is still guaranteed, for convex problems.

3.5 Optimality conditions and stopping criteria

The optimality conditions for the original problem (3) are primal feasibility $Aw = b$ and dual feasibility

$$0 = Pw + q + A^T\nu + h,$$

where ν is the dual variable associated with the $Aw = b$ constraint and $h \in \partial g(w)$. From these we define primal and dual residuals for the original problem as

$$r^k_{\text{p, orig}} = A\tilde{w}^k - b, \qquad r^k_{\text{d, orig}} = P\tilde{w}^k + q + A^T\nu^k + R_k u^k,$$

noting that $R_k u^k \in \partial g(\tilde{w}^k)$ when $\alpha = 1$.

The associated primal and dual residuals of the consensus form of the problem (6) at iteration k are

$$r^k_{\text{p, cons}} = w^k - \tilde{w}^k, \qquad r^k_{\text{d, cons}} = R_k(\tilde{w}^{k-1} - \tilde{w}^k).$$

We terminate when the norms of the primal and dual residuals for the original problem are below tolerances levels $\epsilon_{\text{prim}} > 0$ and $\epsilon_{\text{dual}} > 0$, that is, when

$$\|r^k_{\text{prim}}\|_\infty \le \epsilon_{\text{prim}} \quad \text{and} \quad \|r^k_{\text{dual}}\|_\infty \le \epsilon_{\text{dual}}.$$

In our case, we choose to use the infinity norm as in [SBG+20]. We take the tolerances to be

$$\begin{aligned} \epsilon_{\text{prim}} &= \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\{\|A\tilde{w}^k\|_\infty, \|b\|_\infty\}, \\ \epsilon_{\text{dual}} &= \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\{\|P\tilde{w}^k\|_\infty, \|q\|_\infty, \|A^T\nu^k\|_\infty, \|R_k u^k\|_\infty\}, \end{aligned}$$

where ϵ_{abs} and ϵ_{dual} are user-defined absolute and relative tolerances.

3.6 Preconditioning via initial diagonal scaling

The convergence of ADMM in practice is known to benefit from diagonal scaling, which can be effective in reducing the condition number of the matrix in the KKT system (10); see, *e.g.*, [BPC+11, SBG+20, OCPB16]. In our setting, this involves choosing a diagonal matrix with positive entries

$$S = \begin{bmatrix} D & \\ & E \end{bmatrix},$$

$S \in \mathbf{R}^{n+m}_{++}$, so that the condition number of SMS is smaller than that of M, where $M \in \mathbf{R}^{n+m}_{+}$ is the KKT system matrix

$$M = \begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix}.$$

To achieve this we choose S by equilibrating M, which has been shown in practice to be effective in reducing its condition number of SMS [Bra10]. Matrix equilibration is the task of choosing S so that SMS has row and column norm equal to one. There are many algorithms for finding such an S, one of which we describe below. In addition to diagonal scaling, we scale the objective function by $c > 0$. This scaling results in the new problem

$$\begin{array}{ll} \text{minimize} & \bar{f}(\bar{w}) + \bar{g}(\bar{w}) \\ \text{subject to} & \bar{A}\bar{w} = \bar{b}, \end{array} \tag{11}$$

with $\bar{w} = D^{-1}w$, $\bar{f}(\bar{w}) = cf(D\bar{w})$, $\bar{g}(\bar{w}) = cg(D\bar{w})$, $\bar{A} = EAD$, and $\bar{b} = Eb$. It is readily seen that if the original f and g are quadratic and separable, respectively, then so are the scaled functions $\bar{f}$ and $\bar{g}$, so the scaled problem is also a QS problem. We perform scaling as a preconditioning step once at the beginning, then find an optimal $\bar{w}^\star$ (or approximately optimal, when the problem is nonconvex), and finally recover the optimal $w^\star$ via $w^\star = D\bar{w}^\star$.

A specific choice for diagonal scaling. We choose our diagonal scaling matrices D and E through a matrix equilibration scheme called Ruiz equilibration [Rui01], modified to also yield an objective scaling c. This process is described in algorithm 3.1, based on the scheme described in [SBG+20]. Note that M_i in the algorithm refers to the ith column of the KKT matrix M. The algorithm produces a modified problem defined by $\bar{P}$, $\bar{q}$, $\bar{A}$, and $\bar{b}$ whose KTT matrix has columns with infinity norm close to 1.

```
Algorithm 3.1 Ruiz equilibration for quadratic-separable problems
Initialize. Set $c = 1$, $S = I$, $\delta = 0$, $\bar{P} = P$, $\bar{q} = q$, $\bar{A} = A$, $\bar{b} = b$.
while $\|1 - \delta\|_\infty > \epsilon_{\text{equil}}$
    1. for $i = 1, \ldots, n+m$
           $\delta_i = 1/\sqrt{\|M_i\|_\infty}$
    2. Update $\bar{P}, \bar{q}, \bar{A}, \bar{b}$ by scaling using $\mathbf{diag}(\delta)$.
    3. $\gamma = 1/\max\{\text{mean}(\|\bar{P}_i\|_\infty), \|q\|_\infty\}$
    4. $\bar{P} = \gamma\bar{P}$, $\bar{q} = \gamma\bar{q}$
    5. $S = \mathbf{diag}(\delta)S$, $c = \gamma c$
return S, c
```

4 Implementation

4.1 SD software

Our proposed SD framework has been implemented in software at

> https://github.com/cvxgrp/signal-decomposition.

The software comes with many built-in component classes that users can use to define SD problems. Users then provide data, with which the SD software formulates the problem as a quadratic-separable optimization problem, handing it off to QSS (described below) for solution.

We list the supported component classes in Appendix B. These classes can be arbitrarily combined to form "aggregate" classes. This allows one to, *e.g.*, define a component class that penalizes the ℓ_1 norm of the first difference of the component *and* enforces the component's entries to be bounded between two values.

As an example, consider the component class described above (2) that is the sum of squared second differences with the ℓ_∞ norm of the component constrained to be less than or equal to 1. This component class can be defined in software as follows.

```
from gfosd.components import SumSquare, Inequality
component1 = Aggregate([SumSquare(diff=2), Inequality(vmax=1, vmin=-1)])
```

4.2 QSS

The general QS problem solution method described in §3 has been implemented in the open-source package QSS (Quadratic-Separable Solver), available at

> https://github.com/cvxgrp/qss.

QSS comes with a built-in library of separable functions whose proximal operators have been implemented. These functions can be weighted, scaled, and shifted. We list the functions in Appendix A. QSS additionally comes with support for solving nonconvex problems and support for constructing and solving problems involving abstract linear operators. It has been extensively tested on an array of problems from signal decomposition and other application domains.

5 Examples

In what follows, we explore the performance of our signal decomposition framework and solution method on three problems. All timings reported are for a 2017 MacBook Pro with a 2.3 GHz Intel Core i5 processor.

5.1 ℓ_1 trend filtering

The ℓ_1 trend filtering problem [KKBG09] takes the form

$$\text{minimize} \quad \tfrac{1}{2}\|y - x\|_2^2 + \lambda \|D_2 x\|_1, \tag{12}$$

where $y \in \mathbf{R}^n$ is the problem data (a time series), $x \in \mathbf{R}^n$ is the optimization variable, $\lambda \geq 0$ is a smoothing parameter, and $D_2 \in \mathbf{R}^{(n-2)\times n}$ is the second difference operator, given by

$$D_2 = \begin{bmatrix} 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 & 1 \end{bmatrix},$$

with entries not shown taken to be zero. With appropriate choice of λ, the solution to (12) will be sparse in its second difference, which is to say it will be piecewise linear. So the problem is to fit a piecewise linear function to the input time series y.

Signal decomposition model. To perform ℓ_1 trend filtering using the SD framework, we construct a scalar SD problem with $T = 100000$ and $K = 2$ component classes: mean-square small and ℓ_1 norm-penalized second order difference.

Data set. We generate a synthetic signal y of length $T = 100000$ as a sum of two 'true' signal components, one that is Gaussian noise and one that is continuous and piecewise linear with 4 kinks. The first component is denoted $\tilde{x}^1$ and has IID entries $\mathcal{N}(0, 0.2^2)$, while the second component is denoted $\tilde{x}^2$. We discard 20% of the entries of y at random, *i.e.*, we replace them with ?.

Decomposition. We perform signal decomposition with weights $70/T$ and 1 for the mean-square small and ℓ_1 norm-penalized second difference components, respectively. The decomposition returned by QSS is shown in figure 1. The components found can be seen to match the true components quite well. Indeed, the RMS (root mean-square) error of the second component x^2 is about 0.006. Additionally, the second difference of the x^2 component, plotted in figure 2, is sparse with the correct number (four) of nonzeros, which correspond to kinks in x^2. The four kink points identified are not exactly at the same points as the original component but are very close, with the highest deviation between the true and predicted kink points being 409 index entries away.

Comparison with other solution methods. While (12) is a strongly convex problem and therefore has a unique global optimum, different solvers will terminate at different apprpoximate solutions deemed acceptable as defined by their tolerance levels. As a point of comparison, we solved the same trend filtering problem using two other solution methods.

Table 1: Summary of runtimes of solution methods on the ℓ_1 trend filtering problem.

Solution method	Time (s)
QSS	6.44
OSQP	2.15
l1_tf	0.53

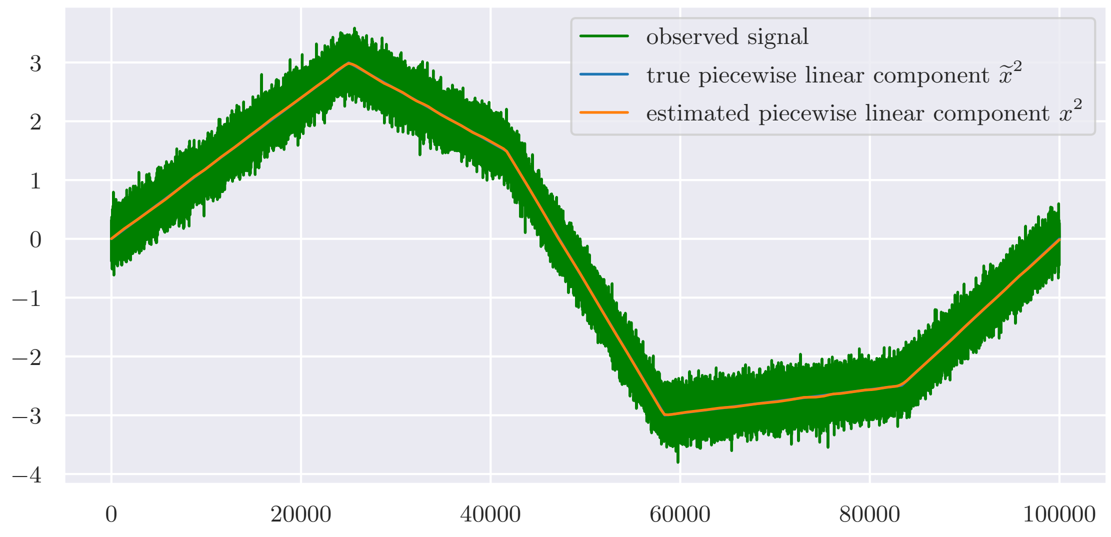

Figure 1: QSS solution to ℓ_1 trend filtering signal decomposition example. Not visible in the observed signal at this scale are the 20% of the data points that have been removed at random.

The first is OSQP [SBG+20], an ADMM-based quadratic program solver. The second is l1_tf [KKBG09], an implementation of an interior point method designed specifically for the ℓ_1 trend filtering problem. Since l1_tf cannot handle missing values, we run it on the full signal with no missing entries.

We summarize the runtimes of the three methods in table 1. Most apparent is the extremely fast runtime of l1_tf, perhaps expected as it is a low-level C implementation of a solution method designed specifically for this problem. Somewhere between QSS and l1_tf in runtime is OSQP.

It is worth noting that unlike QSS, OSQP and l1_tf were not able to produce x^2 components with appropriately sparse second differences. Indeed, even as OSQP's tolerances were tightened to increase solve time to more than a minute, the sparsity pattern it produced did not feature the four kink points one would expect upon inspection of the data, as shown in figure 2.

5.2 A simple nonconvex example

In this section, we revisit the example from §2.9 of [MB22].

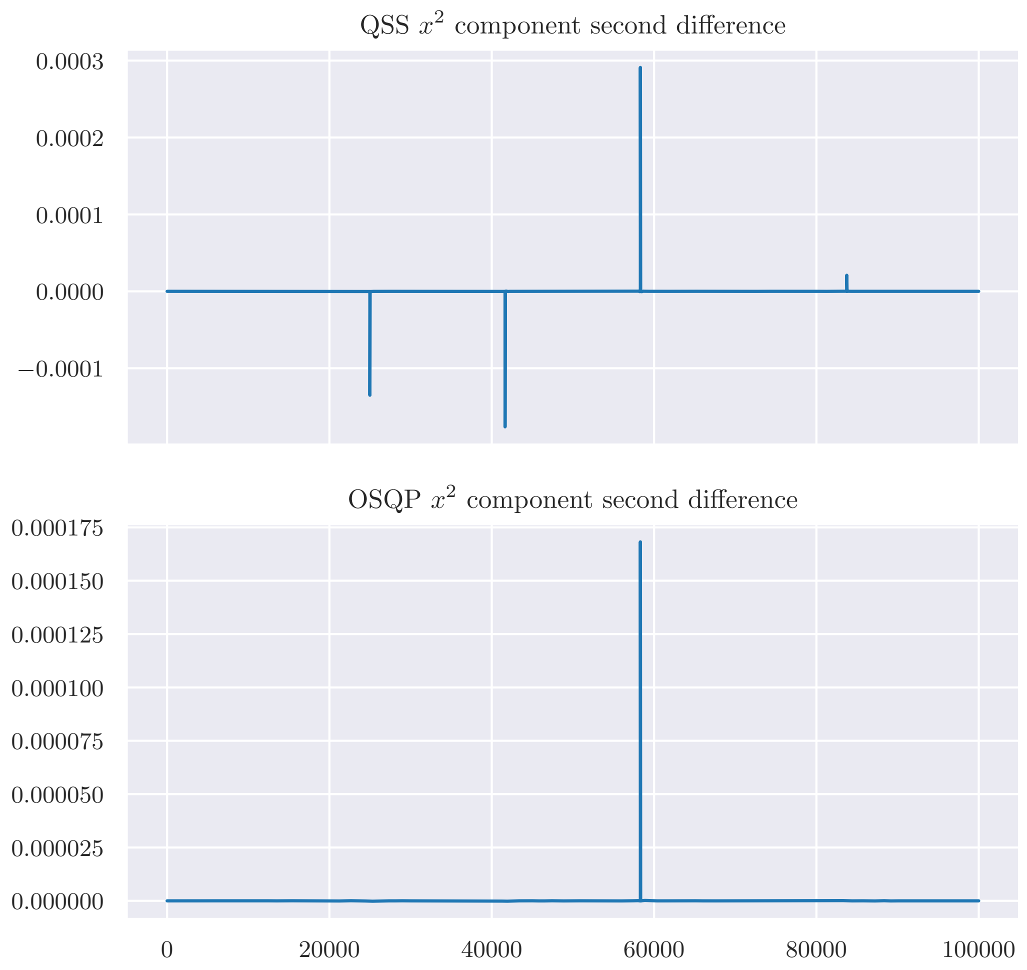

Figure 2: Second difference of x^2 components of the QSS and OSQP solutions to the ℓ_1 trend filtering signal decomposition example.

Data. We generate data as in [MB22, §2.9], which yields a scalar signal of length $T = 500$. This signal is composed of three known components, a noise term, a smooth term that is a sum of sinusoids, and a switching term.

Signal decomposition model. The example from [MB22, §2.9] concerns itself with an SD problem of length $T = 500$ with $K = 3$ component classes: mean-square small, mean-square second-order smooth (with weight parameter θ_1), and scaled Boolean. The scaled Boolean component class requires all entries of x^3 to be in $\{0, \theta_2\}$, parametrized by θ_2. As this class is not convex, this SD problem is itself not convex. In this example, we use the values for the two problem parameters, θ_1 and θ_2, chosen via a grid search in [MB22, §2.9].

Decomposition. The final decomposition generated by QSS is shown in figure 3. The components are nearly exactly recovered. This decomposition took about 2 seconds using QSS.

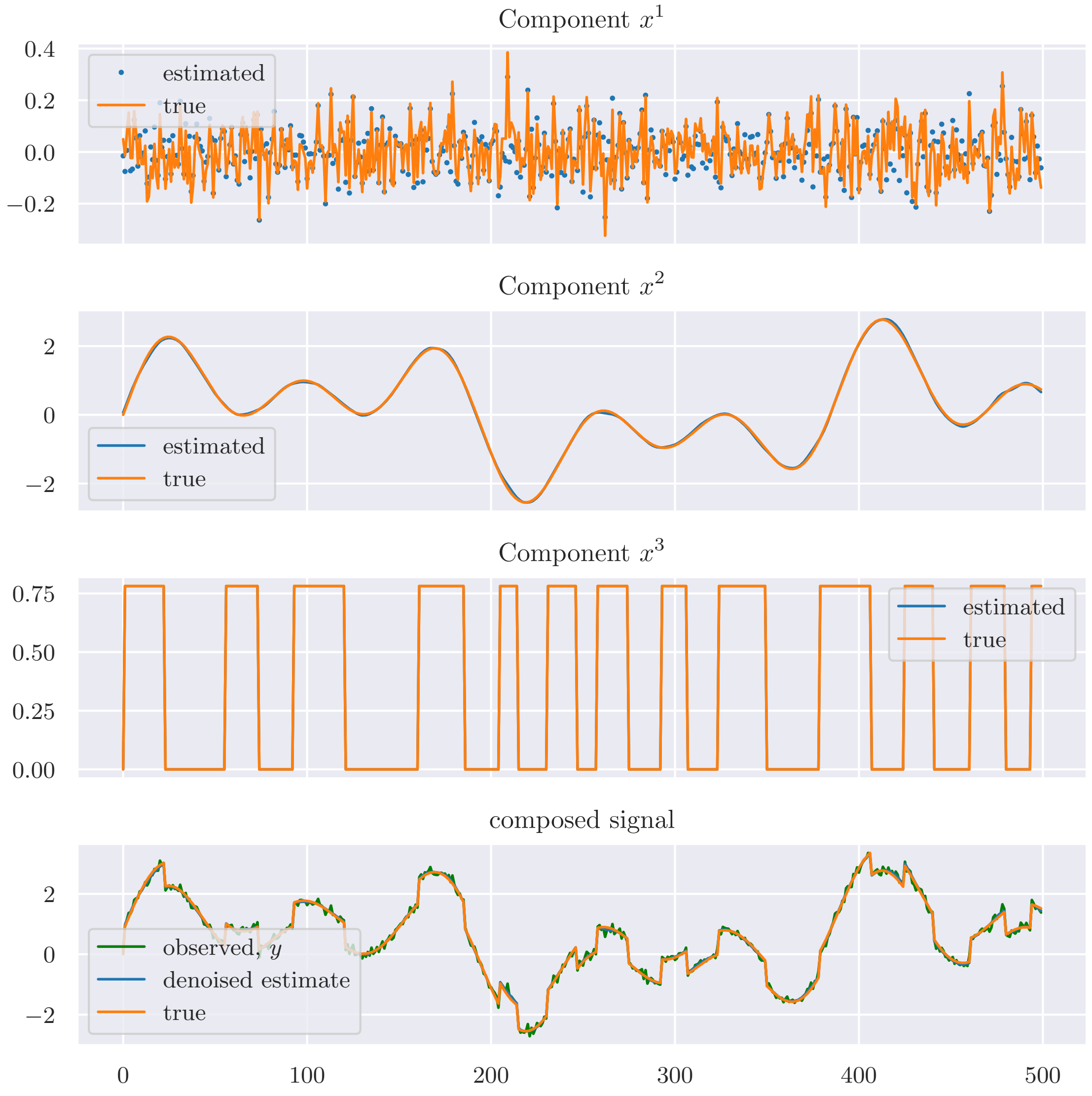

Figure 3: QSS decomposition for simple nonconvex example.

5.3 Photovoltaic soiling

Here we consider a practical problem from the domain of photo-voltaic (PV) performance analysis, the soiling estimation problem. Soiling losses in the context of PV power generation refers to the reduction of system power output due to the accumulation of dirt, soot, or other small particles on the surface of the PV modules, and soiling can result in losses as high as -1%/day [IMF+19]. The soiling estimation problem involves determining the magnitude of soiling losses over time, given measurements of photovoltaic system energy output over time (typically along with some reference data such as irradiance and temperature).

Data. Here, we consider synthetic data which models the performance index of a PV system experiencing soiling losses in addition to seasonal variability and a long-term degradation mode, as discussed in [SD20, Mey22]. We consider a single example of data generated by this model, shown in figure 4. The data are constructed by generating four components—noise, seasonal, degradation, and soiling—shown in the top four plots. These components are multiplied together to generate the observed data, shown in the bottom plot.

Data preprocessing. We first apply a log transform to the data, converting the multiplicative component model to an additive one (see [MB22, §2.8]). Then we standardize the data by applying min-max scaling to the range $[0, 10]$, a common transform available in popular packages such as `sklearn` [skl]. The range was chosen to provide a good dynamic range for the soiling and degradation components. The impact of this preprocessing on the component data model is as follows. Defining the original generative data model as

$$y = x^1 x^3 x^4 x^5,$$

where the term x^2 has been intentionally skipped (the reason for which will be clear momentarily), then the resulting data model after preprocessing is

$$\frac{c(\log y - a)}{b - a} = \frac{c}{b - a}\left(\log x^1 + \log x^3 + \log x^4 + \log x^5 - a\right),$$

where a is the minimum value of $\log y$, b is the maximum value of $\log y$, and $c = 10$ is the top of the data standardization range. We note that this has introduced an offset term on the right-hand side, which we will now include in the model as the second term. So, we let

$$\begin{aligned} \widetilde{y} &= \frac{c(\log y - a)}{b - a} \\ \widetilde{x}^k &= \frac{c}{b - a}\log x^k, \quad k = 1, 3, 4, 5 \\ \widetilde{x}^2 &= \frac{-ac}{b - c}, \end{aligned}$$

which gives us a data model in standard SD form,

$$\widetilde{y} = \widetilde{x}^1 + \widetilde{x}^2 + \widetilde{x}^3 + \widetilde{x}^4 + \widetilde{x}^5.$$

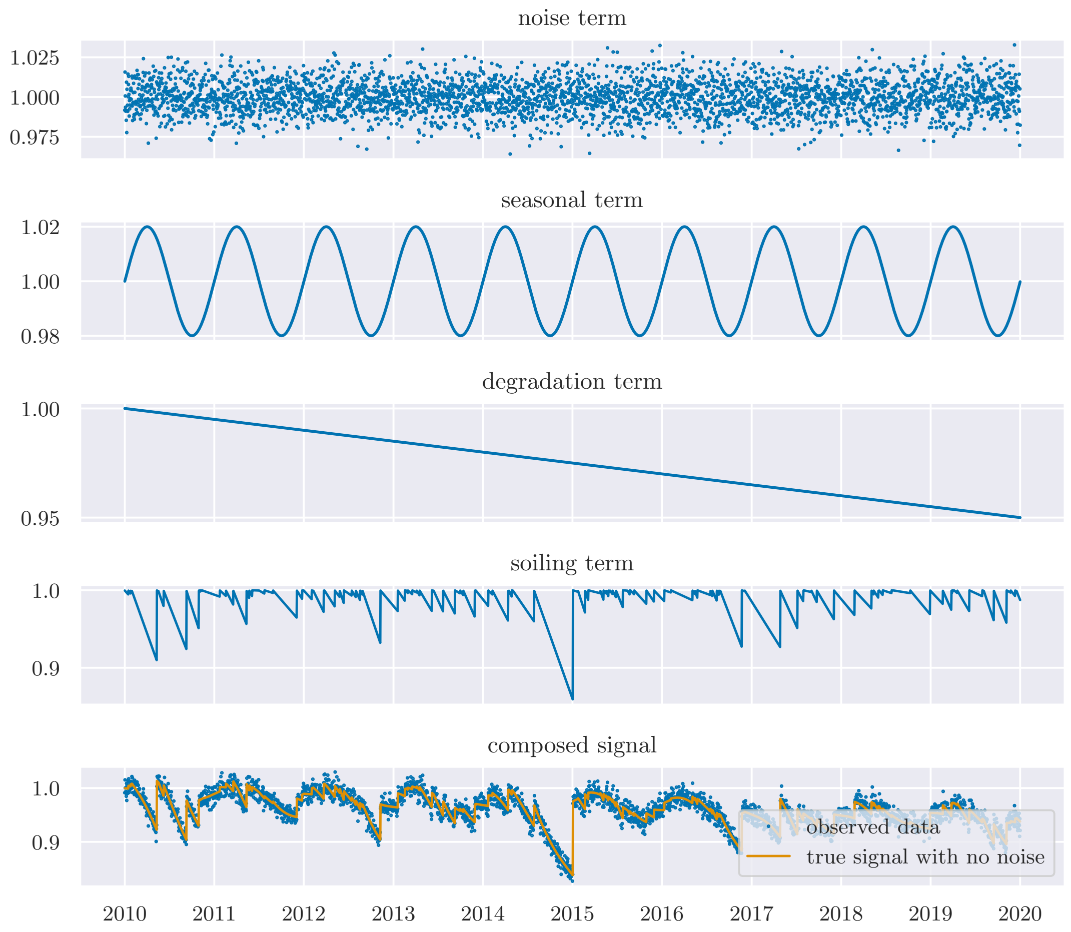

Figure 4: Synthetic soiling performance index data.

Table 2: SD component definitions for PV soiling

k	Name	$\phi_k(x)$
1	mean-square small	$(1/T)\left\|x\right\|_2^2$
2	bias	$\mathcal{I}(D_1 x = 0)$
3	linear	$\mathcal{I}(D_2 x = 0)$
4	smooth & periodic	$\lambda_3 \left\|D_2 x\right\|_2^2 + \mathcal{I}\left(x_t = x_{t+365} \text{ for } t = 1, \ldots, T-365\right)$
5	soiling	$\ell_1(x) + \ell_2(x) + \ell_3(x) + \ell_4(x)$

Table 3: PV soiling SD problem weights

term	value
λ_3	5
λ_{5a}	1×10^{-5}
λ_{5b}	5×10^{-3}
λ_{5c}	1×10^{-5}

Signal decomposition model. Following the methods section of [Mey22], we define a $K = 5$ SD model with five component class costs, ordered in terms of increasing complexity, summarized in table 2. We let D_2 be the second-order difference matrix as previously defined, D_1 be the first-order difference matrix, and λ_3 be a weight parameter. We drop the tilde and superscripts on the component variables in the interest of notational simplicity. The fifth term, which we name "soiling", is defined as a sum of four simpler loss functions,

$$\begin{aligned}
\ell_1(x) &= \mathcal{I}(x \leq 0) \\
\ell_2(x) &= \lambda_{5a} \left\|x\right\|_1 \\
\ell_3(x) &= \lambda_{5b} \sum_{t=1}^{T-1} \left[(1/2)\left|(D_1 x)_t\right| + (2/5)(D_1 x)_t\right] \\
\ell_4(x) &= \lambda_{5c} \left\|D_2 x\right\|_1,
\end{aligned}$$

with weight parameters λ_{5a}, λ_{5b}, and λ_{5c}. In order, this costs selects for signals that (ℓ_1) are non-positive, (ℓ_2) are not too large (and prefer zero values), (ℓ_3) have more negative slope values than positive, and (ℓ_4) are sparse in second-order differences. We note that, unlike the first four component cost functions, ϕ_5 does not have a closed-form solution for its proximal operator. However, it is easily expressed in quadratic-separable form. Weight parameters are chosen to provide a satisfactory decomposition but could be chosen through a holdout procedure, as described in [MB22, §2.7]. The chosen weights are summarized in table 3.

Results. We find very close agreement between the decomposition generated by QSS and the known underlying components. A comparison of the estimated and actual components is shown in figure 5. The root mean square error between the actual and estimated components are all less than 0.01, as illustrated by figure 6.

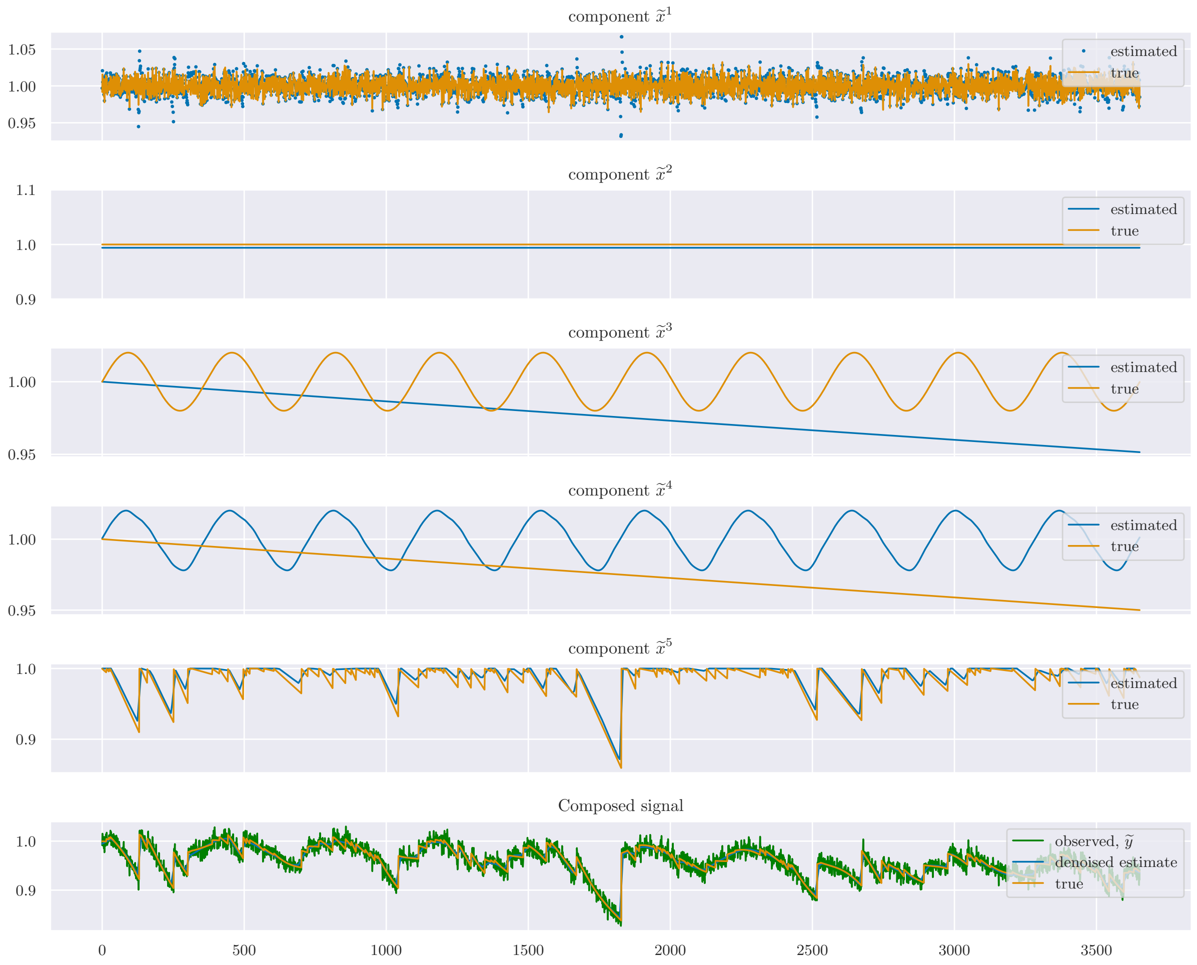

Figure 5: Estimated and true signal components in the synthetic soiling data.

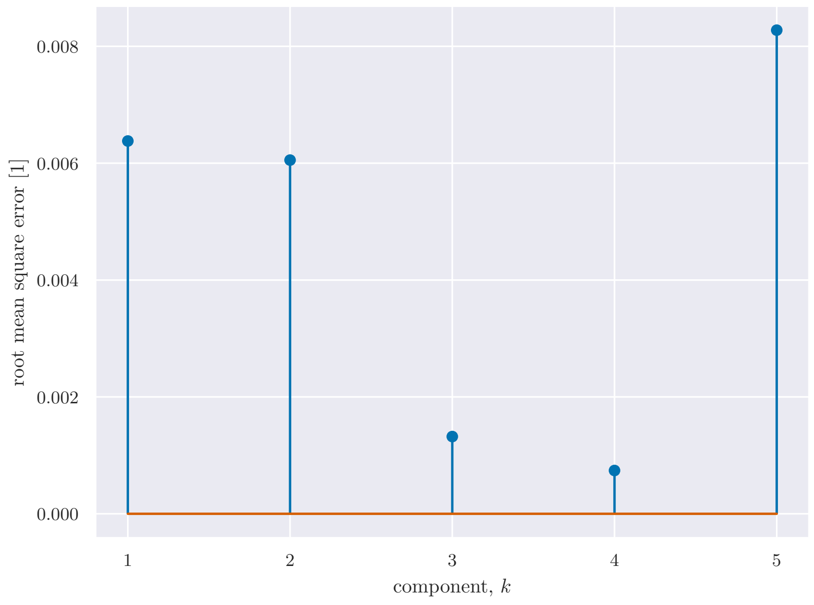

Figure 6: Root mean square error for the five component estimates.

References

[BPC+11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[Bra10] A. Bradley. *Algorithms for the equilibration of matrices and their application to limited-memory Quasi-Newton methods.* PhD thesis, 2010.

[BV09] S. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge University Press, 2009.

[DB16] S. Diamond and S. Boyd. Matrix-free convex optimization modeling. In *Optimization and its applications in control and data sciences*, pages 221–264. Springer, 2016.

[DTB18] S. Diamond, R. Takapoui, and S. Boyd. A general system for heuristic minimization of convex functions over non-convex sets. *Optimization Methods and Software*, 33(1):165–193, 2018.

[FB18] C. Fougner and S. Boyd. Parameter selection and preconditioning for a graph form solver. In *Emerging Applications of Control and Systems Theory*, pages 41–61. Springer, 2018.

[IMF+19] K. Ilse, L. Micheli, B. W. Figgis, K. Lange, D. Daßler, H. Hanifi, F. Wolfertstetter, V. Naumann, C. Hagendorf, R. Gottschalg, and J. Bagdahn. Techno-economic assessment of soiling losses and mitigation strategies for solar power generation. *Joule*, 3(10):2303–2321, 2019.

[KK18] Y. Kanno and S. Kitayama. Alternating direction method of multipliers as a simple effective heuristic for mixed-integer nonlinear optimization. *Structural and Multidisciplinary Optimization*, 58(3):1291–1295, 2018.

[KKBG09] S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky. ℓ_1 Trend Filtering. *SIAM Review*, 51(2):339–360, 2009.

[MB12] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[MB22] B. Meyers and S. Boyd. Signal decomposition using masked proximal operators. 2022.

[Mey22] B. Meyers. Estimation of soiling losses in unlabeled PV data. *Proceedings of the 49th Photovoltaic Specialists Conference*, 2022.

[MGBK21] N. Moehle, J. Gindi, S. Boyd, and M. Kochenderfer. Portfolio construction as linearly constrained separable optimization. 2021.

[OCPB16] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

[OSB13] B. O'Donoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, 21(6):2432–2442, 2013.

[PB14] N. Parikh and S. Boyd. Block splitting for distributed optimization. *Mathematical Programming Computation*, 6(1):77–102, 2014.

[PS75] C. Paige and M. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis*, 12(4):617–629, 1975.

[Rui01] D. Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. 2001.

[SBG+20] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[SD20] Å. Skomedal and M. Deceglie. Combined estimation of degradation and soiling losses in photovoltaic systems. *IEEE Journal of Photovoltaics*, 10(6):1788–1796, nov 2020.

[skl] MinMaxScaler from the Scikit Learn preprocessing module. *Scikit Learn Documentation.*

A Separable functions supported by QSS

Table 4: List of convex separable functions supported by QSS. $\mathcal{I}$ is the $\{0, \infty\}$ indicator function.

Function	Parameters	$g_i(x)$
`zero`		0
`abs`		$\lvert x\rvert$
`is_pos`		$\mathcal{I}(x \geq 0)$
`is_neg`		$\mathcal{I}(x \leq 0)$
`is_bound`	`lb`: lower bound `ub`: upper bound	$\mathcal{I}(l \leq x \leq u)$
`is_zero`		$\mathcal{I}(x = 0)$
`pos`		$\max\{x, 0\}$
`neg`		$\max\{-x, 0\}$
`quantile`	`tau`: scalar in $(0, 1)$	$0.5\lvert x\rvert + (\tau - 0.5)x$
`huber`	`M`: positive scalar	$\begin{cases} x^2 \text{ if } \lvert x\rvert \leq M, \\ 2M\lvert x\rvert - M^2 \text{ else} \end{cases}$

Table 5: List of nonconvex separable functions supported by QSS. $\mathcal{I}$ is the $\{0, \infty\}$ indicator function.

Function	Parameters	$g_i(x)$
`sqrt`		$\sqrt{\lvert x\rvert}$
`card`		0 for $x = 0$; 1 for $x \neq 0$
`is_int`		$\mathcal{I}(x \text{ is an integer})$
`is_finite_set`	`S`: list of scalars	$\mathcal{I}(x \in S)$
`is_bool`		$\mathcal{I}(x \in \{0, 1\})$

B Components classes supported by SD software

Table 6: List of convex component classes supported by SD software. $\mathcal{I}$ is the $\{0, \infty\}$ indicator function. The Huber and quantile functions are as defined in Appendix A.

Component class	Parameters	$\phi(x)$
`SumSquare`		$\sum_i (x_i)^2$
`SumAbs`		$\sum_i \lvert x_i \rvert$
`SumHuber`	`M`: positive scalar	$\sum_i \text{Huber}(x_i; M)$
`SumQuantile`	`tau`: scalar in (0, 1)	$\sum_i \text{quantile}(x_i; \tau)$
`Inequality`	`vmin`: lower bound `vmax`: upper bound	$\sum_i \mathcal{I}(v_{\text{min}} \leq x_i \leq v_{\text{max}})$
`Basis`	`B`: basis matrix	$\mathcal{I}(x = Bz \text{ for some } z)$
`Periodic`	`period`: positive integer	$\mathcal{I}(x$ is `period`-periodic$)$

Table 7: List of nonconvex component classes supported by SD software. $\mathcal{I}$ is the $\{0, \infty\}$ indicator function. The card function is as defined n Appendix A.

Component class	Parameters	$\phi(x)$
`SumCard`		$\sum_i \text{card}(x_i)$
`FiniteSet`	`values`: list of scalars	$\sum_i \mathcal{I}(x_i \in$ `values`$)$
`Boolean`		$\sum_i \mathcal{I}(x_i \in \{0, 1\})$