SSNCVX: A semismooth Newton algorithms based solver for convex composite optimization

Author One ONE@STAT.WASHINGTON.EDU
Department of Statistics
University of Washington
Seattle, WA 98195-4322, USA

Author Two TWO@CS.BERKELEY.EDU
Division of Computer Science
University of California
Berkeley, CA 94720-1776, USA

Abstract

We develop a semismooth Newton based solver to solve a class of convex composite optimization called SSNCVX. Different from many SSN based solvers which are designed case by case or only able to solve two blocks problems. The solver proposed in this paper is able to solve multi-block problems such as conic programming, Lasso type problems and quadratic programming. By dealing internally with the intrinsic nonsmooth property of $p(\boldsymbol{x})$, all the different problems are solved in a unified saddle point framework induced from augment Lagrangian strong duality which lowers the entrance barrier and hence user-friently. Such structured constraints appear pervasively in image processing and machine learging such as roubst PCA clustering problems. This software is not a single, monolithic solver; rather, it is a suite of programs and routines designed to serve as building blocks for constructing complete algorithms.

Keywords: convex composite optimization, semismooth Newton method, Matlab software package.

1 Interface

In this paper, we aim to develop a Matlab software package for the following convex composite optimization problem:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{X}} \quad & \langle \boldsymbol{c}, \boldsymbol{x} \rangle + \frac{1}{2} \langle \boldsymbol{x}, \mathcal{Q}(\boldsymbol{x}) \rangle + f(\mathcal{B}(\boldsymbol{x})) + p(\boldsymbol{x}), \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{P}_1, \quad \mathcal{A}(\boldsymbol{x}) \in \mathcal{P}_2. \end{aligned} \tag{1.1}$$

where $\boldsymbol{c} \in \mathbb{R}^n$, $\mathcal{A} : \mathbb{R}^n \to \mathbb{R}^m, \mathcal{B} : \mathbb{R}^n \to \mathbb{R}^p$ are linear operators, $f : \mathbb{R}^p \to \mathbb{R}$ is a convex function, $\mathcal{P}_1 = \{\boldsymbol{x} \in \mathbb{R}^n | \mathtt{l} \le \boldsymbol{x} \le \mathtt{u}\}$ and $\mathcal{P}_2 = \{\boldsymbol{x} \in \mathbb{R}^m | \mathtt{lb} \le \boldsymbol{x} \le \mathtt{ub}\}$, $\mathcal{Q} \in \mathbb{S}^n_+$ is a positive semidefinite matrix or operator, $p(x)$ is a convex and nonsmooth function. The choices of $p(\boldsymbol{x})$ provide flexibility to handle many kinds of problems. Some examples for (1.1) are listed subsequently. Furthermore, the $\mathcal{A}, \mathcal{B}$ and $\mathcal{Q}$ can be inputed as matrix form or operator form.

The corresponding solver is terrm as SSNCVX. The corresponding calling syntax is defined as:

```
(x,out) = SSNCVX (x0,pblk,f,Bt,Q,c,l,u,At,bl,bu,opts,y,z,v,r).
```
(1.2)

We next give a detailed introduction of the input arguments and output arguments.

1.1 Input arguments

The details of the input for `SSNCVX` is introduced in this subsection.

- `x0` (optional): the initial point, which is set as zero if not provided.
- `pblk`: The function p, can be a cone or other regularization term.
- `f`: The function f and its corresponding proximal operator. We require that either the proximal operator of $f(\mathcal{B}\boldsymbol{x})$ is differentiable or its proximal operator can be obtained easily.
- `At,Bt`: The given linear maps of $\mathcal{A}$ and $\mathcal{B}$ respectively (optional). `At` is the linear map for linear constraint, `Bt` is the linear map for f. They can be the map form or the matrix form.
- `l,u`: The box constraints that $\boldsymbol{x}$ satisfies. If $\boldsymbol{x}$ does not have the box constraint, we can set $\texttt{l} = [\,], \texttt{u} = [\,]$. We note that `l` may be a sparse matrix. For the index that is not provided, we set it as zero.
- `opts`: a structure array of parameters (optional).
- `bl,bu`: The box constraints that $\mathcal{A}(\boldsymbol{x})$ should satisfy. Similarly, if the linear map $\mathcal{A}$ is not present, one can set $\texttt{bl} = [\,], \texttt{bu} = [\,]$.
- `y,z,v,r`: other initial point for dual variables (optional).

We note that the `pblk` is a nonsmooth term. One of the `pblk` and $\texttt{l}, \texttt{u}$ must be nontrival.

1.1.1 structure of pblk

The format of the input data in `SSNCVX` is an extension of the format of conic programming.

(1) `pblk.type`: type of the nonsmooth term for $p(\boldsymbol{x})$ in (1.1).

(2) `pblk.size`: the dimension of the variable $\boldsymbol{x}$.

(3) `pblk.shift`: the shift term for the original variables $\boldsymbol{x}$. If no shift term is used, we can set $\texttt{pblk.shift} = [\,]$.

(4) `pblk.cofficient` (optional): the cofficient λ of the nonsmooth function $p(\boldsymbol{x})$. We set it as 1 if not provided.

(5) `pblk.cofficient2` (optional): other coefficient if needed of the nonsmooth function $p(\boldsymbol{x})$.

We next present some examples of `pblk` tpye. The differentiable example is marked with $*$.

- `cone:` structure for conic programing. If the k-th block `Xk` of the variable $\boldsymbol{x}$ is a nonnegative vector block with dimension n_k, the `pblk` can be represented by

$$\texttt{pblk\{k\}.type = `l'}, \texttt{pblk\{k\}.size} = n_k.$$

For semidefinite programming, the `pblk` can be represented by

$$\texttt{pblk\{k\}.type = `s'}, \texttt{pblk\{k\}.size} = n_k.$$

For SOCP, the `pblk` can be represented by

$$\texttt{pblk\{k\}.type = `q'}, \texttt{pblk\{k\}.size} = n_k.$$

- `l1:` structure for ℓ_1 norm regularizer.

$$\begin{aligned}&\texttt{pblk\{k\}.type = `l1'}, \texttt{pblk\{k\}.size} = n_k,\\ &\texttt{pblk\{k\}.coefficient} = \lambda_1.\end{aligned}$$

- `linfty:` structure for ℓ_∞ norm regularizer.

$$\begin{aligned}&\texttt{pblk\{k\}.type = `linfy'}, \texttt{pblk\{k\}.size} = n_k,\\ &\texttt{pblk\{k\}.coefficient} = \lambda_1.\end{aligned}$$

- `fused:` structure for ℓ_1 norm pule ℓ_2 regulatizer.

$$\begin{aligned}&\texttt{pblk\{k\}.type = `fused'}, \;\; \texttt{pblk\{k\}.size} = n_k,\\ &\texttt{pblk\{k\}.coefficient} = \lambda_1, \;\; \texttt{pblk\{k\}.coefficient2} = \lambda_2.\end{aligned}$$

- `l2:` structure for ℓ_2 norm regularizer.
- `nuclear:` structure for nuclear norm.
- `l1con:` structer for constraint $\|\boldsymbol{x}\|_1 < \lambda_1$.

$$\begin{aligned}&\texttt{pblk\{k\}.type = `l1con'}, \texttt{pblk\{k\}.size} = n_k,\\ &\texttt{pblk\{k\}.coefficient} = \lambda_1.\end{aligned}$$

- `linfcon:` structer for $\|\boldsymbol{x}\|_\infty < \lambda_1$.

$$\begin{aligned}&\texttt{pblk\{k\}.type = `linfcon'}, \texttt{pblk\{k\}.size} = n_k,\\ &\texttt{pblk\{k\}.coefficient} = \lambda_1.\end{aligned}$$

- `box`: structer for the box constraint $\texttt{l} \le \boldsymbol{x} \le \texttt{u}$.

$$\texttt{pblk\{k\}.type = `boxc'}, \texttt{pblk\{k\}.size} = n_k.$$

We note that the `boxc` can not appear with the linear constraint $\texttt{l} \le \boldsymbol{x} \le \texttt{u}$ at the same time.

- `square`*`:` structure for square of ℓ_2.

$$\begin{aligned}&\texttt{pblk\{k\}.type = `square'}, \texttt{pblk\{k\}.size} = n_k,\\ &\texttt{pblk\{k\}.shift = } b.\end{aligned}$$

We summerize the example of `pblk` as follows:

f	atom	f^*	$\mathrm{prox}_{\lambda p}(\boldsymbol{x})[\boldsymbol{u}]$	$\partial\mathrm{prox}_{\lambda p}(\boldsymbol{x})$ or $\nabla p^*(\boldsymbol{x})$	Assumptions
$\lambda\lVert\boldsymbol{x}\rVert^2$	`square`	$\frac{1}{4\lambda}\lVert\boldsymbol{y}\rVert^2$	-	$2\lambda\boldsymbol{x}$	-
$\lambda\sum_{i=1}^n e^{\boldsymbol{x}_i}$	`exp`	$\sum_{i=1}^n \boldsymbol{y}_i\log\boldsymbol{y}_i - \boldsymbol{y}_i$	-	$\lambda\log(\boldsymbol{x}/\lambda)$	$(\mathrm{dom}(f^*) = \mathbb{R}_+^n)$
$-\lambda\sum_{i=1}^n \log\boldsymbol{x}_i$	`nlog`	$-n - \sum_{i=1}^n \log(-\boldsymbol{y}_i)$	-	$[\frac{-\lambda}{\log(\boldsymbol{x})_1}, \cdots, \frac{-\lambda}{\log(\boldsymbol{x})_n}]$	$(\mathrm{dom}(f^*) = \mathbb{R}_+^n)$
$\lambda\log(\sum_{i=1}^n e^{\boldsymbol{x}_i})$	`exp`	$\sum_{i=1}^n \boldsymbol{y}_i\log(\boldsymbol{y}_i)$, dom $= \Delta_n$.	-	$[e^{\boldsymbol{x}_1}, \cdots, e^{\boldsymbol{x}_n}]/\sum_{i=1}^n e^{\boldsymbol{x}_i}$	$(\mathrm{dom}(f^*) = \mathbb{R}_+^n)$
$\lambda\lVert\boldsymbol{x}\rVert_1$	`l1`	$\delta_{B_{\lVert\cdot\rVert_\infty}[0,\lambda]}(\boldsymbol{y})$	$(\lvert\boldsymbol{x}\rvert - \lambda\boldsymbol{e})_+ \odot \mathrm{sgn}(\boldsymbol{x})$	$\mathrm{Diag}(\boldsymbol{u}),\ \boldsymbol{u}_i = \begin{cases} 0, & \text{if } \lvert(\boldsymbol{x})_i\rvert < \lambda, \\ 1, & \text{otherwise.} \end{cases}$	-
$\lambda\lVert\boldsymbol{x}\rVert_2$	`l2`	$\delta_{B_{\lVert\cdot\rVert_2}[0,\lambda]}(\boldsymbol{y})$	$\begin{cases} \boldsymbol{x} - \lambda\boldsymbol{x}/\lVert\boldsymbol{x}\rVert, & \text{if } \lVert\boldsymbol{x}\rVert > \lambda, \\ 0, & \text{otherwise.} \end{cases}$	$\begin{cases} I - \lambda(I - \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}/\lVert\boldsymbol{x}\rVert^2)/\lVert\boldsymbol{x}\rVert, & \text{if } \lVert\boldsymbol{x}\rVert > \lambda, \\ 0, & \text{otherwise.} \end{cases}$	-
$\lambda\lVert\boldsymbol{x}\rVert_\infty$	`linfty`	$\delta_{B_{\lVert\cdot\rVert_1}[0,\lambda]}(\boldsymbol{y})$	$\boldsymbol{x} - \lambda P_{B_{\lVert\cdot\rVert_1}[0,1]}(\boldsymbol{x}/\lambda)$	$\mathrm{Diag}(u),\ u_i = \begin{cases} 0, & \text{if } \lvert\boldsymbol{x}/\lambda\rvert > \mu_*, \text{where } \mu_* \text{ satisfy } \mathbf{1}^{\mathrm{T}}[\boldsymbol{x} - \mu_*\mathbf{1}]_+ = \mathbf{1} \\ 1, & \text{otherwise.} \end{cases}$	-
$\delta_{\mathtt{l}\le\boldsymbol{x}\le\mathtt{u}}(\boldsymbol{x})$	`box`	$\langle\mathtt{u}, \max\{\boldsymbol{x}, 0\}\rangle + \langle\mathtt{l}, \min\{\boldsymbol{x}, 0\}\rangle$	$P_{\mathtt{l}\le\boldsymbol{x}\le\mathtt{u}}(\boldsymbol{x})$	$\mathrm{Diag}(u), u_i = \begin{cases} 1, & \text{if } x_i/\lambda \in C, \\ 0, & \text{otherwise.} \end{cases}$	
$\delta_{\boldsymbol{x}\in\mathcal{K}}(\boldsymbol{x})$	`cone`	$\delta_{\mathcal{K}}(-\boldsymbol{y})$	$P_{\mathcal{K}}(\boldsymbol{x})$	Depend on $\mathcal{K}$	
$\lambda\max\{\boldsymbol{x}_i\}$	`max`	$\delta_{\Delta_n}(\boldsymbol{y})$	$\boldsymbol{x} - \lambda P_{\Delta_n}(\boldsymbol{x}/\lambda)$	$\mathrm{Diag}(\boldsymbol{u}), \boldsymbol{u}_i = \begin{cases} 0, & \text{if } \boldsymbol{x}_i/\lambda \in C, \\ 1, & \text{otherwise.} \end{cases}$	-
$\lambda\sum_{i=1}^k \boldsymbol{x}_{[i]}$	`topk`	-	$x - \lambda P_{C_{e,k}}(\boldsymbol{x}/\lambda)$, $C = H_{e,k} \cap \mathrm{Box}[\mathbf{0}, \boldsymbol{e}]$	$\mathrm{Diag}(\boldsymbol{u}), \boldsymbol{u}_i = \begin{cases} 0, & \text{if } \boldsymbol{x}_i/\lambda \in C, \\ 1, & \text{otherwise.} \end{cases}$	-
$\lambda\sum_{k=1}^n \lvert\boldsymbol{x}_{[i]}\rvert$	`topkabs`	-	$\boldsymbol{x} - \lambda P_C(\boldsymbol{x}/\lambda)$ $C = B_{\lVert\cdot\rVert_1,[0,k]} \cap \mathrm{Box}[-e, e]$	$\mathrm{Diag}(\boldsymbol{u}), \boldsymbol{u}_i = \begin{cases} 0, & \text{if } \boldsymbol{x}_i/\lambda \in C, \\ 1, & \text{otherwise.} \end{cases}$	-
$\lambda H_\mu(\boldsymbol{x})$	`huber`	-	-	$\lambda\boldsymbol{x}$	-
$\lambda\lVert\boldsymbol{X}\rVert_F^2$	`frobenius`	$\frac{1}{4\lambda}\lVert\boldsymbol{Y}\rVert_{\mathrm{F}}^2$	-	$2\lambda\boldsymbol{X}$	-
$\lambda\lVert\boldsymbol{X}\rVert_F$	`frobenius`	$\delta_{B_{\lVert\cdot\rVert_{\mathrm{F}}}[0,\lambda]}(\boldsymbol{Y})$	$\left(1 - \frac{\lambda}{\max\{\lVert\boldsymbol{X}\rVert_F, \lambda\}}\right)\boldsymbol{X}$	$\begin{cases} I - \lambda(I - \frac{\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}}{\lVert\boldsymbol{X}\rVert_{\mathrm{F}}^2})/\lVert\boldsymbol{X}\rVert_{\mathrm{F}}, & \text{if } \lvert(\boldsymbol{X})_i\rvert < \lambda, \\ 0, & \text{otherwise.} \end{cases}$	-
$\lambda\lVert\boldsymbol{X}\rVert_*$	`nuclear`	$\delta_{B_{\lVert\cdot\rVert_{S_\infty}}[0,\lambda]}(\boldsymbol{Y})$	(2.9)	(2.10)	-
$\lambda\lVert\boldsymbol{X}\rVert_{1,2}$	`l1l2`	$\delta_{\lVert\boldsymbol{Y}\rVert_{\infty,2}<\lambda}(\boldsymbol{Y})$	$[\max\left(0, 1 - \frac{\lambda\lambda}{\lVert X_1\rVert_2}X_1\right), \cdots, \max\left(0, 1 - \frac{\lambda\lambda}{\lVert X_n\rVert_2}X_n\right)]$	$\begin{cases} I - \lambda(I - \boldsymbol{X_i}\boldsymbol{X_i}^{\mathrm{T}}/\lVert\boldsymbol{X}_i\rVert^2)/\lVert\boldsymbol{X}_i\rVert, & \text{if } \lVert\boldsymbol{X}_i\rVert > \lambda, \\ 0, & \text{otherwise.} \end{cases}$	-
$\lambda\lVert\boldsymbol{X}\rVert_{1,\infty}$	`l1linfty`	$\delta_{\lVert\boldsymbol{Y}\rVert_{\infty,1}<\lambda}(\boldsymbol{Y})$	$\boldsymbol{X}_i - \lambda P_{B_{\lVert\cdot\rVert_1}[0,1]}(\boldsymbol{X}_i/\lambda)$	$u_{i,j} = \begin{cases} 0, & \text{if } \lvert\boldsymbol{X}_j/\lambda\rvert > \mu_{*,j}, \text{where } \mu_* \text{ satisfy } \mathbf{1}^{\mathrm{T}}[\boldsymbol{x} - \mu_{*,j}\mathbf{1}]_+ = \mathbf{1} \\ 1, & \text{otherwise.} \end{cases}$	-
$-\lambda\log\det(\boldsymbol{X})$	`logdet`	$-n - \log\det(-\boldsymbol{Y})$	$U\mathrm{diag}\left(\frac{\lambda_j(\boldsymbol{X}) + \sqrt{\lambda_j(X)^2 + 4\lambda}}{2}\right)\boldsymbol{U}^T$	-	$\mathbb{S}_{--}^n$
$\lambda\sigma_1(\boldsymbol{X})$	`mmax`	$\delta_{B_{\lVert\cdot\rVert_*}[0,\lambda]}(\boldsymbol{Y})$	$\boldsymbol{U}\mathrm{diag}\left(\lambda(\boldsymbol{X}) - \lambda P_{\Delta_n}(\lambda(\boldsymbol{X})/\lambda)\right)\boldsymbol{V}^T$	-	γ_n
$\lambda_1\lVert\boldsymbol{x}\rVert_1 + \lambda_2\lVert B\boldsymbol{x}\rVert_1$	`fused`	$\delta_{\lVert\boldsymbol{y}\rVert_\infty<\lambda_1}(\boldsymbol{y}) + \delta_{\lVert B^{\mathrm{T}}\boldsymbol{y}\rVert_\infty<\lambda_2}(\boldsymbol{y})$	(**??**)	(**??**)	-

Table 1: Combined table of functions, their duals, proximal operators, and subdifferentials

1.1.2 Structure of `opts`

The parameters are set in out solver are set in the structure approach to control the solving process. The significant parameters which the user is likely to reset are introduced as follows.

(1) `opts.tol`: accuracy tolerance to terminate the algorithm, default is 10^{-6}.

(2) `opts.maxiter`: the maximum iteration number of, default is 1×10^5 times.

(3) `opts.maxtime`: the maximum time of, default is 2×10^5 seconds.

(4) `opts.record`: the parameter to decide whether to print the message.

(5) `opts.prestop`: the parameter to stop the solver in advance. To prevent the solver from stopping prematurely before the required accuracy is attained, set `opts.prestop=0`.

(6) `opts.cgmaxiter`: the maximum iteration number of CG iteration number.

(7) `opts.cgtol`: the minimun tolerance of CG iteration.

(8) `opts.projectiongap`: the option to decide whether the projection strategy is utilized or not.

1.2 Output arguments

- `x` : the solution structure of Problem (1.1).
- `out` : the structer array of the iteration information structure which records various performance measures of the solver such as

  `out.pinf, out.dinf, out.K1, out.K2`

  corrsdponds to $\eta_P, \eta_D, \eta_{K1}, \eta_{K2}$ that will be defined later.

1.3 Stopping criteria

In SSNCVX, the criteria to mearsue the accuracy of $(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{v}, \boldsymbol{r}, \boldsymbol{x})$ is based on KKT optimality conditions

$$\eta = \max\{\eta_P, \eta_D, \eta_K, \eta_{\mathcal{P}}\},$$

where

$$\begin{aligned}
\eta_P &:= \frac{\|\mathcal{A}\boldsymbol{x} - \Pi_{\mathcal{P}_2}(\mathcal{A}\boldsymbol{x} - \boldsymbol{y})\|}{1+\|\boldsymbol{x}\|}, \eta_D := \frac{\|\mathcal{A}^*(\boldsymbol{y}) + \mathcal{B}^*(\boldsymbol{z}) + \boldsymbol{s} - \mathcal{Q}(\boldsymbol{v}) - \boldsymbol{c}\|}{1+\|\boldsymbol{c}\|}, \\
\eta_K &:= \min\left\{ \frac{\|\boldsymbol{x} - \mathrm{prox}_p(\boldsymbol{x} - \boldsymbol{s})\|}{1+\|\boldsymbol{s}\|+\|\boldsymbol{x}\|}, \frac{\|\mathcal{Q}(\boldsymbol{v}) - \mathcal{Q}(\boldsymbol{x})\|_{\mathrm{F}}}{1+\|\mathcal{Q}(\boldsymbol{v})\|+\|\mathcal{Q}(\boldsymbol{x})\|} \right\}, \\
\eta_{\mathcal{P}} &= \min\left\{ \frac{\|\mathrm{prox}_f(\mathcal{B}\boldsymbol{x} - \boldsymbol{z}) - \mathcal{B}(\boldsymbol{x})\|}{1+\|\mathcal{B}(\boldsymbol{x})\|+\|\boldsymbol{z}\|} \text{ or } \frac{\|-\nabla f^*(-\boldsymbol{z}) - \mathcal{B}(\boldsymbol{x})\|}{1+\|\mathcal{B}(\boldsymbol{x})\|+\|\boldsymbol{z}\|}, \frac{\|\Pi_{\mathcal{P}_1}(\boldsymbol{x} - \boldsymbol{r}) - \boldsymbol{x}\|}{1+\|\boldsymbol{x}\|+\|\boldsymbol{r}\|} \right\}.
\end{aligned}$$

We also compute the relative gap by

$$\eta_g = \frac{\texttt{|pobj - dobj|}}{1 + \texttt{|pobj| + |dobj|}}.$$

For given accuracy η, we terminate SSNCVX when $\eta <$ `opts.tol`.

2 Examples

The classical models that (1.1) includes are classified as follows:

2.1 convex constraint programming

Consider the following convex optimization problems with constraint:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \|\boldsymbol{x}\|_1, \\ \text{s.t.} \quad & \|\mathcal{B}\boldsymbol{x} - \boldsymbol{b}\|_\infty < \lambda_1. \end{aligned} \tag{2.1}$$

The problem include Dantzig selector introduced in Candes and Tao (2009) and constrainted Lasso. The calling syntax for (2.1) can be represented as

```
(xopt, out) = SSNCVX (x0,pblk,f,Bt,[],[],[],[],[],[],[]),
```

where `pblk.type = ‘l1’`, `f.type = ‘linfcon’` and `f.shift` $= \boldsymbol{b}$. For the ℓ_1 constrainted problem:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \|\mathcal{A}\boldsymbol{x} - \boldsymbol{b}\|_2, \\ \text{s.t.} \quad & \|\boldsymbol{x}\|_1 \le \lambda_1, \end{aligned} \tag{2.2}$$

the corresponding calling syntax is

```
(xopt, out) = SSNCVX (x0,pblk,f,Bt,[],[],[],[],[],[],[]),
```

where `pblk.type = ‘l1con’`,`f.type = ‘l2’` and `f.shift` $= \boldsymbol{b}$. For constrained Lasso type problem such as (2.2), we note that our algorithm can solve it directly instead of solving a series of subproblems of level set method Li et al. (2018).

2.2 conic programming

We use a special option “`conic`” to represent problems that have conic structure. We note that we focus on the conic programing with the following structure:

$$\min_{\boldsymbol{x}} \langle \boldsymbol{c}, \boldsymbol{x} \rangle, \quad \text{s.t.} \;\; \boldsymbol{x} \in \mathbb{S}^n_+, \;\; \texttt{bl} \le \mathcal{A}\boldsymbol{x} \le \texttt{bu} \;\; \texttt{l} \le \boldsymbol{x} \le \texttt{u}. \tag{2.3}$$

Model (2.3) also includes SDP with affian constraints such as the relaxation SDP problem arising from BIQ and clustering problems (RCP). In this case, the calling syntax is

```
(xopt, out) = SSNCVX (x0,pblk,[],[],[],c,l,u,At,bl,bu),
```

where `pblk.type = ‘s’`,`l = 0`,`u = inf`.

When $p(\boldsymbol{x}) = \delta_{\mathcal{K}}(\boldsymbol{x})$ denotes the nonnegative cone, semidefinite cone, or second-order cone, the corresponding problem is linear programming (LP), semidefinite programming (SDP), or second-order cone programming (SOCP) respectively, i.e. the following classical conic programming:

$$\min_{\boldsymbol{x}} \langle \boldsymbol{c}, \boldsymbol{x} \rangle \quad \text{s.t.}\, \mathcal{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \in \mathbb{S}^n_+. \tag{2.4}$$

For the classical conic programming, the calling syntax can be represented as

```
(xopt,out) = SSNCVX (x0,pblk,[],[],[],c,[],[],At,bl,bu),
```

where `At,c` denotes the linear map and the objective matrix respectively, `blk` denotes the structure of the cone.

2.3 quadratic programmin

When h, p are two singleton sets, then it corresponds to the classical quadratic programming (QP):

$$\min_{\boldsymbol{x}} \frac{1}{2}\langle \boldsymbol{x}, \mathcal{Q}(\boldsymbol{x})\rangle + \langle \boldsymbol{c}, \boldsymbol{x}\rangle \quad \text{s.t.}\, \mathcal{A}\boldsymbol{x} = \boldsymbol{b},\, \mathtt{l} \le \boldsymbol{x} \le \mathtt{u}. \tag{2.5}$$

It is noted in ApS (2019) that it is more welcomed to transform QP into second order conic programming. This transformation is numerically more robust than the one for quadratic problems. However, in this paper, we consider solving the dual of (2.5). This approach can make use of the sparsity of the problem. Furthermore, it can be extended to solve the ℓ_1-QP problem directly.

$$\min_{\boldsymbol{x}} \frac{1}{2}\langle \boldsymbol{x}, Q\boldsymbol{x}\rangle + \|\boldsymbol{x}\|_1 \quad \text{s.t.}\, \mathcal{A}\boldsymbol{x} = \boldsymbol{b},\, \mathtt{l} \le \boldsymbol{x} \le \mathtt{u}. \tag{2.6}$$

The calling synatx is represented by:

```
(xopt,out) = SSNCVX (x0,pblk,[],[],Q,[],l,u,[],[],[]),
```

where `f.type = 'square'`,`pblk.type = 'l1'`, `Q` is the given quadratic term.

2.4 Lasso type problem

When $Q = 0$ and $\boldsymbol{c} = 0$, the corresponding two-block composite optimization is:

$$\min_{\boldsymbol{x}} \quad \frac{1}{2}\|\mathcal{A}\boldsymbol{x} - \boldsymbol{b}\|^2 + p(\boldsymbol{x}), \tag{2.7}$$

Specially, when $f(\boldsymbol{x}) = \frac{1}{2}\|\mathcal{A}\boldsymbol{x} - \boldsymbol{b}\|^2$ and $p(\boldsymbol{x})$ is a nonnegative positively homogeneous convex function such that $p(0) = 0$, i.e. a gauge function. Model (2.7) covers typical problems that arise in statistical learning. We list a few of them as follows:

- When $p(\boldsymbol{x}) = \lambda\|\boldsymbol{x}\|_1$, the problem corresponds to the classical Lasso problem.
- When $p(\boldsymbol{x}) = \lambda_1\|B\boldsymbol{x}\|_1 + \lambda_2\|\boldsymbol{x}\|_1$, where $B\boldsymbol{x} = [\boldsymbol{x}_2 - \boldsymbol{x}_1, \cdots, \boldsymbol{x}_n - \boldsymbol{x}_{n-1}]^\top \in \mathbb{R}^{n-1}$, then the problem corresponds to the classical Fused Lasso problem.
- When $p(\boldsymbol{x}) = \lambda_1\|\boldsymbol{x}\|_1 + \lambda_2\sum_{l=1}^{G} w_l\|\boldsymbol{x}_{G_l}\|_2$, then the problem corresponds to the classical Group Lasso problem.

Consider the Lasso case, the calling syntax can be

```
(xopt,out) = SSNCVX (x0,pblk,f,Bt,[],[],[],[],[],[],[]),
```

where `f.type = 'square'`,`pblk.type = 'l1'` or `pblk.type = 'fused'`, `Bt` is the given linear map.

2.5 Image restoration model

Furthermore, the robust PCA problem for video segment can be represented by:

$$\min_{\boldsymbol{X}} \|\boldsymbol{X}\|_* + \|\boldsymbol{D} - \boldsymbol{X}\|_1. \tag{2.8}$$

For nuclear norm $\|\boldsymbol{x}\|_*$, let the signular value decomptation of $\boldsymbol{X}$ denoted by $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$, where the its proximal operator can be presented by:

$$\operatorname{prox}_{\lambda\|\cdot\|_*}(\boldsymbol{X}) = U\operatorname{diag}\left(T_\alpha(\boldsymbol{\lambda}(\boldsymbol{X}))\right)V^T. \tag{2.9}$$

Denote

$$\hat{D}_2(G) = U\left[\frac{\Omega^\mu_{\sigma,\sigma} + \Omega^\mu_{\sigma,-\sigma}}{2} \odot G_1 + \frac{\Omega^\mu_{\sigma,\sigma} - \Omega^\mu_{\sigma,-\sigma}}{2} \odot G_1^\top, (\Omega^\mu_{\sigma,0} \odot (G_2))\right]V^\top, \tag{2.10}$$

where $\sigma = [\sigma^{(1)}, \cdots, \sigma^{(m)}]$ is the tensor singular value of $\boldsymbol{X}$, $G_1 = U^\top G V_1 \in \mathbb{R}^{n_1\times n_1}, G_2 = U^\top G V_2 \in \mathbb{R}^{n_1\times(n_2-n_1)}$ and $(\Omega^\mu_{\sigma,\sigma})$ is defined by:

$$(\Omega^\mu_{\sigma,\sigma})_{ij} := \begin{cases} \partial_B \operatorname{prox}_{\mu\|\cdot\|_1}(\sigma_i), & \text{if } \sigma_i = \sigma_j, \\ \left\{\frac{\operatorname{prox}_{\mu\|\cdot\|_1}(\sigma_i) - \operatorname{prox}_{\mu\|\cdot\|_1}(\sigma_j)}{\sigma_i - \sigma_j}\right\}, & \text{otherwise.} \end{cases} \tag{2.11}$$

Then for any $G \in \mathbb{R}^{m\times n}$, we have $\hat{D}[G] = D[G]$. The corresponding calling syntax is

```
(xopt,out) = SSNCVX (x0,pblk,f,Bt,[],[],l,u,[],[],[]).
```

2.6 Clustering problems

The convex clustering model is presented as:

$$\min_{\boldsymbol{X}\in\mathbb{R}^{d\times n}} \frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{a}_i\|^2 + \gamma \sum_{(i,j)\in\mathcal{E}} w_{ij}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_q, \tag{2.12}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n]$, denotes the calssification feature, $\gamma > 0$ is a tuning parameter, $\mathcal{E} = \cup_{i=1}^n \{(i,j) | j \,\text{is}\, i\text{'s } k\text{-nearest neighbors}, i < j \le n\}$ is the edge set. Typically, p is chosen to be $1, 2$ or ∞. After solving (2.12) and obtaining the optimal solution $\boldsymbol{X}^* = [\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_n^*]$, we assign the data vector $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$ to the same cluster if and only if $\boldsymbol{x}_i^* = \boldsymbol{x}_j^*$. In other words, $\boldsymbol{x}_i^*$ is the centroid for observation $\boldsymbol{a}_i$. Sparse PCA has the following form:

$$\min_{\boldsymbol{x}} -\langle \boldsymbol{L}, \boldsymbol{x}\rangle + \lambda\|\boldsymbol{x}\|_1, \text{ s.t.} \operatorname{Tr}(\boldsymbol{x}) = 1, \boldsymbol{x} \succeq 0. \tag{2.13}$$

The problems examples are summarized in Table 2.

Problem Type	**Objective Function**	**Constraints**	**Function block**	**Remarks**
General Problem	$\underbrace{\langle \boldsymbol{c}, \boldsymbol{x} \rangle}_{(1)} + \underbrace{\frac{1}{2} \langle \boldsymbol{x}, \mathcal{Q}(\boldsymbol{x}) \rangle}_{(2)} + \underbrace{f(\mathcal{B}(\boldsymbol{x}))}_{(3)} + p(\boldsymbol{x})$	$\underbrace{\boldsymbol{x} \in \mathcal{P}_1}_{(4)}, \underbrace{\mathcal{A}(\boldsymbol{x}) \in \mathcal{P}_2}_{(5)}.$	(1) (2) (3) (4) (5)	Handles various optimization problems
Conic programming	$\langle \boldsymbol{c}, \boldsymbol{x} \rangle$	$\mathcal{A}(\boldsymbol{x}) = \boldsymbol{b}, \boldsymbol{x} \geq 0.$	(1)(5)	Linear programming
	$\langle \boldsymbol{C}, \boldsymbol{X} \rangle$	$\mathcal{A}(\boldsymbol{X}) = \boldsymbol{b}, \boldsymbol{X} \in \mathbb{S}_+^n.$	(1)(5)	semidefinite programming
	$\langle \boldsymbol{c}, \boldsymbol{x} \rangle$	$\mathcal{A}(\boldsymbol{x}) = \boldsymbol{b}, \boldsymbol{x} \in \mathcal{Q}^n.$	(1)(5)	quadratic cone programming
SDP with box constraints	$\langle \boldsymbol{C}, \boldsymbol{X} \rangle$	$\mathcal{A}(\boldsymbol{X}) = \boldsymbol{b}, \boldsymbol{x} \in \mathcal{P}_1, \boldsymbol{X} \in \mathbb{S}_+^n.$	(1)(4)(5)	SDP with box constraints
Lasso type Problems	$\frac{1}{2}\|\mathcal{B}(\boldsymbol{x}) - \boldsymbol{b}\|^2 + \lambda\|\boldsymbol{x}\|_1$	-	(3)	Lasso problem
	$\frac{1}{2}\|\mathcal{B}(\boldsymbol{x}) - \boldsymbol{b}\|^2 + \lambda_1\|\boldsymbol{x}\|_1 + \lambda_2\|D\boldsymbol{x}\|_1$	-	(3)	Fused lasso problem
	$\frac{1}{2}\|\mathcal{B}(\boldsymbol{x}) - \boldsymbol{b}\|^2 + \lambda\|\boldsymbol{x}\|_2$	-	(3)	Group Lasso
	$\frac{1}{2}\|\mathcal{B}(\boldsymbol{x}) - \boldsymbol{b}\|^2 + \lambda \sum_{i=1}^k \boldsymbol{x}_{[i]}$	-	(3)	Top-k regression
Matrix Completion	$\|\mathcal{B}(\boldsymbol{X}) - \boldsymbol{B}\|_{\mathrm{F}}^2 + \lambda\|\boldsymbol{X}\|_*$	-	(3)	Low-rank matrix recovery
QP	$\langle \boldsymbol{x}, \mathcal{Q}(\boldsymbol{x}) \rangle + \langle \boldsymbol{x}, \boldsymbol{c} \rangle$	$\mathtt{l} \leq \boldsymbol{x} \leq \mathtt{u}, \mathcal{A}(\boldsymbol{x}) = \boldsymbol{b}.$	(1)(2)(4)(5)	Quadratic Programming
QP with regularizer	$\langle \boldsymbol{x}, \mathcal{Q}(\boldsymbol{x}) \rangle + \lambda\|\boldsymbol{x}\|_1$	$\mathtt{l} \leq \boldsymbol{x} \leq \mathtt{u}, \mathcal{A}(\boldsymbol{x}) = \boldsymbol{b}.$	(1)(2)(3)(5)	QP with ℓ_1 norm
convex constraint problems	$\|\boldsymbol{x}\|_1$	$\|\mathcal{B}\boldsymbol{x} - \boldsymbol{b}\|_\infty < \lambda$	(3)	ℓ_∞ constraint problem
	$\|\mathcal{B}(\boldsymbol{x}) - \boldsymbol{b}\|_1$	$\|\boldsymbol{x}\|_1 < \lambda$	(3)	ℓ_1 constraint problem
Statistical learning	$-\langle \boldsymbol{L}, \boldsymbol{x} \rangle + \lambda\|\boldsymbol{x}\|_1,$	$\mathrm{Tr}(\boldsymbol{x}) = 1, \boldsymbol{x} \succeq 0$	(3)	Sparse PCA
	$\|\boldsymbol{x}\|_1$	$\mathcal{A}(\boldsymbol{x}) = \boldsymbol{b},$	(5)	Base pursuit
	$\|\boldsymbol{x}_1\|_* + \mu\|\boldsymbol{x}_2\|_1$	$\boldsymbol{x}_1 + \boldsymbol{x}_2 = \boldsymbol{D}$	(3)(5)	Roubst PCA
	$-\log(\det(\boldsymbol{X})) + \mathrm{Tr}(\boldsymbol{X}\boldsymbol{S}) + \lambda\|\boldsymbol{X}\|_1$	-	(1)(3)	Sparse covariance matrix estimation
	$\frac{1}{2}\sum_{i=1}^n \|\boldsymbol{x}_i - \boldsymbol{a}_i\|^2 + \gamma \sum_{(i,j)\in\mathcal{E}} w_{ij}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_q$	-	(3)	convex clustering problem

Table 2: The examples of problems Model (1.1) able to solve.

References

M. ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 10.1.0.*, 2019. URL `http://docs.mosek.com/10.1/toolbox/index.html`.

E. Candes and T. Tao. The dantzig selector: Statistical estimation when p is much larger than n. *Quality control and applied statistics*, 54(1):83–84, 2009.

X. Li, D. Sun, and K.-C. Toh. On efficiently solving the subproblems of a level-set method for fused Lasso problems. *SIAM Journal on Optimization*, 28(2):1842–1866, 2018.