Homework 5 Report 1

李知含

1600010653

November 28, 2017

1 Answers to questions

Question 1 (1) The functions are `l1_cvx_mosek` and `l1_cvx_gurobi` in module `l1_cvx_mosek`. Note that the codes are implemented using CVXPY, which is an alternative of CVX in Python.

Question 2 (2) The model is equivalent to QP

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}y^{\mathrm{T}}y + \mu 1^{\mathrm{T}}t, \\ \text{subject to} & Ax - y = b, \\ & -t \preceq x \preceq t, \end{array} \tag{1}$$

The functions are `l1_mosek_qp` in module `solver_mosek` and `l1_gurobi_noexpand` in module `solver_gurobi`.

Question 3 (3 (a)) The model is first equivalent to QP

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\left(Ax - b\right)^{\mathrm{T}}\left(Ax - b\right) + \mu 1^{\mathrm{T}}t, \\ \text{subject to} & -t \preceq x \preceq t. \end{array} \tag{2}$$

Denote $t + x$ and $t - x$ by u, v respectively, and this model is equivalent to

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\left(\frac{1}{2}A\left(u - v\right) - b\right)^{\mathrm{T}}\left(\frac{1}{2}A\left(u - v\right) - b\right) + \frac{1}{2}\mu 1^{\mathrm{T}}\left(u + v\right), \\ \text{subject to} & u \succeq 0, \\ & v \succeq 0, \end{array} \tag{3}$$

which is quadratic program with box constraints.

Data: A, b, mu, $x^{(0)}$, η
$i \leftarrow 0$;
$u^{(0)} \leftarrow 2\max\left\{x^{(0)}, 0\right\}$;
$v^{(0)} \leftarrow 2\min\left\{x^{(0)}, 0\right\}$;
while *not satisfies stop condition* **do**
  $\nabla u^{(i)} \leftarrow \frac{1}{2}A^{\mathrm{T}}\left(\frac{1}{2}A\left(u^{(i)} - v^{(i)}\right) - b\right) + \frac{1}{2}\mu 1^{\mathrm{T}}$;
  $\nabla v^{(i)} \leftarrow -\frac{1}{2}A^{\mathrm{T}}\left(\frac{1}{2}A\left(u^{(i)} - v^{(i)}\right) - b\right) + \frac{1}{2}\mu 1^{\mathrm{T}}$;
  $u^{(i+1)} \leftarrow u^{(i)} - \eta\nabla u^{(i)}$;
  $v^{(i+1)} \leftarrow v^{(i)} - \eta\nabla v^{(i)}$;
  $u^{(i+1)} \leftarrow \max\left\{u^{(i+1)}, 0\right\}$;
  $v^{(i+1)} \leftarrow \max\left\{v^{(i+1)}, 0\right\}$;
  $i \leftarrow i + 1$;
end
$x^{(i)} = \frac{1}{2}\left(u^{(i)} - v^{(i)}\right)$;

Algorithm 1: Projection gradient method with fixed step size

The corresponding algorithm is shown in Algorithm 1.

First increasing μ, and then decreasing μ exponentially to the original value result in better numerical result. The algorithm is shown in Algorithm 2.

Data: A, b, mu, $x^{(0)}$, η, i_0, k
$i \leftarrow 0$;
$u^{(0)} \leftarrow 2\max\left\{x^{(0)}, 0\right\}$;
$v^{(0)} \leftarrow 2\min\left\{x^{(0)}, 0\right\}$;
while *not satisfies stop condition* **do**
  $\mu^{(i)} \leftarrow \mu \mathrm{e}^{k\max\{i_0 - i, 0\}}$;
  $\nabla u^{(i)} \leftarrow \frac{1}{2}A^{\mathrm{T}}\left(\frac{1}{2}A\left(u^{(i)} - v^{(i)}\right) - b\right) + \frac{1}{2}\mu^{(i)} 1^{\mathrm{T}}$;
  $\nabla v^{(i)} \leftarrow -\frac{1}{2}A^{\mathrm{T}}\left(\frac{1}{2}A\left(u^{(i)} - v^{(i)}\right) - b\right) + \frac{1}{2}\mu^{(i)} 1^{\mathrm{T}}$;
  $u^{(i+1)} \leftarrow u^{(i)} - \eta\nabla u^{(i)}$;
  $v^{(i+1)} \leftarrow v^{(i)} - \eta\nabla v^{(i)}$;
  $u^{(i+1)} \leftarrow \max\left\{u^{(i+1)}, 0\right\}$;
  $v^{(i+1)} \leftarrow \max\left\{v^{(i+1)}, 0\right\}$;
  $i \leftarrow i + 1$;
end
$x^{(i)} = \frac{1}{2}\left(u^{(i)} - v^{(i)}\right)$;

Algorithm 2: Projection gradient method with fixed step size and modified μ

These two algorithms are implemented by `l1_proj_grad` in module `method_proj_grad`.

Question 4 (3 (b)) The corresponding algorithm is shown in Algorithm 3, where the sign functions are used for sub-gradients.

The same strategy of adjusting μ here, as shown in Algorithm .

Data: A, b, mu, $x^{(0)}$, η
$i \leftarrow 0$;
while *not satisfies stop condition* **do**
  $\nabla x^{(i)} \leftarrow A^{\mathrm{T}}(Ax - b) + \mu \operatorname{sgn} x^{(i)}$;
  $x^{(i+1)} \leftarrow x^{(i)} - \eta \nabla x^{(i)}$;
  $i \leftarrow i + 1$;
end

Algorithm 3: Sub-gradient method with fixed step size

Data: A, b, mu, $x^{(0)}$, η, i_0, k
$i \leftarrow 0$;
while *not satisfies stop condition* **do**
  $\mu^{(i)} \leftarrow \mu \mathrm{e}^{k \max\{i_0 - i\}, 0}$;
  $\nabla x^{(i)} \leftarrow A^{\mathrm{T}}(Ax - b) + \mu^{(i)} \operatorname{sgn} x^{(i)}$;
  $x^{(i+1)} \leftarrow x^{(i)} - \eta \nabla x^{(i)}$;
  $i \leftarrow i + 1$;
end

Algorithm 4: Sub-gradient method with fixed step size and modified μ

These two algorithms are implemented by `l1_sub_grad` in module `method_sub_grad`.

2 Numerical results

Numerical results are shown in Table 1.

In Table 1, "approximation error" stands for $\frac{1}{2}\|Ax - b\|_2^2$, "error to known" stands for the error to known solution obtained by MOSEK from CVXPY, and "error to GT" stands for the error to `u`. (Note that the optimization problem is derived from compressive sensing, which finds a sparse representation of a optimization problem. Therefore, we denotes `u` to be "ground-truth", which is solution to be reconstructed, even if it is not the exact solution of the optimization problem)

All the codes are implemented in Python. These results are carried out on a computer with Intel Core i7-6500U (4 threads) and 7894 MiB RAM. The operating system is Arch Linux 4.13.12 64-bit and the Python environment is provided by Anaconda 4.3.30. The versions of Python, NumPy, CVXPY, MOSEK and Gurobi are 3.6.2, 1.13.1, 0.4.8, 8.1.31, 7.5.1 respectively.

In Algorithm 1, $\eta = 0.001$. In Algorithm 2, $\eta = 0.002$, $i_0 = 1500$ and $\mu^{(0)} = 1000$. In Algorithm 3, $\eta = 0.0002$. In Algorithm 4, $\eta = 0.0005$, $i_0 = 1500$ and $\mu^{(0)} = 1000$.

$x^{(0)}$ is not used when direct calling CVXPY or solvers.

	time (s)	setup time (s)	solve time (s)	variables	iterations
CVXPY-MOSEK	1.088	0.077	0.620	NA	9
CVXPY-Gurobi	15.799	NA	2.783	NA	NA
MOSEK	0.785	0.094	0.681	2560	9
Gurobi	18.729	17.195	1.074	2560	14
Algorithm 1	1.618	NA	NA	2048	2000
Algorithm 2	1.602	NA	NA	2048	2000
Algorithm 3	1.149	NA	NA	1024	2000
Algorithm 4	1.142	NA	NA	1024	2000

	primal objective	approximation error	error to known	error to GT
CVXPY-MOSEK	7.561e-02	2.060e-07	0.000e+00	1.641e-05
CVXPY-Gurobi	7.561e-02	1.449e-07	1.093e-05	6.298e-06
MOSEK	7.561e-02	1.785e-07	6.618e-05	1.242e-05
Gurobi	7.561e-02	1.082e-07	1.446e-05	2.555e-06
Algorithm 1	5.764e-01	1.041e-07	1.253e+00	1.253e+00
Algorithm 2	7.561e-02	1.081e-07	1.450e-05	2.509e-06
Algorithm 3	3.830e-01	2.525e-07	1.405e+00	1.405e+00
Algorithm 4	7.561e-02	4.965e-07	1.305e-05	4.469e-06

Table 1 Numerical results

3 Interpretations and Discussions

From the result, all algorithms except Algorithm 1 and 3 converges to the optimal solution.

The intuition of modifications in Algorithm 2 and 4 follows. In the results of Algorithm 1 and 3, the approximation error is rather low, which indicates the regularization fails. Figure 1 proves this fact. To settle this issue, the regularization should be increased, and therefore an

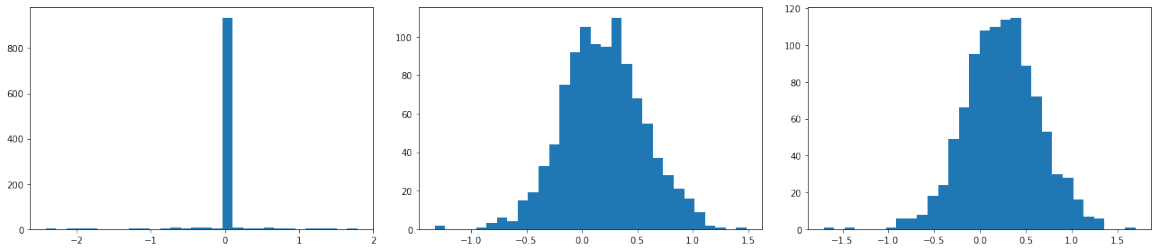

Figure 1 Histogram of solutions (Left: CVX-MOSEK, middle: Algorithm 1, right: Algorithm 3)

exponentially decreasing sequence of μ is introduced. Note that $i_0 = 1500$, and therefore the gradient method runs for 500 iterations for the original optimization problem, which ensures the solution not distorted.

Note that the Gurobi is not compatible with the memory model of Python, which results in a difficiency in setup time. To be exact, the Python version of Gurobi uses a class

named `tupledict` to handle matrices and tensors which cannot be directly converted to `numpy.array`, and therefore conversion accounts for the abnormal long setup time. The solve time makes sense anyway.