Mirror Prox Solver for $\min_x \{\|x\|_1 : \|Ax-b\|_p \le \delta\}$ User's Guide

A. Juditsky, F. Kilinc-Karzan, A. Nemirovski

January 3, 2012

1 Purpose

The solver `MirrorProx` implements the (deterministic) Mirror Prox algorithm [1] as applied to the problem

$$\mathrm{Opt} = \min_x \{\|x\|_1 : \|Ax-b\|_p \le \delta\}, \tag{1}$$

where A is an $m \times n$ matrix and p is either ∞ or 2.

On successful run, `MirrorProx` returns a vector x such that

$$\begin{array}{ll}(a) & \|x\|_1 \le (1+\omega)\mathrm{Opt} \\ (b) & \|Ax-b\|_p \le \delta + \epsilon,\end{array} \tag{2}$$

where $\omega \ge 0$, $\epsilon > 0$ are desired tolerances.

There is also an option to solve the problem

$$\mathrm{Opt}(\rho) = \min_x \{\|Ax-b\|_p : \|x\| \le 1/\rho\} \tag{3}$$

with $\rho > 0$ given; with this option (referred to as the `rhofixed`), on successful run the returned solution x satisfies

$$\begin{array}{ll}(a) & \|x\|_1 \le 1/\rho \\ (b) & \|Ax-b\|_p \le \rho^{-1}\mathrm{Opt}(\rho) + \epsilon,\end{array} \tag{4}$$

where $\epsilon > 0$ is a desired tolerance.

Absolute and Relative tolerances. The user has an option to specify the desired tolerance ϵ either in the *absolute*, or in the *relative* scale. With the absolute scale, the numerical value of ϵ is specified. With the relative scale, user specifies relative accuracy ν which is linked with ϵ via the relation (cf. section 3.1)

$$\epsilon = \nu \cdot [\|A\|_{1\to p}\mathrm{Opt}],$$

where $\|A\|_{1\to p}$ is the maximum of $\|\cdot\|_p$-norms of columns in A and Opt is given by (1).

2 Synopsis

The calling sequence is

```
sol=MirrorProx(data,control);
```

The first argument is obligatory, the second is optional.

Specifying the data. `data` is a user-provided MATLAB structure specifying A, b, p, with obligatory fields as follows:

name	type	default value	meaning
`data.fit`	double	2	the value of p (that is, either 2, or `inf`)
`data.ydim`	int	–	m (A is $m \times n$)
`data.xdim`	int	–	n (A is $m \times n$)
`data.L`	double	–	the quantity $\|A\|_{1\to p} := \max\limits_{1\le j\le n} \|\mathrm{Column}_j(A)\|_p$
`data.b`	`double(`m`,1)`	–	array representing b
`data.delta`	≥ 0	`0.0005`	`data.delta` is δ in (1), **Note:** `data.delta` is irrelevant when `control.rhofixed='on'` (see below)
`data.Mult`	function handle	–	handle to "*multiplication oracle*" — a user-supplied function computing matrix-vector products involving A and A^T (see below).

Defining *Multiplication oracle* **routine:** the calling sequence to the routine should be

```
res=<data.Mult>(x,Tflag,data);
```

• when `Tflag=0`, `x` should be `double`$(n,1)$ representing $x \in \mathbf{R}^n$, and `res` should be `double(`m`,1)` representing Ax
• when `Tflag=1`, `x` should be `double`$(m,1)$ representing $x \in \mathbf{R}^m$, and `res` should be `double(`n`,1)` representing $A^T x$.

Note: the structure `data` may include additional user-specified fields, to be used by the multiplication oracle.

Example 1 Let A be a general-type dense $m \times n$ matrix. In this case one can specify `data.A` as a `double(`m`,`n`)` array containing A; define

```
data.Mult=@PlainMult;
```

the corresponding multiplication oracle being

```
function res=PlainMult(x,Tflag,data);
if Tflag,
   res=data.A'*x;
else
   res=data.A*x;
end;
```

Example 2 Let $2k \times n$ matrix A represent $k \times n$ submatrix F of the $n \times n$ DFT matrix F_n (complex-valued rows r in F are represented in A by pairs of rows $\Re$`(r)`, $\Im$`(r)`). This can be captured by specifying $k \times 1$ array `data.freq` of distinct positive integers $\le n$ – the indexes of the rows of F, and the corresponding multiplication oracle could be as follows:

```
function res=FFTMult(x,Tflag,data);
myi=sqrt(-1);
k=data.ydim/2;
n=data.xdim;
if Tflag,
   tmp=zeros(n,1);
   tmp(data.freq)=x(1:k)+myi*x(k+1:end);
   res=real(n*ifft(tmp));
else
   tmp=fft(x);
   res=[real(tmp(data.freq));imag(tmp(data.freq))];
end;
```

The corresponding pointer may be defined as

```
data.Mult=@FFTMult;
```

Specifying the controls. `control` is a user-specified structure with control parameters (integers, doubles and characters) as follows:

name	range	default value	meaning
`control.rhofixed`	`'on'`/`'off'`	`'off'`	`'on'`: solving problem (3) `'off'`: solving problem (1)
`control.rho`	> 0	1	rhofixed on: ρ in (3) rhofixed off: `control.rho` is irrelevant
`control.accuracy`	'abs'/'rel'	`'rel'`	`control.accuracy='abs'` means that a solution of *absolute* accuracy `control.eps` is sought; `control.accuracy='rel'` means that a solution of *relative* accuracy `control.eps` is sought
`control.eps`	> 0	0.0005	required absolute (when `control.accuracy='abs'`) or relative (when `control.accuracy='rel'`) accuracy
`control.omega`	≥ 0	0	rhofixed off: `control.omega` is ω in (2), rhofixed on: `control.omega` is irrelevant
`control.ItrMax`	$1, 2, ...$	50000	maximal # of steps
`control.cpumax`	$(0, \infty)$	1800	maximum running time, sec
`control.kappa`	$(0.01, 0.99)$	0.75	see section 3
`control.printlevel`	$0/1$	0	printlevel (0 – no display output)
`control.dgf`	`'e'/'p'`	`'p'`	switch from 'p' to 'e' is expected to increase somehow the iteration count while reducing the computational effort per iteration

Note: There is no necessity for a user to specify all control fields; the fields unspecified on a call to `MirrorProx` will be assigned their default values.

Output. On successful run, solver returns structure `sol` with the fields as follows:

name	semantics
`sol.rhofixed`	`'on'` when rhofixed is on, and `'off'` otherwise
`sol.p`	p in (1)
`sol.ok`	0 – successful run (termination when desired accuracy is reached), 1 – termination when maximum iteration count/running time is reached
`sol.cpu`	running time, sec
`sol.Steps`	total number of iterations
`sol.Calls`	total number of *Calls* – computations of the form $[x, y] \mapsto [A^T y; Ax]$
`sol.Stages`	total number of *Stages*, see section 3
`sol.x`	array with resulting primal solution x
`sol.Ax`	array Ax
`sol.rho`	rhofixed off: the final value of ρ, see section 3 rhofixed on: `control.rho`
`sol.ell1norm`	$\|x\|_1$
`sol.err.R`	rhofixed off: upper bound on $[\|x\|_1 - \mathrm{Opt}]$ rhofixed on: 0
`sol.err.abs`	upper bound on $\|Ax - b\|_p - \delta$
`sol.err.rel`	rhofixed off: upper bound on $[\|Ax - b\|_p - \delta]/[\|A\|_{1\to p}\mathrm{Opt}]$ rhofixed on: ∞
`sol.opt_lwb`	rhofixed off: a lower bound on Opt, see (1) rhofixed on: 0
`sol.res`	$\|Ax - b\|_p$
`sol.res_rel`	$\|Ax - b\|_p/\|b\|_p$
`sol.res_lwb`	a lower bound on $\mathrm{Opt}(\rho)$ for $\rho =$ `sol.rho`, see (3)
`sol.y`	array with the resulting dual approximate solution y, see section 3
`sol.ATy`	array $A^T y$
`sol.control`	structure representing controls used in the solution process

3 What is inside

To use the software in an intelligent manner, a potential user should have an idea of the strategy implemented in the solver. We are about to outline this strategy for the situation where the problem of interest is (1) (i.e., rhofixed is off); the much simpler case of problem (3) will be addressed later.

Observe that the problem of interest (1) can be rewritten equivalently as

$$\mathrm{Opt} = \min_{\xi,R} \{R \geq 0 : \|AR\xi - b\|_p \leq \delta, \|\xi\|_1 \leq 1\}$$

(substitution $x = R\xi$). In the only nontrivial case of $\mathrm{Opt} > 0$ (or, which is the same, of $\|b\|_p > \delta$), passing from the variables R, ξ to the variables $\rho = 1/R, \xi$, we can rewrite the latter problem as

$$\frac{1}{\mathrm{Opt}} = \max_{\rho>0} \left\{\rho : \Phi(\rho) := \min_{\xi:\|\xi\|_1\leq 1} \|A\xi - \rho b\|_p - \rho\delta \leq 0\right\}. \tag{5}$$

When (1) is feasible and nontrivial (i.e., $0 < \mathrm{Opt} < \infty$), the function $\Phi(\rho)$ looks as shown on Figure 1 (magenta). It is a convex function of $\rho \geq 0$ which vanishes at $\rho = 0$ and $\rho_* = \frac{1}{\mathrm{Opt}}$,

An approximate solution $\widehat{x}$ satisfying (3) can be represented as $\widehat{\rho}^{-1}\widehat{\xi}$, where $\widehat{\rho} \geq \rho_*/(1+\omega)$ and $\|\widehat{\xi}\|_1 \leq 1$, implying that $\|\widehat{x}\|_1 \leq 1/\rho \leq (1+\omega)/\rho_* = (1+\Omega)\mathrm{Opt}$, which is the first requirement in (3). The second requirement in (3) amounts to $\|A\widehat{\xi} - \widehat{\rho}b\|_p - \widehat{\rho}\delta \leq \widehat{\rho}\epsilon$. This relation definitely will hold if $\widehat{\rho}$ is close enough to ρ_*, so that $\Phi(\widehat{\rho})$ is small, and $\widehat{\xi}$ is a near-minimizer of $\|A\xi - \widehat{\rho}b\|_p$ over $\|\xi\|_1 \leq 1$, so that $\|A\widehat{\xi} - \widehat{\rho}b\|_p - \widehat{\rho}\delta$ is close to $\Phi(\widehat{\rho}) = \min\limits_{\|\xi\|_1 \leq 1} \|A\xi - \widehat{\rho}b\|_p - \widehat{\rho}\delta$. To achieve the outlined goals, we apply to Φ a Newton-type root-finding routine aimed at fast approximation of ρ_* (cf. [2]). Specifically,

1. The execution of the algorithm is split into *stages* $s = 1, 2, ...$; at stage s. we work with some value $\rho_s \geq \frac{\rho_*}{1+\omega}$ of ρ, and ρ_1 is chosen to be $> \rho_*$.

2. At s-th stage, we apply the Saddle Point Mirror Prox algorithm [1] to build sequences $u_1^s \geq u_2^s \geq ...$, $\ell_1^s \leq \ell_2^s \leq ...$ of upper and lower bounds on the quantity $\Phi(\rho_s)$.

   - Every upper bound u_t^s is augmented with an approximate solution ξ_t^s satisfying $\|\xi_t^s\| \leq 1$ and $\|A\xi_t^s - \rho_s b\|_p - \rho_s\delta = u_t^s$. It follows that the vector $x_t^s = \xi_t^s/\rho_s$ satisfies $\|x_t^s\|_1 \leq 1/\rho_s \leq (1+\omega)/\rho_* = (1+\omega)\mathrm{Opt}$ and $\|Ax_t^s - b\|_p \leq \delta + u_t^s/\rho_s$. In particular, in the case of $u_t^s \leq \epsilon\rho_s$, the vector x_t^s satisfies our target relation (2), and we terminate.
   - Every lower bound ℓ_t^s is augmented with *dual solution* y_t^s – a vector satisfying $\|y_t^s\|_{\frac{p}{p-1}} \leq 1$ For a vector y satisfying the latter relation, we clearly have $y^T(A\xi - \rho b) \leq \|A\xi - \rho b\|_p$, whence $\min_{\|\xi\|_1 \leq 1} y^T A\xi - \rho y^T b - \rho\delta = -\|A^T y\|_\infty - \rho(b^T y + \delta)$ is a lower bound on $\Phi(\rho)$ for all ρ. In particular, $\phi_t^s(\rho) := -\|A^T y_t^s\|_\infty - \rho(b^T y_t^s + \delta)$ is an affine in ρ lower bound on $\Phi(\rho)$. The quantity ℓ_t^s is nothing but $\phi_t^s(\rho_s)$.
   - The Saddle Point Mirror Prox algorithm ensures that $u_t^s - \ell_t^s$ goes to 0 as $t \to \infty$, meaning that eventually either u_t^s will become $\leq \epsilon\rho_s$ (this is our termination condition), or ℓ_t^s will be positive, and the ratio u_t^s/ℓ_t^s will be less than $1+\kappa$, where $\kappa \in (0,1)$ is a parameter of the method. Whenever the latter happens, $\phi_t^s(\rho_s)$ *has unique root* $\widehat{\rho}_s$*, and this root is a smaller than* ρ_s *upper bound on* ρ_* (since $\phi_t^s(\rho)$ is a lower bound on $\Phi(\rho)$ and $\psi_t^s(\rho_s) = \ell_t^s > 0$). In the situation in question, we pass to phase $s+1$, setting $\rho_{s+1} = \widehat{\rho}_s/(1+\omega)$ (and thus ensuring that $\rho_{s+1} \geq \rho_*/(1+\omega)$.

   The outlined mechanism is illustrated on Figure 1.

3.1 Complexity Bound

The number of *stages* in the above algorithm provably does not exceed

$$S = c(\kappa)\ln\left(\frac{\|A\|_{1\to p}\mathrm{Opt}}{\epsilon}\right), \quad \|A\|_{1\to p} = \max_j \|A_j\|_p,$$

where $A_1, ..., A_n$ are the columns of A, and the number of iterations at a stage does not exceed

$$c(\kappa)\chi_p\frac{\|A\|_{1\to p}\mathrm{Opt}}{\epsilon}, \ \chi_p = \begin{cases} \sqrt{\ln(m+1)\ln(n+1)}, & p = \infty \\ \sqrt{\ln(n+1)}, & p = 2 \end{cases}$$

Here $c(\kappa)$ depends solely on $\kappa \in (0,1)$.

Effort per iteration is dominated by the necessity to carry out a small number (3 at average) of *calls* — matrix-vector multiplications of the form $(x, y) \mapsto (Ax, A^T y)$.

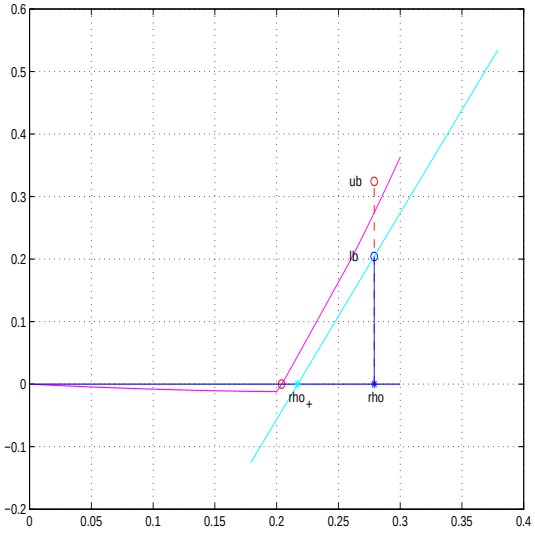

Figure 1: Magenta: function $\Phi(\cdot)$. ρ_* is the positive root of Φ, $\rho > \rho_*$ is ρ_s. ub and lb are the current upper and lower bounds u_t^s, ℓ_t^s on $\Phi(\rho_s)$ satisfying $u_t^s/\ell_t^s \le 1+\kappa$. Cyan: $\rho_t^s(\cdot)$, ρ_+ is ρ_{s+1} ($\omega = 0$).

	$p=2$			$p=\infty$		
Sizes of A	# of stages	# of calls	CPU, sec	# of stages	# of calls	CPU, sec
256×1024	8	666	5.5 (3.4)	7	6332	78.6 (3.6)
2048×4096	7	390	23.0 (829.6)	7	9064	712.4 (1684.9)

Table 1: Sample numerical results for Mirror Prox. Red: cpu time of finding a solution of required quality by a commercial interior point solver `mosekopt`.
Platform: Lenovo T410 laptop with Intel(R) Core(TM) i7 CPU 2.67 GHz, 8.00 GB RAM, 64 bit

Some impression on actual running times can be obtained from Table 1 below. In these experiments, A is drawn at random from the *Rademacher ensemble* (random matrices with independent entries taking values ± 1 with probability 1/2) and in the case of $p = 2$ are further divided by $\sqrt{m}$ (to make the Euclidean lengths of the columns A to become equal to 1). For the problems we were generating, Opt is close to 1, and $\|A\|_{1\to p}$, meaning that the relative and absolute accuracy scales nearly coincide. The absolute accuracy we sought for was $\epsilon =$ 5.e-4, and up to this parameter, the controls were set to their default values.

Warning: As far as large-scale dense matrices A are concerned, the above complexity bounds are the best known so far. This being said, note that the number of iterations is inverse proportional to the relative accuracy, meaning that *it is unreasonable to ask for really high (something like 10^{-6} or less) relative accuracy.*

References

[1] Nemirovski, A., "Prox-method with rate of convergence $O(1/t)$ for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems" – *SIAM Journal on Optimization* **15** (2004), 229–251.

[2] Juditsky, A., Kilinç Karzan, F., Nemirovski, A. (2011), "L_1 Minimization via Randomized First Order Algorithms" – submitted to *Mathematical Programming*
E-print: http://www.optimization-online.org/DB_FILE/2010/05/2618.pdf