User's Guide for YALL1: Your ALgorithms for L1 Optimization

Yin Zhang
Department of CAAM
Rice University, Houston, Texas, 77005

(CAAM Technical Report TR09-17)

(Versions: beta3 5-13-09; beta4 5-25-09; beta5 6-18-09)

Abstract

This User's Guide describes the functionality and basic usage of the Matlab package YALL1 for L1 minimization. The one-for-six algorithm used in the YALL1 solver is briefly introduced in the appendix.

1 Introduction

YALL1 stands for "Your ALgorithms for L1". It is a Matlab solver that at present can be applied to the following six L1-minimization models:

$$\text{(BP)} \quad \min_{x\in\mathbb{C}^n} \|Wx\|_{w,1}, \ \text{s.t.} \ Ax = b, \tag{1}$$

$$\text{(L1/L1)} \quad \min_{x\in\mathbb{C}^n} \|Wx\|_{w,1} + (1/\nu)\|Ax-b\|_1, \tag{2}$$

$$\text{(L1/L2)} \quad \min_{x\in\mathbb{C}^n} \|Wx\|_{w,1} + (1/2\rho)\|Ax-b\|_2^2, \tag{3}$$

$$\text{(BP+)} \quad \min_{x\in\mathbb{R}^n} \|x\|_{w,1}, \ \text{s.t.} \ Ax = b \quad \text{and} \ x \geq 0, \tag{4}$$

$$\text{(L1/L1+)} \quad \min_{x\in\mathbb{R}^n} \|x\|_{w,1} + (1/\nu)\|Ax-b\|_1, \quad \text{s.t.} \ x \geq 0, \tag{5}$$

$$\text{(L1/L2+)} \quad \min_{x\in\mathbb{R}^n} \|x\|_{w,1} + (1/2\rho)\|Ax-b\|_2^2, \quad \text{s.t.} \ x \geq 0, \tag{6}$$

where $\nu, \rho \geq 0$, $A \in \mathbb{C}^{m\times n}$, $b \in \mathbb{C}^m$, $x \in \mathbb{C}^n$ (for the first three) or $x \in \mathbb{R}^n$ (for the last three) and $W \in \mathbb{C}^{n\times n}$ is a unitary matrix serving as a sparsifying basis. Also $\|\cdot\|_{w,1}$ is the weighted L1 (semi-) norm defined as

$$\|x\|_{w,1} \triangleq \sum_{i=1}^{n} w_i|x_i| \tag{7}$$

for any given nonnegative vector $w \in \mathbb{R}^n$. Here BP stands for "basis pursuit", a name commonly attached to model (1) in the signal processing community [1]. Other names are more or less self-evident. The second terms in objective functions are commonly referred to as fidelity terms.

In general, the first two problems are not linear programs for a complex variable $x \in \mathbb{C}^n$, but they reduce to linear programs when x is real. Moreover, it is well-known that unlike the L2-norm square used in the L1/L2 and L1/L2+ models, the L1-norm penalty functions used in the L1/L1 and L1/L1+ models are *exact*; that is, when ν is sufficiently small, the L1/L1 (L1/L1+) model reduces to the BP (BP+) model.

1.1 Scaling of A

The scaling of the measurement matrix A is important to the performance of YALL1. YALL1 version beta-3 requires A to have, or to be scaled to have, orthonormal rows so that

$$AA^* = I.$$

This requirement is no longer necessary for YALL1 version beta-4. However, such orthonormalization is still recommended, whenever it is computationally reasonable to do, because it generally makes our algorithms more robust.

If orthonormalization is computationally too expensive, then one should at least consider to normalize the rows of A so that they all have the unit length, i.e.,

$$\|e_i^T A\|_2 = 1, \;\; i = 1, 2, \cdots, m.$$

If the rows of A are neither orthonormalized nor normalized, then YALL1 may be prone to slow convergence. (Normalizing the columns, instead of rows, of A will also help.)

2 Quick Start

YALL1 can be downloaded from the website:

```
http://www.caam.rice.edu/~optimization/L1/YALL1/
```

It has a simple Matlab interface with 3 inputs and 1 output:

```
x = yall1(A, b, opts);
```

where the input A is either a matrix $A \in \mathbb{C}^{m\times n}$ or a structure of two function handles (see more information below), $b \in \mathbb{C}^m$ and the output is $x \in \mathbb{C}^n$. All these quantities can be real or complex. The input variable `opts` is a structure carrying input options that control the behavior of the algorithm. An optional output, `Out`, can also be requested that contains some secondary output information.

After downloading and unzipping the package, you will have a new directory: `YALL1-xx` where "xx" will vary with YALL1 versions. Get into this directory and run the Matlab script `Run_Me_1st.m`, which sets necessary path and tries to compile a C++ code for fast Walsh-Hadamard transform into a Matlab `mex` file (as such you will need a relevant compiler installed on your system).

Assuming that everything goes as it is supposed, then you will be able to run most of the `demo` files in the `Demos` directory, though a few `demo` files require extra material to be downloaded (see the `Readme.txt` file in the directory for more details).

Among the 3 input variables of YALL1, b is an m-vector, and A can be (i) an $m \times n$ matrix, (ii) a Matlab structure, or (iii) a Matlab function handle. In case (ii), the Matlab structure, say `A`, should have two fields:

```
A.times -- a function handle for A*x,
A.trans -- a function handle for A'*y,
```

representing the action of an $m \times n$ matrix. That is, `A.times`$(\cdot)$ is a linear function from $\mathbb{C}^n$ to $\mathbb{C}^m$, and `A.trans`$(\cdot)$ from $\mathbb{C}^m$ to $\mathbb{C}^n$. For an example on how such a structure `A` is defined, see the function `pdct_operator.m` in the `Utilities` directory. Case (iii) is added to YALL1 version beta-5, where the Matlab function represented by a handle `A` should have two modes:

```
A(x,1) represents A*x,
A(x,2) represents A'*y.
```

The input variable `opts` will be specified in detail later. Here we just mention that the option `opts.tol` specifies the stopping tolerance and must be set by user.

The YALL1 directory contains a simple Matlab script, `quick_start.m` that invokes YALL1 to solve a random problem. Version beta-5 adds another script `quick_start2.m` that demonstrates the use of partial discrete Walsh-Hadamart matrix as sensing matrix, and the use of the discrete cosine transform (DCT) matrix as a sparsifying basis.

3 How to Specify a Model?

As is mentioned, YALL1 can be applied to 6 models: BP, L1/L1, L1/L2, BP+, L1/L1+ and L1/L2+. A model is selected through the input Matlab structure `opts`.

- **BP Model.** By default, YALL1 solves the BP model (1) without the need for any specifications.

- **L1/L1 Model.** To solve the L1/L1 model (2), assign a positive value to the parameter ν represented by the field `opts.nu`. For example, set `opts.nu = 0.5`. Experiments show that usually $\nu < 1$ should be advisable. As is noted earlier, L1/L1 reduces to BP whenever ν is sufficiently small (but how small is small is problem-dependent). Also, assigning zero to the field `opts.nu` has the same effect as not setting this field at all.

- **L1/L2 Model.** To solve the L1/L2 model (3), assign a positive value to the parameter ρ represented by the field `opts.rho`. For example, set `opts.rho = 1e-3`. Usually, the ρ value should be chosen fairly close to zero.

  (The behavior of the mixed model has not been carefully studied. Although setting both $\nu > 0$ and $\rho > 0$ at the same time is allowable, it is not advisable at this point.)

- **Models with Nonnegativity.** Simply set `opts.nonneg = 1`. For example, the L1L1+ model (5) is solved when `opts.nu = 0.5` and `opts.nonneg = 1`. If you set `opts.nonneg = 1` and also use a sparsifying basis W, then you are really assuming that Wx is nonnegative.

- **Weighted L1-Norm.** By default, YALL1 assumes the uniformly weighted L1-norm:

  $$w_i = 1, \quad i = 1, \cdots, n.$$

  To specify a non-unformly weighted L1-norm, set `opts.weights = w;` where $w \in \mathbb{R}^n$ can be any nonnegative vector.

- **Sparsifying Basis.** A sparsifying basis is a unitary matrix $W \in \mathbb{C}^{n\times n}$. In YALL1, such a W is represented by a structure `opts.basis` with two fields:

  ```
  opts.basis.times -- a function handle for W*x,
  opts.basis.trans -- a function handle for W'*x.
  ```

  See the function `wavelet_basis.m` in the `Utilities` directory for an example. Please be cautioned that not every wavelet transform in the Matlab Wavelet Toolbox can be represented by a unitary matrix or be applied to complex vectors.

- **General Matrix** A. In YALL1 version beta-4, if A does not satisfy $AA^* = I$, set `opts.nonorth = 1`.

4 List of Input Options

The following are the input options that can be reset by the user. The values in the brackets "[ ]" are default values.

```
opts.tol = [set by user]      (stopping tolerance)
opts.print = [0],1,2;         (levels of printout)
opts.maxit = [9999];          (maximum iterations)
opts.nu = [0];                (> 0 for L1/L1 model)
opts.rho = [0];               (> 0 for L1/L2 model)
opts.nonneg = [0];            (1 for nonnegativity)
opts.basis = [identity];      (sparsifying basis W)
opts.mu = [set by code]       (penalty parameter)
opts.weights = [1];           (weights for L1-norm)
opts.nonorth = [0];           (set to 1 if A*A' =/= I)
opts.stepfreq = [1];          (see explanation below)
```

The penalty parameter μ can be altered by `opts.mu`, but this should be done only when the code cannot satisfactorily solve your problem using its default value for μ.

The option `opts.nonorth` is available after version beta-4 (for specifying that $AA^* \neq I$), and `opts.stepfreq` after beta-5 (see below).

4.1 Tolerance Value

The option `opts.tol` specifies the stopping tolerance and must be set by the user. To get the best performance, its value should be set more or less consistent with the noise levels in the observed data b. In most practical situations where noise inevitably exists, `opts.tol` values between 1E-2 to 1E-4 should generally be sufficient. Setting a tolerance value much lower than noise level would only prolong computational time without the benefit of getting a higher accuracy.

4.2 Step Calculation Frequency ($AA^* \neq I$)

A new option `opts.stepfreq` is added after version beta-5 that specifies the frequency of calculating the exact steepest descent step-length when $AA^* \neq I$ (which requires computing AA^*v for a vector v). The default value for `opts.stepfreq` is 1, meaning that the exact step-length calculation is preformed at every iteration. Setting `opts.stepfreq` to integers greater than 1 would reduce computational time, but increase the risk of non-convergence.

(Our computational experience suggests that `opts.stepfreq = 10` could still work well on average problems, but for difficult problems even `opts.stepfreq = 2` could cause failure.)

5 Feedbacks

The author welcomes and appreciates feedbacks from users. Please send your bug reports and suggestions to the email address: `yzhang@rice.edu`.

Appendix: A 3-line Algorithm for 6 Problems

YALL1 versions beta-3 and beta-4 utilize a single and simple algorithm to solve all the six models (1)–(6), hence a one-for-six algorithm. It is derived from the classic approach of Alternating Direction Method, or ADM, that minimizes augmented Lagrangian functions through an alternating minimization scheme and updates the multiplier after each sweep. The convergence of such algorithms has been well analyzed in the literature (see [2], for example, and the references thereof). Such algorithms can also be characterized as *first-order primal-dual algorithms* because they explicitly update both primal and dual variables at every iteration.

ADM-like approaches can be applied to models (1)–(6) and their duals, potentially leading to a large number of possible first-order primal-dual algorithm variants. A study of a few such first-order primal-dual algorithms will be presented in a forthcoming paper [3].

In the sequel, let us assume that $W = I$; otherwise a change of variable $\hat{x} = Wx$ would bring the problem into the desired form with the matrix A replaced by AW^* that still has orthonormal rows.

L1/L1 "$\subset$" BP

We first demonstrate that the L1/L1 model (2) can be readily solved as a BP problem. Note that (2) (with $W = I$) has an equivalent form

$$\min_{x\in\mathbb{C}^n, r\in\mathbb{C}^m} \left\{ \|x\|_{w,1} + \frac{1}{\nu}\|r\|_1 : Ax + r = b \right\},$$

or, after a change of variable and multiplying both sides by a constant,

$$\min_{(x,r)\in\mathbb{C}^{n+m}} \left\{ \left\| \begin{pmatrix} x \\ r \end{pmatrix} \right\|_{\hat{w},1} : \frac{[\,A \quad \nu I\,]}{\sqrt{1+\nu^2}} \begin{pmatrix} x \\ r \end{pmatrix} = \frac{b}{\sqrt{1+\nu^2}} \right\}. \tag{A-1}$$

Model (A-1) has the same form as the BP model (1), but with (A, b, x) replaced by (A_ν, b_ν, x_r) and $w \in \mathbb{R}^n$ augmented to $\hat{w} \in \mathbb{R}^{n+m}$ (here weighted L1-norm for fidelity can be easily handled as well), where

$$A_\nu = \frac{\left[\begin{array}{cc} A & \nu I \end{array}\right]}{\sqrt{1+\nu^2}}, \quad b_\nu = \frac{b}{\sqrt{1+\nu^2}} \quad \text{and} \quad x_r = \left(\begin{array}{c} x \\ r \end{array}\right). \tag{A-2}$$

The new coefficient matrix A_ν has the property that

$$AA^* = I \implies A_\nu A_\nu^* = I, \tag{A-3}$$

which has a nice consequence in our implementation. Consequently, the L1/L1 model (2) can be solved by any algorithm for the BP model (1).

Dual BP "⊂" Dual L1/L2

It is well known that as $\rho \to 0$, the solution of the L1/L2 model (3) converges to that of the BP model (1). On the other hand, when we consider the duals of the two, then the dual of (3) reduces to that of (1) exactly when $\rho = 0$. To see this relationship, we only need to derive and compare the dual of (3),

$$\max_{y \in \mathbb{C}^m} \{\mathbf{Re}(b^* y) - \frac{\rho}{2}\|y\|_2^2 : A^* y \in \mathbf{B}_w\}, \tag{A-4}$$

with that of (1),

$$\max_{y \in \mathbb{C}^m} \{\mathbf{Re}(b^* y) : A^* y \in \mathbf{B}_w\}, \tag{A-5}$$

where

$$\mathbf{B}_w \triangleq \{z \in \mathbb{C}^n : |z_i| \le w_i,\ i = 1, \cdots, n\}. \tag{A-6}$$

Therefore, one can use a single algorithm to solve both (A-4) with $\rho > 0$ and (A-5) with $\rho = 0$, while the latter contains model (2) as a special case.

Models with Nonnegativity

Models with nonnegativity can be treated in a totally analogous manner. The dual of the L1/L2+ model (6) and the BP+ model (4) have exactly the same forms as those of (A-4) and (A-5), respectively, except that the definition of $\mathbf{B}_w$ is changed to

$$\mathbf{B}_w \triangleq \{z \in \mathbb{R}^n : z_i \le w_i,\ i = 1, \cdots, n\}. \tag{A-7}$$

We note that a minor modification to the above definition of $\mathbf{B}_w$ is required if a BP+ model is from an L1/L1+ model where the variable is x_r defined as in (A-2), consisting of two parts in which the x-part is nonnegative but the r-part is not.

A crucial fact is that the projections onto these slightly different sets $\mathbf{B}_w$ are all trivial and inexpensive because they are all "boxes".

Your 1-for-6 Algorithm

Models (1)-(6) can be solved by a single algorithm. Such an algorithm can be called *one-for-six (or 1-for-6): one algorithm solving six problems.* (In fact, this algorithm can solve more than 6 models if one counts the mixed model where both $\nu > 0$ and $\rho > 0$).

A number of 1-for-6 algorithms can be derived. The main algorithm used in YALL1 beta versions is derived from applying ADM to an augmented Lagrangian function of the dual program (A-4) with a penalty parameter $\mu > 0$. Let P_w be the orthogonal projection onto the box $\mathbf{B}_w$ where the definition of $\mathbf{B}_w$ can vary. The algorithm in YALL1 version beta-3 has the basic form:

$$\begin{aligned}
y^{k+1} &= \alpha A z^k - \beta(Ax^k - b), && \text{(A-8a)}\\
z^{k+1} &= P_w\left(A^* y^{k+1} + x^k/\mu\right), && \text{(A-8b)}\\
x^{k+1} &= x^k + \gamma\mu\left(A^* y^{k+1} - z^{k+1}\right), && \text{(A-8c)}
\end{aligned}$$

where $\gamma \in (0, (1+\sqrt{5})/2)$,

$$\alpha = \frac{\mu}{\mu+\rho} \quad \text{and} \quad \beta = \frac{1}{\mu+\rho}. \tag{A-9}$$

This algorithm is a *first-order primal-dual algorithm* since that the primal variable x and dual variables y and z (a dual slack) are updated at every iteration. The algorithm used in YALL1 version beta-4 is a modifed version of the above algorithm that allows general A-matrices not necessarily satisfying $AA^* = I$.

Depending on the definition of $\mathbf{B}_w$, when $\rho > 0$ the algorithm solves either the L1/L2 or the L1/L2+ model; and when $\rho = 0$ it solves either the BP or the BP+ model if $\nu = 0$, or either the L1/L1 or the L1/L1+ model if $\nu > 0$ and (A, b) is replaced by (A_ν, b_ν). (Is this 1-for-6 algorithm wonderful? At the very least, it saved me tons of programming time.)

As is mentioned earlier, a number of other first-order primal dual algorithms can be derived in which primal and dual variables are alternatingly updated. Some of these algorithms will be presented in a forthcoming paper [3]. We plan to implement other algorithms in YALL1 once we determine that they can bring enhanced performance and/or further problem-solving capacities.

Acknowledgments

The author would like to thank a group of users, colleagues and students inside and outside of Rice University who helped test previous versions of the code and provided useful feedbacks.

References

[1] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic Decomposition by Basis Pursuit. SIAM J. Scientific Computing 20: 33-61, 1998.

[2] R. Glowinski. Numerical Methods for Nonlinear Variational Problems. Springer-Verlag, New York, Berlin, Heidelberg, Tokyo, (1984).

[3] J. Yang and Y. Zhang. A Study of Algorithms and Models for Sparse Solution Recovery via ℓ_1-Minimization. In preparation.