

# l1\_ls: A Matlab Solver for Large-Scale $\ell_1$ -Regularized Least Squares Problems

Kwangmoo Koh  
deneb1@stanford.edu

Seungjean Kim  
sjkim@stanford.edu

Stephen Boyd  
boyd@stanford.edu

May 15, 2008

`l1_ls` solves  $\ell_1$ -regularized least squares problems (LSPs) using the truncated Newton interior-point method described in [KKL<sup>+</sup>07].

## 1 The problems

`l1_ls` solves an optimization problem of the form

$$\text{minimize} \quad \|Ax - y\|^2 + \lambda \|x\|_1, \quad (1)$$

where the variable is  $x \in \mathbf{R}^n$  and the problem data are  $A \in \mathbf{R}^{m \times n}$  and  $y \in \mathbf{R}^m$ . Here,  $\lambda \geq 0$  is the regularization parameter. We refer to the problem (1) as an  $\ell_1$ -regularized least squares problem.

`l1_ls` can also solve  $\ell_1$ -regularized LSPs with nonnegativity constraints:

$$\begin{aligned} &\text{minimize} \quad \|Ax - y\|^2 + \lambda \sum_{i=1}^n x_i \\ &\text{subject to} \quad x_i \geq 0, \quad i = 1, \dots, n. \end{aligned} \quad (2)$$

## 2 Calling sequences

The package `l1_ls` has two solver files, `l1_ls.m` and `l1_ls_nonneg.m`: `l1_ls.m` solves problem (1) and `l1_ls_nonneg.m` solves problem (2). Both solvers supports two calling styles. The simple calling style works when  $A$  is given as a matrix. The more complex calling style handles the case when  $A$  and its transpose are given as operators.

The simple calling sequence of `l1_ls` is

```
>> [x,status] = l1_ls(A,y,lambda,rel_tol,quiet);
```

The more complex calling sequence of `l1_ls` is

```
>> [x,status] = l1_ls(A,At,m,n,y,lambda,rel_tol,quiet);
```

In both cases, the last two arguments are optional. Input arguments represent the problem data of (1). Output arguments are the optimal point (up to the given tolerance) and its status.

`l1_ls_nonneg.m` has the same calling styles as `l1_ls.m`.

## 2.1 Input arguments

- **A**: data matrix with  $n$  columns and  $m$  rows. For the simple calling sequence, **A** is a matrix in dense or sparse format. For the complex calling sequence, **A** is a Matlab object with which we can evaluate  $\mathbf{A}*\mathbf{z}$  with a vector  $\mathbf{z}$  in  $\mathbf{R}^n$  by overloading the multiplication operator.
- **At**: transpose of **A**. For the complex calling sequence, **At** is a Matlab object with which we can evaluate  $\mathbf{At}*\mathbf{w}$  with a vector  $\mathbf{w}$  in  $\mathbf{R}^m$  by overloading the multiplication operator.
- **m**: number of observations.
- **n**: number of unknowns.
- **y**: vector of length  $m$ .
- **lambda**: regularization parameter.
- **rel\_tol**: target relative tolerance, defined as duality gap divided by the dual objective value, which is an upper bound on the relative suboptimality.
- **quiet**: boolean. Suppresses print messages during execution if **true**. The default value is **false**.

For the simple calling sequence, **m** and **n** are obtained internally from the size of **A**.

## 2.2 Output arguments

- **x**:  $n$ -vector. If **status** is 'Solved', **x** is a solution (up to the given relative tolerance); otherwise it is the last iterate of the truncated Newton interior-point method. (If **status** is 'Failed', **x** is the last value before the failure.)
- **status**: string; possible values are 'Solved' and 'Failed'.

## 3 Hints and caveats

Here are some hints on using `l1_ls`.

- If your problem is large and sparse, be sure that **A** (and **At**, if you use the complex calling style) are in sparse format.

- If  $A$  and  $At$  are dense but there are fast algorithms for the matrix-vector products  $A*z$  and  $At*w$ , be sure that  $A$  and  $At$  are Matlab operators and the fast algorithms are supported by overloading the multiplication operator  $*$ , using the object-oriented programming of Matlab. For example, when  $A$  is a matrix formed by sampling  $m$  rows of the discrete cosine transform (DCT) matrix  $F \in \mathbf{R}^{n \times n}$ ,  $A*z$  can be computed by using the fast DCT algorithm, and  $At*w$  can be by using the fast inverse transform algorithm.

Now, a caveat. The truncated Newton interior-point method can work poorly (*i.e.*, be slow) when the regularization parameter is too small (which corresponds to the case when the solution  $x$  is not very sparse), and the mutual coherence of the data matrix  $A$ ,

$$\mu = \max_{i \neq j} \frac{|A_i^T A_j|}{\|A_i\| \|A_j\|}$$

where  $A_i$  is the  $i$ th column of  $A$ , is high (*i.e.*, near 1). We recommend keeping  $\lambda/\lambda_{\max}$  greater than  $10^{-4}$  or so.

## 4 Utility functions

The utility function `find_lambdamax_l1_ls.m` computes  $\lambda_{\max}$  of  $\ell_1$ -regularized LSP (1). For any larger value of  $\lambda$ , the optimal solution obtained from  $\ell_1$ -regularized least squares is zero. The function `find_lambdamax_l1_ls_nonneg.m` computes  $\lambda_{\max}$  of  $\ell_1$ -regularized LSP with nonnegativity constraints (2).

## 5 Examples

### 5.1 A small example for the simple calling sequence

We give a simple example to illustrate the usage of the simple calling sequence, with the problem data

$$A = \begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 0.2 & 0.3 \\ 0 & 0.1 & 1 & 0.2 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 0.2 \\ 1 \end{bmatrix}.$$

The Matlab code for finding a point which is guaranteed to be no more 1%-suboptimal is shown below.

```
>> % Matlab script for solving the simple problem given above.
>> A = [1 0 0 0.5;... \
        0 1 0.2 0.3;... \
        0 0.1 1 0.2];
>> x0 = [1 0 1 0]'; % original signal
```

```

>> y = A*x0;           % measurements with no noise
>> lambda = 0.01;      % regularization parameter
>> rel_tol = 0.01;     % relative target duality gap

[x,status]=l1_ls(A,y,lambda,rel_tol);

```

After executing the code, you can see the result by typing `x` in Matlab.

```

>> x

x =

    0.9930
    0.0004
    0.9941
    0.0040

```

## 5.2 An example for the complex calling sequence

We give a simple example to illustrate the usage of the complex calling sequence that involves the object-oriented programming feature of Matlab. We consider a sparse signal recovery problem with a signal  $x \in \mathbf{R}^{1024}$  which consists of 10 spikes with amplitude  $\pm 1$ , shown at the top plot of figure 1. Suppose we observe

$$y = Ax + v \in \mathbf{R}^{128}$$

where  $v$  is drawn according to the Gaussian distribution  $\mathcal{N}(0, 0.01^2 I)$  on  $\mathbf{R}^{128}$ . Here,  $Ax$  gives the discrete cosine transform of  $x$  at  $m = 128$  frequencies, chosen from uniform distribution over the indices  $1, \dots, 1024$ .

The Matlab code for finding a point which is guaranteed to be no more than 1%-suboptimal is shown below.

```

>> % Matlab script for solving the sparse signal recovery problem
>> % using the object-oriented programming feature of Matlab.
>> % The three m files in ./@partialDCT/ implement the partial DCT class
>> % with the multiplication and transpose operators overloaded.
>>
>> rand('state',0);randn('state',0); %initialize (for reproducibility)
>>
>> n = 1024; % signal dimension
>> m = 128; % number of measurements
>>
>> J = randperm(n); J = J(1:m); % m randomly chosen indices
>>

```

```

>> % generate the m*n partial DCT matrix whose m rows are
>> % the rows of the n*n DCT matrix at the indices specified by J
>> % see files at @partialDCT/
>> A = partialDCT(n,m,J); % A
>> At = A'; % transpose of A
>>
>> % spiky signal generation
>> T = 10; % number of spikes
>> x0 = zeros(n,1);
>> q = randperm(n);
>> x0(q(1:T)) = sign(randn(T,1));
>>
>> % noisy observations
>> sigma = 0.01; % noise standard deviation
>> y = A*x0 + sigma*randn(m,1);
>>
>> lambda = 0.01; % regularization parameter
>> rel_tol = 0.01; % relative target duality gap
>>
>> %run the l1-regularized least squares solver
>> [x,status]=l1_ls(A,At,m,n,y,lambda,rel_tol);

```

After executing the code, you can see the result by typing x in Matlab.

```

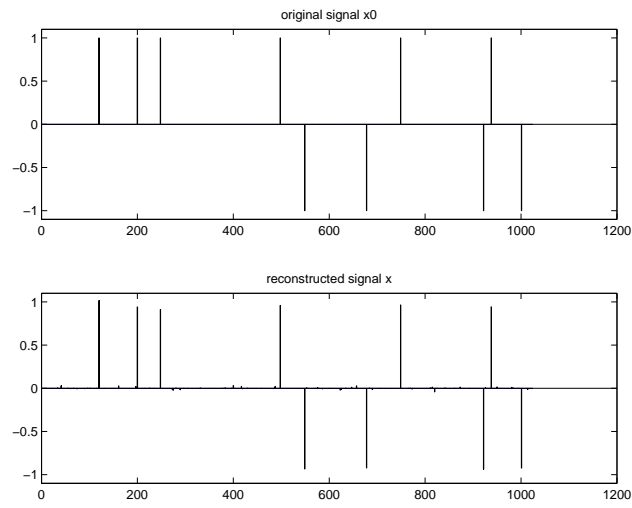
>> figure(1)
>> subplot(2,1,1); bar(x0); ylim([-1.1 1.1]); title('original signal x0');
>> subplot(2,1,2); bar(x); ylim([-1.1 1.1]); title('reconstructed signal x');

```

Figure 1 shows the result.

## References

- [KKL<sup>+</sup>07] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. A method for large-scale  $l_1$ -regularized least squares. *IEEE Journal on Selected Topics in Signal Processing*, 1(4):606–617, 2007.



**Figure 1:** Reconstruction results. Original spike signal (Top). Reconstructed signal via  $\ell_1$ -regularized least squares (Bottom).