User Manual
FPC_AS
A MATLAB Solver for ℓ_1-Regularized Least Squares Problems

Zaiwen Wen
Department of Industrial Engineering and Operations Research, Columbia University
Wotao Yin
Department of Computational and Applied Mathematics, Rice University

Version 1.0, September, 2008

1 Summary and History

FPC_AS stands for fixed-point continuation and active set. It solves the ℓ_1-regularized minimization problem

$$\min_{x \in \mathbb{R}^n} \quad \psi_\mu(x) := \mu\|x\|_1 + \frac{1}{2}\|Ax - b\|_M^2, \tag{1.1}$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $M \in \mathbb{R}^{m\times m}$, and $b \in \mathbb{R}^m$, and $\mu > 0$ is the regularization parameter. It is based upon the active-set algorithm with a continuation strategy described in [1, 2].

FPC_AS is a successor of **FPC** [3]. While **FPC_AS** still performs shrinkage iterations and continuation as its predecessor, most of the code has been rewritten. Compared to **FPC**, which has good performance on large-scale problems with highly sparse solutions, **FPC_AS** works better overall and much better on certain difficult problems arising in *compressed sensing*, to name a few, those with sparse, but not highly sparse, solutions and those whose solutions have both very large and very small nonzero components (i.e., the solutions have huge dynamic ranges). In the solutions of these problems, there are certain nonzero components difficult to identify because they are either too small or have only slight advantage to represent b over some of the others. **FPC_AS** was designed with active set identification and sub-optimization to help recover these components in the solutions.

2 Installation

To install the package, please follow the instructions the file "README.m".

3 Usage

- The calling sequence of **FPC_AS** at the MATLAB command line is

```
>> [x,Out] = FPC_AS(n,A,b,mu,M,opts);
```

  The last two input arguments are optional. The input and output arguments are described below in Subsections 3.2 and 3.3.

- A quick test:

```
>> cd FPC_AS_folder      % replace FPC_AS_folder by the actual folder of FPC_AS
>> addpath(genpath(pwd));
>> one_run;
```

- Run a test on a set of difficult problems:

```
>> Test_Difficult_Problems;
```

3.1 Important notices

- **FPC_AS** usually works better if the maximum eigenvalue of $A^\top MA$ is close to, but no larger than, 1. If it is larger than 1, $\frac{1}{2}||Ax-b||_M^2$ will dominate $\|x\|_1$, and this may affect the performance of the algorithm. To make sure the maximum eigenvalue of $A^\top MA$ is 1, please either scale the input argument A by letting $A \leftarrow \theta A$ for some appropriate $\theta < 1$ or leave this to **FPC_AS** by setting opts.scale_A = 1.
- The default values of certain solver options given below in Subsection 3.2 are set to solve problem (1.1) to a high accuracy. However, it is sometimes NOT necessary to solve problem (1.1) very accurately, for example, when 'b' is contaminated by noise and/or μ is relatively large. In these cases, the default values will be over tight, so please relax them for better performance. For instance, for problems with noisy 'b' and highly sparse solutions, one can use

  ```
  opts.sub_mxitr = 10; opts.gtol = 1e-3; opts.gtol_scale_x = 1e-6;
  ```

  The performance of **FPC_AS** is quite sensitive to these options; see next subsections for their meanings.

3.2 Input arguments

- n: the dimension of x. The number of rows of A must equal n.
- A: either an explicit $m \times n$ matrix or an A_operator object representing a matrix implicitly. When the operations A*x and A.'*x can be computed much faster through certain means, it is recommend that A be created as an A_operator object, the source code of which is provided with the solver. To create an A_operator object, two functions or function handles for computing A*x and A.'*x, respectively, must be given. Suppose they are AA and AT,
  - AA: a function handle such that AA(x) = A*x,
  - AT: a function handle such that AT(x) = A.'*x.

  Then A can be created as an A_operator by

  ```
  A = A_operator(@(x) AA(x), @(x) AT(x));
  ```

  An example for A being an implicit partial DCT matrix, which performs a complete DCT but returns only the subset of the results corresponding to omega, is

  ```
  function y=pdct(x,picks); y=dct(x); y=y(picks); end
  function y=pidct(x,n,picks); y=zeros(n,1); y(picks)=x; y=idct(y); end
  A = A_operator(@(x) pdct(x,omega), @(x) pidct(x,n,omega));
  ```

- b: an $m \times 1$ vector
- mu: the ℓ_1 regularization parameter μ
- M: either an $m \times m$ positive definite matrix or the empty matrix [ ]. If M=[ ], **FPC_AS** treats M = I, which reduces the last term in (1.1) to $\frac{1}{2}||Ax-b||_2^2$.

- opts: a structure of options. It is an optional argument, so it can be ignored or set empty. Some of the frequently used fields include:
  - 'mxitr': max number of iterations
    default: 1000, valid range: [1, 100000]
  - 'gtol': termination criterion on "crit2", the maximum norm of sub-gradient, where the meaning of "crit2" is described in subsection 3.3
    default: 1e-06, valid range: [0, 1]
  - 'gtol_scale_x': termination criterion on "crit2" scaled by max(norm(x), 1).
    default: 1e-12, valid range: [0, 1]
  - 'f_value_tol': Tolerance on the optimal objective value. Stop if $\psi_\mu(x)$ less than or equal to f_value_tol.
    default: 0, valid range: [0, inf]
  - 'sub_mxitr': max number of iterations in each sub-optimization
    default: 80, valid range: [1, 100000]
  - 'sub_opt_meth': choice of sub-optimization methods
    default: 'lbfgs', valid values: {'lbfgs','lbfgsb','pcg'};
  - 'scale_A': on/off switch for scaling the input matrix A so that the max of eigs(A*A.') equals 1
    default: 0, valid range: {0, 1}
  - 'minK': an estimate of the number of nonzero components in optimal solution
    default: m/2, valid range: [1, n]
  - 'zero': a lower bound of minimal magnitude of nonzero components of optimal solution
    default: 1e-08, valid range: [0, 1e+10]
  - 'dynamic_zero': on/off switch for setting 'zero' dynamically
    default: 0, valid range: {0, 1}
  - 'xs': optimal solution for non-algorithmic purposes such as progress display
    default: empty vector, valid range: [-Inf, Inf]
  - 'record': print level, -1=quiet, 0=some output, 1=more output.
    default: 0, valid range: {-1,0, 1}
  - 'PrintOptions': print options, 0=quiet, 1=output
    default: 0, valid range: {0, 1}

  To set a field of opts, do "opts.[fieldname] = value'.

  A complete list of options are given in Appendix B.

3.3 Output arguments

- x: exit solution, which is the point obtained at last iteration
- Out: a structure having the following fields
  - cpu: total CPU time
  - exit: exit flag, 1='normal', 10='max number of iterations reached'
  - mesg: message of exit status
  - itr: number of iterations taken
  - f: exit function value, i.e., $\mu||x||_1 + \frac{1}{2}||Ax - b||_M^2$
  - nrm1x: exit ℓ_1 norm, i.e., $||x||_1$
  - rNorm: exit l_2 discrepancy term, i.e., $||Ax - b||_M$
  - g: exit gradient of $\frac{1}{2}||Ax - b||_M^2$

- zero: final tolerance for zero, which is used for computing termination criteria
- crit2: violation of optimality, which is computed as

  ```
  nz_x = x>Out.zero; z_xa = ~nz_x & (||g|-mu| > Out.zero);
  T = union(nz_x, z_xa); crit2 = norm(g(T)-mu,'inf');
  ```

  Above nz_x is the set of nonzero components whose magnitude of x are larger than Out.zero and z_xa is the set of nonzero components whose magnitude of $|g| - \mu$ are larger than Out.zero
- nnz_x: number of the components in x whose magnitudes are larger than Out.zero
- nCont: number of continuation steps taken
- nSubOpt: number of sub-optimization problems solved
- nProdA, nProdAt: total number of operations A*x and A.'*x, respectively
- nfe, nge: number of A*x and A.'*x performed in shrinkage, respectively.
- nfe_sub, nge_sub: number of A*x and A.'*x performed in sub-optimization
- opts: options used
- The following fields are available if an optimal solution opts.xs is provided in the input. opts.xs is compared to x after x is truncated in the way that $x_i = 0$ if $x_i \leq$ Out.zero. The comparison results are given in
  - sgn: number of nonzero components of x that have different sign compared to those of opts.xs,
  - miss: number of (missed) components that are zero in x but nonzero in opts.xs
  - over: number of (overshot) components that are nonzero in x but zero in opts.xs

  They are computed as

  ```
  jnt  = union(nz_xs, nz_x); nz_xs = abs(xs) > opts.eps;
  sgn  = nnz(sign(x(jnt))~=sign(xs(jnt)));
  miss = nnz(nz_xs&(~nz_x))
  over = nnz(nz_x&(~nz_xs))
  ```

3.4 Auxiliary routines

- The operator "A_operator" is defined in the subdirectory "prob_gen\classes"
- A collection of test problems are stored in the subdirectory "prob_gen"
  - A random problem generator: "getData.m". Please see the code for a description.
  - Six problems in the ".mat" format with variables "n","Omega", "b" and "xs", where b=A*xs and the matrix A is the partial DCT matrix whose rows are select from the DCT matrix with indices "Omega". The four problems: "CaltechTest1", "CaltechTest2", "CaltechTest3" and "CaltechTest4" are provided by [6] and the two problems "Ameth6Xmeth2seed200" and "Ameth6Xmeth6seed200" are from [2].

4 Examples

4.1 An explicit matrix A example

In the following example, an explicit matrix A and vector b are generated by the auxiliary function getData.m, which is able to generate a variety of test data. A usage instruction is given in the file.

```
mu=1e-10;
seed = 200;
n = 2^9;
delta = 0.5;                % m/n, m = round(delta*n)
rho = 0.3;                  % k/m, k = round(rho*m)
Ameth = 0;                  % see getData.m for codes
xmeth = 1;                  %  "      "      "    "
% set noise level
sigma1 = 0;    %- standard deviation of signal noise (added to xs)
sigma2 = 0;    %- standard deviation of meas. noise (added to b)
% problem size
m = round(delta*n); k = round(rho*m);
% initialization, get problem
[A,b,xs,xsn] = getData(m,n,k,Ameth,xmeth,sigma1,sigma2,seed); M = []; opts.xs = xs;
% call FPC_AS
opts.gtol = 1e-8;
[x, Out] = FPC_AS(n,A,b,mu,M,opts);
```

The recovered solution is depicted in Figure 1, and the screen output is

```
solver: /home/code/FPC_code/FPC_AS
Problem information: n=512, m=256, K=77
abs.err=1.37e-08, rel.err2=1.09e-08, nnz(x)=79, sgn=0, miss=0, over=2
time:  0.480s, crit2: 9.03e-10, nrm1x: 3.69e+01, ||Ax-b||: 1.18e-08
cost: num. of shrinkage: 68, num. of sub-opt: 7, num. of continuation: 6
num. of A*x from (total, shrink, sub-opt): (289, 69, 220), num. of A'*x: (296, 76, 220)
Message: sub-opt optimal
```

The above message indicates that the output solution, compared to the true sparse solution xs, has a absolute error of 1.37e-8, a relative error of 1.09e-8, and has two extra nonzero entries. This solution was obtained through 68 shrinkage iterations, 7 sub-optimization problems, and 6 continuation steps. The computation was dominated by a total number of 289 and 296 operations in the form of A*x and A'*x, respectively. Among the 289 A*x operations, 69 were performed in shrinkage iterations, and 220 in sub-optimization. Among the 296 A'*x operations, 76 and 220 were performed in shrinkage iterations and sub-optimization, respectively. The total CPU time was 0.48 second.

4.2 An A_operator example

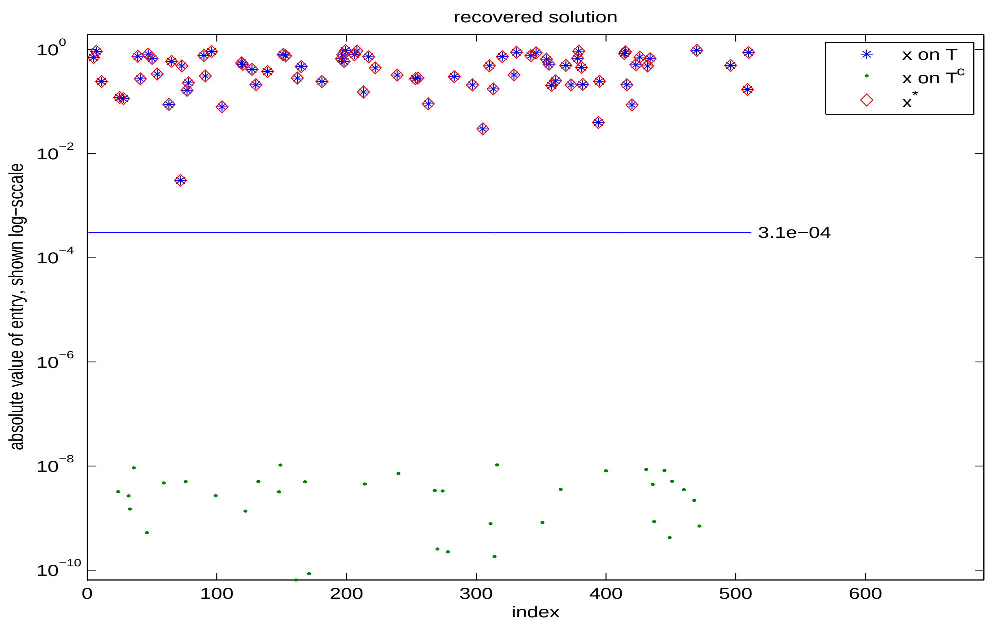
In this example, an implicit matrix A will generated as an A_operator object. This implicit matrix A is a partial DCT matrix, whose matrix-vector products in the form of A*x and A'*x are computed by the function pdct. pdct(·,1,n,Omega) and pdct(·,2,n,Omega) computes A*· and A'*·, respectively. The data of this test problem were provided in [6].

```
mu=1e-10;
% set up problem
load('CaltechTest3', 'b','Omega','n','xs');
A = A_operator(@(x) pdct(x,1,n,Omega), @(x) pdct(x,2,n,Omega)); M = []; opts.xs = xs;
% call FPC_AS
opts.gtol = 1e-14;
[x, Out] = FPC_AS(n,A,b,mu,M,opts);
```

The recovered solution is depicted in Figure 2, and the screen output is

```
solver: /home/code/FPC_code/FPC_AS
Problem information: n=512, m=128, K=32
abs.err=5.85e-10, rel.err2=1.41e-09, nnz(x)=32, sgn=0, miss=0, over=0
```

Figure 1: recovered solution for the example in section 4.1. T is the support of the exact solution and T^c is the complement of T.



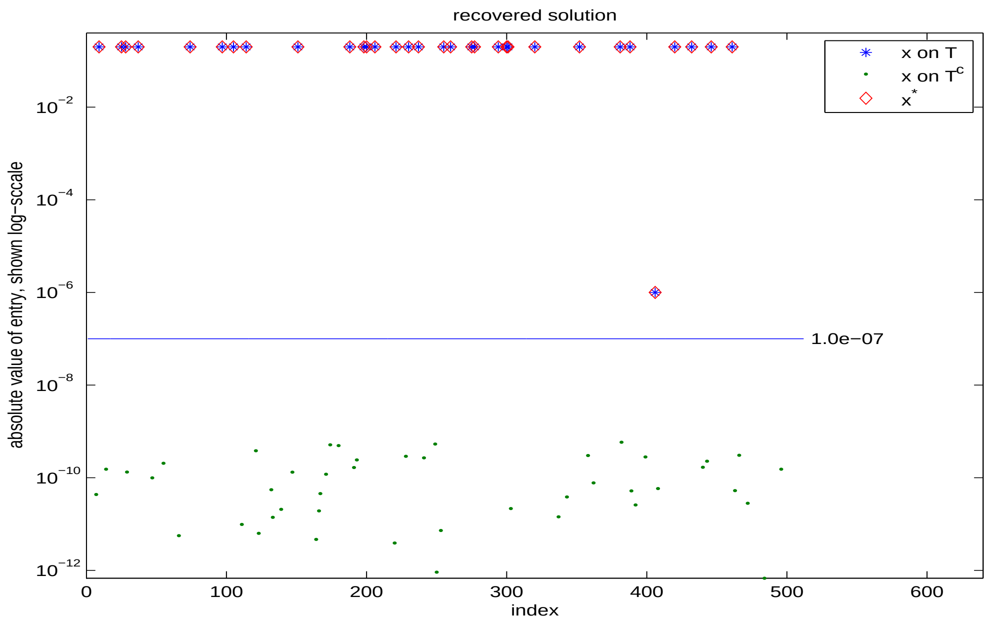
```
time:  0.460s, crit2: 9.33e-12, nrm1x: 6.20e+00, ||Ax-b||: 1.29e-09
cost: num. of shrinkage: 61, num. of sub-opt: 4, num. of continuation: 4
num. of A*x from (total, shrink, sub-opt): (235, 62, 173), num. of A'*x: (235, 62, 173)
Message: shrinkage optimal
```

5 License

A Preliminary Numerical Results

The purpose of this section is to demonstrate the recoverability of **FPC_AS** on problems which might be "pathological". The first test set has four problems ("CaltechTest1", "CaltechTest2", "CaltechTest3" and

Figure 2: recovered solution for the example in section 4.2. T is the support of the exact solution and T^c is the complement of T.



"CaltechTest4") from [6]. These problems are difficulty because the magnitudes of the components fall into a big range, i.e., the largest nonzero magnitudes are significantly larger than the smallest nonzero magnitudes. We summarize the dimension of these four problems and the magnitudes of the exact solutions in table 1. In the last column of the table, we show the pairs of the magnitude of the exact solution and the number of elements on this level. For example, in CaltechTest3, there are thirty-one components which have a magnitudes of 0.2 and there is one component which has a small magnitude 10^{-6}. The second test set has two problems ("Ameth6Xmeth2seed200" and "Ameth6Xmeth6seed200") which were encountered during our development of **FPC_AS**. Our numerical experience indicates that they might have solutions which are not sparse at least for the l_1-regularized problem (1.1), although we can not confirm our observation theoretically. The coefficient matrix A here is the partial discrete cosine transform (DCT) matrix whose m rows were chosen randomly from the $n \times n$ DCT matrix.

To give an idea of the relative performance of **FPC_AS** on these problems, we did some comparison with the solver **l1eq_pd** (a new version, private communication) in the l_1**-magic** software package [5] and the solver **spg_bp** in the software package **SPGL1** (version 1.5) [4] for solving the basis pursuit problem

$$\text{(A.1)} \qquad \text{(Basis Pursuit)} \qquad \min_{x\in\mathbb{R}^n} \|x\|_1 \text{ subject to } Ax = b.$$

We should pointed out that the comparison here is not meant to be a rigorous assessment of the performance of these three classes of algorithms, as this would require very careful handling of subtle details such as comparable termination criteria, and would be outside the scope of this paper. In addition, a comparison might quickly be out of date since all the three software packages are continuously improved. The main purpose of the comparison here is to encourage readers to consider **FPC_AS** as a potential candidate when looking for a practical solver for Compressed sensing. We set the termination criteria sufficiently small for each solver. Specifically, we set the parameter $pdtol = 10^{-8}$ for **l1eq_pd**, the parameter $bpTol = 10^{-10}$, $optTol = 10^{-10}$, $decTol = 10^{-10}$ and $iteration = 10^4$ for **spg_bp** and the parameter $\mu = 10^{-10}$ and $gtol = 10^{-14}$ for **FPC_AS**. All other parameters of each solver were set to their default values. The termination criteria are not direct comparable due to different formulation of the problems, but we believe that on average the chosen criteria for **FPC_AS** is tighter than those of the other two solvers. All codes were written in MATLAB (Release 7.3.0) and all experiments were performed on a Dell Precision 670 workstation with an Intel Xeon 3.4GHZ CPU and 6GB of RAM.

Table 1: Problem information

Problem	n	m	K	(magnitude, num. of elements on this level)
CaltechTest1	512	128	38	$(10^5, 33)$, $(1, 5)$
CaltechTest2	512	128	37	$(10^5, 32)$, $(1, 5)$
CaltechTest3	512	128	32	$(0.2, 31)$, $(10^{-6}, 1)$
CaltechTest4	512	102	26	$(10^4, 13)$, $(1, 12)$, $(10^{-2}, 1)$
Ameth6Xmeth2seed200	1024	512	154	$(1, 154)$
Ameth6Xmeth6seed200	1024	512	154	$(10^5, 154)$

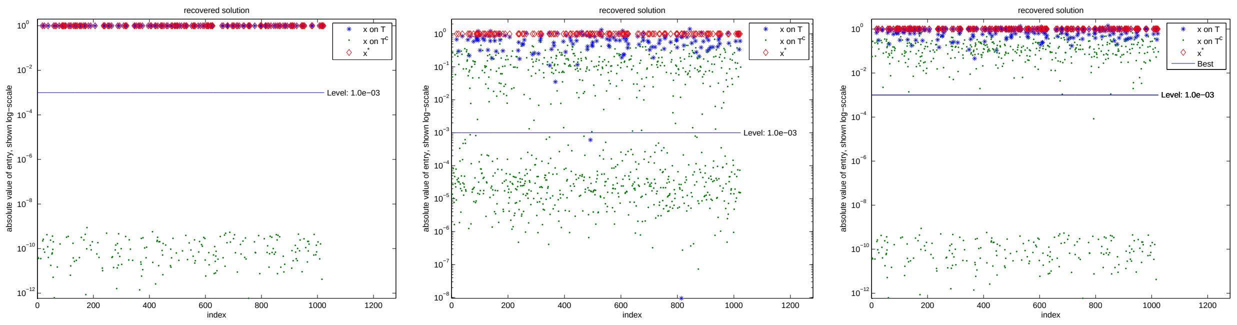
Table 2: Computational results for the difficult problems

Problem	solver	CPU(sec.)	rel.err	$\|x\|_1$	$\|r\|_2$	nMat	(sgn,miss,over)
CaltechTest1	**FPC_AS**	0.570	5.1e-12	3.3e+06	2.1e-09	627	(0, 0, 0)
	l1eq_pd	56.280	5.1e-12	3.3e+06	1.0e-10	91833	(0, 0, 0)
	spg_bp	23.070	3.9e-06	3.3e+06	4.3e-03	29667	(0, 0, 32)
CaltechTest2	**FPC_AS**	0.380	7.1e-14	3.2e+06	1.6e-09	417	(0, 0, 0)
	l1eq_pd	34.720	7.2e-14	3.2e+06	9.6e-11	56177	(0, 0, 0)
	spg_bp	20.820	1.0e-09	3.2e+06	2.2e-05	25775	(0, 0, 0)
CaltechTest3	**FPC_AS**	0.460	1.4e-09	6.2e+00	1.3e-09	471	(0, 0, 0)
	l1eq_pd	7.430	2.5e-09	6.2e+00	1.5e-15	11951	(0, 0, 0)
	spg_bp	13.520	4.3e-09	6.2e+00	1.0e-10	17250	(0, 0, 0)
CaltechTest4	**FPC_AS**	0.680	7.6e-14	1.3e+05	1.3e-09	817	(0, 0, 0)
	l1eq_pd	20.700	7.1e-14	1.3e+05	5.5e-12	32675	(0, 0, 0)
	spg_bp	19.350	2.6e-12	1.3e+05	9.8e-11	25142	(0, 0, 0)
Ameth6Xmeth2seed200	**FPC_AS**	0.980	6.6e-10	1.5e+02	2.2e-09	681	(0, 0, 0)
	l1eq_pd	38.090	5.5e-01	1.5e+02	4.5e-06	47451	(0, 3, 207)
	spg_bp	30.950	5.5e-01	1.5e+02	3.4e-03	29528	(0, 3, 205)
Ameth6Xmeth6seed200	**FPC_AS**	0.600	7.7e-15	1.5e+07	2.8e-09	525	(0, 0, 0)
	l1eq_pd	56.690	1.0e-00	4.1e+03	7.8e+05	69707	(0, 154, 0)
	spg_bp	31.300	5.5e-01	1.5e+07	3.5e+02	29549	(0, 3, 206)

We introduce some symbols used in the tables of the following numerical reports. Here, "CPU" denotes CPU time measured in seconds, "rel.err" denotes the relative error between the recovered solution x and the exact sparsest solution $\bar{x}$, i.e., rel.err $= \frac{\|x^k - \bar{x}\|}{\|\bar{x}\|}$, $\|x\|_1$ denotes the l_1-norm of the recovered solution x, $\|r\|_2 := \|Ax - b\|$ denotes the l_2-norm of the residual, and nMat denotes the total number matrix-vector products with A and $A^\top$. Since a truncation of the recovered solution is useful in practice, we compute three numbers "sgn", "miss" and "over" to measure the recoverability of the truncated solution. First, we set a thresholding value $\xi = 0.1|\bar{x}_k|$, where $\bar{x}_k$ has the smallest magnitude among all the nonzero components of $\bar{x}$. We define a vector y with $y_i = x_i$ if $|x_i| \geq \xi$ and $y_i = 0$, otherwise. Then "sgn" denotes the number of components of y which have different signs compared to $\bar{x}$ in the union of the supports of y and $\bar{x}$, "miss" denotes the number of zero components of y which are nonzero in $\bar{x}$, and "over" denotes the number of nonzero components of y which are zero in $\bar{x}$. The values of "sgn", "miss" and "over" should be zero if x is a good approximation to $\bar{x}$.

A summary of the computational results for all of the four problems is presented in Table 2. From the table, the superiority of **FPC_AS** is obvious. For the last two problems, only **FPC_AS** was able to recover the solution successfully. Both **FPC_AS** and **l1eq_pd** were able to achieve very small relative error "rel.err" and to obtain small residual $\|r\|$ on the first four problems, which implies that these two solvers are able to recover the magnitudes of the solution successfully. The signs of the sparsest solution were also identified indicated by the pairs of "(sgn,miss,over)". However, **FPC_AS** is much cheaper than **l1eq_pd** and **spg_bp** in terms of CPU time and the total number of matrix-vector products. Although **spg_bp** gave similar value on the l_1-norm $\|x\|_1$ as the other two solvers on "CaltechTest1", some components should be zero were not eliminated. Adjusting other parameters of **spg_bp** might give better results, but it is outside the scope of this paper. Finally, we depicted the recovered solutions from all of the three solvers for "Ameth6Xmeth2seed200" in Figure 3.

Figure 3: Recovered solutions of "Ameth6Xmeth2seed200"



(a) **FPC_AS** (b) **l1eq_pd** (c) **spg_bp**

B Options of FPC_AS

The available options are:

- 'x0': initial solution
  default: , valid range: [-Inf, Inf]
- 'init': methods of initialization, integer
  default: 2, valid range: {0,1,2}
- 'xs': exact solution whose purpose is only for comparison
  default: , valid range: [-Inf, Inf]
- 'tol_eig': tolerance for eigs
  default: 0.0001, valid range: [0, 1]
- 'scale_A': scale the matrix A so that max of eigs(A*AT) equals 1, integer
  default: 0, valid range: {0, 1}
- 'eigs_mxitr': max number of iterations for eigs(A*AT), integer
  default: 20, valid range: [1, 100]
- 'eps': machine accuarcy
  default: 1e-16, valid range: [0, 1]
- 'zero': minimal magnitude of x
  default: 1e-08, valid range: [0, 1e+10]
- 'dynamic_zero': set the thresholding level dynamically or not, integer
  default: 0, valid range: {0, 1}
- 'minK': estimate of the number of the nonzero components of the exact solution, integer
  default: m/2, valid range: {1, n}
- 'tauD': a parameter for shrinkage
  default: min(1.999,-1.665*m/n + 2.665), valid range: [0, 100000]
- 'tau_min': minimal tau
  default: 0.0001, valid range: [0, 100000]
- 'tau_max': minimal tau
  default: 1000, valid range: [0, 100000]
- 'mxitr': max number of iterations, integer
  default: 1000, valid range: [1, 100000]

- 'gtol': Tolerance on norm of sub-gradient
  default: 1e-06, valid range: [0, 1]
- 'gtol_scale_x': Tolerance on norm of sub-gradient
  default: 1e-12, valid range: [0, 1]
- 'f_rel_tol': Tolerance on the relative changes of function value
  default: 1e-12, valid range: [0, 1]
- 'f_value_tol': Tolerance on the optimal objective value. Stop if $\psi_\mu(x)$ less than or equal to f_value_tol.
  default: 0, valid range: [0, inf]
- 'ls_mxitr': max number of iterations of of line search subroutine, integer
  default: 5, valid range: [2, 100]
- 'gamma': a parameter for the nonmonotone line search
  default: 0.85, valid range: [0, 1]
- 'c': a parameter for Armijo condition
  default: 0.001, valid range: [0, 1]
- 'beta': a parameter for decreasing the step size in the nonmonotone line search
  default: 0.5, valid range: [0, 1]
- 'eta': a parameter for decreasing mu
  default: 0.5, valid range: [0, 1]
- 'eta_rho': a parameter for decreasing eta
  default: 0.5, valid range: [0, 1]
- 'eta_min': minimal eta
  default: 0.001, valid range: [0, 1]
- 'eta_max': maximal eta
  default: 0.8, valid range: [0, 1]
- 'max_itr_mu': control the decreasing of eta, iteration number between the changes of mu, integer
  default: 3, valid range: [1, 100]
- 'sub_mxitr': max number of iterations for doing sub-optimization, integer
  default: 80, valid range: [1, 100000]
- 'lbfgs_m': storage number of L-BFGS, integer
  default: 5, valid range: [1, 100]
- 'kappa_g_d': tolerance for checking whether do sub-optimization or not
  default: 10, valid range: [1, 1000]
- 'kappa_rho': a parameter for increasing kappa_g_d
  default: 10, valid range: [1, 1000]
- 'tol_start_sub': Tolerance of starting sub-optimization
  default: 1e-06, valid range: [0, 1]
- 'min_itr_shrink': min number of iterations between two sub-optimization, integer
  default: 3, valid range: [1, 1000]
- 'max_itr_shrink': max number of iterations between two sub-optimization, integer
  default: 20, valid range: [1, 1000]
- 'record': print information, -1=quiet, 0=some output, 1=more output. integer
  default: 0, valid range: {-1,0,1}

- ’PrintOptions’: print options, integer
  default: 0, valid range: {0, 1}

References

[1] Zaiwen Wen, Wotao Yin, et al. On the convergence of an active set method for l_1 minimization. working paper, 2008.

[2] Zaiwen Wen, Wotao Yin, et al. An algorithm for l_1 minimization using shrinkage and subspace optimization. working paper, 2008.

[3] Elain Hale, Wotao Yin, and Yin Zhang. FPC: A MATLAB solver for minimizing $\|x\|_1 + p\|Ax - b\|_2^2$. `http://www.caam.rice.edu/~optimization/L1/fpc/`, 2008.

[4] E. van den Berg and M. P. Friedlander. SPGL1: A solver for large-scale sparse reconstruction, June 2007. http://www.cs.ubc.ca/labs/scl/spgl1.

[5] E.Candes, J.Romberg, and S. Becker. l_1-magic, 2007. http://www.acm.caltech.edu/l1magic.

[6] Stephen Becker and Emmanuel Candes. Some test problems for compressed sensing. private communication, 2008.