SPAMS: a SPArse Modeling Software, v2.4

Julien Mairal
`julien.mairal@m4x.org`

December 5, 2013

Contents

1 Introduction

SPAMS (SPArse Modeling Software) is an open-source optimization toolbox for sparse estimation with licence GPLv3. It implements algorithms for solving machine learning and signal processing problems involving sparse regularizations.

The library is coded in C++, is compatible with Linux, Mac, and Windows 32bits and 64bits Operating Systems. It is interfaced with Matlab, but can be called from any C++ application. R and Python interfaces have been developed by Jean-Paul Chieze.

It requires an implementation of BLAS and LAPACK for performing linear algebra operations. The ones shipped with matlab and R can be used, but also external libraries such as atlas, the netlib implementation, or the Intel Math Kernel Library. It also exploits multi-core CPUs when this feature is supported by the compiler, through OpenMP.

The current licence is GPLv3, which is available at `http://www.gnu.org/licenses/gpl.html`. For other usages (such as the use in proprietary softwares), please contact the author.

Version 2.4 of SPAMS is divided into four "toolboxes" and has a few additional miscellaneous functions:

- The **Dictionary learning and matrix factorization toolbox** contains the online learning technique of [19, 20] and its variants for solving various matrix factorization problems:
  - dictionary Learning for sparse coding;
  - sparse principal component analysis (seen as a sparse matrix factorization problem);
  - non-negative matrix factorization;
  - non-negative sparse coding;
  - dictionary learning with structured sparsity.
- The **Sparse decomposition toolbox** contains efficient implementations of
  - Orthogonal Matching Pursuit, (or Forward Selection) [34, 26];

  - the LARS/homotopy algorithm [29, 8] (variants for solving Lasso and Elastic-Net problems);
  - a weighted version of LARS;
  - OMP and LARS when data come with a binary mask;
  - a coordinate-descent algorithm for ℓ_1-decomposition problems [11, 9, 35];
  - a greedy solver for simultaneous signal approximation as defined in [33, 32] (SOMP);
  - a solver for simulatneous signal approximation with ℓ_1/ℓ_2-regularization based on block-coordinate descent;
  - a homotopy method for the Fused-Lasso Signal Approximation as defined in [9] with the homotopy method presented in the appendix of [20];
  - a tool for projecting efficiently onto a few convex sets inducing sparsity such as the ℓ_1-ball using the method of [3, 17, 7], and Elastic-Net or Fused Lasso constraint sets as proposed in the appendix of [20].

- The **Proximal toolbox**: An implementation of proximal methods (ISTA and FISTA [1]) for solving a large class of sparse approximation problems with different combinations of loss and regularizations. One of the main features of this toolbox is to provide a robust stopping criterion based on *duality gaps* to control the quality of the optimization, whenever possible. It also handles sparse feature matrices for large-scale problems. The following regularizations are implemented:

  - Tikhonov regularization (squared ℓ_2-norm);
  - ℓ_1-norm, ℓ_2, ℓ_∞-norms;
  - Elastic-Net [37];
  - Fused Lasso [31];
  - tree-structured sum of ℓ_2-norms (see [14, 15]);
  - tree-structured sum of ℓ_∞-norms (see [14, 15]);
  - general sum of ℓ_∞-norms (see [21, 22]);
  - mixed ℓ_1/ℓ_2-norms on matrices [36, 28];
  - mixed ℓ_1/ℓ_∞-norms on matrices [36, 28];
  - mixed ℓ_1/ℓ_2-norms on matrices plus ℓ_1 [30, 10];
  - mixed ℓ_1/ℓ_∞-norms on matrices plus ℓ_1;
  - group-lasso with ℓ_2 or ℓ_∞-norms;
  - group-lasso+ℓ_1;
  - multi-task tree-structured sum of ℓ_∞-norms (see [21, 22]);
  - trace norm;
  - ℓ_0 pseudo-norm (only with ISTA);
  - tree-structured ℓ_0 (only with ISTA);
  - rank regularization for matrices (only with ISTA);
  - the path-coding penalties of [23].

  All of these regularization functions can be used with the following losses

  - square loss;
  - square loss with missing observations;
  - logistic loss, weighted logistic loss;
  - multi-class logistic.

  This toolbox can also enforce non-negativity constraints, handle intercepts and sparse matrices. There are also a few additional undocumented functionalities, which are available in the source code. For some combinations of loss and regularizers, stochastic and incremental proximal gradient solvers are also implemented [25, 24].

- A few tools for performing linear algebra operations such as a conjugate gradient algorithm, manipulating sparse matrices and graphs.

The toolbox was written by Julien Mairal at INRIA, with the collaboration of Francis Bach (INRIA), Jean Ponce (Ecole Normale Supérieure), Guillermo Sapiro (University of Minnesota), Guillaume Obozinski (INRIA) and Rodolphe Jenatton (INRIA).

R and Python interfaces have been written by Jean-Paul Chieze (INRIA).

2 Installation

The toolbox used to come with pre-compiled binaries for various plateforms. Now the sources are available and you will have to compile it yourself.

The user has the choice of the BLAS library, but the Intel MKL is recommended for the best performance. Note that the builtin blas library of Matlab is a version of the Intel MKL (not the most recent one though), and offers a good performance.

The folder `doc` contains the documentation in pdf and html. `build/` contains the binary files, including the help for each command. `test_release` contains various matlab scripts which call the different functions of this toolbox.

The software package comes also with a script bash that has to be used to launch matlab for Linux and/or Mac OS versions. The install procedure is described in a file called `HOW_TO_INSTALL` and in the file `compile.m` for the Matlab version.

3 Dictionary Learning and Matrix Factorization Toolbox

This is the section for dictionary learning and matrix factorization, corresponding to [19, 20].

3.1 Function mexTrainDL

This is the main function of the toolbox, implementing the learning algorithms of [20]. Given a training set $\mathbf{x}^1, \ldots,$. It aims at solving

$$\min_{\mathbf{D} \in \mathcal{C}} \lim_{n \to +\infty} \frac{1}{n} \sum_{i=1}^{n} \min_{\boldsymbol{\alpha}^i} \Big(\frac{1}{2} \|\mathbf{x}^i - \mathbf{D}\boldsymbol{\alpha}^i\|_2^2 + \psi(\boldsymbol{\alpha}^i) \Big). \tag{1}$$

ψ is a sparsity-inducing regularizer and $\mathcal{C}$ is a constraint set for the dictionary. As shown in [20] and in the help file below, various combinations can be used for ψ and $\mathcal{C}$ for solving different matrix factorization problems. What is more, positivity constraints can be added to $\boldsymbol{\alpha}$ as well. The function admits several modes for choosing the optimization parameters, using the parameter-free strategy proposed in [19], or using the parameters t_0 and ρ presented in [20]. **Note that for problems of a reasonable size, and when ψ is the ℓ_1-norm, the function mexTrainDL_Memory can be faster but uses more memory.**

```
%
% Usage:   [D [model]]=mexTrainDL(X,param[,model]);
%          model is optional
%
% Name: mexTrainDL
%
% Description: mexTrainDL is an efficient implementation of the
%     dictionary learning technique presented in
%
%     "Online Learning for Matrix Factorization and Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     arXiv:0908.0050
%
%     "Online Dictionary Learning for Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     ICML 2009.
%
%     Note that if you use param.mode=1 or 2, if the training set has a
%     reasonable size and you have enough memory on your computer, you
%     should use mexTrainDL_Memory instead.
%
%
%     It addresses the dictionary learning problems
%        1) if param.mode=0
%     min_{D in C} (1/n) sum_{i=1}^n (1/2)||x_i-Dalpha_i||_2^2  s.t. ...
%                                                  ||alpha_i||_1 <= lambda
%        2) if param.mode=1
%     min_{D in C} (1/n) sum_{i=1}^n  ||alpha_i||_1  s.t.  ...
%                                           ||x_i-Dalpha_i||_2^2 <= lambda
%        3) if param.mode=2
%     min_{D in C} (1/n) sum_{i=1}^n (1/2)||x_i-Dalpha_i||_2^2 + ...
%                                  lambda||alpha_i||_1 + lambda_2||alpha_i||_2^2
%        4) if param.mode=3, the sparse coding is done with OMP
%     min_{D in C} (1/n) sum_{i=1}^n (1/2)||x_i-Dalpha_i||_2^2  s.t. ...
%                                                  ||alpha_i||_0 <= lambda
%        5) if param.mode=4, the sparse coding is done with OMP
%     min_{D in C} (1/n) sum_{i=1}^n  ||alpha_i||_0  s.t.  ...
%                                           ||x_i-Dalpha_i||_2^2 <= lambda
%        6) if param.mode=5, the sparse coding is done with OMP
%     min_{D in C} (1/n) sum_{i=1}^n 0.5||x_i-Dalpha_i||_2^2 +lambda||alpha_i||_0
%
%
%%     C is a convex set verifying
%        1) if param.modeD=0
%           C={  D in Real^{m x p}  s.t.  forall j,  ||d_j||_2^2 <= 1 }
%        2) if param.modeD=1
%           C={  D in Real^{m x p}  s.t.  forall j,  ||d_j||_2^2 + ...
%                                                  gamma1||d_j||_1 <= 1 }
%        3) if param.modeD=2
%           C={  D in Real^{m x p}  s.t.  forall j,  ||d_j||_2^2 + ...
%                                  gamma1||d_j||_1 + gamma2 FL(d_j) <= 1 }
%        4) if param.modeD=3
%           C={  D in Real^{m x p}  s.t.  forall j,  (1-gamma1)||d_j||_2^2 + ...
%                                  gamma1||d_j||_1 <= 1 }

%
%     Potentially, n can be very large with this algorithm.
%
% Inputs: X:  double m x n matrix   (input signals)
%               m is the signal size
%               n is the number of signals to decompose
%         param: struct
%            param.D: (optional) double m x p matrix   (dictionary)
%              p is the number of elements in the dictionary
%              When D is not provided, the dictionary is initialized
%              with random elements from the training set.
%           param.K (size of the dictionary, optional is param.D is provided)
%           param.lambda  (parameter)
%           param.lambda2  (optional, by default 0)
%           param.iter (number of iterations).  If a negative number is
%              provided it will perform the computation during the
%              corresponding number of seconds. For instance param.iter=-5
%              learns the dictionary during 5 seconds.
%           param.mode (optional, see above, by default 2)
%           param.posAlpha (optional, adds positivity constraints on the
%             coefficients, false by default, not compatible with
%             param.mode =3,4)
%           param.modeD (optional, see above, by default 0)
%           param.posD (optional, adds positivity constraints on the
%             dictionary, false by default, not compatible with
%             param.modeD=2)
%           param.gamma1 (optional parameter for param.modeD >= 1)
%           param.gamma2 (optional parameter for param.modeD = 2)
%           param.batchsize (optional, size of the minibatch, by default
%              512)
%           param.iter_updateD (optional, number of BCD iterations for the dictionary
%              update step, by default 1)
%           param.modeParam (optimization mode).
%              1) if param.modeParam=0, the optimization uses the
%                 parameter free strategy of the ICML paper
%              2) if param.modeParam=1, the optimization uses the
%                 parameters rho as in arXiv:0908.0050
%              3) if param.modeParam=2, the optimization uses exponential
%                 decay weights with updates of the form
%                 A_{t} <- rho A_{t-1} + alpha_t alpha_t^T
%           param.rho (optional) tuning parameter (see paper arXiv:0908.0050)
%           param.t0 (optional) tuning parameter (see paper arXiv:0908.0050)
%           param.clean (optional, true by default. prunes
%              automatically the dictionary from unused elements).
%           param.verbose (optional, true by default, increase verbosity)
%           param.numThreads (optional, number of threads for exploiting
%              multi-core / multi-cpus. By default, it takes the value -1,
%              which automatically selects all the available CPUs/cores).
%
% Output:
%         param.D: double m x p matrix   (dictionary)
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%         - single precision setting
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
clear all;

I=double(imread('data/lena.png'))/255;
% extract 8 x 8 patches
X=im2col(I,[8 8],'sliding');
X=X-repmat(mean(X),[size(X,1) 1]);
X=X ./ repmat(sqrt(sum(X.^2)),[size(X,1) 1]);

param.K=256;  % learns a dictionary with 100 elements
param.lambda=0.15;
param.numThreads=-1; % number of threads
param.batchsize=400;
param.verbose=false;

param.iter=1000;  % let us see what happens after 1000 iterations.

%%%%%%%%%% FIRST EXPERIMENT %%%%%%%%%%%
tic
D = mexTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);

fprintf('Evaluating cost function...\n');
alpha=mexLasso(X,D,param);
R=mean(0.5*sum((X-D*alpha).^2)+param.lambda*sum(abs(alpha)));
ImD=displayPatches(D);
subplot(1,3,1);
imagesc(ImD); colormap('gray');
fprintf('objective function: %f\n',R);
drawnow;

fprintf('*********** SECOND EXPERIMENT ***********\n');
%%%%%%%%%% SECOND EXPERIMENT %%%%%%%%%%%
% Train on half of the training set, then retrain on the second part
X1=X(:,1:floor(size(X,2)/2));
X2=X(:,floor(size(X,2)/2):end);
param.iter=500;
tic
[D model] = mexTrainDL(X1,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);
fprintf('Evaluating cost function...\n');
alpha=mexLasso(X,D,param);
R=mean(0.5*sum((X-D*alpha).^2)+param.lambda*sum(abs(alpha)));
fprintf('objective function: %f\n',R);
tic
% Then reuse the learned model to retrain a few iterations more.
param2=param;
param2.D=D;
[D model] = mexTrainDL(X2,param2,model);
%[D] = mexTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);
fprintf('Evaluating cost function...\n');
alpha=mexLasso(X,D,param);
R=mean(0.5*sum((X-D*alpha).^2)+param.lambda*sum(abs(alpha)));
fprintf('objective function: %f\n',R);

% let us add sparsity to the dictionary itself
fprintf('*********** THIRD EXPERIMENT ***********\n');
param.modeParam=0;
param.iter=1000;
param.gamma1=0.3;
param.modeD=1;
tic
[D] = mexTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);
fprintf('Evaluating cost function...\n');
alpha=mexLasso(X,D,param);
R=mean(0.5*sum((X-D*alpha).^2)+param.lambda*sum(abs(alpha)));
fprintf('objective function: %f\n',R);
tic
subplot(1,3,2);
ImD=displayPatches(D);
imagesc(ImD); colormap('gray');
drawnow;

fprintf('*********** FOURTH EXPERIMENT ***********\n');
param.modeParam=0;
param.iter=1000;
param.gamma1=0.3;
param.modeD=3;
tic
[D] = mexTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);
fprintf('Evaluating cost function...\n');
alpha=mexLasso(X,D,param);
R=mean(0.5*sum((X-D*alpha).^2)+param.lambda*sum(abs(alpha)));
fprintf('objective function: %f\n',R);
tic
subplot(1,3,3);
ImD=displayPatches(D);
imagesc(ImD); colormap('gray');
drawnow;
```

3.2 Function mexTrainDL_Memory

Memory-consuming version of mexTrainDL. This function is well adapted to small/medium-size problems: It requires storing all the coefficients $\boldsymbol{\alpha}$ and is therefore impractical for very large datasets. However, in many situations, one can afford this memory cost and it is better to use this method, which is faster than mexTrainDL. Note that unlike mexTrainDL this function does not allow warm-restart.

```
%
% Usage:   [D]=mexTrainDL(X,param);
%
% Name: mexTrainDL_Memory
%
% Description: mexTrainDL_Memory is an efficient but memory consuming
%     variant of the dictionary learning technique presented in
%
%     "Online Learning for Matrix Factorization and Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     arXiv:0908.0050
%
%     "Online Dictionary Learning for Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     ICML 2009.
%
%     Contrary to the approaches above, the algorithm here
%        does require to store all the coefficients from all the training
%        signals. For this reason this variant can not be used with large
%        training sets, but is more efficient than the regular online
%        approach for training sets of reasonable size.
%
%     It addresses the dictionary learning problems
%        1) if param.mode=1
%     min_{D in C} (1/n) sum_{i=1}^n  ||alpha_i||_1  s.t.  ...
%                                         ||x_i-Dalpha_i||_2^2 <= lambda
%        2) if param.mode=2
%     min_{D in C} (1/n) sum_{i=1}^n (1/2)||x_i-Dalpha_i||_2^2 + ...
%                                                      lambda||alpha_i||_1
%
%     C is a convex set verifying
%        1) if param.modeD=0
%           C={  D in Real^{m x p}  s.t.  forall j,  ||d_j||_2^2 <= 1 }
%        1) if param.modeD=1
%           C={  D in Real^{m x p}  s.t.  forall j,  ||d_j||_2^2 + ...
%                                                gamma1||d_j||_1 <= 1 }
%        1) if param.modeD=2
%           C={  D in Real^{m x p}  s.t.  forall j,  ||d_j||_2^2 + ...
%                                gamma1||d_j||_1 + gamma2 FL(d_j) <= 1 }
%
%     Potentially, n can be very large with this algorithm.
%
% Inputs: X:  double m x n matrix   (input signals)
%               m is the signal size
%               n is the number of signals to decompose
%         param: struct
%            param.D: (optional) double m x p matrix   (dictionary)
%              p is the number of elements in the dictionary
%              When D is not provided, the dictionary is initialized
%              with random elements from the training set.
%           param.K (size of the dictionary, optional is param.D is provided)
%           param.lambda  (parameter)
%           param.iter (number of iterations).  If a negative number is
%              provided it will perform the computation during the
%              corresponding number of seconds. For instance param.iter=-5
%              learns the dictionary during 5 seconds.
%            param.mode (optional, see above, by default 2)
%            param.modeD (optional, see above, by default 0)
%            param.posD (optional, adds positivity constraints on the
%              dictionary, false by default, not compatible with
%              param.modeD=2)
%            param.gamma1 (optional parameter for param.modeD >= 1)
%            param.gamma2 (optional parameter for param.modeD = 2)
%            param.batchsize (optional, size of the minibatch, by default
%              512)
%            param.iter_updateD (optional, number of BCD iterations for the dictionary
%                update step, by default 1)
%            param.modeParam (optimization mode).
%               1) if param.modeParam=0, the optimization uses the
%                  parameter free strategy of the ICML paper
%               2) if param.modeParam=1, the optimization uses the
%                  parameters rho as in arXiv:0908.0050
%               3) if param.modeParam=2, the optimization uses exponential
%                  decay weights with updates of the form
%                  A_{t} <- rho A_{t-1} + alpha_t alpha_t^T
%             param.rho (optional) tuning parameter (see paper arXiv:0908.0050)
%             param.t0 (optional) tuning parameter (see paper arXiv:0908.0050)
%             param.clean (optional, true by default. prunes
%               automatically the dictionary from unused elements).
%             param.numThreads (optional, number of threads for exploiting
%               multi-core / multi-cpus. By default, it takes the value -1,
%               which automatically selects all the available CPUs/cores).
%
% Output:
%         param.D: double m x p matrix   (dictionary)
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%         - single precision setting (even though the output alpha is double
%           precision)
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
clear all;

I=double(imread('data/lena.png'))/255;
% extract 8 x 8 patches
X=im2col(I,[8 8],'sliding');
X=X-repmat(mean(X),[size(X,1) 1]);
X=X ./ repmat(sqrt(sum(X.^2)),[size(X,1) 1]);
X=X(:,1:10:end);

param.K=200;  % learns a dictionary with 100 elements
param.lambda=0.15;
param.numThreads=4; % number of threads

param.iter=100;  % let us see what happens after 100 iterations.

%%%%%%%%%% FIRST EXPERIMENT %%%%%%%%%%%
tic
D = mexTrainDL_Memory(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);

fprintf('Evaluating cost function...\n');
alpha=mexLasso(X,D,param);
R=mean(0.5*sum((X-D*alpha).^2)+param.lambda*sum(abs(alpha)));
ImD=displayPatches(D);
subplot(1,3,1);
imagesc(ImD); colormap('gray');
fprintf('objective function: %f\n',R);
```

```
%%%%%%%%%% SECOND EXPERIMENT %%%%%%%%%%%
tic
D = mexTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);

fprintf('Evaluating cost function...\n');
alpha=mexLasso(X,D,param);
R=mean(0.5*sum((X-D*alpha).^2)+param.lambda*sum(abs(alpha)));
ImD=displayPatches(D);
subplot(1,3,2);
imagesc(ImD); colormap('gray');
fprintf('objective function: %f\n',R);
```

3.3 Function mexStructTrainDL

This function allows to use mexTrainDL with structured regularization functions for the coefficients $\boldsymbol{\alpha}$. It internally uses the FISTA algorithm.

```
%
% Usage:   [D [model]]=mexStructTrainDL(X,param[,model]);
%          model is optional
%
% Name: mexStructTrainDL
%
% Description: mexStructTrainDL is an efficient implementation of the
%     dictionary learning technique presented in
%
%     "Online Learning for Matrix Factorization and Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     arXiv:0908.0050
%
%     "Online Dictionary Learning for Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     ICML 2009.
%
%
%     It addresses the dictionary learning problems
%        min_{D in C} (1/n) sum_{i=1}^n 0.5||x_i-Dalpha_i||_2^2 + lambda psi(alpha)
%        where the regularization function psi depends on param.regul
%        (see mexProximalFlat for the description of psi,
%         and param.regul below for allowed values of regul)
%
%%     C is a convex set verifying
%        1) if param.modeD=0
%           C={  D in Real^{m x p}  s.t.  forall j,  ||d_j||_2^2 <= 1 }
%        2) if param.modeD=1
%           C={  D in Real^{m x p}  s.t.  forall j,  ||d_j||_2^2 + ...
%                                                  gamma1||d_j||_1 <= 1 }
%        3) if param.modeD=2
%           C={  D in Real^{m x p}  s.t.  forall j,  ||d_j||_2^2 + ...
%                                   gamma1||d_j||_1 + gamma2 FL(d_j) <= 1 }
%        4) if param.modeD=3
%           C={  D in Real^{m x p}  s.t.  forall j,  (1-gamma1)||d_j||_2^2 + ...
%                                  gamma1||d_j||_1 <= 1 }

%
%     Potentially, n can be very large with this algorithm.
%
% Inputs: X:  double m x n matrix   (input signals)
%               m is the signal size
%               n is the number of signals to decompose
%         param: struct
%            param.D: (optional) double m x p matrix   (dictionary)
%              p is the number of elements in the dictionary
%              When D is not provided, the dictionary is initialized
%              with random elements from the training set.
%           param.K (size of the dictionary, optional is param.D is provided)
%           param.lambda  (parameter)
%           param.lambda2  (optional, by default 0)
%           param.lambda3 (optional, regularization parameter, 0 by default)
%           param.iter (number of iterations).  If a negative number is
%              provided it will perform the computation during the
%              corresponding number of seconds. For instance param.iter=-5
%              learns the dictionary during 5 seconds.
%           param.regul choice of regularization : one of
%               'l0' 'l1' 'l2' 'linf' 'none' 'elastic-net' 'fused-lasso'
%               'graph' 'graph-ridge' 'graph-l2' 'tree-l0' 'tree-l2' 'tree-linf'
%           param.tree struct (see documentation of mexProximalTree);
%               needed for param.regul of graph kind.
%           param.graph struct (see documentation of mexProximalGraph);
%               needed for param.regul of tree kind.
%           param.posAlpha (optional, adds positivity constraints on the
%               coefficients, false by default.
%           param.modeD (optional, see above, by default 0)
%           param.posD (optional, adds positivity constraints on the
%             dictionary, false by default, not compatible with
%             param.modeD=2)
%           param.gamma1 (optional parameter for param.modeD >= 1)
%           param.gamma2 (optional parameter for param.modeD = 2)
%           param.batchsize (optional, size of the minibatch, by default
%              512)
%           param.iter_updateD (optional, number of BCD iterations for the dictionary
%              update step, by default 1)
%           param.modeParam (optimization mode).
%              1) if param.modeParam=0, the optimization uses the
%                 parameter free strategy of the ICML paper
%              2) if param.modeParam=1, the optimization uses the
%                 parameters rho as in arXiv:0908.0050
%              3) if param.modeParam=2, the optimization uses exponential
%                 decay weights with updates of the form
%                 A_{t} <- rho A_{t-1} + alpha_t alpha_t^T
%            param.ista (optional, use ista instead of fista, false by default).
%            param.tol (optional, tolerance for stopping criteration, which is a relative
  duality gap
%            param.fixed_step (deactive the line search for L in fista and use param.K instead)
%           param.rho (optional) tuning parameter (see paper arXiv:0908.0050)
%           param.t0 (optional) tuning parameter (see paper arXiv:0908.0050)
%           param.clean (optional, true by default. prunes
%              automatically the dictionary from unused elements).
%           param.verbose (optional, true by default, increase verbosity)
%           param.numThreads (optional, number of threads for exploiting
%              multi-core / multi-cpus. By default, it takes the value -1,
%              which automatically selects all the available CPUs/cores).
%
% Output:
%         param.D: double m x p matrix   (dictionary)
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%         - single precision setting
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
clear all;

I=double(imread('data/lena.png'))/255;
% extract 8 x 8 patches
X=im2col(I,[8 8],'sliding');
X=X-repmat(mean(X),[size(X,1) 1]);
X=X ./ repmat(sqrt(sum(X.^2)),[size(X,1) 1]);

param.K=64;  % learns a dictionary with 64 elements
param.lambda=0.05;
param.numThreads=4; % number of threads
param.batchsize=400;
param.tol = 1e-3

param.iter=200;  %

if false
param.regul = 'l1';
fprintf('with Fista Regression %s\n',param.regul);
tic
D = mexStructTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);
%
param.regul = 'l2';
fprintf('with Fista Regression %s\n',param.regul);
tic
D = mexStructTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);
%
param.regul = 'elastic-net';
fprintf('with Fista %s\n',param.regul);
param.lambda2=0.1;
tic
D = mexStructTrainDL(X,param);
t=toc;
```

```
fprintf('time of computation for Dictionary Learning: %f\n',t);

%%% GRAPH
param.lambda=0.1; % regularization parameter
param.tol=1e-5;
param.K = 10
graph.eta_g=[1 1 1 1 1];
graph.groups=sparse([0 0 0 1 0;
                     0 0 0 0 0;
                     0 0 0 0 0;
                     0 0 0 0 0;
                     0 0 1 0 0]);   % g5 is included in g3, and g2 is included in g4
graph.groups_var=sparse([1 0 0 0 0;
                         1 0 0 0 0;
                         1 0 0 0 0 ;
                         1 1 0 0 0;
                         0 1 0 1 0;
                         0 1 0 1 0;
                         0 1 0 0 1;
                         0 0 0 0 1;
                         0 0 0 0 1;
                         0 0 1 0 0]); % represents direct inclusion relations

param.graph = graph

param.regul = 'graph';
fprintf('with Fista %s\n',param.regul);
tic
D = mexStructTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);

%%
%%% TREE
%?pause;
param = rmfield(param,'graph');

end


param.lambda=0.1; % regularization parameter
param.tol=1e-5;
param.K = 10

tree.own_variables=  int32([0 0 3 5 6 6 8 9]);   % pointer to the first variable of each group
tree.N_own_variables=int32([0 3 2 1 0 2 1 1]); % number of "root" variables in each group
tree.eta_g=[1 1 1 2 2 2 2.5 2.5];
tree.groups=sparse([0 0 0 0 0 0 0 0; ...
                    1 0 0 0 0 0 0 0; ...
                    0 1 0 0 0 0 0 0; ...
                    0 1 0 0 0 0 0 0; ...
                    1 0 0 0 0 0 0 0; ...
                    0 0 0 0 1 0 0 0; ...
                    0 0 0 0 1 0 0 0; ...
                    0 0 0 0 0 0 1 0]);  % first group should always be the root of the tree

param.tree = tree;

param.regul = 'tree-l0';
fprintf('with Fista %s\n',param.regul);

tic
D = mexStructTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);

%
param.regul = 'tree-l2';
fprintf('with Fista %s\n',param.regul);
tic
D = mexStructTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);

%
param.regul = 'tree-linf';
fprintf('with Fista %s\n',param.regul);

tic
D = mexStructTrainDL(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);
```

3.4 Function nmf

This function is an example on how to use the function mexTrainDL for the problem of non-negative matrix factorization formulated in [16]. Note that mexTrainDL can be replaced by mexTrainDL_Memory in this function for small or medium datasets.

```
%
% Usage:   [U [,V]]=nmf(X,param);
%
% Name: nmf
%
% Description: mexTrainDL is an efficient implementation of the
%     non-negative matrix factorization technique presented in
%
%     "Online Learning for Matrix Factorization and Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     arXiv:0908.0050
%
%     "Online Dictionary Learning for Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     ICML 2009.
%
%     Potentially, n can be very large with this algorithm.
%
% Inputs: X:  double m x n matrix   (input signals)
%               m is the signal size
%                n is the number of signals to decompose
%          param: struct
%             param.K (number of required factors)
%             param.iter (number of iterations).  If a negative number
%               is provided it will perform the computation during the
%               corresponding number of seconds. For instance param.iter=-5
%               learns the dictionary during 5 seconds.
%             param.batchsize (optional, size of the minibatch, by default
%                512)
%             param.modeParam (optimization mode).
%                1) if param.modeParam=0, the optimization uses the
%                   parameter free strategy of the ICML paper
%                2) if param.modeParam=1, the optimization uses the
%                   parameters rho as in arXiv:0908.0050
%                3) if param.modeParam=2, the optimization uses exponential
%                   decay weights with updates of the form
%                   A_{t} <- rho A_{t-1} + alpha_t alpha_t^T
%             param.rho (optional) tuning parameter (see paper
%               arXiv:0908.0050)
%             param.t0 (optional) tuning parameter (see paper
%               arXiv:0908.0050)
%             param.clean (optional, true by default. prunes automatically
%               the dictionary from unused elements).
%             param.batch (optional, false by default, use batch learning
%               instead of online learning)
%             param.numThreads (optional, number of threads for exploiting
%                 multi-core / multi-cpus. By default, it takes the value -1,
%                 which automatically selects all the available CPUs/cores).
%          model: struct (optional) learned model for "retraining" the data.
%
% Output:
%          U: double m x p matrix
%          V: double p x n matrix   (optional)
%          model: struct (optional) learned model to be used for
%            "retraining" the data.
%
% Author: Julien Mairal, 2009
function [U V] = nmf(X,param)

param.lambda=0;
param.mode=2;
param.posAlpha=1;
param.posD=1;
param.whiten=0;
U=mexTrainDL(X,param);
param.pos=1;
if nargout == 2
   if issparse(X) % todo allow sparse matrices X for mexLasso
      maxbatch=ceil(10000000/size(X,1));
      for jj = 1:maxbatch:size(X,2)
         indbatch=jj:min((jj+maxbatch-1),size(X,2));
         Xb=full(X(:,indbatch));
         V(:,indbatch)=mexLasso(Xb,U,param);
      end
   else
      V=mexLasso(X,U,param);
   end
end
```

The following piece of code illustrates how to use this function.

```
clear all;

I=double(imread('data/lena.png'))/255;
% extract 8 x 8 patches
X=im2col(I,[16 16],'sliding');
X=X(:,1:10:end);
X=X ./ repmat(sqrt(sum(X.^2)),[size(X,1) 1]);

param.K=49;  % learns a dictionary with 100 elements
param.numThreads=4; % number of threads

param.iter=-5;  % let us see what happens after 100 iterations.

%%%%%%%%%% FIRST EXPERIMENT %%%%%%%%%%%
tic
[U V] = nmf(X,param);
t=toc;
fprintf('time of computation for Dictionary Learning: %f\n',t);

fprintf('Evaluating cost function...\n');
R=mean(0.5*sum((X-U*V).^2));
ImD=displayPatches(U);
imagesc(ImD); colormap('gray');
fprintf('objective function: %f\n',R);
```

3.5 Function nnsc

This function is an example on how to use the function mexTrainDL for the problem of non-negative sparse coding as defined in [13]. Note that mexTrainDL can be replaced by mexTrainDL_Memory in this function for small or medium datasets.

```
%
% Usage:   [U [,V]]=nnsc(X,param);
%
% Name: nmf
%
% Description: mexTrainDL is an efficient implementation of the
%     non-negative sparse coding technique presented in
%
%     "Online Learning for Matrix Factorization and Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     arXiv:0908.0050
%
%     "Online Dictionary Learning for Sparse Coding"
%     by Julien Mairal, Francis Bach, Jean Ponce and Guillermo Sapiro
%     ICML 2009.
%
%     Potentially, n can be very large with this algorithm.
%
```

```
% Inputs: X:  double m x n matrix   (input signals)
%                m is the signal size
%                n is the number of signals to decompose
%         param: struct
%           param.K (number of required factors)
%           param.lambda (parameter)
%           param.iter (number of iterations).  If a negative number
%              is provided it will perform the computation during the
%              corresponding number of seconds. For instance param.iter=-5
%              learns the dictionary during 5 seconds.
%           param.batchsize (optional, size of the minibatch, by default
%              512)
%           param.modeParam (optimization mode).
%              1) if param.modeParam=0, the optimization uses the
%                 parameter free strategy of the ICML paper
%              2) if param.modeParam=1, the optimization uses the
%                 parameters rho as in arXiv:0908.0050
%              3) if param.modeParam=2, the optimization uses exponential
%                 decay weights with updates of the form
%                 A_{t} <- rho A_{t-1} + alpha_t alpha_t^T
%           param.rho (optional) tuning parameter (see paper
%              arXiv:0908.0050)
%           param.t0 (optional) tuning parameter (see paper
%              arXiv:0908.0050)
%           param.clean (optional, true by default. prunes automatically
%              the dictionary from unused elements).
%           param.batch (optional, false by default, use batch learning
%              instead of online learning)
%           param.numThreads (optional, number of threads for exploiting
%              multi-core / multi-cpus. By default, it takes the value -1,
%              which automatically selects all the available CPUs/cores).
%         model: struct (optional) learned model for "retraining" the data.
%
% Output:
%         U: double m x p matrix
%         V: double p x n matrix   (optional)
%         model: struct (optional) learned model to be used for
%            "retraining" the data.
%
% Author: Julien Mairal, 2009
function [U V] = nnsc(X,param)

param.mode=2;
param.posAlpha=1;
param.posD=1;
param.whiten=0;
U=mexTrainDL(X,param);
param.pos=1;
if nargout == 2
   V=mexLasso(X,U,param);
end
```

4 Sparse Decomposition Toolbox

This toolbox implements several algorithms for solving signal reconstruction problems. It is mostly adapted for solving a large number of small/medium scale problems, but can be also efficient sometimes with large scale ones.

4.1 Function mexOMP

This is a fast implementation of the Orthogonal Matching Pursuit algorithm (or forward selection) [26, 34]. Given a matrix of signals $\mathbf{X} = [\mathbf{x}^1, \ldots, \mathbf{x}^n]$ in $\mathbb{R}^{m\times n}$ and a dictionary $\mathbf{D} = [\mathbf{d}^1, \ldots, \mathbf{d}^p]$ in $\mathbb{R}^{m\times p}$, the algorithm computes a matrix $\mathbf{A} = [\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^n]$ in $\mathbb{R}^{p\times n}$, where for each column $\mathbf{x}$ of $\mathbf{X}$, it returns a coefficient vector $\boldsymbol{\alpha}$ which is an approximate solution of the following NP-hard problem

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^p} \|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 \;\text{ s.t. }\; \|\boldsymbol{\alpha}\|_0 \leq L, \tag{2}$$

or

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^p} \|\boldsymbol{\alpha}\|_0 \;\text{ s.t. }\; \|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 \leq \varepsilon, \tag{3}$$

or

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_0. \tag{4}$$

For efficienty reasons, the method first computes the covariance matrix $\mathbf{D}^T\mathbf{D}$, then for each signal, it computes $\mathbf{D}^T\mathbf{x}$ and performs the decomposition with a Cholesky-based algorithm (see [6] for instance).

Note that mexOMP can return the “greedy” regularization path if needed (see below):

```
%
% Usage:   A=mexOMP(X,D,param);
% or       [A path]=mexOMP(X,D,param);
%
% Name: mexOMP
%
% Description: mexOMP is an efficient implementation of the
%     Orthogonal Matching Pursuit algorithm. It is optimized
%     for solving a large number of small or medium-sized
%     decomposition problem (and not for a single large one).
%     It first computes the Gram matrix D'D and then perform
%     a Cholesky-based OMP of the input signals in parallel.
%     X=[x^1,...,x^n] is a matrix of signals, and it returns
%     a matrix A=[alpha^1,...,alpha^n] of coefficients.
%
%     it addresses for all columns x of X,
%         min_{alpha} ||alpha||_0  s.t  ||x-Dalpha||_2^2 <= eps
%         or
%         min_{alpha} ||x-Dalpha||_2^2  s.t. ||alpha||_0 <= L
%         or
%         min_{alpha} 0.5||x-Dalpha||_2^2 + lambda||alpha||_0
%
%
% Inputs: X:  double m x n matrix   (input signals)
%            m is the signal size
%            n is the number of signals to decompose
%         D:  double m x p matrix   (dictionary)
%            p is the number of elements in the dictionary
%            All the columns of D should have unit-norm !
%         param: struct
%            param.L (optional, maximum number of elements in each decomposition,
%               min(m,p) by default)
%            param.eps (optional, threshold on the squared l2-norm of the residual,
%               0 by default
%            param.lambda (optional, penalty parameter, 0 by default
%            param.numThreads (optional, number of threads for exploiting
%            multi-core / multi-cpus. By default, it takes the value -1,
%            which automatically selects all the available CPUs/cores).
%
% Output: A: double sparse p x n matrix (output coefficients)
%         path (optional): double dense p x L matrix (regularization path of the first signal)
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%      - single precision setting (even though the output alpha is double
%        precision)
%      - Passing an int32 vector of length n to param.L provides
%        a different parameter L for each input signal x_i
%      - Passing a double vector of length n to param.eps and or param.lambda
%        provides a different parameter eps (or lambda) for each input signal x_i
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
clear all;
randn('seed',0);

fprintf('test mexOMP');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Decomposition of a large number of signals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X=randn(64,100000);
D=randn(64,200);
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);
% parameter of the optimization procedure are chosen
param.L=10; % not more than 10 non-zeros coefficients
param.eps=0.1; % squared norm of the residual should be less than 0.1
param.numThreads=-1; % number of processors/cores to use; the default choice is -1
                    % and uses all the cores of the machine
tic
alpha=mexOMP(X,D,param);
t=toc
fprintf('%f signals processed per second\n',size(X,2)/t);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Regularization path of a single signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X=randn(64,1);
D=randn(64,10);
param.L=5;
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);
[alpha path]=mexOMP(X,D,param);
```

4.2 Function mexOMPMask

This is a variant of mexOMP with the possibility of handling a binary mask. Given a binary mask $\mathbf{B} = [\boldsymbol{\beta}^1, \ldots, \boldsymbol{\beta}^n]$ in $\{0,1\}^{m \times n}$, it returns a matrix $\mathbf{A} = [\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^n]$ such that for every column $\mathbf{x}$ of $\mathbf{X}$, $\boldsymbol{\beta}$ of $\mathbf{B}$, it computes a column $\boldsymbol{\alpha}$ of $\mathbf{A}$ by addressing

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \| \operatorname{diag}(\boldsymbol{\beta})(\mathbf{x} - \mathbf{D}\boldsymbol{\alpha})\|_2^2 \;\; \text{s.t.} \;\; \|\boldsymbol{\alpha}\|_0 \leq L, \tag{5}$$

or

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \|\boldsymbol{\alpha}\|_0 \;\; \text{s.t.} \;\; \| \operatorname{diag}(\boldsymbol{\beta})(\mathbf{x} - \mathbf{D}\boldsymbol{\alpha})\|_2^2 \leq \varepsilon \frac{\|\boldsymbol{\beta}\|_0}{m}, \tag{6}$$

or

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \frac{1}{2} \| \operatorname{diag}(\boldsymbol{\beta})(\mathbf{x} - \mathbf{D}\boldsymbol{\alpha})\|_2^2 + \lambda \|\boldsymbol{\alpha}\|_0. \tag{7}$$

where $\operatorname{diag}(\boldsymbol{\beta})$ is a diagonal matrix with the entries of $\boldsymbol{\beta}$ on the diagonal.

```
%
% Usage:   A=mexOMPMask(X,D,B,param);
% or       [A path]=mexOMPMask(X,D,B,param);
%
% Name: mexOMPMask
%
% Description: mexOMPMask is a variant of mexOMP that allow using
%     a binary mask B
%
%     for all columns x of X, and columns beta of B, it computes a column
%         alpha of A by addressing
%         min_{alpha} ||alpha||_0  s.t  ||diag(beta)*(x-Dalpha)||_2^2
%                                                               <= eps*||beta||_0/m
%         or
%         min_{alpha} ||diag(beta)*(x-Dalpha)||_2^2  s.t. ||alpha||_0 <= L
%         or
%         min_{alpha} 0.5||diag(beta)*(x-Dalpha)||_2^2  + lambda||alpha||_0
%
%
% Inputs: X:  double m x n matrix   (input signals)
%            m is the signal size
%            n is the number of signals to decompose
%         D:  double m x p matrix   (dictionary)
%            p is the number of elements in the dictionary
%            All the columns of D should have unit-norm !
%         B:  boolean m x n matrix   (mask)
%               p is the number of elements in the dictionary
%         param: struct
%            param.L (optional, maximum number of elements in each decomposition,
%               min(m,p) by default)
%            param.eps (optional, threshold on the squared l2-norm of the residual,
%               0 by default
%            param.lambda (optional, penalty parameter, 0 by default
%            param.numThreads (optional, number of threads for exploiting
%            multi-core / multi-cpus. By default, it takes the value -1,
%            which automatically selects all the available CPUs/cores).
%
% Output: A: double sparse p x n matrix (output coefficients)
%         path (optional): double dense p x L matrix
%                                      (regularization path of the first signal)
%
```

```
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%      - single precision setting (even though the output alpha is double
%        precision)
%      - Passing an int32 vector of length n to param.L provides
%        a different parameter L for each input signal x_i
%      - Passing a double vector of length n to param.eps and or param.lambda
%        provides a different parameter eps (or lambda) for each input signal x_i
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
clear all;

randn('seed',0);
fprintf('test mexOMPMask\n');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Decomposition of a large number of signals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Data are generated
X=randn(100,100);
X=X./repmat(sqrt(sum(X.^2)),[size(X,1) 1]);
D=randn(100,20);
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);
mask=(X > 0); % generating a binary mask

% parameter of the optimization procedure are chosen
param.L=20; % not more than 20 non-zeros coefficients (default: min(size(D,1),size(D,2)))
param.eps=0.01; %
param.numThreads=-1; % number of processors/cores to use; the default choice is -1
                     % and uses all the cores of the machine
tic
alpha=mexOMPMask(X,D,mask,param);
t=toc;
toc

fprintf('%f signals processed per second\n',size(X,2)/t);
```

4.3 Function mexRidgeRegression

This is a ridge regression solver using a conjugate gradient solver.

```
%
% Usage: [W]=mexRidgeRegression(Y,X,W0,param);
%
% Name: mexRidgeRegression
%
% Description: mexFistaFlat solves sparse regularized problems.
%         X is a design matrix of size m x p
%         X=[x^1,...,x^n]', where the x_i's are the rows of X
%         Y=[y^1,...,y^n] is a matrix of size m x n
%         It implements a conjugate gradient solver for ridge regression
%
% Inputs: Y:  double dense m x n matrix
%         X:  double dense or sparse m x p matrix
%         W0:  double dense p x n matrix or p x Nn matrix (for multi-logistic loss)
%              initial guess
%         param: struct
%            param.lambda (regularization parameter)
%            param.numThreads (optional, number of threads for exploiting
%                multi-core / multi-cpus. By default, it takes the value -1,
%                which automatically selects all the available CPUs/cores).
%            param.itermax (optional, maximum number of iterations, 100 by default)
%            param.tol (optional, tolerance for stopping criteration, which is a relative
   duality gap
%                if it is available, or a relative change of parameters).
%
% Output:  W:  double dense p x n matrix

% Author: Julien Mairal, 2013
```

The following piece of code illustrates how to use this function.

```
format compact;
randn('seed',0);
param.numThreads=-1; % all cores (-1 by default)
param.lambda=0.05; % regularization parameter

X=randn(100,200);
X=X-repmat(mean(X),[size(X,1) 1]);
X=mexNormalize(X);
Y=randn(100,1);
Y=Y-repmat(mean(Y),[size(Y,1) 1]);
Y=mexNormalize(Y);
W0=zeros(size(X,2),size(Y,2));
% Regression experiments
% 100 regression problems with the same design matrix X.
fprintf('\nVarious regression experiments\n');
fprintf('\nRidge Regression with conjugate gradient solver\n');
tic
[W]=mexRidgeRegression(Y,X,W0,param);
t=toc
```

4.4 Function mexLasso

This is a fast implementation of the LARS algorithm [8] (variant for solving the Lasso) for solving the Lasso or Elastic-Net. Given a matrix of signals $\mathbf{X} = [\mathbf{x}^1, \ldots, \mathbf{x}^n]$ in $\mathbb{R}^{m \times n}$ and a dictionary $\mathbf{D}$ in $\mathbb{R}^{m \times p}$, depending on the input parameters, the algorithm returns a matrix of coefficients $\mathbf{A} = [\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^n]$ in $\mathbb{R}^{p \times n}$ such that for every column $\mathbf{x}$ of $\mathbf{X}$, the corresponding column $\boldsymbol{\alpha}$ of $\mathbf{A}$ is the solution of

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 \;\; \text{s.t.} \;\; \|\boldsymbol{\alpha}\|_1 \leq \lambda, \tag{8}$$

or

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \|\boldsymbol{\alpha}\|_1 \;\; \text{s.t.} \;\; \|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 \leq \lambda, \tag{9}$$

or

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \frac{1}{2}\|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{\alpha}\|_2^2. \tag{10}$$

For efficiency reasons, the method first compute the covariance matrix $\mathbf{D}^T\mathbf{D}$, then for each signal, it computes $\mathbf{D}^T\mathbf{x}$ and performs the decomposition with a Cholesky-based algorithm (see [8] for instance). The implementation has also an option to add **positivity constraints** on the solutions $\boldsymbol{\alpha}$. When the solution is very sparse and the problem size is reasonable, this approach can be very efficient. Moreover, it gives the solution with an exact precision, and its performance does not depend on the correlation of the dictionary elements, except when the solution is not unique (the algorithm breaks in this case).

Note that mexLasso can return the whole regularization path of the first signal $\mathbf{x}_1$ and can handle implicitely the matrix $\mathbf{D}$ if the quantities $\mathbf{D}^T\mathbf{D}$ and $\mathbf{D}^T\mathbf{x}$ are passed as an argument, see below:

```
%
% Usage:  [A [path]]=mexLasso(X,D,param);
%  or:    [A [path]]=mexLasso(X,Q,q,param);
%
% Name: mexLasso
%
% Description: mexLasso is an efficient implementation of the
%     homotopy-LARS algorithm for solving the Lasso.
%
%     if the function is called this way [A [path]]=mexLasso(X,D,param),
%     it aims at addressing the following problems
%     for all columns x of X, it computes one column alpha of A
%     that solves
%       1) when param.mode=0
%         min_{alpha} ||x-Dalpha||_2^2 s.t. ||alpha||_1 <= lambda
%       2) when param.mode=1
%         min_{alpha} ||alpha||_1 s.t. ||x-Dalpha||_2^2 <= lambda
%       3) when param.mode=2
%         min_{alpha} 0.5||x-Dalpha||_2^2 + lambda||alpha||_1 +0.5 lambda2||alpha||_2^2
%
%     if the function is called this way [A [path]]=mexLasso(X,Q,q,param),
%     it solves the above optimisation problem, when Q=D'D and q=D'x.
%
%     Possibly, when param.pos=true, it solves the previous problems
%     with positivity constraints on the vectors alpha
%
% Inputs: X:  double m x n matrix   (input signals)
%               m is the signal size
%               n is the number of signals to decompose
%         D:  double m x p matrix   (dictionary)
%               p is the number of elements in the dictionary
%         param: struct
%               param.lambda  (parameter)
%               param.lambda2  (optional parameter for solving the Elastic-Net)
%                              for mode=0 and mode=1, it adds a ridge on the Gram Matrix
%               param.L (optional), maximum number of steps of the homotopy algorithm (can
%                        be used as a stopping criterion)
%               param.pos (optional, adds non-negativity constraints on the
%                 coefficients, false by default)
%               param.mode (see above, by default: 2)
%               param.numThreads (optional, number of threads for exploiting
%                 multi-core / multi-cpus. By default, it takes the value -1,
%                 which automatically selects all the available CPUs/cores).
%               param.cholesky (optional, default false),  choose between Cholesky
%                 implementation or one based on the matrix inversion Lemma
%               param.ols (optional, default false), perform an orthogonal projection
%                 before returning the solution.
%               param.max_length_path (optional) maximum length of the path, by default 4*p
%
```

```
% Output: A: double sparse p x n matrix (output coefficients)
%         path: optional,  returns the regularisation path for the first signal
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%         - single precision setting (even though the output alpha is double
%           precision)
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
clear all;

randn('seed',0);
fprintf('test mexLasso\n');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Decomposition of a large number of signals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Data are generated
X=randn(100,100000);
X=X./repmat(sqrt(sum(X.^2)),[size(X,1) 1]);
D=randn(100,200);
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);

% parameter of the optimization procedure are chosen
%param.L=20; % not more than 20 non-zeros coefficients (default: min(size(D,1),size(D,2)))
param.lambda=0.15; % not more than 20 non-zeros coefficients
param.numThreads=-1; % number of processors/cores to use; the default choice is -1
                     % and uses all the cores of the machine
param.mode=2;        % penalized formulation

tic
alpha=mexLasso(X,D,param);
t=toc
fprintf('%f signals processed per second\n',size(X,2)/t);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Regularization path of a single signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X=randn(64,1);
D=randn(64,10);
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);
param.lambda=0;
[alpha path]=mexLasso(X,D,param);
```

4.5 Function mexLassoWeighted

This is a fast implementation of a weighted version of LARS [8]. Given a matrix of signals $\mathbf{X} = [\mathbf{x}^1, \ldots, \mathbf{x}^n]$ in $\mathbb{R}^{m \times n}$, a matrix of weights $\mathbf{W} = [\mathbf{w}^1, \ldots, \mathbf{w}^n] \in \mathbb{R}^{p \times n}$, and a dictionary $\mathbf{D}$ in $\mathbb{R}^{m \times p}$, depending on the input parameters, the algorithm returns a matrix of coefficients $\mathbf{A} = [\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^n]$ in $\mathbb{R}^{p \times n}$, such that for every column $\mathbf{x}$ of $\mathbf{X}$, $\mathbf{w}$ of $\mathbf{W}$, it computes a column $\boldsymbol{\alpha}$ of $\mathbf{A}$, which is the solution of

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 \ \text{ s.t. } \ \|\operatorname{diag}(\mathbf{w})\boldsymbol{\alpha}\|_1 \leq \lambda, \tag{11}$$

or

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \|\operatorname{diag}(\mathbf{w})\boldsymbol{\alpha}\|_1 \ \text{ s.t. } \ \|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 \leq \lambda, \tag{12}$$

or

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \frac{1}{2}\|\mathbf{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2 + \lambda\|\operatorname{diag}(\mathbf{w})\boldsymbol{\alpha}\|_1. \tag{13}$$

The implementation has also an option to add **positivity constraints** on the solutions $\boldsymbol{\alpha}$. This function is potentially useful for implementing efficiently the randomized Lasso of [27], or reweighted-ℓ_1 schemes [4].

```
%
% Usage:  A=mexLassoWeighted(X,D,W,param);
%
% Name: mexLassoWeighted.
%
% WARNING: This function has not been tested intensively
%
% Description: mexLassoWeighted is an efficient implementation of the
%     LARS algorithm for solving the weighted Lasso. It is optimized
%     for solving a large number of small or medium-sized
%     decomposition problem (and not for a single large one).
%     It first computes the Gram matrix D'D and then perform
%     a Cholesky-based OMP of the input signals in parallel.
%     For all columns x of X, and w of W, it computes one column alpha of A
%     which is the solution of
%       1) when param.mode=0
%         min_{alpha} ||x-Dalpha||_2^2   s.t.
%                                     ||diag(w)alpha||_1 <= lambda
%       2) when param.mode=1
%         min_{alpha} ||diag(w)alpha||_1  s.t.
%                                        ||x-Dalpha||_2^2 <= lambda
%       3) when param.mode=2
%         min_{alpha} 0.5||x-Dalpha||_2^2  +
%                                         lambda||diag(w)alpha||_1
%     Possibly, when param.pos=true, it solves the previous problems
%     with positivity constraints on the vectors alpha
%
% Inputs: X:  double m x n matrix   (input signals)
%               m is the signal size
%               n is the number of signals to decompose
%         D:  double m x p matrix   (dictionary)
%               p is the number of elements in the dictionary
%         W:  double p x n matrix   (weights)
%         param: struct
%            param.lambda  (parameter)
%            param.L (optional, maximum number of elements of each
%            decomposition)
%            param.pos (optional, adds positivity constraints on the
%            coefficients, false by default)
%            param.mode (see above, by default: 2)
%            param.numThreads (optional, number of threads for exploiting
%            multi-core / multi-cpus. By default, it takes the value -1,
%            which automatically selects all the available CPUs/cores).
%
% Output:  A: double sparse p x n matrix (output coefficients)
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%         - single precision setting (even though the output alpha is double
%           precision)
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
clear all;

fprintf('test Lasso weighted\n');
randn('seed',0);
% Data are generated
X=randn(64,10000);
X=X./repmat(sqrt(sum(X.^2)),[size(X,1) 1]);
D=randn(64,256);
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);

% parameter of the optimization procedure are chosen
param.L=20; % not more than 20 non-zeros coefficients (default: min(size(D,1),size(D,2)))
param.lambda=0.15; % not more than 20 non-zeros coefficients
param.numThreads=8; % number of processors/cores to use; the default choice is -1
                    % and uses all the cores of the machine
param.mode=2;       % penalized formulation

W=rand(size(D,2),size(X,2));

tic
alpha=mexLassoWeighted(X,D,W,param);
t=toc;
toc

fprintf('%f signals processed per second\n',size(X,2)/t);
```

4.6 Function mexLassoMask

This is a variant of mexLasso with the possibility of adding a mask $\mathbf{B} = [\boldsymbol{\beta}^1, \ldots, \boldsymbol{\beta}^n]$, as in mexOMPMask. For every column $\mathbf{x}$ of $\mathbf{X}$, $\boldsymbol{\beta}$ of $\mathbf{B}$, it computes a column $\boldsymbol{\alpha}$ of $\mathbf{A}$, which is the solution of

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \| \operatorname{diag}(\boldsymbol{\beta})(\mathbf{x} - \mathbf{D}\boldsymbol{\alpha})\|_2^2 \ \text{ s.t. } \ \|\boldsymbol{\alpha}\|_1 \leq \lambda, \tag{14}$$

or

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \|\boldsymbol{\alpha}\|_1 \ \text{ s.t. } \ \| \operatorname{diag}(\boldsymbol{\beta})(\mathbf{x} - \mathbf{D}\boldsymbol{\alpha})\|_2^2 \leq \lambda \frac{\|\boldsymbol{\beta}\|_0}{m}, \tag{15}$$

or

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \frac{1}{2}\| \operatorname{diag}(\boldsymbol{\beta})(\mathbf{x} - \mathbf{D}\boldsymbol{\alpha})\|_2^2 + \lambda \frac{\|\boldsymbol{\beta}\|_0}{m} \|\boldsymbol{\alpha}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{\alpha}\|_2^2. \tag{16}$$

```
%
% Usage:   A=mexLassoMask(X,D,B,param);
%
% Name: mexLassoMask
%
% Description: mexLasso is a variant of mexLasso that handles
%     binary masks. It aims at addressing the following problems
%     for all columns x of X, and beta of B, it computes one column alpha of A
%     that solves
%       1) when param.mode=0
%         min_{alpha} ||diag(beta)(x-Dalpha)||_2^2 s.t. ||alpha||_1 <= lambda
%       2) when param.mode=1
%         min_{alpha} ||alpha||_1 s.t. ||diag(beta)(x-Dalpha)||_2^2
%                                                            <= lambda*||beta||_0/m
%       3) when param.mode=2
%         min_{alpha} 0.5||diag(beta)(x-Dalpha)||_2^2 +
%                                                 lambda*(||beta||_0/m)*||alpha||_1 +
%                                                 (lambda2/2)||alpha||_2^2
%     Possibly, when param.pos=true, it solves the previous problems
%     with positivity constraints on the vectors alpha
%
% Inputs: X:  double m x n matrix   (input signals)
%               m is the signal size
%               n is the number of signals to decompose
%         D:  double m x p matrix   (dictionary)
%               p is the number of elements in the dictionary
%         B:  boolean m x n matrix   (mask)
%               p is the number of elements in the dictionary
%         param: struct
%               param.lambda  (parameter)
%               param.L (optional, maximum number of elements of each
%                 decomposition)
%               param.pos (optional, adds positivity constraints on the
%                 coefficients, false by default)
%               param.mode (see above, by default: 2)
%               param.lambda2  (optional parameter for solving the Elastic-Net)
%                              for mode=0 and mode=1, it adds a ridge on the Gram Matrix
%               param.numThreads (optional, number of threads for exploiting
%                 multi-core / multi-cpus. By default, it takes the value -1,
%                 which automatically selects all the available CPUs/cores).
%
% Output: A: double sparse p x n matrix (output coefficients)
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%         - single precision setting (even though the output alpha is double
%           precision)
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
clear all;

randn('seed',0);
fprintf('test mexLasso\n');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Decomposition of a large number of signals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Data are generated
X=randn(100,100);
X=X./repmat(sqrt(sum(X.^2)),[size(X,1) 1]);
D=randn(100,20);
```

```
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);
mask=(X > 0); % generating a binary mask

% parameter of the optimization procedure are chosen
%param.L=20; % not more than 20 non-zeros coefficients (default: min(size(D,1),size(D,2)))
param.lambda=0.15; % not more than 20 non-zeros coefficients
param.numThreads=-1; % number of processors/cores to use; the default choice is -1
                     % and uses all the cores of the machine
param.mode=2;        % penalized formulation

tic
alpha=mexLassoMask(X,D,mask,param);
t=toc;
toc

fprintf('%f signals processed per second\n',size(X,2)/t);
```

4.7 Function mexCD

Coordinate-descent approach for solving Eq. (10) and Eq. (9). Note that unlike mexLasso, it is not implemented to solve the Elastic-Net formulation. To solve Eq. (9), the algorithm solves a sequence of problems of the form (10) using simple heuristics. Coordinate descent is very simple and in practice very powerful. It performs better when the correlation between the dictionary elements is small.

```
%
% Usage:   A=mexCD(X,D,A0,param);
%
% Name: mexCD
%
% Description: mexCD addresses l1-decomposition problem with a
%     coordinate descent type of approach.
%     It is optimized for solving a large number of small or medium-sized
%     decomposition problem (and not for a single large one).
%     It first computes the Gram matrix D'D.
%     This method is particularly well adapted when there is low
%     correlation between the dictionary elements and when one can benefit
%     from a warm restart.
%     It aims at addressing the two following problems
%     for all columns x of X, it computes a column alpha of A such that
%       2) when param.mode=1
%         min_{alpha} ||alpha||_1 s.t. ||x-Dalpha||_2^2 <= lambda
%         For this constraint setting, the method solves a sequence of
%         penalized problems (corresponding to param.mode=2) and looks
%         for the corresponding Lagrange multplier with a simple but
%         efficient heuristic.
%       3) when param.mode=2
%         min_{alpha} 0.5||x-Dalpha||_2^2 + lambda||alpha||_1
%
% Inputs: X:  double m x n matrix   (input signals)
%               m is the signal size
%               n is the number of signals to decompose
%         D:  double m x p matrix   (dictionary)
%               p is the number of elements in the dictionary
%               All the columns of D should have unit-norm !
%         A0:  double sparse p x n matrix   (initial guess)
%         param: struct
%            param.lambda  (parameter)
%            param.mode (optional, see above, by default 2)
%            param.itermax (maximum number of iterations)
%            param.tol (tolerance parameter)
%            param.numThreads (optional, number of threads for exploiting
%            multi-core / multi-cpus. By default, it takes the value -1,
%            which automatically selects all the available CPUs/cores).
%
% Output: A: double sparse p x n matrix (output coefficients)
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%         - single precision setting (even though the output alpha
%           is double precision)
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
clear all;

fprintf('test mexCD\n');
randn('seed',0);
% Data are generated
X=randn(64,100);
X=X./repmat(sqrt(sum(X.^2)),[size(X,1) 1]);
D=randn(64,100);
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);

% parameter of the optimization procedure are chosen
param.lambda=0.015; % not more than 20 non-zeros coefficients
param.numThreads=4; % number of processors/cores to use; the default choice is -1
param.mode=2;       % penalized formulation

tic
alpha=mexLasso(X,D,param);
t=toc;
toc
E=mean(0.5*sum((X-D*alpha).^2)+param.lambda*sum(abs(alpha)));
fprintf('%f signals processed per second for LARS\n',size(X,2)/t);
fprintf('Objective function for LARS: %g\n',E);

param.tol=0.001;
param.itermax=1000;
tic
alpha2=mexCD(X,D,sparse(size(alpha,1),size(alpha,2)),param);
t=toc;
toc

fprintf('%f signals processed per second for CD\n',size(X,2)/t);
E=mean(0.5*sum((X-D*alpha2).^2)+param.lambda*sum(abs(alpha2)));
fprintf('Objective function for CD: %g\n',E);
fprintf('With Random Design, CD can be much faster than LARS\n');
```

4.8 Function mexSOMP

This is a fast implementation of the Simultaneous Orthogonal Matching Pursuit algorithm. Given a set of matrices $\mathbf{X} = [\mathbf{X}^1, \dots, \mathbf{X}^n]$ in $\mathbb{R}^{m \times N}$, where the $\mathbf{X}^i$'s are in $\mathbb{R}^{m \times n_i}$, and a dictionary $\mathbf{D}$ in $\mathbb{R}^{m \times p}$, the algorithm returns a matrix of coefficients $\mathbf{A} = [\mathbf{A}^1, \dots, \mathbf{A}^n]$ in $\mathbb{R}^{p \times N}$ which is an approximate solution of the following NP-hard problem

$$\forall i \quad \min_{\mathbf{A}^i \in \mathbb{R}^{p \times n_i}} \|\mathbf{X}^i - \mathbf{D}\mathbf{A}^i\|_F^2 \;\; \text{s.t.} \;\; \|\mathbf{A}^i\|_{0,\infty} \leq L. \tag{17}$$

or

$$\forall i \quad \min_{\mathbf{A}^i \in \mathbb{R}^{p \times n_i}} \|\mathbf{A}^i\|_{0,\infty} \;\; \text{s.t.} \;\; \|\mathbf{X}^i - \mathbf{D}\mathbf{A}^i\|_F^2 \leq \varepsilon n_i. \tag{18}$$

To be efficient, the method first compute the covariance matrix $\mathbf{D}^T\mathbf{D}$, then for each signal, it computes $\mathbf{D}^T\mathbf{X}^i$ and performs the decomposition with a Cholesky-based algorithm.

```
%
% Usage:   alpha=mexSOMP(X,D,list_groups,param);
%
% Name: mexSOMP
%     (this function has not been intensively tested).
%
% Description: mexSOMP is an efficient implementation of a
%     Simultaneous Orthogonal Matching Pursuit algorithm. It is optimized
%     for solving a large number of small or medium-sized
%     decomposition problem (and not for a single large one).
%     It first computes the Gram matrix D'D and then perform
%     a Cholesky-based OMP of the input signals in parallel.
%     It aims at addressing the following NP-hard problem
%
%     X is a matrix structured in groups of signals, which we denote
%     by X=[X_1,...,X_n]
%
%     for all matrices X_i of X,
%         min_{A_i} ||A_i||_{0,infty}  s.t  ||X_i-D A_i||_2^2 <= eps*n_i
%         where n_i is the number of columns of X_i
%
%         or
%
%         min_{A_i} ||X_i-D A_i||_2^2  s.t. ||A_i||_{0,infty} <= L
%
%
% Inputs: X:  double m x N matrix   (input signals)
%            m is the signal size
%            N is the total number of signals
%         D:  double m x p matrix   (dictionary)
%            p is the number of elements in the dictionary
%            All the columns of D should have unit-norm !
%         list_groups : int32 vector containing the indices (starting at 0)
%            of the first elements of each groups.
%         param: struct
%            param.L (maximum number of elements in each decomposition)
%            param.eps (threshold on the squared l2-norm of the residual
%            param.numThreads (optional, number of threads for exploiting
%            multi-core / multi-cpus. By default, it takes the value -1,
%            which automatically selects all the available CPUs/cores).
%
% Output: alpha: double sparse p x N matrix (output coefficients)
```

```
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%      - single precision setting (even though the output alpha is double
%        precision)
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
clear all;
randn('seed',0);

fprintf('test mexSOMP\n');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Decomposition of a large number of groups
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X=randn(64,10000);
D=randn(64,200);
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);
ind_groups=int32(0:10:9999); % indices of the first signals in each group

% parameter of the optimization procedure are chosen
param.L=10; % not more than 10 non-zeros coefficients
param.eps=0.1; % squared norm of the residual should be less than 0.1
param.numThreads=-1; % number of processors/cores to use; the default choice is -1
                    % and uses all the cores of the machine
tic
alpha=mexSOMP(X,D,ind_groups,param);
t=toc
fprintf('%f signals processed per second\n',size(X,2)/t);
```

4.9 Function mexL1L2BCD

This is a fast implementation of a simultaneous signal decomposition formulation. Given a set of matrices $\mathbf{X} = [\mathbf{X}^1, \ldots, \mathbf{X}^n]$ in $\mathbb{R}^{m \times N}$, where the $\mathbf{X}^i$'s are in $\mathbb{R}^{m \times n_i}$, and a dictionary $\mathbf{D}$ in $\mathbb{R}^{m \times p}$, the algorithm returns a matrix of coefficients $\mathbf{A} = [\mathbf{A}^1, \ldots, \mathbf{A}^n]$ in $\mathbb{R}^{p \times N}$ which is an approximate solution of the following NP-hard problem

$$\forall i \quad \min_{\mathbf{A}^i \in \mathbb{R}^{p \times n_i}} \|\mathbf{X}^i - \mathbf{D}\mathbf{A}^i\|_F^2 \ \text{ s.t. } \ \|\mathbf{A}^i\|_{1,2} \leq \frac{\lambda}{n_i}. \tag{19}$$

or

$$\forall i \quad \min_{\mathbf{A}^i \in \mathbb{R}^{p \times n_i}} \|\mathbf{A}^i\|_{1,2} \ \text{ s.t. } \ \|\mathbf{X}^i - \mathbf{D}\mathbf{A}^i\|_F^2 \leq \lambda n_i. \tag{20}$$

To be efficient, the method first compute the covariance matrix $\mathbf{D}^T\mathbf{D}$, then for each signal, it computes $\mathbf{D}^T\mathbf{X}^i$ and performs the decomposition with a Cholesky-based algorithm.

```
%
% Usage:   alpha=mexL1L2BCD(X,D,alpha0,list_groups,param);
%
% Name: mexL1L2BCD
%     (this function has not been intensively tested).
%
% Description: mexL1L2BCD is a solver for a
%     Simultaneous signal decomposition formulation based on block
%     coordinate descent.
```

```
%
%     X is a matrix structured in groups of signals, which we denote
%     by X=[X_1,...,X_n]
%
%     if param.mode=2, it solves
%         for all matrices X_i of X,
%         min_{A_i} 0.5||X_i-D A_i||_2^2 + lambda/sqrt(n_i)||A_i||_{1,2}
%         where n_i is the number of columns of X_i
%     if param.mode=1, it solves
%         min_{A_i} ||A_i||_{1,2} s.t. ||X_i-D A_i||_2^2  <= n_i lambda
%
%
% Inputs: X:  double m x N matrix   (input signals)
%            m is the signal size
%            N is the total number of signals
%         D:  double m x p matrix   (dictionary)
%            p is the number of elements in the dictionary
%         alpha0: double dense p x N matrix (initial solution)
%         list_groups : int32 vector containing the indices (starting at 0)
%            of the first elements of each groups.
%         param: struct
%            param.lambda (regularization parameter)
%            param.mode (see above, by default 2)
%            param.itermax (maximum number of iterations, by default 100)
%            param.tol (tolerance parameter, by default 0.001)
%            param.numThreads (optional, number of threads for exploiting
%            multi-core / multi-cpus. By default, it takes the value -1,
%            which automatically selects all the available CPUs/cores).
%
% Output: alpha: double sparse p x N matrix (output coefficients)
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%      - single precision setting (even though the output alpha is double
%        precision)
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
clear all;
randn('seed',0);

fprintf('test mexL1L2BCD\n');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Decomposition of a large number of groups
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
X=randn(64,100);
D=randn(64,200);
D=D./repmat(sqrt(sum(D.^2)),[size(D,1) 1]);
ind_groups=int32(0:10:size(X,2)-1); % indices of the first signals in each group

% parameter of the optimization procedure are chosen
param.itermax=100;
param.tol=1e-3;
param.mode=2; % penalty mode
param.lambda=0.15; % squared norm of the residual should be less than 0.1
param.numThreads=-1; % number of processors/cores to use; the default choice is -1
                    % and uses all the cores of the machine
tic
alpha0=zeros(size(D,2),size(X,2));
alpha=mexL1L2BCD(X,D,alpha0,ind_groups,param);
t=toc
fprintf('%f signals processed per second\n',size(X,2)/t);
```

4.10 Function mexSparseProject

This is a multi-purpose function, implementing fast algorithms for projecting on convex sets, but it also solves the fused lasso signal approximation problem. The proposed method is detailed in [20]. The main problems addressed by this function are the following: Given a matrix $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_n]$ in $\mathbb{R}^{m \times n}$, it finds a matrix $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ in $\mathbb{R}^{m \times n}$ so that for all column $\mathbf{u}$ of $\mathbf{U}$, it computes a column $\mathbf{v}$ of $\mathbf{V}$ solving

$$\min_{\mathbf{v} \in \mathbb{R}^m} \|\mathbf{u} - \mathbf{v}\|_2^2 \ \text{ s.t. } \ \|\mathbf{v}\|_1 \leq \tau, \tag{21}$$

or

$$\min_{\mathbf{v} \in \mathbb{R}^m} \|\mathbf{u} - \mathbf{v}\|_2^2 \ \text{ s.t. } \ \lambda_1 \|\mathbf{v}\|_1 + \lambda_2 \|\mathbf{v}\|_2^2 \leq \tau, \tag{22}$$

or

$$\min_{\mathbf{v} \in \mathbb{R}^m} \|\mathbf{u} - \mathbf{v}\|_2^2 \ \text{ s.t. } \ \lambda_1 \|\mathbf{v}\|_1 + \lambda_2 \|\mathbf{v}\|_2^2 + \lambda_3 FL(\mathbf{v}) \leq \tau, \tag{23}$$

or

$$\min_{\mathbf{v} \in \mathbb{R}^m} \frac{1}{2} \|\mathbf{u} - \mathbf{v}\|_2^2 + \lambda_1 \|\mathbf{v}\|_1 + \lambda_2 \|\mathbf{v}\|_2^2 + \lambda_3 FL(\mathbf{v}). \tag{24}$$

Note that for the two last cases, the method performs a small approximation. The method follows the regularization path, goes from one kink to another, and stop whenever the constraint is not satisfied anymore. The solution returned by the algorithm is the one obtained at the last kink of the regularization path, which is in practice close, but not exactly the same as the solution. This will be corrected in a future release of the toolbox.

```
%
% Usage:  V=mexSparseProject(U,param);
%
% Name: mexSparseProject
%
% Description: mexSparseProject solves various optimization
%     problems, including projections on a few convex sets.
%     It aims at addressing the following problems
%     for all columns u of U in parallel
%       1) when param.mode=1 (projection on the l1-ball)
%           min_v ||u-v||_2^2  s.t.  ||v||_1 <= thrs
%       2) when param.mode=2
%           min_v ||u-v||_2^2  s.t. ||v||_2^2 + lamuda1||v||_1 <= thrs
%       3) when param.mode=3
%           min_v ||u-v||_2^2  s.t  ||v||_1 + 0.5lamuda1||v||_2^2 <= thrs
%       4) when param.mode=4
%           min_v 0.5||u-v||_2^2 + lamuda1||v||_1  s.t  ||v||_2^2 <= thrs
%       5) when param.mode=5
%           min_v 0.5||u-v||_2^2 + lamuda1||v||_1 +lamuda2 FL(v) + ...
%                                                 0.5lamuda_3 ||v||_2^2
%          where FL denotes a "fused lasso" regularization term.
%       6) when param.mode=6
%          min_v ||u-v||_2^2 s.t lamuda1||v||_1 +lamuda2 FL(v) + ...
%                                              0.5lamuda3||v||_2^2 <= thrs
%
%        When param.pos=true and param.mode <= 4,
%        it solves the previous problems with positivity constraints
%
% Inputs: U:  double m x n matrix   (input signals)
%               m is the signal size
%               n is the number of signals to project
%         param: struct
%           param.thrs (parameter)
%           param.lambda1 (parameter)
%           param.lambda2 (parameter)
%           param.lambda3 (parameter)
%           param.mode (see above)
%           param.pos (optional, false by default)
%           param.numThreads (optional, number of threads for exploiting
%             multi-core / multi-cpus. By default, it takes the value -1,
%             which automatically selects all the available CPUs/cores).
%
% Output: V: double m x n matrix (output matrix)
%
% Note: this function admits a few experimental usages, which have not
%     been extensively tested:
%         - single precision setting
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
clear all;
randn('seed',0);
% Data are generated
X=randn(20000,100);
X=X./repmat(sqrt(sum(X.^2)),[size(X,1) 1]);

% parameter of the optimization procedure are chosen
param.numThreads=-1; % number of processors/cores to use; the default choice is -1
                    % and uses all the cores of the machine

param.pos=0;
param.mode=1;       % projection on the l1 ball
param.thrs=2;
tic
X1=mexSparseProject(X,param);
t=toc;
toc
fprintf('%f signals of size %d projected per second\n',size(X,2)/t,size(X,1));
fprintf('Checking constraint: %f, %f\n',min(sum(abs(X1))),max(sum(abs(X1))));



param.mode=2;       % projection on the Elastic-Net
param.lambda1=0.15;

tic
```

```
X1=mexSparseProject(X,param);
t=toc;
toc
fprintf('%f signals of size %d projected per second\n',size(X,2)/t,size(X,1));
constraints=sum((X1.^2))+param.lambda1*sum(abs(X1));
fprintf('Checking constraint: %f, %f\n',min(constraints),max(constraints));

param.mode=6;       % projection on the FLSA
param.lambda1=0.7;
param.lambda2=0.7;
param.lambda3=1.0;

X=rand(2000,100);
X=X./repmat(sqrt(sum(X.^2)),[size(X,1) 1]);

tic
X1=mexSparseProject(X,param);
t=toc;
toc
fprintf('%f signals of size %d projected per second\n',size(X,2)/t,size(X,1));
constraints=0.5*param.lambda3*sum(X1.^2)+param.lambda1*sum(abs(X1))+param.lambda2*sum(abs(X1(2:
    end,:)-X1(1:end-1,:)));
fprintf('Checking constraint: %f, %f\n',mean(constraints),max(constraints));
fprintf('Projection is approximate (stops at a kink)\n',mean(constraints),max(constraints));

param.mode=6;       % projection on the FLSA
param.lambda1=0.7;
param.lambda2=0.7;
param.lambda3=1.0;

X=rand(2000,100);
X=X./repmat(sqrt(sum(X.^2)),[size(X,1) 1]);

tic
X1=mexSparseProject(X,param);
t=toc;
toc
fprintf('%f signals of size %d projected per second\n',size(X,2)/t,size(X,1));
constraints=0.5*param.lambda3*sum(X1.^2)+param.lambda1*sum(abs(X1))+param.lambda2*sum(abs(X1(2:
    end,:)-X1(1:end-1,:)));
fprintf('Checking constraint: %f, %f\n',mean(constraints),max(constraints));
fprintf('Projection is approximate (stops at a kink)\n',mean(constraints),max(constraints));
```

5 Proximal Toolbox

The previous toolbox we have presented is well adapted for solving a large number of small and medium-scale sparse decomposition problems with the square loss, which is typical from the classical dictionary learning framework. We now present a new software package that is adapted for solving a wide range of possibly large-scale learning problems, with several combinations of losses and regularization terms. The method implements the proximal methods of [1], and includes the proximal solvers for the tree-structured regularization of [14], and the solver of [21] for general structured sparse regularization. The solver for structured sparse regularization norms includes a C++ max-flow implementation of the push-relabel algorithm of [12], with heuristics proposed by [5].

This implementation also provides robust stopping criteria based on *duality gaps*, which are presented in Appendix A. It can handle intercepts (unregularized variables). The general formulation that our software can solve take the form

$$\min_{\mathbf{w}\in\mathbb{R}^p} [g(\mathbf{w}) \triangleq f(\mathbf{w}) + \lambda\psi(\mathbf{w})],$$

where f is a smooth loss function and ψ is a regularization function. When one optimizes a matrix $\mathbf{W}$ in $\mathbb{R}^{p\times r}$ instead of a vector $\mathbf{w}$ in $\mathbb{R}^p$, we will write

$$\min_{\mathbf{W}\in\mathbb{R}^{p\times r}} [g(\mathbf{W}) \triangleq f(\mathbf{W}) + \lambda\psi(\mathbf{W})].$$

Note that the software can possibly handle nonnegativity constraints.

We start by presenting the type of regularization implemented in the software

5.1 Regularization Functions

Our software can handle the following regularization functions ψ for vectors $\mathbf{w}$ in $\mathbb{R}^p$:

- **The Tikhonov regularization**: $\psi(\mathbf{w}) \triangleq \frac{1}{2}\|\mathbf{w}\|_2^2$.
- **The ℓ_1-norm**: $\psi(\mathbf{w}) \triangleq \|\mathbf{w}\|_1$.
- **The Elastic-Net**: $\psi(\mathbf{w}) \triangleq \|\mathbf{w}\|_1 + \gamma\|\mathbf{w}\|_2^2$.
- **The Fused-Lasso**: $\psi(\mathbf{w}) \triangleq \|\mathbf{w}\|_1 + \gamma\|\mathbf{w}\|_2^2 + \gamma_2\sum_{i=1}^{p-1}|\mathbf{w}_{i+1} - \mathbf{w}_i|$.
- **The group Lasso**: $\psi(\mathbf{w}) \triangleq \sum_{g\in\mathcal{G}}\eta_g\|\mathbf{w}_g\|_2$, where $\mathcal{G}$ are groups of variables.
- **The group Lasso with ℓ_∞-norm**: $\psi(\mathbf{w}) \triangleq \sum_{g\in\mathcal{G}}\eta_g\|\mathbf{w}_g\|_\infty$, where $\mathcal{G}$ are groups of variables.
- **The sparse group Lasso**: same as above but with an additional ℓ_1 term.
- **The tree-structured sum of ℓ_2-norms**: $\psi(\mathbf{w}) \triangleq \sum_{g\in\mathcal{G}}\eta_g\|\mathbf{w}_g\|_2$, where $\mathcal{G}$ is a tree-structured set of groups [14], and the η_g are positive weights.
- **The tree-structured sum of ℓ_∞-norms**: $\psi(\mathbf{w}) \triangleq \sum_{g\in\mathcal{G}}\eta_g\|\mathbf{w}_g\|_\infty$. See [14]
- **General sum of ℓ_∞-norms**: $\psi(\mathbf{w}) \triangleq \sum_{g\in\mathcal{G}}\eta_g\|\mathbf{w}_g\|_\infty$, where no assumption are made on the groups $\mathcal{G}$.

Our software also handles regularization functions ψ on matrices $\mathbf{W}$ in $\mathbb{R}^{p\times r}$ (note that $\mathbf{W}$ can be transposed in these formulations). In particular,

- **The ℓ_1/ℓ_2-norm**: $\psi(\mathbf{W}) \triangleq \sum_{i=1}^p\|\mathbf{W}_i\|_2$, where $\mathbf{W}_i$ denotes the i-th row of $\mathbf{W}$.
- **The ℓ_1/ℓ_∞-norm**: $\psi(\mathbf{W}) \triangleq \sum_{i=1}^p\|\mathbf{W}_i\|_\infty$,
- **The ℓ_1/ℓ_2+ℓ_1-norm**: $\psi(\mathbf{W}) \triangleq \sum_{i=1}^p\|\mathbf{W}_i\|_2 + \lambda_2\sum_{i,j}|\mathbf{W}_{ij}|$.
- **The ℓ_1/ℓ_∞+ℓ_1-norm**: $\psi(\mathbf{W}) \triangleq \sum_{i=1}^p\|\mathbf{W}_i\|_\infty + \lambda_2\sum_{i,j}|\mathbf{W}_{ij}|$,
- **The ℓ_1/ℓ_∞-norm on rows and columns**: $\psi(\mathbf{W}) \triangleq \sum_{i=1}^p\|\mathbf{W}_i\|_\infty + \lambda_2\sum_{j=1}^r\|\mathbf{W}^j\|_\infty$, where $\mathbf{W}^j$ denotes the j-th column of $\mathbf{W}$.
- **The multi-task tree-structured sum of ℓ_∞-norms**:

  $$\psi(\mathbf{W}) \triangleq \sum_{i=1}^r\sum_{g\in\mathcal{G}}\eta_g\|\mathbf{w}_g^i\|_\infty + \gamma\sum_{g\in\mathcal{G}}\eta_g\max_{j\in g}\|\mathbf{W}_j\|_\infty, \tag{25}$$

  where the first double sums is in fact a sum of independent structured norms on the columns $\mathbf{w}^i$ of $\mathbf{W}$, and the right term is a tree-structured regularization norm applied to the ℓ_∞-norm of the rows of $\mathbf{W}$, thereby inducing the tree-structured regularization at the row level. $\mathcal{G}$ is here a tree-structured set of groups.

- **The multi-task general sum of ℓ_∞-norms** is the same as Eq. (25) except that the groups $\mathcal{G}$ are general overlapping groups.
- **The trace norm**: $\psi(\mathbf{W}) \triangleq \|\mathbf{W}\|_*$.

Non-convex regularizations are also implemented with the ISTA algorithm (no duality gaps are of course provided in these cases):

- **The ℓ_0-pseudo-norm**: $\psi(\mathbf{w}) \triangleq \|\mathbf{w}\|_0$.
- **The rank**: $\psi(\mathbf{W}) \triangleq \text{randk}(\mathbf{W})$.
- **The tree-structured ℓ_0-pseudo-norm**: $\psi(\mathbf{w}) \triangleq \sum_{g\in\mathcal{G}} \delta_{\mathbf{w}_g \neq 0}$.

All of these regularization terms for vectors or matrices can be coupled with nonnegativity constraints. It is also possible to add an intercept, which one wishes not to regularize, and we will include this possibility in the next sections. There are also a few hidden undocumented options which are available in the source code.

We now present 3 functions for computing proximal operators associated to the previous regularization functions.

5.2 Function mexProximalFlat

This function computes the proximal operators associated to many regularization functions, for input signals $\mathbf{U} = [\mathbf{u}^1, \ldots, \mathbf{u}^n]$ in $\mathbb{R}^{p\times n}$, it finds a matrix $\mathbf{V} = [\mathbf{v}^1, \ldots, \mathbf{v}^n]$ in $\mathbb{R}^{p\times n}$ such that:

• If one chooses a regularization function on vectors, for every column $\mathbf{u}$ of $\mathbf{U}$, it computes one column $\mathbf{v}$ of $\mathbf{V}$ solving

$$\min_{\mathbf{v}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda\|\mathbf{v}\|_0, \tag{26}$$

or

$$\min_{\mathbf{v}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda\|\mathbf{v}\|_1, \tag{27}$$

or

$$\min_{\mathbf{v}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \frac{\lambda}{2}\|\mathbf{v}\|_2^2, \tag{28}$$

or

$$\min_{\mathbf{v}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda\|\mathbf{v}\|_1 + \lambda_2\|\mathbf{v}\|_2^2, \tag{29}$$

or

$$\min_{\mathbf{v}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda\sum_{j=1}^{p-1}|\mathbf{v}^i_{j+1}-\mathbf{v}^i_j| + \lambda_2\|\mathbf{v}\|_1 + \lambda_3\|\mathbf{v}\|_2^2, \tag{30}$$

or

$$\min_{\mathbf{v}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda\sum_{g\in\mathcal{T}}\delta^g(\mathbf{v}), \tag{31}$$

where $\mathcal{T}$ is a tree-structured set of groups (see [15]), and $\delta^g(\mathbf{v}) = 0$ if $\mathbf{v}_g = 0$ and 1 otherwise. It can also solve

$$\min_{\mathbf{v}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda\sum_{g\in\mathcal{T}}\eta^g\|\mathbf{v}_g\|_2, \tag{32}$$

or

$$\min_{\mathbf{v}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda\sum_{g\in\mathcal{T}}\eta^g\|\mathbf{v}_g\|_\infty, \tag{33}$$

or

$$\min_{\mathbf{v}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda\sum_{g\in\mathcal{G}}\eta^g\|\mathbf{v}_g\|_\infty, \tag{34}$$

where $\mathcal{G}$ is any kind of set of groups.

This function can also solve the following proximal operators on matrices

$$\min_{\mathbf{V}\in\mathbb{R}^{p\times n}} \frac{1}{2}\|\mathbf{U}-\mathbf{V}\|_F^2 + \lambda\sum_{i=1}^{p}\|\mathbf{V}_i\|_2, \tag{35}$$

where $\mathbf{V}_i$ is the i-th row of $\mathbf{V}$, or

$$\min_{\mathbf{V}\in\mathbb{R}^{p\times n}} \frac{1}{2}\|\mathbf{U}-\mathbf{V}\|_F^2 + \lambda\sum_{i=1}^{p}\|\mathbf{V}_i\|_\infty, \tag{36}$$

or

$$\min_{\mathbf{V}\in\mathbb{R}^{p\times n}} \frac{1}{2}\|\mathbf{U}-\mathbf{V}\|_F^2 + \lambda\sum_{i=1}^{p}\|\mathbf{V}_i\|_2 + \lambda_2\sum_{i=1}^{p}\sum_{j=1}^{n}|\mathbf{V}_{ij}|, \tag{37}$$

or

$$\min_{\mathbf{V}\in\mathbb{R}^{p\times n}} \frac{1}{2}\|\mathbf{U}-\mathbf{V}\|_F^2 + \lambda\sum_{i=1}^{p}\|\mathbf{V}_i\|_\infty + \lambda_2\sum_{i=1}^{p}\sum_{j=1}^{n}|\mathbf{V}_{ij}|, \tag{38}$$

or

$$\min_{\mathbf{V}\in\mathbb{R}^{p\times n}} \frac{1}{2}\|\mathbf{U}-\mathbf{V}\|_F^2 + \lambda\sum_{i=1}^{p}\|\mathbf{V}_i\|_\infty + \lambda_2\sum_{j=1}^{n}\|\mathbf{V}^j\|_\infty. \tag{39}$$

where $\mathbf{V}^j$ is the j-th column of $\mathbf{V}$.

See usage details below:

```
%
% Usage:  [V [val_regularizer]]=mexProximalFlat(U,param);
%
% Name: mexProximalFlat
%
% Description: mexProximalFlat computes proximal operators. Depending
%         on the value of param.regul, it computes
%
%         Given an input matrix U=[u^1,\ldots,u^n], it computes a matrix
%         V=[v^1,\ldots,v^n] such that
%         if one chooses a regularization functions on vectors, it computes
%         for each column u of U, a column v of V solving
%         if param.regul='l0'
%             argmin 0.5||u-v||_2^2 + lambda||v||_0
%         if param.regul='l1'
%             argmin 0.5||u-v||_2^2 + lambda||v||_1
%         if param.regul='l2'
%             argmin 0.5||u-v||_2^2 + 0.5lambda||v||_2^2
%         if param.regul='elastic-net'
%             argmin 0.5||u-v||_2^2 + lambda||v||_1 + lambda_2||v||_2^2
%         if param.regul='fused-lasso'
%             argmin 0.5||u-v||_2^2 + lambda FL(v) + ...
%                               ...  lambda_2||v||_1 + lambda_3||v||_2^2
%         if param.regul='linf'
%             argmin 0.5||u-v||_2^2 + lambda||v||_inf
%         if param.regul='l1-constraint'
%             argmin 0.5||u-v||_2^2 s.t. ||v||_1 <= lambda
%         if param.regul='l2-not-squared'
%             argmin 0.5||u-v||_2^2 + lambda||v||_2
%         if param.regul='group-lasso-l2'
%             argmin 0.5||u-v||_2^2 + lambda sum_g ||v_g||_2
%             where the groups are either defined by param.groups or by param.size_group,
%         if param.regul='group-lasso-linf'
%             argmin 0.5||u-v||_2^2 + lambda sum_g ||v_g||_inf
%         if param.regul='sparse-group-lasso-l2'
%             argmin 0.5||u-v||_2^2 + lambda sum_g ||v_g||_2 + lambda_2 ||v||_1
%             where the groups are either defined by param.groups or by param.size_group,
%         if param.regul='sparse-group-lasso-linf'
%             argmin 0.5||u-v||_2^2 + lambda sum_g ||v_g||_inf + lambda_2 ||v||_1
%         if param.regul='trace-norm-vec'
%             argmin 0.5||u-v||_2^2 + lambda ||mat(v)||_*
%            where mat(v) has param.size_group rows
%
%         if one chooses a regularization function on matrices
%         if param.regul='l1l2',  V=
%             argmin 0.5||U-V||_F^2 + lambda||V||_{1/2}
%         if param.regul='l1linf',  V=
%             argmin 0.5||U-V||_F^2 + lambda||V||_{1/inf}
%         if param.regul='l1l2+l1',  V=
%             argmin 0.5||U-V||_F^2 + lambda||V||_{1/2} + lambda_2||V||_{1/1}
%         if param.regul='l1linf+l1',  V=
%             argmin 0.5||U-V||_F^2 + lambda||V||_{1/inf} + lambda_2||V||_{1/1}
%         if param.regul='l1linf+row-column',  V=
%             argmin 0.5||U-V||_F^2 + lambda||V||_{1/inf} + lambda_2||V'||_{1/inf}
%         if param.regul='trace-norm',  V=
%             argmin 0.5||U-V||_F^2 + lambda||V||_*
%         if param.regul='rank',  V=
%             argmin 0.5||U-V||_F^2 + lambda rank(V)
%         if param.regul='none',  V=
%             argmin 0.5||U-V||_F^2
%
%         for all these regularizations, it is possible to enforce non-negativity constraints
%         with the option param.pos, and to prevent the last row of U to be regularized, with
%         the option param.intercept
%
% Inputs: U:  double m x n matrix   (input signals)
%               m is the signal size
%         param: struct
%               param.lambda  (regularization parameter)
%               param.regul (choice of regularization, see above)
%               param.lambda2  (optional, regularization parameter)
%               param.lambda3  (optional, regularization parameter)
%               param.verbose (optional, verbosity level, false by default)
%               param.intercept (optional, last row of U is not regularized,
%                 false by default)
%               param.transpose (optional, transpose the matrix in the regularization function)
%               param.size_group (optional, for regularization functions assuming a group
%                 structure). It is a scalar. When param.groups is not specified, it assumes
%                 that the groups are the sets of consecutive elements of size param.size_group
%               param.groups (int32, optional, for regularization functions assuming a group
%                 structure. It is an int32 vector of size m containing the group indices of
   the
%                 variables (first group is 1).
%               param.pos (optional, adds positivity constraints on the
%                 coefficients, false by default)
%               param.numThreads (optional, number of threads for exploiting
%                 multi-core / multi-cpus. By default, it takes the value -1,
%                 which automatically selects all the available CPUs/cores).
%
% Output: V: double m x n matrix (output coefficients)
%         val_regularizer: double 1 x n vector (value of the regularization
%         term at the optimum).
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
U=randn(100,1000);

param.lambda=0.1; % regularization parameter
param.num_threads=-1; % all cores (-1 by default)
param.verbose=true;   % verbosity, false by default

% test l0
fprintf('\nprox l0\n');
param.regul='l0';
param.pos=false;       % false by default
param.intercept=false; % false by default
alpha=mexProximalFlat(U,param);

% test l1
fprintf('\nprox l1, intercept, positivity constraint\n');
param.regul='l1';
param.pos=true;       % can be used with all the other regularizations
param.intercept=true; % can be used with all the other regularizations
alpha=mexProximalFlat(U,param);

% test l2
fprintf('\nprox squared-l2\n');
param.regul='l2';
param.pos=false;
param.intercept=false;
alpha=mexProximalFlat(U,param);

% test elastic-net
fprintf('\nprox elastic-net\n');
param.regul='elastic-net';
param.lambda2=0.1;
alpha=mexProximalFlat(U,param);

% test fused-lasso
fprintf('\nprox fused lasso\n');
param.regul='fused-lasso';
param.lambda2=0.1;
param.lambda3=0.1;
alpha=mexProximalFlat(U,param);

% test l1l2
fprintf('\nprox mixed norm l1/l2\n');
param.regul='l1l2';
alpha=mexProximalFlat(U,param);

% test l1linf
fprintf('\nprox mixed norm l1/linf\n');
param.regul='l1linf';
alpha=mexProximalFlat(U,param);
```

```
% test l1l2+l1
fprintf('\nprox mixed norm l1/l2 + l1\n');
param.regul='l1l2+l1';
param.lambda2=0.1;
alpha=mexProximalFlat(U,param);

% test l1linf+l1
fprintf('\nprox mixed norm l1/linf + l1\n');
param.regul='l1linf+l1';
param.lambda2=0.1;
alpha=mexProximalFlat(U,param);

% test l1linf-row-column
fprintf('\nprox mixed norm l1/linf on rows and columns\n');
param.regul='l1linf-row-column';
param.lambda2=0.1;
alpha=mexProximalFlat(U,param);

% test none
fprintf('\nprox no regularization\n');
param.regul='none';
alpha=mexProximalFlat(U,param);
```

5.3 Function mexProximalTree

This function computes the proximal operators associated to tree-structured regularization functions, for input signals $\mathbf{U} = [\mathbf{u}^1, \dots, \mathbf{u}^n]$ in $\mathbb{R}^{p \times n}$, and a tree-structured set of groups [14], it computes a matrix $\mathbf{V} = [\mathbf{v}^1, \dots, \mathbf{v}^n]$ in $\mathbb{R}^{p \times n}$. When one uses a regularization function on vectors, it computes a column $\mathbf{v}$ of $\mathbf{V}$ for every column $\mathbf{u}$ of $\mathbf{U}$:

$$\min_{\mathbf{v} \in \mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda \sum_{g \in \mathcal{T}} \eta^g \|\mathbf{v}_g\|_2, \tag{40}$$

or

$$\min_{\mathbf{v} \in \mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda \sum_{g \in \mathcal{T}} \eta^g \|\mathbf{v}_g\|_\infty, \tag{41}$$

or

$$\min_{\mathbf{v} \in \mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda \sum_{g \in \mathcal{T}} \delta^g(\mathbf{v}), \tag{42}$$

where $\delta^g(\mathbf{v}) = 0$ if $\mathbf{v}_g = 0$ and 1 otherwise (see appendix of [15]).

When the multi-task tree-structured regularization function is used, it solves

$$\min_{\mathbf{V} \in \mathbb{R}^{p \times n}} \frac{1}{2}\|\mathbf{U}-\mathbf{V}\|_F^2 + \lambda \sum_{i=1}^{n} \sum_{g \in \mathcal{T}} \eta^g \|\mathbf{v}_g^i\|_\infty + \lambda_2 \sum_{g \in \mathcal{T}} \max_{j \in g} \|\mathbf{v}_g^j\|_\infty, \tag{43}$$

which is a formulation presented in [21].

This function can also be used for computing the proximal operators addressed by mexProximalFlat (it will just not take into account the tree structure). The way the tree is incoded is presented below, (and examples are given in the file test_ProximalTree.m, with more usage details:

```
%
% Usage:  [V [val_regularizer]]=mexProximalTree(U,tree,param);
%
% Name: mexProximalTree
%
```

```
% Description: mexProximalTree computes a proximal operator. Depending
%         on the value of param.regul, it computes
%
%         Given an input matrix U=[u^1,\ldots,u^n], and a tree-structured set of groups T,
%         it returns a matrix V=[v^1,\ldots,v^n]:
%
%         when the regularization function is for vectors,
%         for every column u of U, it compute a column v of V solving
%         if param.regul='tree-l0'
%             argmin 0.5||u-v||_2^2 + lambda \sum_{g \in T} \delta^g(v)
%         if param.regul='tree-l2'
%           for all i, v^i =
%             argmin 0.5||u-v||_2^2 + lambda\sum_{g \in T} \eta_g||v_g||_2
%         if param.regul='tree-linf'
%           for all i, v^i =
%             argmin 0.5||u-v||_2^2 + lambda\sum_{g \in T} \eta_g||v_g||_inf
%
%         when the regularization function is for matrices:
%         if param.regul='multi-task-tree'
%            V=argmin 0.5||U-V||_F^2 + lambda \sum_{i=1}^n\sum_{g \in T} \eta_g||v^i_g||_inf +
    ...
%                                                 lambda_2 \sum_{g \in T} \eta_g max_{j in g}||
    V_j||_{inf}
%
%         it can also be used with any non-tree-structured regularization addressed by
    mexProximalFlat
%
%         for all these regularizations, it is possible to enforce non-negativity constraints
%         with the option param.pos, and to prevent the last row of U to be regularized, with
%         the option param.intercept
%
% Inputs: U:  double m x n matrix   (input signals)
%               m is the signal size
%         tree: struct
%               with four fields, eta_g, groups, own_variables and N_own_variables.
%
%               The tree structure requires a particular organization of groups and variables
%                  * Let us denote by N = |T|, the number of groups.
%                    the groups should be ordered T={g1,g2,\ldots,gN} such that if gi is
    included
%                    in gj, then j <= i. g1 should be the group at the root of the tree
%                    and contains every variable.
%                  * Every group is a set of  contiguous indices for instance
%                    gi={3,4,5} or gi={4,5,6,7} or gi={4}, but not {3,5};
%                  * We define root(gi) as the indices of the variables that are in gi,
%                    but not in its descendants. For instance for
%                    T={ g1={1,2,3,4},g2={2,3},g3={4} }, then, root(g1)={1},
%                    root(g2)={2,3}, root(g3)={4},
%                    We assume that for all i, root(gi) is a set of contigous variables
%                  * We assume that the smallest of root(gi) is also the smallest index of gi.
%
%                  For instance,
%                    T={ g1={1,2,3,4},g2={2,3},g3={4} }, is a valid set of groups.
%                    but we can not have
%                    T={ g1={1,2,3,4},g2={1,2},g3={3} }, since root(g1)={4} and 4 is not the
%                    smallest element in g1.
%
%               We do not lose generality with these assumptions since they can be fullfilled
    for any
%               tree-structured set of groups after a permutation of variables and a correct
    ordering of the
%               groups.
%               see more examples in test_ProximalTree.m of valid tree-structured sets of
    groups.
%
%               The first fields sets the weights for every group
%                  tree.eta_g            double N vector
%
%               The next field sets inclusion relations between groups
%               (but not between groups and variables):
%                  tree.groups           sparse (double or boolean) N x N matrix
%                  the (i,j) entry is non-zero if and only if i is different than j and
%                  gi is included in gj.
%                  the first column corresponds to the group at the root of the tree.
%
%               The next field define the smallest index of each group gi,
%               which is also the smallest index of root(gi)
%               tree.own_variables    int32 N vector
%
%               The next field define for each group gi, the size of root(gi)
%               tree.N_own_variables  int32 N vector
%
%               examples are given in test_ProximalTree.m
%
%         param: struct
%               param.lambda  (regularization parameter)
%               param.regul (choice of regularization, see above)
%               param.lambda2  (optional, regularization parameter)
%               param.lambda3  (optional, regularization parameter)
%               param.verbose (optional, verbosity level, false by default)
%               param.intercept (optional, last row of U is not regularized,
%                 false by default)
%               param.pos (optional, adds positivity constraints on the
%                 coefficients, false by default)
%               param.transpose (optional, transpose the matrix in the regularization function)
%               param.size_group (optional, for regularization functions assuming a group
%                 structure). It is a scalar. When param.groups is not specified, it assumes
%                 that the groups are the sets of consecutive elements of size param.size_group
%               param.numThreads (optional, number of threads for exploiting
%                 multi-core / multi-cpus. By default, it takes the value -1,
%                 which automatically selects all the available CPUs/cores).
%
% Output: V: double m x n matrix (output coefficients)
%         val_regularizer: double 1 x n vector (value of the regularization
%         term at the optimum).
%
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
U=randn(10,1000);

param.lambda=0.1; % regularization parameter
param.num_threads=-1; % all cores (-1 by default)
param.verbose=true;   % verbosity, false by default
param.pos=false;       % can be used with all the other regularizations
param.intercept=false; % can be used with all the other regularizations

fprintf('First tree example\n');
% Example 1 of tree structure
% tree structured groups:
% g1= {0 1 2 3 4 5 6 7 8 9}
% g2= {2 3 4}
% g3= {5 6 7 8 9}
tree.own_variables=int32([0 2 5]);   % pointer to the first variable of each group
tree.N_own_variables=int32([2 3 5]); % number of "root" variables in each group
                                    % (variables that are in a group, but not in its descendants).
                                    % for instance root(g1)={0,1}, root(g2)={2 3 4}, root(g3)={5 6 7
                                       8 9}
tree.eta_g=[1 1 1];                 % weights for each group, they should be non-zero to use fenchel
    duality
tree.groups=sparse([0 0 0; ...
                    1 0 0; ...
                    1 0 0]);    % first group should always be the root of the tree
                                  % non-zero entriees mean inclusion relation ship, here g2 is a
                                      children of g1,
                                  % g3 is a children of g1

fprintf('\ntest prox tree-l0\n');
param.regul='tree-l0';
alpha=mexProximalTree(U,tree,param);

fprintf('\ntest prox tree-l2\n');
param.regul='tree-l2';
alpha=mexProximalTree(U,tree,param);

fprintf('\ntest prox tree-linf\n');
param.regul='tree-linf';
alpha=mexProximalTree(U,tree,param);

fprintf('Second tree example\n');
% Example 2 of tree structure
% tree structured groups:
% g1= {0 1 2 3 4 5 6 7 8 9}    root(g1) = { };
% g2= {0 1 2 3 4 5}            root(g2) = {0 1 2};
% g3= {3 4}                    root(g3) = {3 4};
% g4= {5}                      root(g4) = {5};
% g5= {6 7 8 9}                root(g5) = { };
% g6= {6 7}                    root(g6) = {6 7};
% g7= {8 9}                    root(g7) = {8};
% g8 = {9}                     root(g8) = {9};
tree.own_variables=  int32([0 0 3 5 6 6 8 9]);   % pointer to the first variable of each group
tree.N_own_variables=int32([0 3 2 1 0 2 1 1]); % number of "root" variables in each group
tree.eta_g=[1 1 1 2 2 2 2.5 2.5];
tree.groups=sparse([0 0 0 0 0 0 0 0; ...
                    1 0 0 0 0 0 0 0; ...
                    0 1 0 0 0 0 0 0; ...
                    0 1 0 0 0 0 0 0; ...
                    1 0 0 0 0 0 0 0; ...
                    0 0 0 0 1 0 0 0; ...
                    0 0 0 0 1 0 0 0; ...
                    0 0 0 0 0 0 1 0]);  % first group should always be the root of the tree

fprintf('\ntest prox tree-l0\n');
param.regul='tree-l0';
alpha=mexProximalTree(U,tree,param);

fprintf('\ntest prox tree-l2\n');
param.regul='tree-l2';
alpha=mexProximalTree(U,tree,param);

fprintf('\ntest prox tree-linf\n');
param.regul='tree-linf';
alpha=mexProximalTree(U,tree,param);

% mexProximalTree also works with non-tree-structured regularization functions
fprintf('\nprox l1, intercept, positivity constraint\n');
param.regul='l1';
param.pos=true;       % can be used with all the other regularizations
param.intercept=true; % can be used with all the other regularizations
alpha=mexProximalTree([U; ones(1,size(U,2))],tree,param);

% Example of multi-task tree
fprintf('\nprox multi-task tree\n');
param.pos=false;
param.intercept=false;
param.lambda2=param.lambda;
param.regul='multi-task-tree';  % with linf
alpha=mexProximalTree(U,tree,param);


tree.own_variables=int32([0 1 2 3 4 5 6]);   % pointer to the first variable of each group
tree.N_own_variables=int32([1 1 1 1 1 1]); % number of "root" variables in each group
                              % (variables that are in a group, but not in its descendants).
                              % for instance root(g1)={0,1}, root(g2)={2 3 4}, root(g3)={5 6 7
                                 8 9}
tree.eta_g=[1 1 1 1 1 1];           % weights for each group, they should be non-zero to use
    fenchel duality
tree.groups=sparse([0 0 0; ...
                    1 0 0; ...
                    1 0 0]);    % first group should always be the root of the tree
                                % non-zero entriees mean inclusion relation ship, here g2 is a
                                   children of g1,
                                % g3 is a children of g1
```

5.4 Function mexProximalGraph

This function computes the proximal operators associated to structured sparse regularization, for input signals $\mathbf{U} = [\mathbf{u}^1, \ldots, \mathbf{u}^n]$ in $\mathbb{R}^{p \times n}$, and a set of groups [21], it returns a matrix $\mathbf{V} = [\mathbf{v}^1, \ldots, \mathbf{v}^n]$ in $\mathbb{R}^{p \times n}$. When one uses a regularization function on vectors, it computes a column $\mathbf{v}$ of $\mathbf{V}$ for every column $\mathbf{u}$ of $\mathbf{U}$:

$$\min_{\mathbf{v} \in \mathbb{R}^p} \frac{1}{2}\|\mathbf{u}-\mathbf{v}\|_2^2 + \lambda \sum_{g \in \mathcal{G}} \eta^g \|\mathbf{v}_g\|_\infty, \tag{44}$$

or with a regularization function on matrices, it computes $\mathbf{V}$ solving

$$\min_{\mathbf{V} \in \mathbb{R}^{p \times n}} \frac{1}{2}\|\mathbf{U}-\mathbf{V}\|_F^2 + \lambda \sum_{i=1}^n \sum_{g \in \mathcal{G}} \eta^g \|\mathbf{v}_g^i\|_\infty + \lambda_2 \sum_{g \in \mathcal{G}} \max_{j \in g} \|\mathbf{v}_g^j\|_\infty, \tag{45}$$

This function can also be used for computing the proximal operators addressed by mexProximalFlat. The way the graph is incoded is presented below (and also in the example file test_ProximalGraph.m, with more usage details:

```
%
% Usage:   [V [val_regularizer]]=mexProximalGraph(U,graph,param);
%
% Name: mexProximalGraph
%
% Description: mexProximalGraph computes a proximal operator. Depending
%         on the value of param.regul, it computes
%
%         Given an input matrix U=[u^1,\ldots,u^n], and a set of groups G,
%         it computes a matrix V=[v^1,\ldots,v^n] such that
%
%         if param.regul='graph'
%         for every column u of U, it computes a column v of V solving
%             argmin 0.5||u-v||_2^2 + lambda\sum_{g \in G} \eta_g||v_g||_inf
%
%         if param.regul='graph+ridge'
%         for every column u of U, it computes a column v of V solving
%             argmin 0.5||u-v||_2^2 + lambda\sum_{g \in G} \eta_g||v_g||_inf + lambda_2||v||_2
    ^2
%
%
%         if param.regul='multi-task-graph'
%            V=argmin 0.5||U-V||_F^2 + lambda \sum_{i=1}^n\sum_{g \in G} \eta_g||v^i_g||_inf +
    ...
%                                              lambda_2 \sum_{g \in G} \eta_g max_{j in g}||
    V_j||_{inf}
%
%         it can also be used with any regularization addressed by mexProximalFlat
%
%         for all these regularizations, it is possible to enforce non-negativity constraints
%         with the option param.pos, and to prevent the last row of U to be regularized, with
%         the option param.intercept
%
% Inputs: U:  double p x n matrix   (input signals)
%               m is the signal size
%         graph: struct
%               with three fields, eta_g, groups, and groups_var
%
```

```
%               The first fields sets the weights for every group
%                  graph.eta_g            double N vector
%
%               The next field sets inclusion relations between groups
%               (but not between groups and variables):
%                  graph.groups           sparse (double or boolean) N x N matrix
%                  the (i,j) entry is non-zero if and only if i is different than j and
%                  gi is included in gj.
%
%               The next field sets inclusion relations between groups and variables
%                  graph.groups_var       sparse (double or boolean) p x N matrix
%                  the (i,j) entry is non-zero if and only if the variable i is included
%                  in gj, but not in any children of gj.
%
%               examples are given in test_ProximalGraph.m
%
%         param: struct
%               param.lambda  (regularization parameter)
%               param.regul (choice of regularization, see above)
%               param.lambda2  (optional, regularization parameter)
%               param.lambda3  (optional, regularization parameter)
%               param.verbose (optional, verbosity level, false by default)
%               param.intercept (optional, last row of U is not regularized,
%                 false by default)
%               param.pos (optional, adds positivity constraints on the
%                 coefficients, false by default)
%               param.numThreads (optional, number of threads for exploiting
%                 multi-core / multi-cpus. By default, it takes the value -1,
%                 which automatically selects all the available CPUs/cores).
%
% Output: V: double p x n matrix (output coefficients)
%         val_regularizer: double 1 x n vector (value of the regularization
%         term at the optimum).
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
U=randn(10,1000);

param.lambda=0.1; % regularization parameter
param.num_threads=-1; % all cores (-1 by default)
param.verbose=true;   % verbosity, false by default
param.pos=false;       % can be used with all the other regularizations
param.intercept=false; % can be used with all the other regularizations

fprintf('First graph example\n');
% Example 1 of graph structure
% groups:
% g1= {0 1 2 3}
% g2= {3 4 5 6}
% g3= {6 7 8 9}
graph.eta_g=[1 1 1];
graph.groups=sparse(zeros(3));
graph.groups_var=sparse([1 0 0;
                         1 0 0;
                         1 0 0;
                         1 1 0;
                         0 1 0;
                         0 1 0;
                         0 1 1;
                         0 0 1;
                         0 0 1;
                         0 0 1]);

fprintf('\ntest prox graph\n');
param.regul='graph';
alpha=mexProximalGraph(U,graph,param);

% Example 2 of graph structure
% groups:
% g1= {0 1 2 3}
% g2= {3 4 5 6}
% g3= {6 7 8 9}
% g4= {0 1 2 3 4 5}
% g5= {6 7 8}
graph.eta_g=[1 1 1 1 1];
graph.groups=sparse([0 0 0 1 0;
                     0 0 0 0 0;
                     0 0 0 0 0;
                     0 0 0 0 0;
                     0 0 1 0 0]);   % g5 is included in g3, and g2 is included in g4
graph.groups_var=sparse([1 0 0 0 0;
                         1 0 0 0 0;
                         1 0 0 0 0 ;
                         1 1 0 0 0;
                         0 1 0 1 0;
                         0 1 0 1 0;
                         0 1 0 0 1;
                         0 0 0 0 1;
                         0 0 0 0 1;
                         0 0 1 0 0]); % represents direct inclusion relations
                                      % between groups (columns) and variables (rows)

fprintf('\ntest prox graph\n');
param.regul='graph';
alpha=mexProximalGraph(U,graph,param);

fprintf('\ntest prox multi-task-graph\n');
param.regul='multi-task-graph';
param.lambda2=0.1;
alpha=mexProximalGraph(U,graph,param);

fprintf('\ntest no regularization\n');
param.regul='none';
alpha=mexProximalGraph(U,graph,param);
```

5.5 Function mexProximalPathCoding

This function computes the proximal operators associated to the path coding penalties of [23].

```
%
% Usage:   [V [val_regularizer]]=mexProximalPathCoding(U,DAG,param);
%
% Name: mexProximalPathCoding
%
% Description: mexProximalPathCoding computes a proximal operator for
%         the path coding penalties of http://arxiv.org/abs/1204.4539
%
%         Given an input matrix U=[u^1,\ldots,u^n],
%
%
% Inputs: U:  double p x n matrix   (input signals)
%               m is the signal size
%         DAG:  struct
%               with three fields, weights, start_weights, stop_weights
%         for a graph with |V| nodes and |E| arcs,
%         DAG.weights: sparse double |V| x |V| matrix. Adjacency
%               matrix. The non-zero entries represent costs on arcs
%               linking two nodes.
%         DAG.start_weights: dense double |V| vector. Represent the costs
%               of starting a path from a specific node.
%         DAG.stop_weights: dense double |V| vector. Represent the costs
%               of ending a path at a specific node.
%
%         if param.regul='graph-path-l0', non-convex penalty
%         if param.regul='graph-path-conv', convex penalty
%
%         param: struct
%               param.lambda  (regularization parameter)
%               param.regul (choice of regularization, see above)
%               param.verbose (optional, verbosity level, false by default)
%               param.intercept (optional, last row of U is not regularized,
%                 false by default)
%               param.pos (optional, adds positivity constraints on the
%                 coefficients, false by default)
%               param.precision (optional, by default a very large integer.
%                 It returns approximate proximal operator by choosing a small integer,
%                 for example, 100 or 1000.
%               param.numThreads (optional, number of threads for exploiting
%                 multi-core / multi-cpus. By default, it takes the value -1,
%                 which automatically selects all the available CPUs/cores).
%
% Output: V: double p x n matrix (output coefficients)
%         val_regularizer: double 1 x n vector (value of the regularization
%         term at the optimum).
%
% Author: Julien Mairal, 2012
```

The following piece of code illustrates how to use this function.

```
clear all;
rand('seed',0);
randn('seed',0);
```

```
fprintf('test mexProximalPathCoding\n');
p=100;
% generate a DAG
G=sprand(p,p,0.02);
G=mexRemoveCyclesGraph(G);
fprintf('\n');

% generate a data matrix
U=randn(p,10);
U=U-mean(U(:));
U=mexNormalize(U);

% input graph
graph.weights=G;
graph.stop_weights=zeros(1,p);
graph.start_weights=10*ones(1,p);

% FISTA parameters
param.num_threads=-1; % all cores (-1 by default)
param.verbose=true;   % verbosity, false by default
param.lambda=0.05; % regularization parameter

fprintf('Proximal convex path penalty\n');
param.regul='graph-path-conv';
tic
[V1 optim]=mexProximalPathCoding(U,graph,param);
t=toc;
num=mexCountConnexComponents(graph.weights,V1(:,1));
fprintf('Num of connected components: %d\n',num);

fprintf('Proximal non-convex path penalty\n');
param.regul='graph-path-l0';
param.lambda=0.005;
tic
[V2 optim]=mexProximalPathCoding(U,graph,param);
t=toc;
num=mexCountConnexComponents(graph.weights,V2(:,1));
fprintf('Num of connected components: %d\n',num);

graph.start_weights=1*ones(1,p);
param.lambda=0.05;
fprintf('Proximal convex path penalty\n');
param.regul='graph-path-conv';
tic
[V1 optim]=mexProximalPathCoding(U,graph,param);
t=toc;
num=mexCountConnexComponents(graph.weights,V1(:,1));
fprintf('Num of connected components: %d\n',num);

fprintf('Proximal non-convex path penalty\n');
param.regul='graph-path-l0';
param.lambda=0.005;
tic
[V2 optim]=mexProximalPathCoding(U,graph,param);
```

```
t=toc;
num=mexCountConnexComponents(graph.weights,V2(:,1));
fprintf('Num of connected components: %d\n',num);
```

This function is associated to a function to evaluate the penalties:

5.6 Function mexEvalPathCoding

```
%
% Usage:   [val [paths]]=mexEvalPathCoding(U,DAG,param);
%
% Name: mexEvalPathCoding
%
% Description: mexEvalPathCoding evaluate the path coding penalies
%         of http://arxiv.org/abs/1204.4539 and provides a path
%         decomposition of a vector W.
%
%         Given an input matrix U=[u^1,\ldots,u^n],
%
%
% Inputs: U:  double p x n matrix   (input signals)
%               m is the signal size
%         DAG:  struct
%               with three fields, weights, start_weights, stop_weights
%         for a graph with |V| nodes and |E| arcs,
%         DAG.weights: sparse double |V| x |V| matrix. Adjacency
%               matrix. The non-zero entries represent costs on arcs
%               linking two nodes.
%         DAG.start_weights: dense double |V| vector. Represent the costs
%               of starting a path from a specific node.
%         DAG.stop_weights: dense double |V| vector. Represent the costs
%               of ending a path at a specific node.
%
%         if param.regul='graph-path-l0', non-convex penalty
%         if param.regul='graph-path-conv', convex penalty
%
%         param: struct
%               param.regul (choice of regularization, see above)
%               param.verbose (optional, verbosity level, false by default)
%               param.precision (optional, by default a very large integer.
%                 It returns approximate proximal operator by choosing a small integer,
%                 for example, 100 or 1000.
%               param.numThreads (optional, number of threads for exploiting
%                 multi-core / multi-cpus. By default, it takes the value -1,
%                 which automatically selects all the available CPUs/cores).
%
% Output: V: double 1 x n vector (values of the objective function)
%         paths: optional, double sparse p x k matrix. selected paths for the
%                first column of U
%
% Author: Julien Mairal, 2012
```

The following piece of code illustrates how to use this function.

```
clear all;
rand('seed',0);
```

```
randn('seed',0);

p=100;
G=sprand(p,p,0.05);
G=mexRemoveCyclesGraph(G);
fprintf('\n');

% input graph
graph.weights=G;
graph.stop_weights=zeros(1,p);
graph.start_weights=10*ones(1,p);

param.regul='graph-path-l0';
U=randn(p,10);
U=U-mean(U(:));
U=mexNormalize(U);
param.lambda=0.005;
[V2 optim]=mexProximalPathCoding(U,graph,param);
[vals paths]=mexEvalPathCoding(U,graph,param);
```

After having presented the regularization terms which our software can handle, we present the various formulations that we address

5.7 Problems Addressed

We present here regression or classification formulations and their multi-task variants.

5.7.1 Regression Problems with the Square Loss

Given a training set $\{\mathbf{x}^i, y_i\}_{i=1}^n$, with $\mathbf{x}^i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$ for all i in $[1;n]$, we address

$$\min_{\mathbf{w}\in\mathbb{R}^p, b\in\mathbb{R}} \sum_{i=1}^n \frac{1}{2}(y_i - \mathbf{w}^\top \mathbf{x}^i - b)^2 + \lambda\psi(\mathbf{w}),$$

where b is an optional variable acting as an "intercept", which is not regularized, and ψ can be any of the regularization functions presented above. Let us consider the vector $\mathbf{y}$ in $\mathbb{R}^n$ that carries the entries y_i. The problem without the intercept takes the following form, which we have already encountered in the previous toolbox, but with different notations:

$$\min_{\mathbf{w}\in\mathbb{R}^p} \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\psi(\mathbf{w}),$$

where the $\mathbf{X} = [\mathbf{x}^i, \ldots, \mathbf{x}^n]^T$ (the $\mathbf{x}^i$'s are here the rows of $\mathbf{X}$).

5.7.2 Classification Problems with the Logistic Loss

The next formulation that our software can solve is the regularized logistic regression formulation. We are again given a training set $\{\mathbf{x}^i, y_i\}_{i=1}^n$, with $\mathbf{x}^i \in \mathbb{R}^p$, but the variables y_i are now in $\{-1, +1\}$ for all i in $[1;n]$. The optimization problem we address is

$$\min_{\mathbf{w}\in\mathbb{R}^p, b\in\mathbb{R}} \frac{1}{n}\sum_{i=1}^n \log(1 + e^{-y_i(\mathbf{w}^\top \mathbf{x}^i + b)} + \lambda\psi(\mathbf{w}),$$

with again ψ taken to be one of the regularization function presented above, and b is an optional intercept.

5.7.3 Multi-class Classification Problems with the Softmax Loss

We have also implemented a multi-class logistic classifier (or softmax). For a classification problem with r classes, we are given a training set $\{\mathbf{x}^i, y_i\}_{i=1}^n$, where the variables $\mathbf{x}^i$ are still vectors in $\mathbb{R}^p$, but the y_i's have integer values in $\{1, 2, \ldots, r\}$. The formulation we address is the following multi-class learning problem

$$\min_{\mathbf{W}\in\mathbb{R}^{p\times r},\mathbf{b}\in\mathbb{R}^r} \frac{1}{n}\sum_{i=1}^n \log\left(\sum_{j=1}^r e^{(\mathbf{w}^j-\mathbf{w}^{\mathbf{y}_i})^\top\mathbf{x}^i+\mathbf{b}_j-\mathbf{b}_{\mathbf{y}_i}}\right) + \lambda\sum_{j=1}^r \psi(\mathbf{w}^j), \qquad (46)$$

where $\mathbf{W} = [\mathbf{w}^1, \ldots, \mathbf{w}^r]$ and the optional vector $\mathbf{b}$ in $\mathbb{R}^r$ carries intercepts for each class.

5.7.4 Multi-task Regression Problems with the Square Loss

We are now considering a problem with r tasks, and a training set $\{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^n$, where the variables $\mathbf{x}^i$ are still vectors in $\mathbb{R}^p$, and $\mathbf{y}^i$ is a vector in $\mathbb{R}^r$. We are looking for r regression vectors $\mathbf{w}^j$, for $j \in [1; r]$, or equivalently for a matrix $\mathbf{W} = [\mathbf{w}^1, \ldots, \mathbf{w}^r]$ in $\mathbb{R}^{p\times r}$. The formulation we address is the following multi-task regression problem

$$\min_{\mathbf{W}\in\mathbb{R}^{p\times r},\mathbf{b}\in\mathbb{R}^r} \sum_{j=1}^r\sum_{i=1}^n \frac{1}{2}(\mathbf{y}_j^i - \mathbf{w}^\top\mathbf{x}^i - \mathbf{b}_j)^2 + \lambda\psi(\mathbf{W}),$$

where ψ is any of the regularization function on matrices we have presented in the previous section. Note that by introducing the appropriate variables $\mathbf{Y}$, the problem without intercept could be equivalently rewritten

$$\min_{\mathbf{W}\in\mathbb{R}^{p\times r}} \frac{1}{2}\|\mathbf{Y}-\mathbf{X}\mathbf{W}\|_{\text{F}}^2 + \lambda\psi(\mathbf{W}).$$

5.7.5 Multi-task Classification Problems with the Logistic Loss

The multi-task version of the logistic regression follows the same principle. We consider r tasks, and a training set $\{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^n$, with the $\mathbf{x}^i$'s in $\mathbb{R}^p$, and the $\mathbf{y}^i$'s are vectors in $\{-1, +1\}^r$. We look for a matrix $\mathbf{W} = [\mathbf{w}^1, \ldots, \mathbf{w}^r]$ in $\mathbb{R}^{p\times r}$. The formulation is the following multi-task regression problem

$$\min_{\mathbf{W}\in\mathbb{R}^{p\times r},\mathbf{b}\in\mathbb{R}^r} \sum_{j=1}^r \frac{1}{n}\sum_{i=1}^n \log\left(1 + e^{-\mathbf{y}_j^i(\mathbf{w}^\top\mathbf{x}^i+\mathbf{b}_j)}\right) + \lambda\psi(\mathbf{W}).$$

5.7.6 Multi-task and Multi-class Classification Problems with the Softmax Loss

The multi-task/multi-class version directly follows from the formulation of Eq. (46), but associates with each class a task, and as a consequence, regularizes the matrix $\mathbf{W}$ in a particular way:

$$\min_{\mathbf{W}\in\mathbb{R}^{p\times r},\mathbf{b}\in\mathbb{R}^r} \frac{1}{n}\sum_{i=1}^n \log\left(\sum_{j=1}^r e^{(\mathbf{w}^j-\mathbf{w}^{\mathbf{y}_i})^\top\mathbf{x}^i+\mathbf{b}_j-\mathbf{b}_{\mathbf{y}_i}}\right) + \lambda\psi(\mathbf{W}).$$

How duality gaps are computed for any of these formulations is presented in Appendix A. We now present the main functions for solving these problems

5.8 Function mexFistaFlat

Given a matrix $\mathbf{X} = [\mathbf{x}^1, \ldots, \mathbf{x}^p]^T$ in $\mathbb{R}^{m\times p}$, and a matrix $\mathbf{Y} = [\mathbf{y}^1, \ldots, \mathbf{y}^n]$, it solves the optimization problems presented in the previous section, with the same regularization functions as mexProximalFlat. see usage details below:

```
%
% Usage: [W [optim]]=mexFistaFlat(Y,X,W0,param);
%
% Name: mexFistaFlat
%
% Description: mexFistaFlat solves sparse regularized problems.
%         X is a design matrix of size m x p
%         X=[x^1,...,x^n]', where the x_i's are the rows of X
%         Y=[y^1,...,y^n] is a matrix of size m x n
%         It implements the algorithms FISTA, ISTA and subgradient descent.
%
%           - if param.loss='square' and param.regul is a regularization function for vectors,
%             the entries of Y are real-valued,  W = [w^1,...,w^n] is a matrix of size p x n
%             For all column y of Y, it computes a column w of W such that
%               w = argmin 0.5||y- X w||_2^2 + lambda psi(w)
%
%           - if param.loss='square' and param.regul is a regularization function for matrices
%             the entries of Y are real-valued,  W is a matrix of size p x n.
%             It computes the matrix W such that
%               W = argmin 0.5||Y- X W||_F^2 + lambda psi(W)
%
%           - param.loss='square-missing' : same as param.loss='square', but handles missing
   data
%             represented by NaN (not a number) in the matrix Y
%
%           - if param.loss='logistic' and param.regul is a regularization function for vectors
   ,
%             the entries of Y are either -1 or +1, W = [w^1,...,w^n] is a matrix of size p x n
%             For all column y of Y, it computes a column w of W such that
%               w = argmin (1/m)sum_{j=1}^m log(1+e^(-y_j x^j' w)) + lambda psi(w),
%             where x^j is the j-th row of X.
%
%           - if param.loss='logistic' and param.regul is a regularization function for
   matrices
%             the entries of Y are either -1 or +1, W is a matrix of size p x n
%               W = argmin sum_{i=1}^n(1/m)sum_{j=1}^m log(1+e^(-y^i_j x^j' w^i)) + lambda psi(
   W)
%
%           - if param.loss='multi-logistic' and param.regul is a regularization function for
   vectors,
%             the entries of Y are in {0,1,...,N} where N is the total number of classes
%             W = [W^1,...,W^n] is a matrix of size p x Nn, each submatrix W^i is of size p x N
%             for all submatrix WW of W, and column y of Y, it computes
%               WW = argmin (1/m)sum_{j=1}^m log(sum_{j=1}^r e^(x^j'(ww^j-ww^{y_j}))) + lambda
   sum_{j=1}^N psi(ww^j),
%             where ww^j is the j-th column of WW.
%
%           - if param.loss='multi-logistic' and param.regul is a regularization function for
   matrices,
%             the entries of Y are in {0,1,...,N} where N is the total number of classes
%             W is a matrix of size p x N, it computes
%               W = argmin (1/m)sum_{j=1}^m log(sum_{j=1}^r e^(x^j'(w^j-w^{y_j}))) + lambda psi
   (W)
%             where ww^j is the j-th column of WW.
%
%           - param.loss='cur' : useful to perform sparse CUR matrix decompositions,
%               W = argmin 0.5||Y-X*W*X||_F^2 + lambda psi(W)
%
%
%         The function psi are those used by mexProximalFlat (see documentation)
```

```
%
%         This function can also handle intercepts (last row of W is not regularized),
%         and/or non-negativity constraints on W, and sparse matrices for X
%
% Inputs: Y:  double dense m x n matrix
%         X:  double dense or sparse m x p matrix
%         W0:  double dense p x n matrix or p x Nn matrix (for multi-logistic loss)
%              initial guess
%         param: struct
%            param.loss (choice of loss, see above)
%            param.regul (choice of regularization, see function mexProximalFlat)
%            param.lambda (regularization parameter)
%            param.lambda2 (optional, regularization parameter, 0 by default)
%            param.lambda3 (optional, regularization parameter, 0 by default)
%            param.verbose (optional, verbosity level, false by default)
%            param.pos (optional, adds positivity constraints on the
%                coefficients, false by default)
%            param.transpose (optional, transpose the matrix in the regularization function)
%            param.size_group (optional, for regularization functions assuming a group
%                 structure)
%            param.groups (int32, optional, for regularization functions assuming a group
%                 structure, see mexProximalFlat)
%            param.numThreads (optional, number of threads for exploiting
%                multi-core / multi-cpus. By default, it takes the value -1,
%                which automatically selects all the available CPUs/cores).
%            param.max_it (optional, maximum number of iterations, 100 by default)
%            param.it0 (optional, frequency for computing duality gap, every 10 iterations by
   default)
%            param.tol (optional, tolerance for stopping criteration, which is a relative
   duality gap
%                if it is available, or a relative change of parameters).
%            param.gamma (optional, multiplier for increasing the parameter L in fista, 1.5 by
   default)
%            param.L0 (optional, initial parameter L in fista, 0.1 by default, should be small
   enough)
%            param.fixed_step (deactive the line search for L in fista and use param.L0 instead
   )
%            param.compute_gram (optional, pre-compute X^TX, false by default).
%            param.intercept (optional, do not regularize last row of W, false by default).
%            param.ista (optional, use ista instead of fista, false by default).
%            param.subgrad (optional, if not param.ista, use subradient descent instead of
   fista, false by default).
%            param.a, param.b (optional, if param.subgrad, the gradient step is a/(t+b)
%            also similar options as mexProximalFlat
%
%            the function also implements the ADMM algorithm via an option param.admm=true. It
   is not documented
%            and you need to look at the source code to use it.
%
% Output:  W:  double dense p x n matrix or p x Nn matrix (for multi-logistic loss)
%          optim: optional, double dense 4 x n matrix.
%              first row: values of the objective functions.
%              third row: values of the relative duality gap (if available)
%              fourth row: number of iterations
```

```
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
format compact;
randn('seed',0);
param.numThreads=-1; % all cores (-1 by default)
param.verbose=true;   % verbosity, false by default
param.lambda=0.05; % regularization parameter
param.it0=10;      % frequency for duality gap computations
param.max_it=200; % maximum number of iterations
param.L0=0.1;
param.tol=1e-3;
param.intercept=false;
param.pos=false;

X=randn(100,200);
X=X-repmat(mean(X),[size(X,1) 1]);
X=mexNormalize(X);
Y=randn(100,1);
Y=Y-repmat(mean(Y),[size(Y,1) 1]);
Y=mexNormalize(Y);
W0=zeros(size(X,2),size(Y,2));
% Regression experiments
% 100 regression problems with the same design matrix X.
fprintf('\nVarious regression experiments\n');
param.compute_gram=true;
fprintf('\nFISTA + Regression l1\n');
param.loss='square';
param.regul='l1';
% param.regul='group-lasso-l2';
% param.size_group=10;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
   mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

param.regul='l1';
fprintf('\nISTA + Regression l1\n');
param.ista=true;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
   mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nSubgradient Descent + Regression l1\n');
param.ista=false;
param.subgrad=true;
param.a=0.1;
param.b=1000; % arbitrary parameters
max_it=param.max_it;
it0=param.it0;
```

```
param.max_it=500;
param.it0=50;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
param.subgrad=false;
param.max_it=max_it;
param.it0=it0;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
   mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nFISTA + Regression l2\n');
param.regul='l2';
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
   mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nFISTA + Regression l2 + sparse feature matrix\n');
param.regul='l2';
tic
[W optim_info]=mexFistaFlat(Y,sparse(X),W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
   mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));


fprintf('\nFISTA + Regression Elastic-Net\n');
param.regul='elastic-net';
param.lambda2=0.1;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nFISTA + Group Lasso L2\n');
param.regul='group-lasso-l2';
param.size_group=2;  % all the groups are of size 2
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
   mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nFISTA + Group Lasso L2 with variable size of groups \n');
param.regul='group-lasso-l2';
param2=param;
param2.groups=int32(randi(5,1,size(X,2)));  % all the groups are of size 2
param2.lambda=10*param2.lambda;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param2);
t=toc;
```

```
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nFISTA + Trace Norm\n');
param.regul='trace-norm-vec';
param.size_group=5;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nFISTA + Regression Fused-Lasso\n');
param.regul='fused-lasso';
param.lambda2=0.1;
param.lambda3=0.1; %
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nFISTA + Regression no regularization\n');
param.regul='none';
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nFISTA + Regression l1 with intercept \n');
param.intercept=true;
param.regul='l1';
tic
[W optim_info]=mexFistaFlat(Y,[X ones(size(X,1),1)],[W0; zeros(1,size(W0,2))],param); % adds a
    column of ones to X for the intercept
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nFISTA + Regression l1 with intercept+ non-negative \n');
param.pos=true;
param.regul='l1';
tic
[W optim_info]=mexFistaFlat(Y,[X ones(size(X,1),1)],[W0; zeros(1,size(W0,2))],param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));
param.pos=false;
param.intercept=false;

fprintf('\nISTA + Regression l0\n');
param.regul='l0';
tic
```

```
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nOne classification experiment\n');
Y=2*double(randn(100,1) > 0)-1;
fprintf('\nFISTA + Logistic l1\n');
param.regul='l1';
param.loss='logistic';
param.lambda=0.01;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...
param.regul='l1';
param.loss='weighted-logistic';
param.lambda=0.01;
fprintf('\nFISTA + weighted Logistic l1 + sparse matrix\n');
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
% can be used of course with other regularization functions, intercept,...

param.loss='logistic';
fprintf('\nFISTA + Logistic l1 + sparse matrix\n');
tic
[W optim_info]=mexFistaFlat(Y,sparse(X),W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...

% Multi-Class classification
Y=double(ceil(5*rand(100,1000))-1);
param.loss='multi-logistic';
fprintf('\nFISTA + Multi-Class Logistic l1\n');
tic
nclasses=max(Y(:))+1;
W0=zeros(size(X,2),nclasses*size(Y,2));
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...

% Multi-Task regression
Y=randn(100,100);
Y=Y-repmat(mean(Y),[size(Y,1) 1]);
Y=mexNormalize(Y);
param.compute_gram=false;
W0=zeros(size(X,2),size(Y,2));
```

```
param.loss='square';
fprintf('\nFISTA + Regression l1l2 \n');
param.regul='l1l2';
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
toc
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nFISTA + Regression l1linf \n');
param.regul='l1linf';
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
toc
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nFISTA + Regression l1l2 + l1 \n');
param.regul='l1l2+l1';
param.lambda2=0.1;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
toc
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nFISTA + Regression l1linf + l1 \n');
param.regul='l1linf+l1';
param.lambda2=0.1;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
toc
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nFISTA + Regression l1linf + row + columns \n');
param.regul='l1linf-row-column';
param.lambda2=0.1;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
toc
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

% Multi-Task Classification
fprintf('\nFISTA + Logistic + l1l2 \n');
param.regul='l1l2';
param.loss='logistic';
Y=2*double(randn(100,100) > 0)-1;
tic
[W optim_info]=mexFistaFlat(Y,X,W0,param);
toc
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
```

```
% Multi-Class + Multi-Task Regularization

fprintf('\nFISTA + Multi-Class Logistic l1l2 \n');
Y=double(ceil(5*rand(100,1000))-1);
param.loss='multi-logistic';
param.regul='l1l2';
tic
nclasses=max(Y(:))+1;
W0=zeros(size(X,2),nclasses*size(Y,2));
[W optim_info]=mexFistaFlat(Y,X,W0,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
   mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...
```

5.9 Function mexFistaTree

Given a matrix $\mathbf{X} = [\mathbf{x}^1, \dots, \mathbf{x}^p]^T$ in $\mathbb{R}^{m \times p}$, and a matrix $\mathbf{Y} = [\mathbf{y}^1, \dots, \mathbf{y}^n]$, it solves the optimization problems presented in the previous section, with the same regularization functions as mexProximalTree. see usage details below:

```
%
% Usage: [W [optim]]=mexFistaTree(Y,X,W0,tree,param);
%
% Name: mexFistaTree
%
% Description: mexFistaTree solves sparse regularized problems.
%         X is a design matrix of size m x p
%         X=[x^1,...,x^n]', where the x_i's are the rows of X
%         Y=[y^1,...,y^n] is a matrix of size m x n
%         It implements the algorithms FISTA, ISTA and subgradient descent for solving
%
%           min_W  loss(W) + lambda psi(W)
%
%         The function psi are those used by mexProximalTree (see documentation)
%         for the loss functions, see the documentation of mexFistaFlat
%
%         This function can also handle intercepts (last row of W is not regularized),
%         and/or non-negativity constraints on W and sparse matrices X
%
% Inputs: Y:  double dense m x n matrix
%         X:  double dense or sparse m x p matrix
%         W0:  double dense p x n matrix or p x Nn matrix (for multi-logistic loss)
%              initial guess
%         tree: struct (see documentation of mexProximalTree)
%         param: struct
%            param.loss (choice of loss, see above)
%            param.regul (choice of regularization, see function mexProximalFlat)
%            param.lambda (regularization parameter)
%            param.lambda2 (optional, regularization parameter, 0 by default)
%            param.lambda3 (optional, regularization parameter, 0 by default)
%            param.verbose (optional, verbosity level, false by default)
%            param.pos (optional, adds positivity constraints on the
%                coefficients, false by default)
%            param.transpose (optional, transpose the matrix in the regularization function)
%           param.size_group (optional, for regularization functions assuming a group
%                structure)
%           param.numThreads (optional, number of threads for exploiting
%               multi-core / multi-cpus. By default, it takes the value -1,
%               which automatically selects all the available CPUs/cores).
%           param.max_it (optional, maximum number of iterations, 100 by default)
%           param.it0 (optional, frequency for computing duality gap, every 10 iterations by
    default)
%           param.tol (optional, tolerance for stopping criteration, which is a relative
    duality gap
%               if it is available, or a relative change of parameters).
%           param.gamma (optional, multiplier for increasing the parameter L in fista, 1.5 by
    default)
%           param.L0 (optional, initial parameter L in fista, 0.1 by default, should be small
    enough)
%           param.fixed_step (deactive the line search for L in fista and use param.L0 instead
    )
%           param.compute_gram (optional, pre-compute X^TX, false by default).
%           param.intercept (optional, do not regularize last row of W, false by default).
%           param.ista (optional, use ista instead of fista, false by default).
%           param.subgrad (optional, if not param.ista, use subradient descent instead of
    fista, false by default).
%           param.a, param.b (optional, if param.subgrad, the gradient step is a/(t+b)
%           also similar options as mexProximalTree
%
%           the function also implements the ADMM algorithm via an option param.admm=true. It
    is not documented
%           and you need to look at the source code to use it.
%
% Output:  W:  double dense p x n matrix or p x Nn matrix (for multi-logistic loss)
%          optim: optional, double dense 4 x n matrix.
%              first row: values of the objective functions.
%              third row: values of the relative duality gap (if available)
%              fourth row: number of iterations
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
format compact;
param.num_threads=-1; % all cores (-1 by default)
param.verbose=false;   % verbosity, false by default
param.lambda=0.001; % regularization parameter
param.it0=10;      % frequency for duality gap computations
param.max_it=200; % maximum number of iterations
param.L0=0.1;
param.tol=1e-5;
param.intercept=false;
param.pos=false;

% Example 2 of tree structure
% tree structured groups:
% g1= {0 1 2 3 4 5 6 7 8 9}    root(g1) = { };
% g2= {0 1 2 3 4 5}            root(g2) = {0 1 2};
% g3= {3 4}                    root(g3) = {3 4};
% g4= {5}                        root(g4) = {5};
% g5= {6 7 8 9}                  root(g5) = { };
% g6= {6 7}                      root(g6) = {6 7};
% g7= {8 9}                      root(g7) = {8};
% g8 = {9}                       root(g8) = {9};
tree.own_variables=  int32([0 0 3 5 6 6 8 9]);   % pointer to the first variable of each group
tree.N_own_variables=int32([0 3 2 1 0 2 1 1]); % number of "root" variables in each group
tree.eta_g=[1 1 1 2 2 2 2.5 2.5];
tree.groups=sparse([0 0 0 0 0 0 0 0; ...
                    1 0 0 0 0 0 0 0; ...
                    0 1 0 0 0 0 0 0; ...
                    0 1 0 0 0 0 0 0; ...
                    1 0 0 0 0 0 0 0; ...
                    0 0 0 0 1 0 0 0; ...
                    0 0 0 0 1 0 0 0; ...
                    0 0 0 0 0 0 1 0]);  % first group should always be the root of the tree

X=randn(100,10);
X=X-repmat(mean(X),[size(X,1) 1]);
X=mexNormalize(X);
Y=randn(100,100);
Y=Y-repmat(mean(Y),[size(Y,1) 1]);
Y=mexNormalize(Y);
W0=zeros(size(X,2),size(Y,2));
% Regression experiments
% 100 regression problems with the same design matrix X.
fprintf('\nVarious regression experiments\n');
param.compute_gram=true;
fprintf('\nFISTA + Regression tree-l2\n');
param.loss='square';
param.regul='tree-l2';
tic
[W optim_info]=mexFistaTree(Y,X,W0,tree,param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nFISTA + Regression tree-linf\n');
param.regul='tree-linf';
tic
[W optim_info]=mexFistaTree(Y,X,W0,tree,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

% works also with non tree-structured regularization. tree is ignored
fprintf('\nFISTA + Regression Fused-Lasso\n');
param.regul='fused-lasso';
param.lambda2=0.001;
param.lambda3=0.001; %
tic
[W optim_info]=mexFistaTree(Y,X,W0,tree,param);
t=toc;
```

```
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nISTA + Regression tree-l0\n');
param.regul='tree-l0';
tic
[W optim_info]=mexFistaTree(Y,X,W0,tree,param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nFISTA + Regression tree-l2 with intercept \n');
param.intercept=true;
param.regul='tree-l2';
tic
[W optim_info]=mexFistaTree(Y,[X ones(size(X,1),1)],[W0; zeros(1,size(W0,2))],tree,param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));
param.intercept=false;

% Classification
fprintf('\nOne classification experiment\n');
Y=2*double(randn(100,size(Y,2)) > 0)-1;
fprintf('\nFISTA + Logistic + tree-linf\n');
param.regul='tree-linf';
param.loss='logistic';
param.lambda=0.001;
tic
[W optim_info]=mexFistaTree(Y,X,W0,tree,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...

% Multi-Class classification
Y=double(ceil(5*rand(100,size(Y,2)))-1);
param.loss='multi-logistic';
param.regul='tree-l2';
fprintf('\nFISTA + Multi-Class Logistic + tree-l2 \n');
tic
nclasses=max(Y(:))+1;
W0=zeros(size(X,2),nclasses*size(Y,2));
[W optim_info]=mexFistaTree(Y,X,W0,tree,param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...

% Multi-Task regression
Y=randn(100,size(Y,2));
Y=Y-repmat(mean(Y),[size(Y,1) 1]);
Y=mexNormalize(Y);
param.compute_gram=false;
param.verbose=true;   % verbosity, false by default
W0=zeros(size(X,2),size(Y,2));
param.loss='square';
fprintf('\nFISTA + Regression multi-task-tree \n');
param.regul='multi-task-tree';
param.lambda2=0.001;
tic
[W optim_info]=mexFistaTree(Y,X,W0,tree,param);
toc
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

% Multi-Task Classification
fprintf('\nFISTA + Logistic + multi-task-tree \n');
param.regul='multi-task-tree';
param.lambda2=0.001;
param.loss='logistic';
Y=2*double(randn(100,size(Y,2)) > 0)-1;
tic
[W optim_info]=mexFistaTree(Y,X,W0,tree,param);
toc
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

% Multi-Class + Multi-Task Regularization
param.verbose=false;
fprintf('\nFISTA + Multi-Class Logistic +multi-task-tree \n');
Y=double(ceil(5*rand(100,size(Y,2)))-1);
param.loss='multi-logistic';
param.regul='multi-task-tree';
tic
nclasses=max(Y(:))+1;
W0=zeros(size(X,2),nclasses*size(Y,2));
[W optim_info]=mexFistaTree(Y,X,W0,tree,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...

fprintf('\nFISTA + Multi-Class Logistic +multi-task-tree + sparse matrix \n');
tic
nclasses=max(Y(:))+1;
W0=zeros(size(X,2),nclasses*size(Y,2));
[W optim_info]=mexFistaTree(Y,sparse(X),W0,tree,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
```

5.10 Function mexFistaGraph

Given a matrix $\mathbf{X} = [\mathbf{x}^1, \ldots, \mathbf{x}^p]^T$ in $\mathbb{R}^{m \times p}$, and a matrix $\mathbf{Y} = [\mathbf{y}^1, \ldots, \mathbf{y}^n]$, it solves the optimization problems presented in the previous section, with the same regularization functions as mexProximalGraph. see usage details below:

```
%
```

```
% Usage: [W [optim]]=mexFistaGraph(Y,X,W0,graph,param);
%
% Name: mexFistaGraph
%
% Description: mexFistaGraph solves sparse regularized problems.
%         X is a design matrix of size m x p
%         X=[x^1,...,x^n]', where the x_i's are the rows of X
%         Y=[y^1,...,y^n] is a matrix of size m x n
%         It implements the algorithms FISTA, ISTA and subgradient descent.
%
%         It implements the algorithms FISTA, ISTA and subgradient descent for solving
%
%           min_W  loss(W) + lambda psi(W)
%
%         The function psi are those used by mexProximalGraph (see documentation)
%         for the loss functions, see the documentation of mexFistaFlat
%
%         This function can also handle intercepts (last row of W is not regularized),
%         and/or non-negativity constraints on W.
%
% Inputs: Y:  double dense m x n matrix
%         X:  double dense or sparse m x p matrix
%         W0:  double dense p x n matrix or p x Nn matrix (for multi-logistic loss)
%              initial guess
%         graph: struct (see documentation of mexProximalGraph)
%         param: struct
%            param.loss (choice of loss, see above)
%            param.regul (choice of regularization, see function mexProximalFlat)
%            param.lambda (regularization parameter)
%            param.lambda2 (optional, regularization parameter, 0 by default)
%            param.lambda3 (optional, regularization parameter, 0 by default)
%            param.verbose (optional, verbosity level, false by default)
%            param.pos (optional, adds positivity constraints on the
%                coefficients, false by default)
%            param.numThreads (optional, number of threads for exploiting
%                multi-core / multi-cpus. By default, it takes the value -1,
%                which automatically selects all the available CPUs/cores).
%            param.max_it (optional, maximum number of iterations, 100 by default)
%            param.it0 (optional, frequency for computing duality gap, every 10 iterations by
  default)
%            param.tol (optional, tolerance for stopping criteration, which is a relative
  duality gap
%                if it is available, or a relative change of parameters).
%            param.gamma (optional, multiplier for increasing the parameter L in fista, 1.5 by
  default)
%            param.L0 (optional, initial parameter L in fista, 0.1 by default, should be small
  enough)
%            param.fixed_step (deactive the line search for L in fista and use param.L0 instead
  )
%            param.compute_gram (optional, pre-compute X^TX, false by default).
%            param.intercept (optional, do not regularize last row of W, false by default).
%            param.ista (optional, use ista instead of fista, false by default).
%            param.subgrad (optional, if not param.ista, use subradient descent instead of
  fista, false by default).
```

```
%            param.a, param.b (optional, if param.subgrad, the gradient step is a/(t+b)
%            also similar options as mexProximalTree
%
%            the function also implements the ADMM algorithm via an option param.admm=true. It
    is not documented
%            and you need to look at the source code to use it.
%
%
% Output:  W:  double dense p x n matrix or p x Nn matrix (for multi-logistic loss)
%          optim: optional, double dense 4 x n matrix.
%              first row: values of the objective functions.
%              third row: values of the relative duality gap (if available)
%              fourth row: number of iterations
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
clear all;
randn('seed',0);
format compact;
param.num_threads=-1; % all cores (-1 by default)
param.verbose=false;   % verbosity, false by default
param.lambda=0.1; % regularization parameter
param.it0=1;      % frequency for duality gap computations
param.max_it=100; % maximum number of iterations
param.L0=0.1;
param.tol=1e-5;
param.intercept=false;
param.pos=false;

graph.eta_g=[1 1 1 1 1];
graph.groups=sparse([0 0 0 1 0;
                     0 0 0 0 0;
                     0 0 0 0 0;
                     0 0 0 0 0;
                     0 0 1 0 0]);   % g5 is included in g3, and g2 is included in g4
graph.groups_var=sparse([1 0 0 0 0;
                         1 0 0 0 0;
                         1 0 0 0 0 ;
                         1 1 0 0 0;
                         0 1 0 1 0;
                         0 1 0 1 0;
                         0 1 0 0 1;
                         0 0 0 0 1;
                         0 0 0 0 1;
                         0 0 1 0 0]); % represents direct inclusion relations

X=randn(100,10);
param.verbose=true;
%X=eye(10);
X=X-repmat(mean(X),[size(X,1) 1]);
X=mexNormalize(X);
Y=randn(100,1);
Y=Y-repmat(mean(Y),[size(Y,1) 1]);
```

```
Y=mexNormalize(Y);
W0=zeros(size(X,2),size(Y,2));
% Regression experiments
% 100 regression problems with the same design matrix X.
fprintf('\nVarious regression experiments\n');
param.compute_gram=true;
fprintf('\nFISTA + Regression graph\n');
param.loss='square';
param.regul='graph';
tic
[W optim_info]=mexFistaGraph(Y,X,W0,graph,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

fprintf('\nADMM + Regression graph\n');
param.admm=true;
param.lin_admm=true;
param.c=1;
param.delta=1;
tic
[W optim_info]=mexFistaGraph(Y,X,W0,graph,param);
t=toc;
fprintf('mean loss: %f, stopping criterion: %f, time: %f, number of iterations: %f\n',mean(
    optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

param.admm=false;
param.max_it=5;
param.it0=1;
tic
[W optim_info]=mexFistaGraph(Y,X,W,graph,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

% works also with non graph-structured regularization. graph is ignored
fprintf('\nFISTA + Regression Fused-Lasso\n');
param.regul='fused-lasso';
param.lambda2=0.01;
param.lambda3=0.01; %
tic
[W optim_info]=mexFistaGraph(Y,X,W0,graph,param);
t=toc;
fprintf('mean loss: %f, time: %f, number of iterations: %f\n',mean(optim_info(1,:)),t,mean(
    optim_info(4,:)));

fprintf('\nFISTA + Regression graph with intercept \n');
param.intercept=true;
param.regul='graph';
tic
[W optim_info]=mexFistaGraph(Y,[X ones(size(X,1),1)],[W0; zeros(1,size(W0,2))],graph,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
```

```
param.intercept=false;

% Classification
fprintf('\nOne classification experiment\n');
Y=2*double(randn(100,size(Y,2)) > 0)-1;
fprintf('\nFISTA + Logistic + graph-linf\n');
param.regul='graph';
param.loss='logistic';
param.lambda=0.01;
tic
[W optim_info]=mexFistaGraph(Y,X,W0,graph,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...

% Multi-Class classification
Y=double(ceil(5*rand(100,size(Y,2)))-1);
param.loss='multi-logistic';
param.regul='graph';
fprintf('\nFISTA + Multi-Class Logistic + graph \n');
tic
nclasses=max(Y(:))+1;
W0=zeros(size(X,2),nclasses*size(Y,2));
[W optim_info]=mexFistaGraph(Y,X,W0,graph,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...

% Multi-Task regression
Y=randn(100,size(Y,2));
Y=Y-repmat(mean(Y),[size(Y,1) 1]);
Y=mexNormalize(Y);
param.compute_gram=false;
param.verbose=true;   % verbosity, false by default
W0=zeros(size(X,2),size(Y,2));
param.loss='square';
fprintf('\nFISTA + Regression multi-task-graph \n');
param.regul='multi-task-graph';
param.lambda2=0.01;
tic
[W optim_info]=mexFistaGraph(Y,X,W0,graph,param);
toc
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));

% Multi-Task Classification
fprintf('\nFISTA + Logistic + multi-task-graph \n');
param.regul='multi-task-graph';
param.lambda2=0.01;
param.loss='logistic';
Y=2*double(randn(100,size(Y,2)) > 0)-1;
tic
```

```
[W optim_info]=mexFistaGraph(Y,X,W0,graph,param);
toc
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% Multi-Class + Multi-Task Regularization

param.verbose=false;
fprintf('\nFISTA + Multi-Class Logistic +multi-task-graph \n');
Y=double(ceil(5*rand(100,size(Y,2)))-1);
param.loss='multi-logistic';
param.regul='multi-task-graph';
tic
nclasses=max(Y(:))+1;
W0=zeros(size(X,2),nclasses*size(Y,2));
[W optim_info]=mexFistaGraph(Y,X,W0,graph,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
    mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
% can be used of course with other regularization functions, intercept,...
```

5.11 Function mexFistaPathCoding

Similarly, the toolbox handles the penalties of [23] with the following function

```
%
% Usage: [W [optim]]=mexFistaPathCoding(Y,X,W0,DAG,param);
%
% Name: mexFistaPathCoding
%
% Description: mexFistaPathCoding solves sparse regularized problems for the
%         path coding penalties of http://arxiv.org/abs/1204.4539
%         X is a design matrix of size m x p
%         X=[x^1,...,x^n]', where the x_i's are the rows of X
%         Y=[y^1,...,y^n] is a matrix of size m x n
%         It implements the algorithms FISTA, ISTA and subgradient descent.
%
%         It implements the algorithms FISTA, ISTA and subgradient descent for solving
%
%           min_W  loss(W) + lambda psi(W)
%
%         The function psi are those used by mexProximalPathCoding (see documentation)
%         for the loss functions, see the documentation of mexFistaFlat
%
%         This function can also handle intercepts (last row of W is not regularized),
%         and/or non-negativity constraints on W.
%
% Inputs: Y:  double dense m x n matrix
%         X:  double dense or sparse m x p matrix
%         W0:  double dense p x n matrix or p x Nn matrix (for multi-logistic loss)
%              initial guess
%         DAG: struct (see documentation of mexProximalPathCoding)
%         param: struct
%            param.loss (choice of loss, see above)
%            param.regul (choice of regularization, see function mexProximalPathCoding)
%            param.lambda (regularization parameter)
%           param.lambda2 (optional, regularization parameter, 0 by default)
%           param.lambda3 (optional, regularization parameter, 0 by default)
%           param.verbose (optional, verbosity level, false by default)
%           param.pos (optional, adds positivity constraints on the
%               coefficients, false by default)
%           param.numThreads (optional, number of threads for exploiting
%               multi-core / multi-cpus. By default, it takes the value -1,
%               which automatically selects all the available CPUs/cores).
%           param.max_it (optional, maximum number of iterations, 100 by default)
%           param.it0 (optional, frequency for computing duality gap, every 10 iterations by
    default)
%           param.tol (optional, tolerance for stopping criteration, which is a relative
    duality gap
%               if it is available, or a relative change of parameters).
%           param.gamma (optional, multiplier for increasing the parameter L in fista, 1.5 by
    default)
%           param.L0 (optional, initial parameter L in fista, 0.1 by default, should be small
    enough)
%           param.fixed_step (deactive the line search for L in fista and use param.L0 instead
    )
%           param.compute_gram (optional, pre-compute X^TX, false by default).
%           param.intercept (optional, do not regularize last row of W, false by default).
%           param.ista (optional, use ista instead of fista, false by default).
%           param.subgrad (optional, if not param.ista, use subradient descent instead of
    fista, false by default).
%           param.a, param.b (optional, if param.subgrad, the gradient step is a/(t+b)
%           also similar options as mexProximalPathCoding
%
%
% Output:  W:  double dense p x n matrix or p x Nn matrix (for multi-logistic loss)
%          optim: optional, double dense 4 x n matrix.
%              first row: values of the objective functions.
%              third row: values of the relative duality gap (if available)
%              fourth row: number of iterations
%
% Author: Julien Mairal, 2012
```

The following piece of code illustrates how to use this function.

```
clear all;
rand('seed',0);
randn('seed',0);
fprintf('test mexFistaPathCoding\n');
p=100;
n=1000;
% generate a DAG
G=sprand(p,p,0.05);
G=mexRemoveCyclesGraph(G);
fprintf('\n');

% generate a data matrix
X=randn(n,p);
X=X-repmat(mean(X),[size(X,1) 1]);
X=mexNormalize(X);
Y=randn(n,2);
```

```
Y=Y-repmat(mean(Y),[size(Y,1) 1]);
Y=mexNormalize(Y);
W0=zeros(size(X,2),size(Y,2));

% input graph
graph.weights=G;
graph.stop_weights=zeros(1,p);
graph.start_weights=10*ones(1,p);

% FISTA parameters
param.num_threads=-1; % all cores (-1 by default)
param.verbose=true;   % verbosity, false by default
param.lambda=0.005; % regularization parameter
param.it0=1;      % frequency for duality gap computations
param.max_it=100; % maximum number of iterations
param.L0=0.01;
param.tol=1e-4;
param.precision=10000000;
param.pos=false;

fprintf('Square Loss + convex path penalty\n');
param.loss='square';
param.regul='graph-path-conv';
tic
[W1 optim_info]=mexFistaPathCoding(Y,X,W0,graph,param);
t=toc;
fprintf('mean loss: %f, mean relative duality_gap: %f, time: %f, number of iterations: %f\n',
   mean(optim_info(1,:)),mean(optim_info(3,:)),t,mean(optim_info(4,:)));
num=mexCountConnexComponents(graph.weights,W1(:,1));
fprintf('Num of connected components: %d\n',num);

fprintf('\n');
fprintf('Square Loss + non-convex path penalty\n');
param.loss='square';
param.regul='graph-path-l0';
param.lambda=0.0001; % regularization parameter
param.ista=true;
tic
[W2 optim_info]=mexFistaPathCoding(Y,X,W0,graph,param);
t=toc;
num=mexCountConnexComponents(graph.weights,W2(:,1));
fprintf('Num of connected components: %d\n',num);

fprintf('\n');
fprintf('Note that for non-convex penalties, continuation strategies sometimes perform better:\
   n');
tablambda=param.lambda*sqrt(sqrt(sqrt(2))).^(20:-1:0);
lambda_orig=param.lambda;
tic
W2=W0;
for ii = 1:length(tablambda)
   param.lambda=tablambda(ii);
   param.verbose=false;
   [W2]=mexFistaPathCoding(Y,X,W2,graph,param);
```

```
end
param.verbose=true;
param.lambda=lambda_orig;
[W2 optim_info]=mexFistaPathCoding(Y,X,W2,graph,param);
t=toc;
num=mexCountConnexComponents(graph.weights,W2(:,1));
param.ista=false;
fprintf('Num of connected components: %d\n',num);
```

5.12 Function mexIncrementalProx

This implements the incremental solver MISO [24].

```
%
% Usage:  [W [optim]]=mexIncrementalProx(y,X,W0,param);
%
% Name: mexIncrementalProx
%
% Description: mexIncremrentalProx implements the incremental algorithm MISO
% for composite optimization in a large scale setting.
%        X is a design matrix of size p x n
%        y is a vector of size n
% WARNING, X is transposed compared to the functions mexFista*, and y is a vector
%        param.lambda is a vector that contains nlambda different values of the
%        regularization parameter (it can be a scalar, in that case nlambda=1)
%        W0: is a dense matrix of size p x nlambda   It is in fact ineffective
%        in the current release
%        W: is the output, dense matrix of size p x nlambda
%
%         - if param.loss='square' and param.regul corresponds to a regularization
%           function (currently 'l1' or 'l2'), the following problem is solved
%           w = argmin (1/n)sum_{i=1}^n 0.5(y_i- x_i^T w)^2 + lambda psi(w)
%         - if param.loss='logistic' and param.regul corresponds to a regularization
%           function (currently 'l1' or 'l2'), the following problem is solved
%           w = argmin (1/n)sum_{i=1}^n log(1+ exp(-y_ix_i^T w)) + lambda psi(w)
%           Note that here, the y_i's should be -1 or +1
%
%         The current release does not handle intercepts
%
% Inputs: y: double dense vector of size n
%         X: dense or sparse matrix of size p x n
%         W0: dense matrix of size p x nlambda
%         param: struct
%           param.loss (choice of loss)
%           param.regul (choice of regularization function)
%           param.lambda : vector of size nlambda
%           param.epochs: number of passes over the data
%           param.minibatches: size of the mini-batches: recommended value is 1
%              if X is dense, and min(n,ceil(1/density)) if X is sparse
%           param.warm_restart : (path-following strategy, very efficient when
%              providing an array of lambdas, false by default)
%           param.normalized : (optional, can be set to true if the x_i's have
%              unit l2-norm, false by default)
%           param.strategy (optional, 3 by default)
%                              0: no heuristics, slow  (only for comparison purposes)
%                            1: adjust the constant L on 5% of the data
%                            2: adjust the constant L on 5% of the data + unstable heuristics (
    this strategy does not work)
%                            3: adjust the constant L on 5% of the data + stable heuristic (this
    is by far the best choice)
%             param.numThreads (optional, number of threads)
%             param.verbose (optional)
%             param.seed (optional, choice of the random seed)
%
% Output:  W:  double dense p x nlambda matrix
%          optim: optional, double dense 3 x nlambda matrix.
%              first row: values of the objective functions.
%              third row: computational time
%
% Author: Julien Mairal, 2013
```

The following piece of code illustrates how to use this function.

```
clear all;
n=40000;
p=1000;
density=0.01;

% generate random data
format compact;
randn('seed',0);
rand('seed',0);
X=sprandn(p,n,density);
mean_nrm=mean(sqrt(sum(X.^2)));
X=X/mean_nrm;

% generate some true model
z=double(sign(full(sprandn(p,1,0.05))));
y=X'*z;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXPERIMENT 1: Lasso
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('EXPERIMENT FOR LASSO\n');
nrm=sqrt(sum(y.^2));
y=y+0.01*nrm*randn(n,1);    % add noise to the model
nrm=sqrt(sum(y.^2));
y=y*(sqrt(n)/nrm);

% set optimization parameters
clear param;
param.regul='l1';         % many other regularization functions are available
param.loss='square';      % only square and log are available
param.numThreads=-1;    % uses all possible cores
param.normalized=false;  % if the columns of X have unit norm, set to true.
param.strategy=3;         % MISO with all heuristics
                          % 0: no heuristics, slow  (only for comparison purposes)
                          % 1: adjust the constant L on 5% of the data
                          % 2: adjust the constant L on 5% of the data + unstable heuristics (
                              this strategy does not work)
```

```
                           % 3: adjust the constant L on 5% of the data + stable heuristic (this
                               is by far the best choice)
param.verbose=true;
param.minibatches=min(n,ceil(1/density));  % size of the minibatches, requires to store twice
    the size of X

% set grid of lambda
max_lambda=max(abs(X*y))/n;
tablambda=max_lambda*(2^(1/8)).^(0:-1:-50);  % order from large to small
tabepochs=[1 2 3 5 10];  % in this script, we compare the results obtained when changing the
    number of passes on the data.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compare the solutions obtained with different epochs for one
% value of lambda;
fprintf('EXPERIMENT: SINGLE LAMBDA\n');
param.lambda=tablambda(20);
for ii=1:length(tabepochs)
   fprintf('EXP WITH %d PASS\n',tabepochs(ii));
   param.epochs=tabepochs(ii);   % one pass over the data
   Beta0=zeros(p,1);
   tic
   [Beta tmp]=mexIncrementalProx(y,X,Beta0,param);
   toc
   obj=tmp(1);
   fprintf('Objective functions: %f\n',obj);
   spar=sum(Beta ~= 0);
   fprintf('Sparsity: %d\n',spar);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Now we do the same experiments, but we provide the array of lambda to
% the function, and we compare two strategies, the second one implementing
% a warm restart with
param.lambda=tablambda;
%% The problem which will be solved is
%%   min_beta  1/(2n) ||y-X' beta||_2^2 + lambda ||beta||_1
% the problems for different lambdas are solve INDEPENDENTLY in parallel
fprintf('EXPERIMENT: ALL LAMBDAS WITHOUT WARM RESTART\n');
param.warm_restart=false;
param.verbose=false;
obj=[];
spar=[];
for ii=1:length(tabepochs)
   param.epochs=tabepochs(ii);   % one pass over the data
   fprintf('EXP WITH %d PASS OVER THE DATA\n',tabepochs(ii));
   nlambdas=length(param.lambda);
   Beta0=zeros(p,nlambdas);
   tic
   [Beta tmp]=mexIncrementalProx(y,X,Beta0,param);
   toc
   fprintf('Objective functions: \n');
   obj=[obj; tmp(1,:)];
   obj
   fprintf('Sparsity: \n');
   spar=[spar; sum(Beta ~= 0)];
   spar
end

% the problems are here solved sequentially with warm restart
% this seems to be the prefered choice.
fprintf('EXPERIMENT: SEQUENTIAL LAMBDAS WITH WARM RESTART\n');
fprintf('A SINGLE CORE IS USED\n');
param.warm_restart=true;
param.num_threads=1;
obj=[];
spar=[];
for ii=1:length(tabepochs)
   param.epochs=tabepochs(ii);   % one pass over the data
   fprintf('EXP WITH %d PASS\n',tabepochs(ii));
   nlambdas=length(param.lambda);
   Beta0=zeros(p,nlambdas);
   tic
   [Beta tmp]=mexIncrementalProx(y,X,Beta0,param);
   toc
   fprintf('Objective functions: \n');
   obj=[obj; tmp(1,:)];
   obj
   fprintf('Sparsity: \n');
   spar=[spar; sum(Beta ~= 0)];
   spar
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXPERIMENT 2: L2 logistic regression
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('EXPERIMENT FOR LOGISTIC REGRESSION + l2\n');
y=sign(y);
param.regul='l2';        % many other regularization functions are available
param.loss='logistic';     % only square and log are available
param.num_threads=-1;    % uses all possible cores
%param.strategy=3;         % MISO with all heuristics
fprintf('EXPERIMENT: ALL LAMBDAS WITHOUT WARM RESTART\n');
param.warm_restart=false;
param.verbose=false;
obj=[];
spar=[];
for ii=1:length(tabepochs)
   param.epochs=tabepochs(ii);   % one pass over the data
   fprintf('EXP WITH %d PASS\n',tabepochs(ii));
   nlambdas=length(param.lambda);
   Beta0=zeros(p,nlambdas);
   tic
   [Beta tmp]=mexIncrementalProx(y,X,Beta0,param);
   toc
   yR=repmat(y,[1 nlambdas]);
   fprintf('Objective functions: \n');
   obj=[obj;tmp(1,:)];
   obj
end

fprintf('EXPERIMENT: SEQUENTIAL LAMBDAS WITH WARM RESTART\n');
fprintf('A SINGLE CORE IS USED\n');
param.warm_restart=true;
param.num_threads=1;
obj=[];
spar=[];
for ii=1:length(tabepochs)
   param.epochs=tabepochs(ii);   % one pass over the data
   fprintf('EXP WITH %d PASS\n',tabepochs(ii));
   nlambdas=length(param.lambda);
   Beta0=zeros(p,nlambdas);
   tic
   [Beta tmp]=mexIncrementalProx(y,X,Beta0,param);
   toc
   fprintf('Objective functions: \n');
   obj=[obj; tmp(1,:)];
   obj
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXPERIMENT 3: L1 logistic regression
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y=sign(y);
fprintf('EXPERIMENT FOR LOGISTIC REGRESSION + l1\n');
param.regul='l1';        % many other regularization functions are available
param.loss='logistic';     % only square and log are available
param.num_threads=-1;    % uses all possible cores
%param.strategy=3;        % MISO with all heuristics
fprintf('EXPERIMENT: ALL LAMBDAS WITHOUT WARM RESTART\n');
param.warm_restart=false;
param.verbose=false;
param.lambda=tablambda;
obj=[];
spar=[];
for ii=1:length(tabepochs)
   param.epochs=tabepochs(ii);   % one pass over the data
   fprintf('EXP WITH %d PASS\n',tabepochs(ii));
   nlambdas=length(param.lambda);
   Beta0=zeros(p,nlambdas);
   tic
   [Beta tmp]=mexIncrementalProx(y,X,Beta0,param);
   toc
   yR=repmat(y,[1 nlambdas]);
   fprintf('Objective functions: \n');
   obj=[obj;tmp(1,:)];
   obj
end

fprintf('EXPERIMENT: SEQUENTIAL LAMBDAS WITH WARM RESTART\n');
fprintf('A SINGLE CORE IS USED\n');
```

```
param.warm_restart=true;
param.num_threads=1;
obj=[];
spar=[];
for ii=1:length(tabepochs)
   param.epochs=tabepochs(ii);   % one pass over the data
   fprintf('EXP WITH %d PASS\n',tabepochs(ii));
   nlambdas=length(param.lambda);
   Beta0=zeros(p,nlambdas);
   tic
   [Beta tmp]=mexIncrementalProx(y,X,Beta0,param);
   toc
   fprintf('Objective functions: \n');
   obj=[obj; tmp(1,:)];
   obj
   fprintf('Sparsity: \n');
   spar=[spar; sum(Beta ~= 0)];
   spar
end
```

5.13 Function mexStochasticProx

This implements the stochastic proximal gradient solver [25].

```
%
% Usage:  [W [W2]]=mexStochasticProx(y,X,W0,param);
%
% Name: mexStochasticProx
%
% Description: mexStochasticProx implements a proximal MM stochastic algorithm
% for composite optimization in a large scale setting.
%       X is a design matrix of size p x n
%       y is a vector of size n
% WARNING, X is transposed compared to the functions mexFista*, and y is a vector
%       param.lambda is a vector that contains nlambda different values of the
%       regularization parameter (it can be a scalar, in that case nlambda=1)
%       W0: is a dense matrix of size p x nlambda   It is in fact ineffective
%       in the current release
%       W: is the output, dense matrix of size p x nlambda
%
%        - if param.loss='square' and param.regul corresponds to a regularization
%          function (currently 'l1' or 'l2'), the following problem is solved
%          w = argmin (1/n)sum_{i=1}^n 0.5(y_i- x_i^T w)^2 + lambda psi(w)
%        - if param.loss='logistic' and param.regul corresponds to a regularization
%          function (currently 'l1' or 'l2'), the following problem is solved
%          w = argmin (1/n)sum_{i=1}^n log(1+ exp(-y_ix_i^T w)) + lambda psi(w)
%          Note that here, the y_i's should be -1 or +1
%
%        The current release does not handle intercepts
%
% Inputs: y: double dense vector of size n
%        X: dense or sparse matrix of size p x n
%        W0: dense matrix of size p x nlambda
%        param: struct
%          param.loss (choice of loss)
%           param.regul (choice of regularization function)
%           param.lambda : vector of size nlambda
%           param.iters : number of iterations (n corresponds to one pass over the data)
%           param.minibatches: size of the mini-batches: recommended value is 1
%           param.normalized : (optional, can be set to true if the x_i's have
%              unit l2-norm, false by default)
%           param.weighting_mode : (optional, 1 by default),
%                0:  w_t = (t_0+1)/(t+t_0)
%                1:  w_t = ((t_0+1)/(t+t_0))^(0.75)
%                2:  w_t = ((t_0+1)/(t+t_0))^(5)
%           param.averaging_mode: (optional, false by default)
%                0: no averaging
%                1: first averaging mode for W2
%                2: second averaging mode for W2
%                WARNING: averaging can be very slow for sparse solutions
%           param.determineEta (optional, automatically choose the parameters of the
%             learning weights w_t, true by default)
%           param.t0 (optional, set up t0 for weights w_t = ((1+t0)/(t+t0))^(alpha)
%           param.numThreads (optional, number of threads)
%           param.verbose (optional)
%           param.seed (optional, choice of the random seed)
%
% Output:  W:  double dense p x nlambda matrix (contains the solution without averaging)
%          W2:  double dense p x nlambda matrix (contains the solution with averaging)
%
% Author: Julien Mairal, 2013
```

The following piece of code illustrates how to use this function.

```
clear all;
n=10000;
p=10000;
density=0.01;

% generate random data
format compact;
randn('seed',0);
rand('seed',0);
X=sprandn(p,n,density);
mean_nrm=mean(sqrt(sum(X.^2)));
X=X/mean_nrm;

% generate some true model
z=double(sign(full(sprandn(p,1,0.05))));
y=X'*z;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXPERIMENT 1: Lasso
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('EXPERIMENT FOR LASSO\n');
nrm=sqrt(sum(y.^2));
y=y+0.01*nrm*randn(n,1);    % add noise to the model
nrm=sqrt(sum(y.^2));
y=y*(sqrt(n)/nrm);
```

```
clear param;
param.regul='l1';        % many other regularization functions are available
param.loss='square';     % only square and log are available
param.numThreads=1;    % uses all possible cores
param.normalized=false;  % if the columns of X have unit norm, set to true.
param.averaging_mode=0;  % no averaging, averaging was not really useful in experiments
param.weighting_mode=1;  % weights are in O(1/sqrt(n))
param.optimized_solver=true;
param.verbose=false;

% set grid of lambda
max_lambda=max(abs(X*y))/n;
tablambda=max_lambda*(2^(1/8)).^(0:-1:-50);  % order from large to small
param.lambda=tablambda;    % best to order from large to small
tabepochs=[1 2 3 5 10];  % in this script, we compare the results obtained when varying the
    number of passes over the data.

%% The problem which will be solved is
%%   min_beta  1/(2n) ||y-X' beta||_2^2 + lambda ||beta||_1
fprintf('EXPERIMENT: ALL LAMBDAS IN PARALLEL, no warm restart\n');
% we try different experiments when varying the number of epochs.
% the problems for different lambdas are solve INDEPENDENTLY in parallel
obj=[];
objav=[];
for ii=1:length(tabepochs)
   param.iters=tabepochs(ii)*n;   % one pass over the data
   fprintf('EXP WITH %d PASS\n',tabepochs(ii));
   nlambdas=length(param.lambda);
   Beta0=zeros(p,nlambdas);
   tic
   [Beta tmp]=mexStochasticProx(y,X,Beta0,param);
   toc
   yR=repmat(y,[1 nlambdas]);
   fprintf('Objective functions: \n');
   obj=[obj; 0.5*sum((yR-X'*Beta).^2)/n+param.lambda.*sum(abs(Beta))];
   obj
   if param.averaging_mode
      objav=[objav; 0.5*sum((yR-X'*tmp).^2)/n+param.lambda.*sum(abs(tmp))];
      objav
   end
   fprintf('Sparsity: \n');
   sum(Beta ~= 0)
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXPERIMENT 2: L2 logistic regression
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('EXPERIMENT FOR LOGISTIC REGRESSION + l2\n');
y=sign(y);
param.regul='l2';        % many other regularization functions are available
param.loss='logistic';     % only square and log are available
obj=[];
objav=[];
for ii=1:length(tabepochs)
```

```
   param.iters=tabepochs(ii)*n;   % one pass over the data
   fprintf('EXP WITH %d PASS\n',tabepochs(ii));
   nlambdas=length(param.lambda);
   Beta0=zeros(p,nlambdas);
   tic
   [Beta tmp]=mexStochasticProx(y,X,Beta0,param);
   toc
   yR=repmat(y,[1 nlambdas]);
   fprintf('Objective functions: \n');
   obj=[obj;sum(log(1.0+exp(-yR .* (X'*Beta))))/n+0.5*param.lambda.*sum(abs(Beta.^2))];
   obj
   if param.averaging_mode
      objav=[objav; sum(log(1.0+exp(-yR .* (X'*tmp))))/n+0.5*param.lambda.*sum(abs(tmp))];
      objav
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXPERIMENT 3: L1 logistic regression
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('EXPERIMENT FOR LOGISTIC REGRESSION + l1\n');
y=sign(y);
param.regul='l1';        % many other regularization functions are available
param.loss='logistic';     % only square and log are available
obj=[];
objav=[];
for ii=1:length(tabepochs)
   param.iters=tabepochs(ii)*n;   % one pass over the data
   fprintf('EXP WITH %d PASS\n',tabepochs(ii));
   nlambdas=length(param.lambda);
   Beta0=zeros(p,nlambdas);
   tic
   [Beta tmp]=mexStochasticProx(y,X,Beta0,param);
   toc
   yR=repmat(y,[1 nlambdas]);
   fprintf('Objective functions: \n');
   obj=[obj; sum(log(1.0+exp(-yR .* (X'*Beta))))/n+param.lambda.*sum(abs(Beta))];
   obj
   if param.averaging_mode
      objav=[objav; sum(log(1.0+exp(-yR .* (X'*tmp))))/n+param.lambda.*sum(abs(tmp))];
      objav
   end
   fprintf('Sparsity: \n');
   sum(Beta ~= 0)
end
```

6 Miscellaneous Functions

6.1 Function mexConjGrad

Implementation of a conjugate gradient for solving a linear system $\mathbf{Ax} = \mathbf{b}$ when $\mathbf{A}$ is positive definite. In some cases, it is faster than the Matlab function `pcg`, especially when the library uses the Intel Math Kernel Library.

```
%
% Usage:   x =mexConjGrad(A,b,x0,tol,itermax)
%
% Name: mexConjGrad
%
% Description: Conjugate gradient algorithm, sometimes faster than the
%    equivalent Matlab function pcg. In order to solve Ax=b;
%
% Inputs: A:  double square n x n matrix. HAS TO BE POSITIVE DEFINITE
%         b:  double vector of length n.
%         x0: double vector of length n. (optional) initial guess.
%         tol: (optional) tolerance.
%         itermax: (optional) maximum number of iterations.
%
% Output: x: double vector of length n.
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
A=randn(5000,500);
A=A'*A;
b=ones(500,1);
x0=b;
tol=1e-4;
itermax=0.5*length(b);

tic
for ii = 1:20
x1 = mexConjGrad(A,b,x0,tol,itermax);
end
t=toc;
fprintf('mex-file time: %fs\n',t);

tic
for ii = 1:20
x2 = pcg(A,b);
end
t=toc;
fprintf('Matlab time: %fs\n',t);
sum((x1(:)-x2(:)).^2)
```

6.2 Function mexBayer

Apply a Bayer pattern to an input image

```
%
% Usage:   x =mexConjGrad(A,b,x0,tol,itermax)
%
% Name: mexConjGrad
%
% Description: Conjugate gradient algorithm, sometimes faster than the
%    equivalent Matlab function pcg. In order to solve Ax=b;
%
% Inputs: A:  double square n x n matrix. HAS TO BE POSITIVE DEFINITE
%         b:  double vector of length n.
%         x0: double vector of length n. (optional) initial guess.
%         tol: (optional) tolerance.
%         itermax: (optional) maximum number of iterations.
%
% Output: x: double vector of length n.
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
A=randn(5000,500);
A=A'*A;
b=ones(500,1);
x0=b;
tol=1e-4;
itermax=0.5*length(b);

tic
for ii = 1:20
x1 = mexConjGrad(A,b,x0,tol,itermax);
end
t=toc;
fprintf('mex-file time: %fs\n',t);

tic
for ii = 1:20
x2 = pcg(A,b);
end
t=toc;
fprintf('Matlab time: %fs\n',t);
sum((x1(:)-x2(:)).^2)
```

6.3 Function mexCalcAAt

For an input sparse matrix **A**, this function returns $\mathbf{A}\mathbf{A}^T$. For some reasons, when **A** has a lot more columns than rows, this function can be much faster than the equivalent matlab command `X*A'`.

```
%
% Usage:   AAt =mexCalcAAt(A);
%
% Name: mexCalcAAt
%
% Description: Compute efficiently AAt = A*A', when A is sparse
%   and has a lot more columns than rows. In some cases, it is
%   up to 20 times faster than the equivalent Matlab expression
%   AAt=A*A';
%
% Inputs: A:  double sparse m x n matrix
%
% Output: AAt: double m x m matrix
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
A=sprand(200,200000,0.05);
```

```
tic
AAt=mexCalcAAt(A);
t=toc;
fprintf('mex-file time: %fs\n',t);

tic
AAt2=A*A';
t=toc;
fprintf('matlab time: %fs\n',t);

sum((AAt(:)-AAt2(:)).^2)
```

6.4 Function mexCalcXAt

For an input sparse matrix **A** and a matrix **X**, this function returns $\mathbf{X}\mathbf{A}^T$. For some reasons, when **A** has a lot more columns than rows, this function can be much faster than the equivalent matlab command `X*A'`.

```
%
% Usage:   XAt =mexCalcXAt(X,A);
%
% Name: mexCalcXAt
%
% Description: Compute efficiently XAt = X*A', when A is sparse and has a
%   lot more columns than rows. In some cases, it is up to 20 times
%   faster than the equivalent Matlab expression XAt=X*A';
%
% Inputs: X:  double m x n matrix
%         A:  double sparse p x n matrix
%
% Output: XAt: double m x p matrix
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
X=randn(64,200000);
A=sprand(200,200000,0.05);

tic
XAt=mexCalcXAt(X,A);
t=toc;
fprintf('mex-file time: %fs\n',t);

tic
XAt2=X*A';
t=toc;
fprintf('mex-file time: %fs\n',t);

sum((XAt(:)-XAt2(:)).^2)
```

6.5 Function mexCalcXY

For two input matrices **X** and **Y**, this function returns **XY**.

```
%
% Usage:   Z =mexCalcXY(X,Y);
%
% Name: mexCalcXY
%
% Description: Compute Z=XY using the BLAS library used by SPAMS.
%
% Inputs: X:  double m x n matrix
%         Y:  double n x p matrix
%
% Output: Z: double m x p matrix
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
X=randn(64,200);
Y=randn(200,20000);

tic
XY=mexCalcXY(X,Y);
t=toc;
fprintf('mex-file time: %fs\n',t);

tic
XY2=X*Y;
t=toc;
fprintf('mex-file time: %fs\n',t);

sum((XY(:)-XY2(:)).^2)
```

6.6 Function mexCalcXYt

For two input matrices $\mathbf{X}$ and $\mathbf{Y}$, this function returns $\mathbf{X}\mathbf{Y}^T$.

```
%
% Usage:   Z =mexCalcXYt(X,Y);
%
% Name: mexCalcXYt
%
% Description: Compute Z=XY' using the BLAS library used by SPAMS.
%
% Inputs: X:  double m x n matrix
%         Y:  double p x n matrix
%
% Output: Z: double m x p matrix
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
X=randn(64,200);
Y=randn(200,20000)';

tic
XYt=mexCalcXYt(X,Y);
```

```
t=toc;
fprintf('mex-file time: %fs\n',t);


tic
XYt2=X*Y';
t=toc;
fprintf('matlab-file time: %fs\n',t);

sum((XYt(:)-XYt2(:)).^2)
```

6.7 Function mexCalcXtY

For two input matrices **X** and **Y**, this function returns $\mathbf{X}^T\mathbf{Y}$.

```
%
% Usage:   Z =mexCalcXtY(X,Y);
%
% Name: mexCalcXtY
%
% Description: Compute Z=X'Y using the BLAS library used by SPAMS.
%
% Inputs: X:  double n x m matrix
%         Y:  double n x p matrix
%
% Output: Z: double m x p matrix
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
X=randn(64,200)';
Y=randn(200,20000);

tic
XtY=mexCalcXtY(X,Y);
t=toc;
fprintf('mex-file time: %fs\n',t);

tic
XtY2=X'*Y;
t=toc;
fprintf('matlab-file time: %fs\n',t);

sum((XtY(:)-XtY2(:)).^2)
```

6.8 Function mexInvSym

For an input symmetric matrices **A** in $\mathbb{R}^{n\times n}$, this function returns $\mathbf{A}^{-1}$.

```
%
% Usage:   B =mexInvSym(A);
%
% Name: mexInvSym
%
% Description: returns the inverse of a symmetric matrix A
```

```
%
% Inputs: A:  double n x n matrix
%
% Output: B: double n x n matrix
%
% Author: Julien Mairal, 2009
```

The following piece of code illustrates how to use this function.

```
A=rand(1000,1000);
A=A'*A;

tic
B=mexInvSym(A);
t=toc;
fprintf('mex-file time: %fs\n',t);

tic
B2=inv(A);
t=toc;
fprintf('matlab-file time: %fs\n',t);

sum((B(:)-B2(:)).^2)
```

6.9 Function mexNormalize

```
%
% Usage:   Y =mexNormalize(X);
%
% Name: mexNormalize
%
% Description: rescale the columns of X so that they have
%        unit l2-norm.
%
% Inputs: X:  double m x n matrix
%
% Output: Y: double m x n matrix
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
X=rand(100,1000);

tic
Y=mexNormalize(X);
t=toc;
```

6.10 Function mexSort

```
%
% Usage:   Y =mexSort(X);
%
% Name: mexSort
%
```

```
% Description: sort the elements of X using quicksort
%
% Inputs: X:  double vector of size n
%
% Output: Y: double  vector of size n
%
% Author: Julien Mairal, 2010
```

The following piece of code illustrates how to use this function.

```
X=rand(1,300000);

tic
Y=mexSort(X);
t=toc;
toc

tic
Y2=sort(X,'ascend');
t=toc;
toc

sum((Y2(:)-Y(:)).^2)
```

6.11 Function mexDisplayPatches

Print to the screen a matrix containing as columns image patches.

6.12 Function mexCountPathsDAG

This function counts the number of paths in a DAG using dynamic programming.

```
%
% Usage:   num=mexCountPathsDAG(G);
%
% Name: mexCountPathsDAG
%
% Description: mexCountPathsDAG counts the number of paths
%       in a DAG.
%
%       for a graph G with |V| nodes and |E| arcs,
%       G is a double sparse adjacency matrix of size |V|x|V|,
%       with |E| non-zero entries.
%       (see example in test_CountPathsDAG.m
%
%
% Inputs: G:  double sparse |V| x |V| matrix (full graph)
%
% Output: num: number of paths
%
% Author: Julien Mairal, 2012
```

The following piece of code illustrates how to use this function.

```
% graph corresponding to figure 1 in arXiv:1204.4539v1
fprintf('test mexCountPathsDAG\n');
% this graph is a DAG
```

```
G=[0 0 0 0 0 0 0 0 0 0 0 0 0;
   1 0 0 0 0 0 0 0 0 0 0 0 0;
   1 1 0 0 0 0 0 0 0 0 0 0 0;
   1 1 0 0 0 0 0 0 0 0 0 0 0;
   0 0 0 1 0 0 0 0 0 0 0 0 0;
   0 1 1 0 1 0 0 0 0 0 0 0 0;
   0 1 0 0 1 0 0 0 0 0 0 0 0;
   0 0 0 0 0 1 1 0 0 0 0 0 0;
   1 0 0 1 0 0 0 0 0 0 0 0 0;
   1 0 0 0 0 0 0 0 1 0 0 0 0;
   0 0 0 0 0 0 0 0 1 1 0 0 0;
   0 0 0 0 0 0 0 0 1 0 1 0 0;
   0 0 0 0 1 0 0 0 1 0 0 1 0];
G=sparse(G);
num=mexCountPathsDAG(G);
fprintf('Num of paths: %d\n',num);
```

6.13 Function mexRemoveCyclesGraph

One heuristic to remove cycles from a graph.

```
%
% Usage:   DAG=mexRemoveCycleGraph(G);
%
% Name: mexRemoveCycleGraph
%
% Description: mexRemoveCycleGraph heuristically removes
%       arcs along cycles in the graph G to obtain a DAG.
%       the arcs of G can have weights. The heuristic will
%       remove in priority arcs with the smallest weights.
%
%       for a graph G with |V| nodes and |E| arcs,
%       G is a double sparse adjacency matrix of size |V|x|V|,
%       with |E| non-zero entries. The non-zero entries correspond
%       to the weights of the arcs.
%
%       DAG is a also double sparse adjacency matrix of size |V|x|V|,
%       but the graph is acyclic.
%
%       Note that another heuristic to obtain a DAG from a general
%       graph consists of ordering the vertices.
%
% Inputs: G:  double sparse |V| x |V| matrix
%
% Output: DAG:  double sparse |V| x |V| matrix
%
% Author: Julien Mairal, 2012
```

The following piece of code illustrates how to use this function.

```
fprintf('test mexRemoveCyclesGraph\n');
% this graph is not a DAG
G=[0   0   0   0   0   0   0   0   0   0   0   0   0;
   1   0   0   0.1 0   0   0   0.1 0   0   0   0   0;
   1   1   0   0   0   0.1 0   0   0   0   0   0   0;
   1   1   0   0   0   0   0   0   0   0   0   0.1 0;
   0   0   0   1   0   0   0   0   0   0   0   0   0;
   0   1   1   0   1   0   0   0   0   0   0   0   0;
   0   1   0   0   1   0   0   0   0   0   0   0   0;
   0   0   0   0   0   1   1   0   0   0   0   0   0;
   1   0   0   1   0   0   0   0   0   0   0   0   0;
   1   0   0   0   0   0   0   0   1   0   0   0   0;
   0   0   0   0   0   0   0   0   1   1   0   0   0;
   0   0   0   0   0   0   0   0   1   0   1   0   0;
   0   0   0   0   1   0   0   0   1   0   0   1   0];
G=sparse(G);
DAG=mexRemoveCyclesGraph(G);
format compact;
fprintf('Original graph:\n');
full(G)
fprintf('New graph:\n');
full(DAG)
```

6.14 Function mexCountConnexComponents

Count the number of connected components of a subgraph from a graph.

```
%
% Usage:   num=mexCountConnexComponents(G,N);
%
% Name: mexCountConnexComponents
%
% Description: mexCountConnexComponents counts the number of connected
%       components of the subgraph of G corresponding to set of nodes
%       in N. In other words, the subgraph of G by removing from G all
%       the nodes which are not in N.
%
%       for a graph G with |V| nodes and |E| arcs,
%       G is a double sparse adjacency matrix of size |V|x|V|,
%       with |E| non-zero entries.
%       (see example in test_CountConnexComponents.m)
%       N is a dense vector of size |V|. if  N[i] is non-zero,
%       it means that the node i is selected.
%
%
% Inputs: G:  double sparse |V| x |V| matrix (full graph)
%         N:  double full |V| vector.
%
% Output: num: number of connected components.
%
% Author: Julien Mairal, 2012
```

The following piece of code illustrates how to use this function.

```
% graph corresponding to figure 1 in arXiv:1204.4539v1
fprintf('test mexCountConnexComponents\n');
% this graph is a DAG
G=[0 0 0 0 0 0 0 0 0 0 0 0 0;
   1 0 0 0 0 0 0 0 0 0 0 0 0;
   1 1 0 0 0 0 0 0 0 0 0 0 0;
   1 1 0 0 0 0 0 0 0 0 0 0 0;
   0 0 0 1 0 0 0 0 0 0 0 0 0;
   0 1 1 0 1 0 0 0 0 0 0 0 0;
   0 1 0 0 1 0 0 0 0 0 0 0 0;
   0 0 0 0 0 1 1 0 0 0 0 0 0;
   1 0 0 1 0 0 0 0 0 0 0 0 0;
   1 0 0 0 0 0 0 0 1 0 0 0 0;
   0 0 0 0 0 0 0 0 1 1 0 0 0;
   0 0 0 0 0 0 0 0 1 0 1 0 0;
   0 0 0 0 1 0 0 0 1 0 0 1 0];
G=sparse(G);
nodes=[0 1 1 0 0 1 0 0 1 0 1 1 0];
num=mexCountConnexComponents(G,nodes);
fprintf('Num of connected components: %d\n',num);

% this graph is a not a DAG anymore. This function works
% with general graphs.
G=[0 0 0 0 0 0 0 0 0 0 0 0 0;
   1 0 0 0 0 0 0 0 0 0 0 0 0;
   1 1 0 1 0 0 0 0 0 0 0 0 0;
   1 1 0 0 0 0 0 0 0 0 0 0 0;
   0 0 0 1 0 0 0 0 0 0 0 0 0;
   0 1 1 0 1 0 0 0 0 0 0 0 0;
   0 1 0 0 1 0 0 0 0 0 0 0 0;
   0 0 0 0 0 1 1 0 0 0 0 0 0;
   1 0 0 1 0 0 0 0 0 0 0 0 0;
   1 0 0 0 0 0 0 0 1 0 0 0 0;
   0 0 0 0 0 0 0 0 1 1 0 0 0;
   0 0 0 0 0 0 0 0 1 0 1 0 0;
   0 0 0 0 1 0 0 0 1 0 0 1 0];
nodes=[0 1 1 0 0 1 0 0 1 0 1 1 0];
G=sparse(G);
num=mexCountConnexComponents(G,nodes);
fprintf('Num of connected components: %d\n',num);

nodes=[0 1 1 1 0 1 0 0 1 0 1 1 0];
num=mexCountConnexComponents(G,nodes);
fprintf('Num of connected components: %d\n',num);
```

6.15 Function mexGraphOfGroupStruct

```
%
% Usage:   groups =mexGraphOfGroupStruct(gstruct)
%
% Name: mexGraphOfGroupStruct
%
% Description: converts a group structure into the graph structure
%    used by mexProximalGraph, mexFistaGraph or mexStructTrainDL
%
% Inputs: gstruct: the structure of groups as a cell array, one element per node
%     an element is itself a 4 elements cell array:
%       nodeid (>= 0), weight (double), array of vars associated to the node,
%       array of children (nodeis's)
% Output: graph: struct (see documentation of mexProximalGraph)
%
% Author: Jean-Paul CHIEZE, 2012
```

6.16 Function mexGroupStructOfString

```
%
% Usage:   gstruct =mexGroupStructOfString(s)
%
% Name: mexGroupStructOfString
%
% Description: decode a multi-line string describing "simply" the structure of groups
%    of variables needed by mexProximalGraph, mexProximalTree, mexFistaGraph,
%    mexFistaTree and mexStructTrainDL and builds the corresponding group structure.
%    Each line describes a group of variables as a node of a tree.
%    It has up to 4 fields separated by spaces:
%        node-id node-weight [variables-list] -> children-list
%    Let's define Ng = number of groups, and Nv = number of variables.
%    node-id must be in the range (0 - Ng-1), and there must be Ng nodes
%    weight is a float
%    variables-list : a space separated list of integers, maybe empty,
%         but '[' and '] must be present. Numbers in the range (0 - Nv-1)
%    children-list : a space separated list of node-id's
%        If the list is empty, '->' may be omitted.
%    The data must obey some rules :
%        - A group contains the variables of the corresponding node and of the whole subtree.
%        - Variables attached to a node are those that are not int the subtree.
%        - If the data destination is a Graph, there may be several independant trees,
%          and a varibale may appear in several trees.
%    If the destination is a Tree, there must be only one tree, the root node
%    must have id == 0 and each variable must appear only once.
%
% Inputs: s:  the multi-lines string
%
% Output: groups: cell array, one element for each node
%                an element is itsel a 4 elements cell array:
%                 nodeid (int >= 0), weight (double), array of vars of the node,
%                array of children (nodeid's)
%
% Author: Jean-Paul CHIEZE, 2012
```

6.17 Function mexReadGroupStruct

```
%
% Usage:   gstruct =mexReadGroupStruct(file)
%
% Name: mexReadGroupStruct
%
% Description: reads a text file describing "simply" the structure of groups
%    of variables needed by mexProximalGraph, mexProximalTree, mexFistaGraph,
%    mexFistaTree and mexStructTrainDL and builds the corresponding group structure.
%    weight is a float
%    variables-list : a space separated list of integers, maybe empty,
%        but '[' and '] must be present. Numbers in the range (0 - Nv-1)
%    children-list : a space separated list of node-id's
%         If the list is empty, '->' may be omitted.
%     The data must obey some rules :
%         - A group contains the variables of the corresponding node and of the whole subtree.
%         - Variables attached to a node are those that are not int the subtree.
%         - If the data destination is a Graph, there may be several independant trees,
%            and a varibale may appear in several trees.
%     If the destination is a Tree, there must be only one tree, the root node
%         must have id == 0 and each variable must appear only once.
%
% Inputs: file:  the file name
%
% Output: groups: cell array, one element for each node
%                 an element is itsel a 4 elements cell array:
%                 nodeid (int >= 0), weight (double), array of vars of the node,
%                 array of children (nodeid's)
%
% Author: Jean-Paul CHIEZE, 2012
```

6.18 Function mexTreeOfGroupStruct

```
%
% Usage:   [permutations groups nbvars] =mexTreeOfGroupStruct(gstruct)
%
% Name: mexTreeOfGroupStruct
%
% Description: converts a group structure into the tree structure
%    used by mexProximalTree, mexFistaTree or mexStructTrainDL
%
% Inputs: gstruct: the structure of groups as a cell array, one element per node
%     an element is itself a 4 lements cell array:
%       nodeid (>= 0), weight (double), array of vars associated to the node,
%       array of children (nodeis's)
% Output: permutations: permutation vector that must be applied to the result of the
%                programm using the tree. Empty if no permutation is needed.
%      tree: struct (see documentation of mexProximalTree)
%      nbvars : number of variables in the tree
%
% Author: Jean-Paul CHIEZE, 2012
```

6.19 Function mexSimpleGroupTree

```
%
% Usage:   gstruct =mexSimpleGroupTree(degrees)
%
% Name: mexSimpleGroupTree
%
% Description: makes a structure representing a tree given the
%   degree of each level.
%
% Inputs: degrees:  int vector; degrees(i) is the number of children of each node at level i
%
% Output: group_struct: cell array, one element for each node
%                 an element is itsel a 4 elements cell array :
```

```
%                 nodeid (int >= 0), weight (double), array of vars attached to the node
%                  (here equal to [nodeid]), array of children (nodeid's)
%
% Author: Jean-Paul CHIEZE, 2012
```

A Duality Gaps with Fenchel Duality

This section is taken from the appendix D of Julien Mairal's PhD thesis [18]. We are going to use intensively Fenchel Duality (see [2]). Let us consider the problem

$$\min_{\mathbf{w}\in\mathbb{R}^p} [g(\mathbf{w}) \triangleq f(\mathbf{w}) + \lambda\psi(\mathbf{w})], \tag{47}$$

We first notice that for all the formulations we have been interested in, $g(\mathbf{w})$ can be rewritten

$$g(\mathbf{w}) = \tilde{f}(\mathbf{X}^\top\mathbf{w}) + \lambda\psi(\mathbf{w}), \tag{48}$$

where $\mathbf{X} = [\mathbf{x}^1, \ldots, \mathbf{x}^n]$ are training vectors, and $\tilde{f}$ is an appropriated smooth real-valued function of $\mathbb{R}^n$, and ψ one of the regularization functions we have introduced.

Given a primal variable $\mathbf{w}$ in $\mathbb{R}^p$ and a dual variable $\boldsymbol{\kappa}$ in $\mathbb{R}^n$, we obtain using classical Fenchel duality rules [2], that the following quantity is a duality gap for problem (47):

$$\delta(\mathbf{w}, \boldsymbol{\kappa}) \triangleq g(\mathbf{w}) + \tilde{f}^*(\boldsymbol{\kappa}) + \lambda\psi^*(-\mathbf{X}\boldsymbol{\kappa}/\lambda),$$

where $\tilde{f}^*$ and ψ^* are respectively the Fenchel conjugates of $\tilde{f}$ and ψ. Denoting by $\mathbf{w}^\star$ the solution of Eq. (47), the duality gap is interesting in the sense that it upperbounds the difference with the optimal value of the function:

$$\delta(\mathbf{w}, \boldsymbol{\kappa}) \geq g(\mathbf{w}) - g(\mathbf{w}^\star) \geq 0.$$

Similarly, we will consider pairs of primal-dual variables $(\mathbf{W}, \mathbf{K})$ when dealing with matrices.

During the optimization, sequences of primal variables $\mathbf{w}$ are available, and one wishes to exploit duality gaps for estimating the difference $g(\mathbf{w}) - g(\mathbf{w}^\star)$. This requires the following components:

- being able to efficiently compute $\tilde{f}^*$ and ψ^*.
- being able to obtain a "good" dual variable $\boldsymbol{\kappa}$ given a primal variable $\mathbf{w}$, such that $\delta(\mathbf{w}, \boldsymbol{\kappa})$ is close to $g(\mathbf{w}) - g(\mathbf{w}^\star)$.

We suppose that the first point is satisfied (we will detail these computations for every loss and regularization functions in the sequel), and explain how to choose $\boldsymbol{\kappa}$ in general (details will also be given in the sequel).

Let us first consider the choice that associates with a primal variable $\mathbf{w}$, the dual variable

$$\boldsymbol{\kappa}(\mathbf{w}) \triangleq \nabla\tilde{f}(\mathbf{X}^\top\mathbf{w}), \tag{49}$$

and let us compute $\delta(\mathbf{w}, \boldsymbol{\kappa}(\mathbf{w}))$. First, easy computations show that for all vectors $\mathbf{z}$ in $\mathbb{R}^n$, $\tilde{f}^*(\nabla\tilde{f}(\mathbf{z})) = \mathbf{z}^\top\nabla\tilde{f}(\mathbf{z}) - \tilde{f}(\mathbf{z})$, which gives

$$\begin{aligned}
\delta(\mathbf{w}, \boldsymbol{\kappa}(\mathbf{w})) &= \tilde{f}(\mathbf{X}^\top\mathbf{w}) + \lambda\psi(\mathbf{w}) + \tilde{f}^*(\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w})) + \lambda\psi^*(-\mathbf{X}\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w})/\lambda), \quad (50)\\
&= \lambda\psi(\mathbf{w}) + \mathbf{w}^\top\mathbf{X}\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w}) + \lambda\psi^*(-\mathbf{X}\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w})/\lambda). \quad (51)
\end{aligned}$$

We now use the classical Fenchel-Young inequality (see, Proposition 3.3.4 of [2]) on the function ψ, which gives

$$\delta(\mathbf{w}, \boldsymbol{\kappa}(\mathbf{w})) \geq \mathbf{w}^\top\mathbf{X}\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w}) - \mathbf{w}^\top\mathbf{X}\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w}) = 0,$$

with equality if and only if $-\mathbf{X}\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w})$ belongs to $\partial\psi(\mathbf{w})$. Interestingly, we now that first-order optimality conditions for Eq. (48) gives that $-\mathbf{X}\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w}^\star) \in \partial\psi(\mathbf{w}^\star)$. We have now in hand a non-negative function $\mathbf{w} \mapsto \delta(\mathbf{w}, \boldsymbol{\kappa}(\mathbf{w}))$ of $\mathbf{w}$, that upperbounds $g(\mathbf{w}) - g(\mathbf{w}^\star)$ and satisfying $\delta(\mathbf{w}^\star, \boldsymbol{\kappa}(\mathbf{w}^\star)) = 0$.

This is however not a sufficient property to make it a good measure of the quality of the optimization, and further work is required, that will be dependent on $\tilde{f}$ and ψ. We have indeed proven that $\delta(\mathbf{w}^\star, \boldsymbol{\kappa}(\mathbf{w}^\star))$ is always 0. However, for $\mathbf{w}$ different than $\mathbf{w}^\star$, $\delta(\mathbf{w}^\star, \boldsymbol{\kappa}(\mathbf{w}^\star))$ can be infinite, making it a non-informative duality-gap. The reasons for this can be one of the following:

- The term $\psi^*(-\mathbf{X}\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w})/\lambda)$ might have an infinite value.
- Intercepts make the problem more complicated. One can write the formulation with an intercept by adding a row to $\mathbf{X}$ filled with the value 1, add one dimension to the vector $\mathbf{w}$, and consider a regularization function ψ that does regularize the last entry of $\mathbf{w}$. This further complexifies the computation of ψ^* and its definition, as shown in the next section.

Let us now detail how we proceed to solve these problems, but first without considering the intercept. The analysis is similar when working with matrices $\mathbf{W}$ instead of vectors $\mathbf{w}$.

A.0.1 Duality Gaps without Intercepts

Let us show how to compute the Fenchel conjugate of the functions we have introduced. We now present the Fenchel conjugate of the loss functions $\tilde{f}$.

- **The square loss**

$$\tilde{f}(\mathbf{z}) = \tfrac{1}{2n}\|\mathbf{y}-\mathbf{z}\|_2^2,$$
$$\tilde{f}^*(\boldsymbol{\kappa}) = \tfrac{n}{2}\|\boldsymbol{\kappa}\|_2^2 + \boldsymbol{\kappa}^\top\mathbf{y}.$$

- **The logistic loss**

$$\tilde{f}(\mathbf{z}) = \tfrac{1}{n}\textstyle\sum_{i=1}^n \log(1+e^{-y_i\mathbf{z}_i})$$
$$\tilde{f}^*(\boldsymbol{\kappa}) = \begin{cases} +\infty \text{ if } \exists\, i \in [1;n] \text{ s.t. } y_i\boldsymbol{\kappa}_i \notin [-1,0], \\ \sum_{i=1}^n (1+y_i\boldsymbol{\kappa}_i)\log(1+y_i\boldsymbol{\kappa}_i) - y_i\boldsymbol{\kappa}_i\log(-y_i\boldsymbol{\kappa}_i) \text{ otherwise.}\end{cases}$$

- **The multiclass logistic loss (or softmax)**. The primal variable is now a matrix $\mathbf{Z}$, in $\mathbb{R}^{n\times r}$, which represents the product $\mathbf{X}^\top\mathbf{W}$. We denote by $\mathbf{K}$ the dual variable in $\mathbb{R}^{n\times r}$.

$$\tilde{f}(\mathbf{Z}) = \tfrac{1}{n}\textstyle\sum_{i=1}^n \log\Big(\sum_{j=1}^r e^{\mathbf{Z}_{ij}-\mathbf{Z}_{i\mathbf{y}_i}}\Big)$$
$$\tilde{f}^*(\mathbf{K}) = \begin{cases} +\infty \text{ if } \exists i \in [1;n] \text{ s.t. } \{\mathbf{K}_{ij} < 0 \text{ and } j \neq \mathbf{y}_i\} \text{ or } \mathbf{K}_{i\mathbf{y}_i} < -1, \\ \sum_{i=1}^n \Big[\sum_{j\neq\mathbf{y}_i}\mathbf{K}_{ij}\log(\mathbf{K}_{ij}) + (1+\mathbf{K}_{i\mathbf{y}_i})\log(1+\mathbf{K}_{i\mathbf{y}_i})\Big].\end{cases}$$

Our first remark is that the choice Eq. (49) ensures that $\tilde{f}(\boldsymbol{\kappa})$ is not infinite.

As for the regularization function, except for the Tikhonov regularization which is self-conjugate (it is equal to its Fenchel conjugate), we have considered functions that are norms. There exists therefore a norm $\|.\|$ such that $\psi(\mathbf{w}) = \|\mathbf{w}\|$, and we denote by $\|.\|_*$ its dual-norm. In such a case, the Fenchel conjugate of ψ for a vector $\boldsymbol{\gamma}$ in $\mathbb{R}^p$ takes the form

$$\psi^*(\boldsymbol{\gamma}) = \begin{cases} 0 & \text{if } \|\boldsymbol{\gamma}\|_* \leq 1, \\ +\infty & \text{otherwise.}\end{cases}$$

It turns out that for almost all the norms we have presented, there exists (i) either a closed form for the dual-norm or (ii) there exists an efficient algorithm evaluating it. The only one which does not conform to this statement is the tree-structured sum of ℓ_2-norms, for which we do not know how to evaluate it efficiently.

One can now slightly modify the definition of $\boldsymbol{\kappa}$ to ensure that $\psi^*(-\mathbf{X}\boldsymbol{\kappa}/\lambda) \neq +\infty$. A natural choice is

$$\boldsymbol{\kappa}(\mathbf{w}) \triangleq \min\Big(1, \frac{\lambda}{\|\mathbf{X}\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w})\|_*}\Big)\nabla\tilde{f}(\mathbf{X}^\top\mathbf{w}),$$

which is the one we have implemented. With this new choice, it is easy to see that for all vectors $\mathbf{w}$ in $\mathbb{R}^p$, we still have $\tilde{f}^*(\boldsymbol{\kappa}) \neq +\infty$, and finally, we also have $\delta(\mathbf{w},\boldsymbol{\kappa}(\mathbf{w})) < +\infty$ and $\delta(\mathbf{w}^\star,\boldsymbol{\kappa}(\mathbf{w}^\star)) = 0$, making it potentially a good duality gap.

A.0.2 Duality Gaps with Intercepts

Even though adding an intercept does seem a simple modification to the original problem, it induces difficulties for finding good dual variables.

We recall that having an intercept is equivalent to having a problem of the type (48), by adding a row to $\mathbf{X}$ filled with the value 1, adding one dimension to the vector $\mathbf{w}$ (or one row for matrices $\mathbf{W}$), and by using a regularization function that does not depend on the last entry of $\mathbf{w}$ (or the last row of $\mathbf{W}$).

Suppose that we are considering a problem of type (48) of dimension $p+1$, but we are using a regularization function $\tilde{\psi} : \mathbb{R}^{p+1} \to \mathbb{R}$, such that for a vector $\mathbf{w}$ in $\mathbb{R}^{p+1}$, $\tilde{\psi}(\mathbf{w}) \triangleq \psi(\mathbf{w}_{[1;p]})$, where $\psi : \mathbb{R}^p \to \mathbb{R}$ is one of the regularization function we have introduced. Then, considering a primal variable $\mathbf{w}$, a dual variable $\boldsymbol{\kappa}$, and writing $\boldsymbol{\gamma} \triangleq -\mathbf{X}\boldsymbol{\kappa}/\lambda$, we are interested in computing

$$\tilde{\psi}^*(\boldsymbol{\gamma}) = \begin{cases} +\infty \text{ if } \boldsymbol{\gamma}_{p+1} \neq 0 \\ \psi^*(\boldsymbol{\gamma}_{[1;p]}) \text{ otherwise,} \end{cases}$$

which means that in order the duality gap not to be infinite, one needs in addition to ensure that $\boldsymbol{\gamma}_{p+1}$ be zero. Since the last row of $\mathbf{X}$ is filled with ones, this writes down $\sum_{i=1}^{p+1} \boldsymbol{\kappa}_i = 0$. For the formulation with matrices $\mathbf{W}$ and $\mathbf{K}$, the constraint is similar but for every column of $\mathbf{K}$.

Let us now detail how we proceed for every loss function to find a "good" dual variable $\boldsymbol{\kappa}$ satisfying this additional constraint, given a primal variable $\mathbf{w}$ in $\mathbb{R}^{p+1}$, we first define the auxiliary function

$$\boldsymbol{\kappa}'(\mathbf{w}) \triangleq \nabla \tilde{f}(\mathbf{X}^\top \mathbf{w}),$$

(which becomes $\mathbf{K}'(\mathbf{W}) \triangleq \nabla \tilde{f}(\mathbf{X}^\top \mathbf{W})$ for matrices), and then define another auxiliary function $\boldsymbol{\kappa}''(\mathbf{w})$ as follows, to take into account the additional constraint $\sum_{i=1}^{p+1} \boldsymbol{\kappa}_i = 0$.

- **For the square loss**, we define another auxiliary function:

$$\boldsymbol{\kappa}''(\mathbf{w}) \triangleq \boldsymbol{\kappa}'(\mathbf{w}) - \frac{1}{n} \mathbf{1}_{p+1}^\top \boldsymbol{\kappa}'(\mathbf{w}) \mathbf{1}_{p+1}$$

  where $\mathbf{1}_{p+1}$ is a vector of size $p+1$ filled with ones. This step, ensures that $\sum_{i=1}^{p+1} \boldsymbol{\kappa}''(\mathbf{w})_i = 0$.

- **For the logistic loss**, the situation is slightly more complicated since additional constraints are involved in the definition of $\tilde{f}^*$.

$$\boldsymbol{\kappa}''(\mathbf{w}) \triangleq \operatorname*{arg\,min}_{\boldsymbol{\kappa} \in \mathbb{R}^n} \|\boldsymbol{\kappa} - \boldsymbol{\kappa}'(\mathbf{w})\|_2^2 \ \text{ s.t. } \ \sum_{i=1}^n \boldsymbol{\kappa}_i = 0 \text{ and } \forall i \in [1;n], \ \boldsymbol{\kappa}_i \in [-1, 0].$$

  This problem can be solved in linear-time [3] using a similar algorithm as for the projection onto the ℓ_1-ball, since it is an instance of a *quadratic knapsack problem.*

- **For the multi-class logistic loss**, we proceed in a similar way, for every column $\mathbf{K}^j$ of $\mathbf{K}$, $j \in [1;r]$:

$$\mathbf{K}''^j(\mathbf{w}) \triangleq \operatorname*{arg\,min}_{\boldsymbol{\kappa} \in \mathbb{R}^n} \|\mathbf{K}'^j - \boldsymbol{\kappa}'(\mathbf{w})\|_2^2 \ \text{ s.t. } \ \sum_{i=1}^n \boldsymbol{\kappa}_i = 0 \text{ and }$$
$$\forall i \in [1;n], \ \{\boldsymbol{\kappa}_i \geq 0 \text{ if } j \neq \mathbf{y}_i\} \text{ and } \{\boldsymbol{\kappa}_i \geq -1 \text{ if } \mathbf{y}_i = j\}.$$

When the function ψ is the Tykhonov regularization function, we end the process by setting $\boldsymbol{\kappa}(\mathbf{w}) = \boldsymbol{\kappa}''(\mathbf{w})$. When it is a norm, we choose, as before for taking into account the constraint $\|\mathbf{X}\boldsymbol{\kappa}\|_* \leq \lambda$,

$$\boldsymbol{\kappa}(\mathbf{w}) \triangleq \min\Big(1, \frac{\lambda}{\|\mathbf{X}\boldsymbol{\kappa}''(\mathbf{w})\|_*}\Big) \boldsymbol{\kappa}''(\mathbf{w}),$$

with a similar formulation for matrices $\mathbf{W}$ and $\mathbf{K}$.

Even though finding dual variables while taking into account the intercept requires quite a lot of engineering, notably implementing a quadratic knapsack solver, it can be done efficiently.

References

[1] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[2] J. M. Borwein and A. S. Lewis. *Convex analysis and nonlinear optimization: Theory and examples.* Springer, 2006.

[3] P. Brucker. An O(n) algorithm for quadratic knapsack problems. 3:163–166, 1984.

[4] E. J. Candès, M. Wakin, and S. Boyd. Enhancing sparsity by reweighted l1 minimization. *Journal of Fourier Analysis and Applications*, 14:877–905, 2008.

[5] B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[6] S. F. Cotter, J. Adler, B. Rao, and K. Kreutz-Delgado. Forward sequential algorithms for best basis selection. In *IEEE Proceedings of Vision Image and Signal Processing*, pages 235–244, 1999.

[7] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

[8] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of statistics*, 32(2):407–499, 2004.

[9] J. Friedman, T. Hastie, H. Hölfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of statistics*, 1(2):302–332, 2007.

[10] J. Friedman, T. Hastie, and R. Tibshirani. A note on the group lasso and a sparse group lasso. Technical report, Preprint arXiv:1001.0736, 2010.

[11] W. J. Fu. Penalized regressions: The bridge versus the Lasso. *Journal of computational and graphical statistics*, 7:397–416, 1998.

[12] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *Proc. of ACM Symposium on Theory of Computing*, pages 136–146, 1986.

[13] P. O. Hoyer. Non-negative sparse coding. In *Proc. IEEE Workshop on Neural Networks for Signal Processing*, 2002.

[14] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for sparse hierarchical dictionary learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

[15] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for hierarchical sparse coding. *Journal of Machine Learning Research*, 12:2297–2334, 2011.

[16] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, 2001.

[17] N. Maculan and J. R. G. Galdino de Paula. A linear-time median-finding algorithm for projecting a vector on the simplex of Rn. *Operations research letters*, 8(4):219–222, 1989.

[18] J. Mairal. *Sparse coding for machine learning, image processing and computer vision.* PhD thesis, Ecole Normale Supérieure, Cachan, 2010.

[19] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009.

[20] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11:19–60, 2010.

[21] J. Mairal, R. Jenatton, G. Obozinski, and F. Bach. Network flow algorithms for structured sparsity. In *Advances in Neural Information Processing Systems*, 2010.

[22] J. Mairal, R. Jenatton, G. Obozinski, and F. Bach. Convex and network flow optimization for structured sparsity. *Journal of Machine Learning Research*, 12:2649–2689, 2011.

[23] J. Mairal and B. Yu. Supervised feature selection in graphs with path coding penalties and network flows. *Journal of Machine Learning Research*, 2013.

[24] Julien Mairal. Optimization with first-order surrogate functions. 2013.

[25] Julien Mairal. Stochastic majorization-minimization algorithms for large-scale optimization. 2013.

[26] S. Mallat and Z. Zhang. Matching pursuit in a time-frequency dictionary. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[27] N. Meinshausen and P. Buehlmann. Stability selection. Technical report. ArXiv:0809.2932.

[28] G. Obozinski, B. Taskar, and M.I. Jordan. Joint covariate selection and joint subspace selection for multiple classification problems. *Statistics and Computing*, pages 1–22.

[29] M. R. Osborne, B. Presnell, and B. A. Turlach. On the Lasso and its dual. *Journal of Computational and Graphical Statistics*, 9(2):319–37, 2000.

[30] P. Sprechmann, I. Ramirez, G. Sapiro, and Y. C. Eldar. Collaborative hierarchical sparse modeling. Technical report, 2010. Preprint arXiv:1003.0400v1.

[31] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society Series B*, 67(1):91–108, 2005.

[32] J. A. Tropp. Algorithms for simultaneous sparse approximation. part ii: Convex relaxation. *Signal Processing, special issue "Sparse approximations in signal and image processing"*, 86:589–602, April 2006.

[33] J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. part i: Greedy pursuit. *Signal Processing, special issue "sparse approximations in signal and image processing"*, 86:572–588, April 2006.

[34] S. Weisberg. *Applied Linear Regression.* Wiley, New York, 1980.

[35] T. T. Wu and K. Lange. Coordinate descent algorithms for Lasso penalized regression. *Annals of Applied Statistics*, 2(1):224–244, 2008.

[36] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B*, 68:49–67, 2006.

[37] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B*, 67(2):301–320, 2005.