Documentation for MMDR_ML package

Version 1.2

Alessandro Benfenati

alessandro.benfenati@unimi.it

Overview

MMDR_ML is a MatLab package which contains the functions devoted to solve two optimization problems. The former, solved via the Douglas–Rachford algorithm, is

$$\underset{\mathbf{C}\in\mathcal{S}_n}{\operatorname{argmin}}\ D_f(\mathbf{C},\mathbf{S}) + \mu_0 g_0(\mathbf{C}) + \mu_1 g_1(\mathbf{C}) \tag{1}$$

where D_f is the Bregman divergence of a matrix spectral function f, $\mathbf{S}$ is the given data, g_0 is the spectral regularization term and g_1 is the regularization function acting on the whole matrix. μ_0 and μ_1 are the regularization parameters for g_0 and g_1, respectively.

The second problem which MMDR_ML solves is the non–convex problem

$$\underset{\mathbf{C}\in\mathcal{S}_n}{\operatorname{argmin}}\ \log\det\left(\mathbf{C}^{-1}+\sigma^2\mathrm{I_d}\right) + \operatorname{trace}\left(\left(\mathrm{I_d}+\sigma^2\mathbf{C}\right)^{-1}\mathbf{C}\mathbf{S}\right) + \mu_0 g_0(\mathbf{C}) + \mu_1 g_1(\mathbf{C}) \tag{2}$$

and it employs the Majorization–Minimization approach described in [1]. Matrix $\mathbf{S}$ is (usually) the given empirical covariance matrix, while σ^2 is the noise level perturbing the data. The inner subproblems arising in each MM step are solved using the Douglas–Rachford procedure, exploiting the proximal approaches involving Bregman divergences depicted in [1].

Contents

1 Using MMDR_ML

The archive MMDR_ML .tar.gz contains the following files and directories

- `documentation.pdf`
- `Funs`
- `MAIN_comparison.m`
- `Prox`
- `SideFuns`
- `Solvers`
- `TEST_parameters.mat`
- `GPL_licens.pdf`
- `Molene_Data`
- `MAIN_Molene.m`

Extract the files and the folders in the desired directory. `MAIN_comparison.m` is devoted to illustrate the comparison described in [1, Section 5]: it includes all the necessary folders and it loads the data for the numerical tests regarding the Majorization–Minimization approach described in [1]. This main file shows also how to choose the various options and how to set the parameters for the solvers `DR_matrix` and `MMDR`: the functions in `Solvers` and in `Prox` can be used in any project and do not require any particular packages nor mex files.

`Main_Molene.m` applies the MMDR algorithm (or the GLASSO approach) to real–world data set regarding weather recordings done by several stations of Radome type located in a French region. It includes all the necessary folders and automatically loads the data.

The directory `Funs` contains the m–files devoted to compute the value of the functionals employed in the variational formulation. `LogDet.m`, `VonNeu.m` and `Froben.m` compute the Bregman divergence of the log det, of the negative Von Neumann entropy and of the Frobenius norm, respectively. The other files simply calculate the values of the function.

The directory `Prox` gathers the proximity operators m–functions for several choice of functionals. The spectral approach is adopted, hence the proximity operators for matrix arguments are actually computed via the eigenvalue decomposition (see again [1] for the details).

The directory `SideFuns` collects the functions needed for some particular tasks: for the moment it just contains the function which generates the synthetic data used for the numerical experiments in `MAIN_comparison.m`.

The directory `Solvers` contains the m–functions implementing the optimization method. The novel files are `DR_matrix.m` and `MMDR.m`, while `covsel.m` is also available in [3]

The directory `Molene_Data` contains the `mat` file with the data and the image file of the region map used to depicted the eventual recovered graph.

2 SideFuns

For release 1.2 this folder contains

- the function `GenerateData`, used for the numerical validation of the proposed procedure;
- the m–file `ShowGraphOnMap`, used to show the recovered graph on the map contained in the file `map.png` (in the `Molene_Data` folder).

This directory could be used as a collector for functions having "side tasks": e.g. computing some error measurements, creating new test problems, m–files for figure creation and so on.

2.1 GenerateData.m

This function generate a synthetic data set. It is mainly based on the code available in [3]: some minor adjustments are done to adapt it to the kind of data described in [1]. All the inputs are optional.

Input	type	Description	Default
		INPUT	
'FEATS'	integer	number of features	100
'SAMPLES'	integer	number of samples to generate	10*NFeats.
'PERC'	double	percentage of nonzero elements in ISIGMA	0.001.
'NOISE'	double	noise level	0
'NSEED'	integer	random seed for generating samples.	0

'ESEED'	integer	random seed for generating nonzero positions.	0
		OUTPUT	
'SIGMA'	double array	Generated covariance matrix	
'ISIGMA'	double array	Generated precision matrix	
'S'	double array	Empirical covariance matrix	

FEATS is the number of features, namely the dimension of the desired covariance (and thus also of the precision) matrix. SAMPLES is the number of realizations from the Gaussian multivalued random variable of zero mean and covariance matrix SIGMA.

PERC is the percentage of the nonzero entries of the precision matrix: the default value create a very sparse matrix. NOISE option governs the Gaussian noise affecting the data, as described in [1]. ESEED is the seed used for the random generation of the nonzero entries' position, while the seed NSEED is employed in order to have some control on the noise generation. These options were included to assure the reproducibility of the experiments without saving data on external files.

2.2 ShowGraphOnMap

A simple file which shows the recovered graph (by any method the user prefers) on the the map regarding the region between 47°N–49°N and 2°S–6°S. The file is detailed below.

```
A = imread('map.png');
```

Read the image. If the user need to load a different file, provide the correct path and file name.

```
Ix = [-6,-2];
Fx = [1,1068];
Iy = [49,47];
Fy = [1,802];

f = @(x,I,F) (F(2)-F(1)) / (I(2)-I(1)) *(x-I(1)) + F(1);
```

Since the region of interest lies in the rectangle 47°N–49°N (latitude) 2°W–6°W (longitude), the real coordinates of the weather stations must be transformed by the function f from actual ones to the corresponding pixels values. This transformation depends on the size of the image.

```
figure, imagesc(A), axis image, axis off, hold on

MapCoords = [f(coords(:,2),Ix,Fx), f(coords(:,1),Iy,Fy)];

gplot(C_fin,MapCoords)
```

This part of code shows the map, it transform the actual coordinates of the station in the related ones in the image and finally the graph plot is drawn.

```
for i = 1:length(MapCoords)
    plot(MapCoords(i,1),MapCoords(i,2),'o',...
        'MarkerSize',5,...
        'MarkerFaceColor','w',...
        'Color',[0 0.4470 0.7410])
end
```

This cycle enlighten the weather stations on the map.

3 Funs

The available functions are

- Bounded.m
- ⋆ Froben.m
- ⋆ LogDet.m
- • Schatten2.m
- • ell1.m
- • iSchatten1.m
- • Schatten1.m
- ⋆ VonNeu.m

The files marked with [•] simply compute the value of the functionals, while the ones marked with [⋆] compute the Bregman divergence of functionals between the input arguments. Recall that the Bregman divergence is **not** symmetric in general.

When the functionals are spectral functions, the computation is done on the eigenvalues if they are provided.

4 Prox

A list of proximity operators is given:

- prox_ell1.m
- prox_LogDet.m
- prox_LogDet_Bounded.m
- prox_LogDet_Schatten1.m
- prox_LogDet_iSchatten1.m
- prox_LogDet_Schatten2.m
- prox_VonNeu_Schatten2.m
- prox_Froben_Schatten1.m
- prox_Froben_Schatten2.m
- prox_VonNeu_Schatten1.m
- prox_svd.m

This is a subset of the larger list presented n [1].

These proximity operator are coded in such way that the actual computation is carried only on the eigenvalues, exploiting the spectral properties of the functionals involved.

Suppose that the proximity operator of the function $\log\det(\mathbf{C}) + \mu_0\mathcal{R}_1(\mathbf{C})$ wrt $\mathbf{Y}$ to has to computed:

$$\underset{\mathbf{C}}{\operatorname{argmin}} \log\det(\mathbf{C}) + \mu_0\mathcal{R}_1(\mathbf{C}) + \frac{1}{2\gamma}\|\mathbf{C}-\mathbf{Y}\|_{\mathrm{F}}^2$$

with $\gamma = 1$ and $\mu_0 = 0.1$. The steps to do are

1. create a function handle for the desired functional, namely `prox_LogDet_Schatten1`
2. invoke

```
prox_svd(Y, 1, prox_LogDet_Schatten1, 0.1)
```

In general, the proximity operator of the function `myfun` at $\mathbf{Y}$ is computed by the call

```
prox_svd(Y, gamma, myfun_handle, varargin)
```

The 4th argument of `prox_svd` can be of any length: this input provides the possibility of including some further parameters for `myfun`.

5 Solvers

This folder gather the m-files implementing the optimization methods. `covsel.m` has been downloaded from [3]: its documentation is not included here.

5.1 DR_matrix

Algorithm 1 of [1] is implemented in `DR_matix.m`. The mandatory input consist of the initial estimate, of the given data $\mathbf{S}$, of the parameter γ for the inner proximity operators and of the regularization paramters μ_0 and μ_1. The optional inputs must be given in pairs: `'KEYWORD', value`.

Input	type	Description	Default
MANDATORY INPUT			
`Ck`	double array	initial estimate	none
`gamma`	double	parameter for the proximal operator	none

mu0	double	regularization parameter for g0	none
mu1	double	regularization parameter for g1	none
S	double array	given symmetric semidefinite positive matrix	none
OPTIONAL INPUT			
'ITER'	integer	maximum number of iterations.	1000
'STOP'	integer	stopping criterion for DR. '1' stands for running all the iterations; '2' stands for controlling the relative decreasing of the objective function value below the tolerance inn_tol. '3' stands for checking the relative distance between two successive	1
'LAMBDA'	double	DR parameter	1.5
'F'	string	Fit-to-data functional	'LogDet'
'G0'	string	eigenvalues regularization functional	'Shatten1'
'G1'	string	regularization functional	'ell_1'
'OBJ'	double array	Benchmark for computing performances.	empty
'VERBOSE'	integer	Verbosity. '1' (or greater than 0) prints iterations information on the shell. '0' stands for silence	1
'PARAMS'	double array	parameters needed for computing g0 and its proximal operators. See the help of the related function	[mu0, 0].
OUTPUT			
'Ck'	double array	estimated solution	
'Ck12'	double array	estimated solution (At convergence, Ck=Ck12)	
'fobj'	double array	vector of functional's values	
't_vec'	double array	employed time per iteration	
'FPR'	double array	if obj is not empty, false positive rate wrt obj	
'TPR'	double array	if obj is not empty, true positive rate wrt obj	
'ERR'	double array	if obj is not empty, relative reconstruction error wrt obj	

5.2 MMDR.m

The m–function MMDR.m implements the majorization–minimization approach for solving

$$\underset{\mathbf{C}\in\mathcal{S}_n}{\operatorname{argmin}} \log\det\left(\mathbf{C}^{-1}+\sigma^2\mathrm{I}_\mathrm{d}\right)+\operatorname{trace}\left(\left(\mathrm{I}_\mathrm{d}+\sigma^2\mathbf{C}\right)^{-1}\mathbf{CS}\right)+\mu_0 g_0(\mathbf{C})+\mu_1 g_1(\mathbf{C})$$

and uses the Douglas–Rachford approach to compute the inner steps. It does not invoke the function in DR_matrix.m: the DR solver is hard–coded in this function, future versions of this software will may include further options for the inner solver.

Input	type	Description	Default
MANDATORY INPUT			
Ck	double array	initial estimate	none
gamma	double	parameter for the proximal operator	none
mu0	double	regularization parameter for g0	none
mu1	double	regularization parameter for g1	none
S	double array	given symmetric semidefinite positive matrix	none
sigma	double	noise level	none
OPTIONAL INPUT			
'INN_ITER'	integer	maximum number of DR iterations.	100

'INN_STOP'	integer	stopping criterion for DR. Option 1 stands for running all the iterations; options 2 stands for controlling the relative decreasing of the objective function value below the tolerance inn_tol. Options 3 stands for checking the relative distance between two successive iterations.	1;
'INN_TOL'	double	tolerance for the inner stopping criterion.	1e-6
'EXT_ITER'	integer	maximum number of MM iterations.	1000
'EXT_STOP'	integer	stopping criterion for MM. Option 1 stands for running all the iterations; options 2 stands for controlling the relative decreasing of the objective function value below the tolerance inn_tol.	1
'EXT_TOL'	double	tolerance for the external stopping criterion.	1e-6
'LAMBDA'	double	DR parameter	1.5
'F'	string	Fit-to-data functional	'LogDet'
'G0'	string	eigenvalues regularization functional	'Shatten1'
'G1'	string	regularization functional	'ell_1'
'OBJ'	double array	Benchmark for computing performances.	empty
'VERBOSE'	integer	Verbosity. '1' (or greater than 0) prints iterations information on the shell. '0' stands for silence	1
'PARAMS'	double array	parameters needed for computing g0 and its proximal operators. See the help of the related function	[mu0, 0].
		OUTPUT	
Cn12	double array	estimated solution	
temp	double array	estimated solution (At convergence, Ck=Ck12)	
fobj	double array	vector of functional's values	
Q	cell	cell containing the objective functional related to each subproblem.	
t_vec	double array	employed time per iteration	
ERR	double array	if obj is not empty, relative reconstruction error wrt obj	

6 MAIN_comparison.m

Here the file MAIN_comparison.m is fully explained in its details.

```
clearvars
close all
clc
```

Clean the workspace, close all the windows and clean the shell: actually, the really important one is the first.

```
load TEST_parameters
```

The matrix **A** stored in TEST_parameters.mat contains the noise levels and the values for μ_0 and μ_1 for all of the three solvers employed in the numerical tests. **A** is organized as follows:

	MM		DR		GLASSO
σ	μ_0	μ_1	μ_0	μ_1	μ_1
0.1	0.0700	0.0400	0.0900	0.0400	0.0300
0.2	0.0699	0.0337	0.3000	0.0400	0.0300
0.3	0.0680	0.0322	0.3000	0.0400	0.0300
0.4	0.0700	0.0300	0.5300	0.0400	0.0300
0.5	0.0716	0.0278	0.8158	0.0367	0.0300

0.6	0.0700	0.0233	1.2204	0.0411	0.0300
0.7	0.0700	0.0200	1.6667	0.0479	0.0300
0.8	0.0600	0.0200	2.2100	0.0500	0.0300

The first column stores the noise levels. The second and the third column contain the regularization parameter values for g_0 and g_1, respectively, for the MM approach depending on the noise level. The fourth and the fifth column contain the values for μ_0 and μ_1 for the DR algorithm and finally the last column stores the values for μ_1 for each noise level for GLASSO approach. These values were manually searched, aiming to obtain the best results for each algorithm in terms of reconstructed values, FPR and TPR. Thus, the i–th row contains the values for the regularization parameters for noise level $\sigma = i/10$.

```
% Set the seed and generated the seeds
% for different noise realizations for
% each level
rng('default');
Lmax = 10;
SEED = round(1e4*rand(8,Lmax));
% Allocating memory for storing the performance evaluators
% GLASSO
S_GL = zeros(Lmax,8);
F_GL = zeros(Lmax,8);
% MM
S_MM = zeros(Lmax,8);
F_MM = zeros(Lmax,8);
% DR
S_DR   = zeros(Lmax,8);
F_DR   = zeros(Lmax,8);
```

The seed is set in order to assure the reproducibility of the tests presented. `Lmax` sets the number of simulation for each noise level: with the default option, for $\sigma_i = i/10$ 10 different realization of noise are generated: then, for each realization at i–th level the performance are stored in the allocated vectors. For example, vector `F_MM` stores the FPR provided by the MM approach. The variable `SEED` is used for the synthetic data generation.

```
% Generating data for each realization of noise level. Notice that
% since the eseed option is set to the default, the position of the
% nonzero entries is always the same. The nseed options is set for
% having Lmax different realization of the noise level sigma
[SIGMA, ISIGMA, S]  = GenerateData('NOISE',sigma,'nseed',SEED(K,L));
```

`GenerateData` create the covariance and the precision matrices, together with the empirical covariance matrix computed on the realizations of a Gaussian multivalued random variable of zero mean and covariance `SIGMA`. The variable `SEED` aims to generate different noise realizations, but the setting `rng('default');` previously done assures that every time the code is run the results are the same.

```
%% BOYD
mu1_boyd     = A(K,6);
[X, history] = covsel(S, mu1_boyd, 1, 1,'QUIET',1);
S_GL(L,K)    = norm(inv(X)-SIGMA,'fro')/norm(SIGMA,'fro');
F_GL(L,K)    = sum(X(:)~=0 & ISIGMA(:)==0)/zero_obj;
```

The parameter for ℓ_1 regularization is set (retrieving it from `A`) and then the method described in [2] is invoked. The relative reconstruction error

$$\frac{\left\| \left(\mathbf{C}^{gl}\right)^{-1} - \Sigma \right\|_{\mathrm{F}}^2}{\|\Sigma\|_{\mathrm{F}}^2}$$

wrt to `SIGMA` is stored in position (L, K) of `S_GL` for the L–th realization of noise at level K. In the same fashion the fpr is stored in `F_GL`. The matrix $\mathbf{C}^{gl}$ stands for the precision matrix restored by GLASSO.

```
%% MMDR
J = rand(size(S));
J = J*J' + eye(size(J));
```

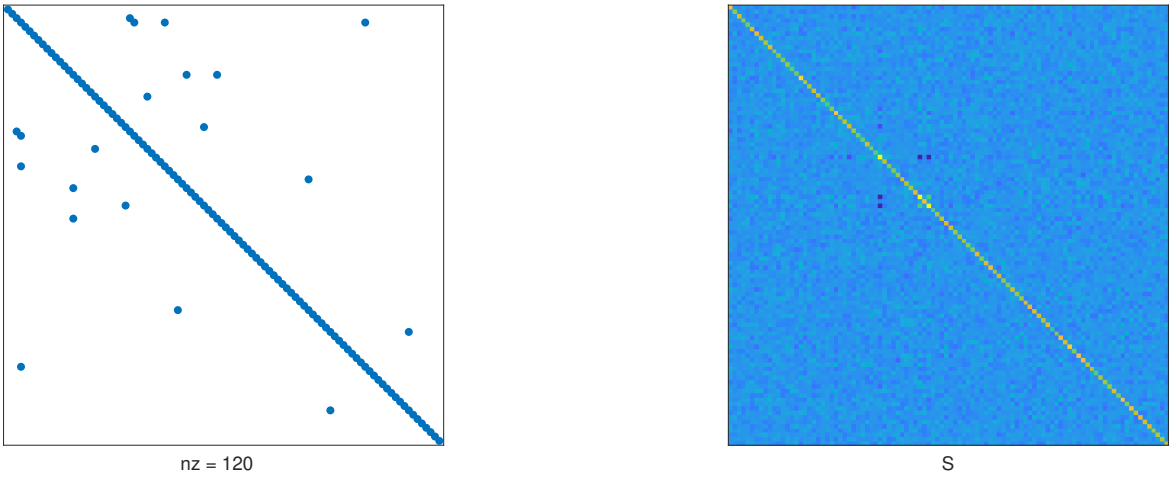

Figure 1: Precision Matrix `ISIGMA` (left) and empirical covariance matrix `S` (right) with `NSEED`=165, `ESEED`=1000*`rand` and `SAMPLES`=100.

```
optionsMMDR.mu0          = A(K,2);
optionsMMDR.mu0b         = 1;
optionsMMDR.mu1          = A(K,3);
optionsMMDR.gamma        = 1;          % Prox parameter
optionsMMDR.INNITER      = 2000;       % number of iterations
optionsMMDR.innstop      = 2;          % stopping criterion
optionsMMDR.inntol       = 1e-10;      % tolerance for the criterion
optionsMMDR.EXTITER      = 20;         % number of iterations
optionsMMDR.extstop      = 2;          % stopping criterion
optionsMMDR.exttol       = 1e-8;       % tolerance for the criterion
optionsMMDR.lambda       = 1;
optionsMMDR.F            = 'LogDet';
optionsMMDR.g0           = 'iSchatten1';
optionsMMDR.init         = J;          %S + eye(size(S));
optionsMMDR.params       = [A(K,2), sigma];

[C_fin, Cn, fobjG, Q, t_vecG, RMSEC] = MMDR(optionsMMDR.init,...
     optionsMMDR.gamma, A(K,2), A(K,3), S, sigma, ...
     'INNITER', optionsMMDR.INNITER,...
     'INNSTOP', optionsMMDR.innstop,...
     'INNTOL',  optionsMMDR.inntol,...
     'EXTITER', optionsMMDR.EXTITER,...
     'EXTSTOP', optionsMMDR.extstop,...
     'EXTTOL',  optionsMMDR.exttol,...
     'LAMBDA',  optionsMMDR.lambda,...
     'F',       optionsMMDR.F,...
     'G0',      optionsMMDR.g0,...
     'OBJ',     ISIGMA,...
     'verbose', 0,...
     'PARAMS',  optionsMMDR.params);

S_MM(L,K) = norm(inv(C_fin)-SIGMA,'fro')/norm(SIGMA,'fro');
F_MM(L,K) = sum(C_fin(:)~=0 & ISIGMA(:)==0)/zero_obj;
```

The `MMDR` function is invoked: all the settings are stored in the structure `optionsMMDR`. The spirit behind this choice is that one can save in a `.mat` file the settings for a particular problem, storing thus just one variable (`optionsMMDR`) instead of the list of choices. `S_MM` and `F_MM` are stored in the same fashion of the previous method.

```
%% DR
optionsDR                = optionsMMDR;
optionsDR.mu0            = A(K,4);
optionsDR.mu1            = A(K,5);
```

```
optionsDR.params         = A(K,4);

[Cf_DR,~,fobjDR,t_vecDR,FPR,TPR,RMSE] = DR_matrix(optionsDR.init,...
     optionsDR.gamma, optionsDR.mu0, optionsDR.mu1, S,...
     'ITER',     optionsDR.INNITER,...
     'STOP',     optionsDR.innstop,...
     'TOL',      optionsDR.inntol,...
     'LAMBDA',   optionsDR.lambda,...
     'F',        optionsDR.F,...
     'G0',       optionsDR.g0,...
     'OBJ',      ISIGMA,...
     'VERBOSE', 0,...
     'PARAMS',   optionsDR.params);

S_DR(L,K) = norm(inv(Cf_DR)-SIGMA,'fro')/norm(SIGMA,'fro');
F_DR(L,K) = FPR(end);
```

The Douglas–Rachford algorithm is applied in the same manner of the MM approach, employing the `optionsDR` variable[1]. `S_DR` and `F_DR` store the same error measurements as the two other methods.

```
%% Plotting the result
h = plot(1:8,mean(S_MM,1),'-o','Linewidth',2);
h.MarkerFaceColor = h.Color;
hold on
h = plot(1:8,mean(S_DR,1),':>','Linewidth',2);
h.MarkerFaceColor = h.Color;
h = plot(1:8,mean(S_GL,1),'--<','Linewidth',2);
h.MarkerFaceColor = h.Color;
legend('MM','DR','GLASSO')
title('Covariance rmse')
axis([1 8 0 1])

figure
h = plot(1:8,mean(F_MM,1),'-o','Linewidth',2);
h.MarkerFaceColor = h.Color;
hold on
h = plot(1:8,mean(F_DR,1),':>','Linewidth',2);
h.MarkerFaceColor = h.Color;
h = plot(1:8,mean(F_GL,1),'--<','Linewidth',2);
h.MarkerFaceColor = h.Color;
legend('MM','DR','GLASSO')
title('fpr')
axis([1 8 0 .45])
```

Last lines of code that plot the results as presented in [1]. When using this code for other tests, please remove the commands for the axis limitation, in order to have the proper visualization of the new results.

7 Molene Dataset

The M–file `MAIN_Molene.m` allows to choose between `MMDR.m`, `covsel.m` [3] or `DR_matrix.m` to address the problem of graph estimation related to data extracted from the Molene weather dataset, available at [4]. The file is fully explained below.

```
clearvars
close all
clc
```

Clean the workspace, close all the windows and clean the shell: actually, the really important one is the first.

```
addpath SideFuns/
addpath Funs/
addpath Solvers/
```

[1] Future version may include the invocation as `[results] = SolverFUN(options)`, where all the options for the method are extracted from the `options` structure.

```
addpath Prox/
addpath Molene_Data/
```

These lines add to the MatLab path the folders containing the m–files and the m–functions needed.

```
load WindData.mat
```

This line loads the data used for the experiment (in the `Molene_Data` folder). This file contains

- `wind_direction`: a double array belonging to $\mathbb{R}^{30\times 72}$ containing the hourly recordings done by 30 weather stations of the direction of the wind from 1st January 2014 to 3rd January 2014;
- `wind_intensity`: a double array belonging to $\mathbb{R}^{30\times 72}$ containing the hourly recordings done by 30 weather stations of the intensity of the wind from 1st January 2014 to 3rd January 2014;
- `wind_dirXint`: a double array belonging to $\mathbb{R}^{30\times 72}$ containing the hourly recordings of the direction multiplied (component–wise) by intensity of the wind from 1st January 2014 to 3rd January 2014: this array is computed as `wind_direction.*wind_intensity`;
- `S_direction`: a double array consisting in the covariance matrix of `wind_direction`;
- `S_intensity`: a double array consisting in the covariance matrix of `wind_intensity`;
- `S_dirXint`: a double array consisting in the covariance matrix of `wind_dirXint`;
- `sigma_direction`: the standard deviation of the elements of `wind_direction`;
- `sigma_intensity`: the standard deviation of the elements of `wind_intensity`;
- `sigma_dirXint`: the standard deviation of the elements of `wind_dirXint`;
- `coords`: double array belonging to $\mathbb{R}^{30\times 3}$ containing the coordinates (latitude and longitude) of the weather stations
- `NameCoords`: cell array containing the names of the weather stations.
- `D`: double array belonging to $\mathbb{R}^{30\times 30}$ containing the relative distances between the stations; it consists of an upper triangular matrix.

The `Molene_Data` folder contains also the `map.png` file, which represent the region between 47°N–49°N and 2°S–6°S: this file is used in the m–file `ShowGraphOnMap.m` for the superimposition of the recovered graph on the actual map, enlightening also the position of the weather stations.

```
S     = S_dirXint;
sigma = sigma_dirXint;
D     = D+D';
D     = 0.1.^D;
S     = D.*S;
```

`S` and `sigma` are the covariance matrix $\mathbf{S}$ and the noise level σ, respectively, in the variational formulation (2). The default values are the ones referring to direction×speed of the wind modulated by the distance matrix $\mathbf{D}$ (see [1, Section 5] for further details).

```
J = S;
```

The starting point for the methods

```
method = 'MMDR';
```

The file `MAIN_molene.m` contemplates the application of MM algorithm, the algorithm [3] or the Douglas–Rachford procedure, by setting the variable `method` to `'MMDR'` (default), `GLASSO` or `DR`, respectively.

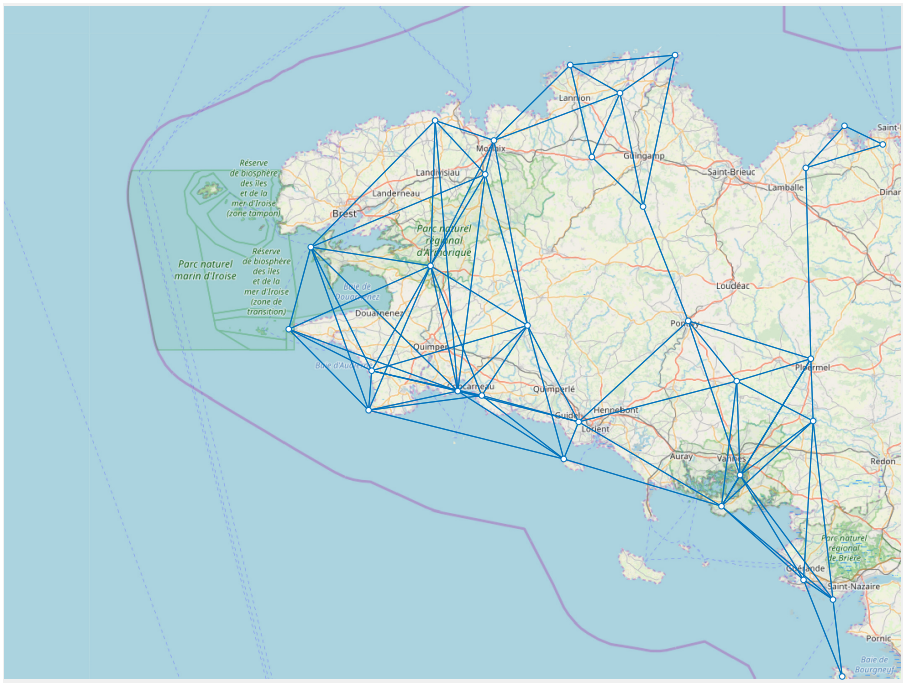

Figure 2: A recovered graph superimposed on the region.

```
if strcmp(method,'MMDR')
     ...
elseif strcmp(method,'GLASSO')
     ...
elseif strcmp(method,'DR')
     ...
else
     error('Unknown method.')
end
```

Invocation of the selected method; an error message will be displayed if the `method` variable refers to an unknown algorithm. Please refer to Section 5.2 for the options and settings for `MMDR.m`, to [3] for the options and settings for GLASSO and to Section 5.1 for the options and settings for `DR_matrix.m`.

`ShowGraphOnMap`

M–file (contained in `SideFuns` folder) which superimposes the recovered graph on the map contained in the file `map.png` (cfr Figure 2).

8 Maintenance

This package is available at `http://www-syscom.univ-mlv.fr/~benfenat/ProjSoft.html` and at `http://www-syscom.univ-mlv.fr/~chouzeno/Logiciel.html`.

It does not need any particular package nor MatLab toolbox.

For bug reporting, suggestions and/or problem please send an email to

`alessandro.benfenati@unimi.it`

References

[1] A. Benfenati, E. Chouzenoux and J.–C. Pesquet, *Proximal Approaches for Matrix Optimization Problems: Application to Robust Precision Matrix Estimation*, submitted

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato and J. Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*, Found. Trends Mach. Learn., **3**, 2011, 1–122,

[3] `http://stanford.edu/~boyd/papers/admm/covsel/covsel_example.html`

[4] `https://www.data.gouv.fr/fr/datasets/donnees-horaires-des-55-stations-terrestres-de-la-zone-large-molene-sur-un-mois/`