nearest_correlation_matrix

thibault.bourgeron@protonmail.com

Abstract

The package `nearcorrmat` provides fast computations of the nearest correlation matrix, in the Euclidian setting.

1 Problem and its dual

A correlation matrix is a real symmetric positive semidefinite matrix with unit diagonal. The problem is to find the closest correlation matrix to a given positive semidefinite matrix, for the Frobenius norm. Five algorithms are implemented.

With symbols, the problem is:

$$\min._X \frac{1}{2} \|G - X\|^2 \quad \text{s.t. } \operatorname{diag}(X) = 1, X \in S_n^+,$$

where S_n^+ (resp. S_n) is the cone (resp. space) of real positive definite matrices (resp. real symmetric matrices), G is a given positive semidefinite matrix, $\|G\|$ is the Frobenius norm, $\operatorname{diag} : S_n \to \mathbb{R}^n$ is the linear operator returning the diagonal vector of a matrix, 1 is the vector of all ones in $\mathbb{R}^n$.

The functional is convex and smooth. The constrained problem has a unique solution. The affine space $\{X \in S_n \mid \operatorname{diag}(X) = 1\}$ and the cone S_n^+ are convex sets of S_n. These properties allow this (primal) minimization problem to be tackled directly, as shown by the first two algorithms. The dual minimization problem leads to more efficient algorithms, as shown by the last three algorithms.

The dual problem is:

$$\min._{y \in \mathbb{R}^n} \theta(y) := \frac{1}{2} \|(G + \operatorname{Diag}(y))_+\|^2 - 1^T y,$$

where $\operatorname{Diag} : \mathbb{R}^n \to S_n$ is the adjoint of diag, $+ : S_n \to S_n^+$ is the projection onto S_n^+. The objective of this minimization problem is used only when a line search is performed. The most important quantity is its gradient:

$$\nabla\theta(y) = \operatorname{diag}(G + \operatorname{Diag}(y)_+) - 1.$$

It is enough to find a zero of the gradient, say y^*, to solve the primal problem, thanks to the formula:

$$x^* := G + \operatorname{Diag}(y^*)_+.$$

The dual problem is convex, unconstrained, and $\nabla\theta$ is strongly semismooth (but not differentiable).

The more use of the regularity of these functionals is performed, the more efficient the algorithm, theoretically. The first algorithm is a gradient descent for this dual problem, the third and fourth algorithms are quasi-Newton methods, while the fifth is a Newton method.

Although the algorithms were not published recently, I am not aware of any freely accessible implementation, apart from the Python module `Statsmodels` implemeting the first algorithm.

2 Five algorithms implemented

As the problem is constrained in the intersection of two convex sets, Von Neumann alternative projection can be used, *cf.* `performances.ipynb`. This algorithm may converge to a sub optimal solution (as S_n^+ is not a vector space). Dykstra's projection does not converge at all, *cf.* `performances.ipynb`.

Five algorithms are implemented in the main function `nearest_corr`. The first two are based on the primal problem, while the last three are directly based on the dual problem.

1. The algorithm 'grad' is based on Higham, 2002. This algorithm is a modified Dykstra's projection algorithm, adapted to find a point in the intersection of an affine subspace and convex subset. As noted in Malick, 2004, this algorithm is exactly a gradient descent for the dual problem. This point is checked in the tests, *cf.* `test.py`.

2. The algorithms 'admm_v0' and 'admm_v1' are ADMMs. Two versions are implemented, depending on the order in which the projections on S_n^+ and $\{X \in S_n \mid \text{diag}(X) = 1\}$ are performed. The two algorithms are different but have similar performances. To the best of my knowledge, the application of the ADMM to this problem is original.

3. The algorithm 'bfgs' is the BFGS algorithm applied to the convex, differentiable, unconstrained dual problem, *cf.* Malick. The implementations are based on BGLS textbook.

4. The algorithm 'l_bfgs' is the limited BFGS algorithm applied to the dual problem, as suggested in Malick. The implementations are based on the same book.

5. The algorithm 'newton' is a Newton's method for the dual problem. It relies on an explicit computation of an element of the Clarke Jacobian of $\nabla\theta$. It has been developed in Qi, Sun, 2006.

3 Performances

These algorihms were tested on random 100×100 matrices, for $err\_max = 10^{-6}$ (*cf.* `performances.ipynb`), which should be enough for a large range of pratical use cases. The ranking of the algorithms' speeds may be significantly different for other ranges of parameters, *cf.* table in Qi, Sun. On a given machine, the algorithms have the following performances.

algo	time	n_iterations
grad	20 s	1500
admm_v0	8 s	500
admm_v1	8 s	500
bfgs	1 s	100
l-bfgs	2 s	100
newton	60 s	100

For all the algorithms, the convergence seem to be linear, *cf.* `performances.ipynb`. This is expected for the modified Dysktra's projection and the ADMMs. Quasi-Newton methods converge superlinearly when the functional is smooth and a Wolfe line search is performed, for instance. The algorithms 'bfgs', 'l_bfgs' may converge only linearly because the gradient of the functional is only semismooth. It is surprising for the Newton's method to have only a linear convergence, as it is proved to be quadratically convergent. The quadratic convergence may be observed for other ranges of parameters; Qi, Sun test their algorithm in higher dimension (1000×1000) and for lower err_max. I could not check this result du to low computational power.

For none of the (dual) algorithms a line search is performed because a Wolfe (or an even simpler) line search considerably slows down all the (dual) algorithms and setting the step to 1 is simple and good enough (this is consistent with a remark in Qi, Sun and with the initial value for quasi-Newton methods; experimentaly, Wolfe condition is often satisfied for the value 1). For 'newton', for the same speed reason, the conjugate gradient is not used directly (as Numpy's solve is a wrapper of a fast Fortran algorithm and no faster conjugate gradient is available); the direction checking is not performed. The resolution of the Newton's linear system is probably responsible for 'newton' algorithm to be slowest, for this set of parameters.