DSPCA: User Guide

Alexandre d'Aspremont[*], Laurent El Ghaoui[†], Michael I. Jordan[‡],
Gert R. G. Lanckriet[§], Ronny Luss[¶]

July 20, 2006

Abstract

Here, we briefly detail how to install and run the sparse PCA code used in [dEGJL05]. Its aim is is to approximate, in the Frobenius-norm sense, a positive, semidefinite symmetric matrix by a rank-one matrix, with an upper bound on the cardinality of its eigenvector. The code is partly written in MATLAB, partly in C with a MEX interface.

1 Introduction

The code provided in the DSPCA package solves a relaxation of the sparse PCA decomposition. Let $A \in \mathbf{S}^n$ be a given $n \times n$ positive semidefinite, symmetric matrix and k be an integer with $1 \leq k \leq n$. The main function looks for a sparse eigenvector associated with the largest eigenvalue in A:

$$\begin{array}{ll} \max & x^T A x \\ \text{subject to} & \|x\| = 1 \\ & \mathbf{Card}(x) \leq k, \end{array} \tag{1}$$

in the variable $x \in \mathbf{R}^n$. This problem is nonconvex and *intractable*, hence (for small scale problems) we solve a semidefinite relaxation given by:

$$\begin{array}{ll} \max & \mathbf{Tr}(AX) \\ \text{subject to} & \mathbf{Tr}(X) = 1 \\ & \mathbf{1}^T |X| \mathbf{1} \leq k \\ & X \succeq 0, \end{array} \tag{2}$$

which is a semidefinite program (SDP) in the variable $X \in \mathbf{S}^n$. For large scale problems, we solve a penalized version of this problem:

$$\begin{array}{ll} \max & \mathbf{Tr}(AX) - \rho \mathbf{1}^T |X| \mathbf{1} \\ \text{subject to} & \mathbf{Tr}(X) = 1 \\ & X \succeq 0, \end{array} \tag{3}$$

[*](Contact author) ORFE, Princeton University, Princeton NJ 08540. `alexandre.daspremont@m4x.org`

[†]SAC Capital, 540 Madison Avenue, New York, NY 10029. `laurent.elghaoui@sac.com` (on leave from the EECS Dept., U.C. Berkeley)

[‡]EECS and Statistics Depts., U.C. Berkeley, Berkeley, CA 94720. `jordan@cs.berkeley.edu`

[§]ECE Dept., U.C. San Diego, La Jolla, CA 92093. `gert@eecs.berkeley.edu`

[¶]ORFE, Princeton University, Princeton NJ 08540. `rluss@princeton.edu`

We refer the reader to [dEGJL05] for further details. Three small scale examples are provided as
`CardversusKPlots.m`, `BasicHastieTest.m` and `PitPropsTest.m` corresponding to §6.1, §6.2 and §6.3 in [dEGJL05] respectively. We also provide a large scale example implementing the smooth minimization code by [Nes05] in C with calls to BLAS and LAPACK.

2 Installation & Sources

The source code, binaries and examples can be downloaded from:

`http://www.princeton.edu/~aspremon/DSPCA.htm`

The code has been tested with MATLAB 6.1 to 7.1 on WINDOWS and Mac OS X. The small scale example use SEDUMI v1.1R2 from [Stu99]. Precompiled binaries for the large scale code are provided for Mac OS X and WINDOWS. Simply copy the `.dll`, `.mexw32` or `.mexmac` file into your working directory or add them to the path.

2.1 Mac OS X

The Mac OS X version was built using gcc 3.3 and Xcode. The Xcode project is provided together with the source files. Simply update the ”search paths” in the project to reflect differences in the MATLAB installation on your machine. The code uses the (vector-optimized) BLAS and LAPACK implementations in the Apple provided vecLIB framework. Note that vecLIB uses a mix of CBLAS and f2c'd LAPACK.

2.2 Windows

The Windows version was built MS VC++, again a project file is provided together with the source files. Here, the code uses the BLAS and LAPACK libraries provided in the MATLAB installation. Again, simply update the paths in the project settings to reflect differences in your MATLAB installation.

2.3 Other Platforms

A MATLAB script `CompileCode.m` will compile the code directly from MATLAB. This has not been tested yet on platforms other than WIN32 or Mac and you should adapt the header file `sparsesvd.h` to the particular version of BLAS/LAPACK available on your system.

3 Content

3.1 Contents

The package contains two main MATLAB functions: `PrimalDec` and `DSPCA` solving small problems of type (2) and large ones of type (3) respectively. Examples and executables for `PrimalDec` and `DSPCA` are contained in the respective folders.

A call to `PrimalDec` is made as:

```
>> [resvec,resval,oval]=PrimalDec(A,k)
```

where, referring to (2)

- $A \in \mathbf{S}_n$ is the input matrix
- $(k + 1)$ is the target cardinality
- $resvec$ is the first eigenvector x^* of the solution X^*
- $resval$ is the explained variance $(x^*)^T Ax^*$
- $oval$ is the objective value $\mathbf{Tr}(AX^*)$

Both parameters to `PrimalDec` are required. Similarly, a call to `DSPCA` is made as:

```
>> [X,U,u]=DSPCA(A,rho,gapchange,maxiter,info,algo)
```

where, referring to (3)

- $A \in \mathbf{S}_n$ is the input matrix
- $\rho > 0$ is a parameter controlling sparsity
- $gapchange$ is the reduction in original gap (derived with target precision set very small)
- $maxiter$ is the maximum number of iterations
- $info$ controls verbosity: 0 is silent, $info > 1$ is the frequency of reporting
- $algo$ controls the method for computing the matrix exponential: 1 is full eigenvalue decomposition (default), 2 is Padé approximation, 3 is partial eigenvalue decomposition
- X is the matrix X solution to the dual above
- U is the solution to the primal
- u is the first eigenvector of U

All parameters are required except for the matrix exponential algorithm option. Note that `DSPCA` is a MATLAB wrapper to the mex function `sparse_rank_one_mex` and both `DSPCA.m` and the appropriate `sparse_rank_one_mex` executable must be copied to the necessary directory.

4 Example

We construct a simple sparse rank one matrix A with uniform noise

```
>> n=10;
>> ratio=100;
>> testvec=[1 0 1 0 1 0 1 0 1 0];
>> testvec=testvec/(norm(testvec));
>> A=rand(n,n);
>> A=A'*A/n+ratio*testvec'*testvec;
```

Such a small example can be solved with the described MATLAB function `PrimalDec` that uses SEDUMI to solve directly.

```
>> [resvec,resval,oval]=PrimalDec(A,4)

resvec =

    0.6123
    0.0000
    0.2863
    0.0000
    0.2773
    0.0000
    0.1604
    0.0000
    0.6637
    0.0000


resval =

   81.2335


oval =

   81.2335
```

We can then run the large-scale function `DSPCA` on this small example as a comparison.

```
>> [X,U,u]=DSPCA(A,7,1e-2,1000,5000,1);
DSPCA starting ...
Iter: 0.000e+000   Obj: 1.0137e+002 Gap:
3.5847e+001   CPU Time:  0h  0m  0s
Iter: 2.500e+002   Obj:
6.6354e+001    Gap: 4.4877e-002   CPU Time:  0h  0m  0s
>> u
```

```
u =

   -0.4449
    0.0003
   -0.4448
    0.0003
   -0.4482
    0.0002
   -0.4501
    0.0003
   -0.4480
    0.0003
```

We can finally compare this with the first eigenvector of A, given here:

```
    0.4483
    0.0057
    0.4465
    0.0066
    0.4465
    0.0063
    0.4459
    0.0045
    0.4486
    0.0037
```

Notice that DSPCA imposed sparsity in the components with the smallest magnitude. But what is more important for real applications of sparse PCA is the performance on large problems. Figure 1 shows the performance of DSPCA applied to a gene expression data set of dimension 500. We see that the partial eigenvalue decomposition (*algo=3*) has the best performance by far among the three implementations.

Examples are available as m-files in the package.

Acknowledgements

Thanks to Andrew Mullhaupt and Francis Bach for useful suggestions. We would like to acknowledge support from ONR MURI N00014-00-1-0637, Eurocontrol-C20052E/BM/03 and C20083E/BM/05, NASA-NCC2-1428.

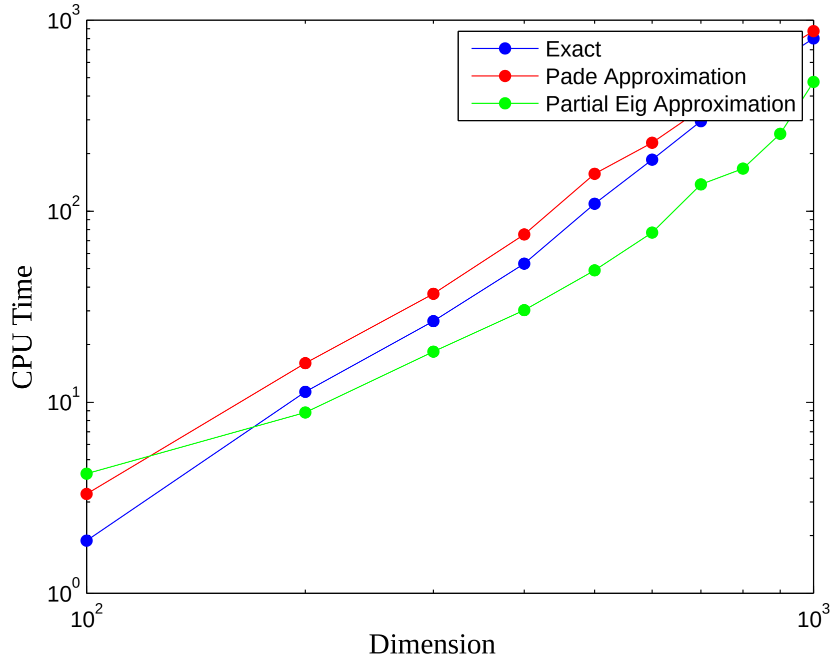

Figure 1: Running Time Comparison

References

[dEGJL05] A. d'Aspremont, L. El Ghaoui, M.I. Jordan, and G. R. G. Lanckriet. A direct formulation for sparse PCA using semidefinite programming. *Advances in Neural Information Processing Systems*, 17:41–48, 2005.

[Nes05] Y. Nesterov. Smooth minimization of nonsmooth functions. *Mathematical Programming, Series A*, 103:127–152, 2005.

[Stu99] J. Sturm. Using SEDUMI 1.0x, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11:625–653, 1999.