PALM: A Package for Solving Quadratic Eigenvalue Problems with Low-rank Damping

Prepared by Ding Lu, Yangfeng Su and Zhaojun Bai

July 1st, 2014.

1 Introduction

PALM is a package in `C++` to compute a partial spectrum of the Quadratic Eigenvalue Problem (QEP) with low-rank damping. A quadratic eigenvalue problem is to find scalar λ (eigenvalue) and vector x (eigenvector) satisfying

$$\mathcal{Q}(\lambda)x \equiv (\lambda^2 M + \lambda C + K)x = 0, \tag{1}$$

where M, C, and K are given mass, stiffness and damping matrix of size n-by-n. The low-rank damping refers to the property of matrix C has a extremely low rank, or $r \equiv \text{rank}(C) \ll n$, so it admits the factorization

$$C = EF^*, \tag{2}$$

where E and F are n-by-r matrices, and $\cdot^*$ denotes the conjugate transpose.

The name "PALM" is an abbreviation of the Pade Approximate Linearzation Method, which is proposed in [1] for computing a few eigenpairs of (1) that is close to a given non-zero shift σ. Algorithmically, PALM first approximates the original QEP by a linear eigenvalue problem of size $n + pm$

$$\mathcal{L}(\mu)x_{\text{L}} \equiv (A - \mu B)x_{\text{L}} = 0. \tag{3}$$

Here m is the order of Padé approximation for the function $\sqrt{\mu+1}$. Then it solves for a few eigenpairs of (3) with smallest magnitudes, and take $\left(\sigma\sqrt{\mu+1}, x_{\text{L}}(1:n)\right)$ as an approximation eigenpair of the QEP. User's can refer to [1] for the discussion of how to produce the LEP (3), and the detailed description of the algorithm.

2 Usage

General Usage. To use PALM in the simplest situation, the user should specify the following parameters:

`int`	`p`	:	A Padé approximant order (usually less than 10).
`int`	`nev`	:	Number of eigenvalues required.
`COMPLEX`	`s`	:	A non-zero shift, around which the eigenvalues are of interest.
`PALMat`	`M,C,K`	:	The mass, stiffness, and damping, matrices. (CSC format sparse matrices.)

Once storage has been declared and the input parameters initialized, an object of class `PALMSol` can be defined and the problem can be solved by calling its member function `EigComp()`. The calling sequence is like:

```
... ...
// Set up a PALMSol class object.
  PALMSol    LowRankQEP( p, s, nev, M, C, K);

// Solve the low-rank QEP by PALM
  LowRankQEP.EigComp();
... ...
```

The computed results are stored as internal data members of `LowRankQEP`, and one can access them through the member functions provided by the `PALMSol` class:

`complex   Eigenvalue( int i )`	:	Return i-th eigenvalue λ_i
`void*     Eigenvector( int i )`	:	Return the pointer to the i-th eigenvector x_i (normalized to unit).
`double    BackErr( int i )`	:	Return i-th norm-wise backward error.

But note that the returned eigenvalues are not necessarily ordered by their magnitudes, or distances to σ. Here, the `BackErr` is an accuracy measure of the computed eigenpair (λ_i, x_i), defined by

$$\texttt{BackErr}(i) \equiv \frac{\|(\lambda_i^2 M + \lambda C + K)x_i\|}{(|\lambda_i|^2\|M\| + |\lambda| \cdot \|C\| + \|K\|)\|x_i\|},$$

where for computational efficiency matrix 1-norm is used.

Matrices M, C and K are stored as `PALMat` class type in PALM. Currently, `PALMat` support sparse matrix in Compressed Sparse Column (CSC) format. To construct a `PALMat` matrix, one can use the constructor

```
PALMat(int m, int n, int nnz, TYPE* val, int* rowind, int* colptr);
```

where `m` and `n` specify the number of rows and columns of the matrix, `nnz` is the number of nonzero element, `val` is an array of non-zero elements of the matrix with type `TYPE` (e.g. double, complex), `rowind` is the row indices corresponding to the elements, and `colptr` is the list of elements indices where each column starts. Note that the indices can either starts from 1 (Fortran style), or 0 (C style).

User Provided Low-rank Factorization. To use PALM, the damping matrix C must be of low-rank (so can be factorized as (2)). By default, PALM will use a built-in function (see appendix) to compute the factorization $C = EF^*$. Users can also apply their own rank-revealing decomposition. To incorporate the pre-factorized $C = EF^*$ into `PALM`, one simply needs to define two `PALMat` matrices E, and F, then construct the `PALSol` object using, for example,

```
PALMSol LowRankQEP(p, s, nev, M, C, K, E, F);
```

Other operations remain the same.

We should mention that, the built-in factorization routine of PALM works well for *extremely sparse* damping matrix C, i.e., nnz(`C`)$\ll$ `n`.[1] Otherwise, the process is computational expensive, so a user provided factorization is desired.

Optional Parameters. The following is a row of optional parameters in PALM.

1. PALM utilizes ARPACK for solving the linear eigenvalue problem (3) (via solving the inverted standard eigenvalue problem $A^{-1}Bx = \frac{1}{\mu}x$). ARPACK is a package for solving large scale eigenvalue problems based on the implicitly restarted Arnoldi methods. Here is several ARPACK related parameters.

`SetARNCV(int ncv)`	This function set the number of Arnoldi vectors used by ARPACK (`2*nev+1` by default) to `ncv`.
`SetARMaxit(int nit)`	This function set the maximum number of IRAM iteration (`300` by default) to `nit`.
`SetARTol(double tol)`	This function set the stopping criteria of ARPACK (machine precision by default) to `tol`.
`SetARStat()`	By calling this function, some outputs will be produced to reflect the progress of the Arnoldi process.

2. For applying the matrix vector multiplication for $A^{-1}B$ in the eigenvalue computation, a sparse LU factorization is required. PALM utilizes the `SuperLU` package for this computation. To control the stability of the factorization, one can use the following function.

`SetSuperThresh( double tol)`	This function set the pivoting threshed (0.1 by default) to `tol`. `tol` should be contained in $(0, 1]$.

3 A Simple Example

As an illustrative example, lets consider how to solve a 5-by-5 QEP with low-rank damping:

$$M = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}, \quad C = \begin{bmatrix} f & & & & \\ & 0 & & & \\ & & 0 & & \\ & & & 0 & \\ & & & & 0 \end{bmatrix} \quad \text{and} \quad K = \begin{bmatrix} a & & & & \\ & b & & & \\ & & c & & \\ & & & d & \\ & & & & e \end{bmatrix},$$

with $a = 4$, $b = 3$, $c = 2$, $d = 1$, $e = 0$, $f = 1$. We use Padé order $m = 3$, and solve `nev` $= 2$ eigenvalues close to $\sigma = 2i$. Below is the example code for using PALM.

```
#include "palsol.h"

int main()
{
```

[1]While a low-rank damping matrix is not necessarily extremely sparse, this is usually the case in practice, so it won't cause any trouble.

```
 /* --------- DATA PREPARATION-----------*/
  double matm[5]={1,1,1,1,1}; //Matrix M
  int nnzm = 5;
  int rowindm[5]={1,2,3,4,5};
  int colptrm[6]={1,2,3,4,5,6};

  double matc[1]={0.1};  // matrix K
  int nnzc = 1;
  int rowindc[1]={1};
  int colptrc[6]={1,2,2,2,2,2};

  double matk[5]={4,3,2,1,0}; // matrix K
  int nnzk = 5;
  int rowindk[5]={1,2,3,4,5};
  int colptrk[6]={1,2,3,4,5,6};

 /*-----------Step 1: DEFINE PAL MATRICES-----------*/
  int nev = 2;  // Number of required eigenvalues.
  int  p = 1;   // Pade order.
  COMPLEX sigma(0.0,2.0);     // Shift sigma = 2i.

  int n  = 5;
  PALMatrix M(n, n, nnzm, matm, rowindm, colptrm);
  PALMatrix C(n, n, nnzc, matc, rowindc, colptrc);
  PALMatrix K(n, n, nnzk, matk, rowindk, colptrk);

  /*-----------Step 2: PROBLEM SET UP-------------*/
  PALSol LowRankQEP(p, sigma, nev, M, C, K);

  /*-----------Step 3: SOLVE THE PROBLEM-----------*/
  LowRankQEP.EigComp(); // Solve the QEP.

  LowRankQEP.PrintEig(); // Output the results.

  return 0;
}
```

4 Member Function List

This is a list of the public member functions (excluding the constructors and destructors) of `PALMSol` class.

`int np();`

Function that returns the dimension of the linearized eigenvalue problem, i.e., $n + rp$.

`double scaling();`

Function that computes and returns the scaling parameters used by PALM. Matrix 1-norm is used for the computation.

`void lufactor()`

Compute LU factorization of the matrix $Q(\sigma) = \sigma^2 M + \sigma C + K$. SuperLU subroutines will be called.

`void MultMv(COMPLEX* v, COMPLEX* w)`

Function that performs the matrix vector multiplication $w = A^{-1}Bv$.

`void EigComp();`

Function that first computes linearized eigenvalue problems by ARPACK, then recovers the QEP eigenvalues from the solution. If `NoEigVec()` is not called, then eigenvectors and the relative backward error of the eigenpairs will also be evaluated.

`void BackErrComp();`

Function that evaluates the relative backward error of the computed eigenpairs. Matrix 1-norm is used for the computation.

`COMPLEX Eigenvalue( int i);`

Return the i-th eigenvalue.

`void* Eigenvector( int i);`

Return the i-th eigenvector

`double BackErr( int i);`

Returns relative backward error of the i-th eigenpair.

`void PrintEig( ); / void PrintEig( int prec );`

Print out the computed eigenvalues (with prec digits) and the corresponding backward error.

`void SetARTol(double tol);`

Set the ARPACK stopping criteria to `tol`. If this is not called, machine precision is used.

`void SetARMaxit(int nitr);`

Set the maximum number if iteration of ARPACK to `nitr`. The default number is 300.

`void SetStat();`

Output the computation information of ARPACK in the process. This function must be called after the problem is set up, and before the eigenvalues are computed.

`void SetLUThresh(double tol);`

Set the SuperLU pivoting threshold to `tol` $\in (0, 1]$. The default number is 0.1.

`void NoEigVec();`

Function that specify that only eigenvalues are required. Eigenvectors, and backward errors are not computed.

Installation and Package Dependency

Installation instructions can be find in the source file package of PALM. To successfully install PALM, the following packages are required.

`ARPACK:`

Available at `http://www.caam.rice.edu/software/ARPACK/`.

`SuperLU:`

Available at `http://crd-legacy.lbl.gov/~xiaoye/SuperLU/`.
Version 4.0 or later.

`BLAS & LAPACK:`

Available at `http://www.netlib.org/`.
Subroutines directly called by PALM includes:
`dzscal`, `dgemv`, `dgesvd`.

For the source code of PALM, users can download from...

Appendix

A. Built-in low-rank factorization. PALM compute $C = EF^*$ for an extremely sparse matrix C based on the following algorithm.

1. Find the row indices $I = [i_1, i_2, \ldots, i_\ell]$ and column indices $J = [j_1, j_2, \ldots, j_q]$, so that $C(I, J)$ is a submatrix containing all non-zero elements in C.[2] Since C is extremely sparse, $C(I, J)$ can be of small size.

2. Compute the singular value decomposition $C(I, J) = U\Sigma V^*$, where U is ℓ-by-r, V is q-by-r and Σ is r-by-r. Then define the n-by-r matrices E and F, such that

$$E(I, :) = U\sqrt{\Sigma} \quad \text{and} \quad F(J, :) = V\sqrt{\Sigma},$$

   and the rest of rows of E and F are set to zero.

[2]We can define I as an ordered array of $\{i : \exists j \text{ s.t. } C(i, j) \neq 0\}$ and J an ordered array of $\{j : \exists i \text{ s.t. } C(i, j) \neq 0\}$.

References

[1] Ding Lu, Xin Huang, Zhaojun Bai and Yangfeng Su, A Pade approximate linearization algorithm for solving the quadratic eigenvalue problem with low-rank damping. Submitted to *Int. J. Numer. Methods Eng.*, 2014.