psfun

Release 0.1

Fabio Durastante

Oct 18, 2020

CONTENTS:

Parallel Sparse Matrix Function library. This library contains routines for the computation of matrix function-vector products

$$y = f(A)x, A \in \mathbb{R}^{n\times n},\ \mathrm{nnz}(A) = O(n),\ f : \mathbb{R} \to \mathbb{R},$$

for large and sparse matrices in a distributed setting.

The parallel environment is managed through the PSBLAS library.

CHAPTER

ONE

SERIAL MODULE

This module contains the routines needed for the computation of $f(A)x$ for A a matrix of small size. It interfaces external codes and algorithms that usually work with matrix memorized in dense storage. The intended use of the functions contained here is to use them at the lower level of a Krylov subspace method. The library directly contains the EXPOKIT code [4] for the computation of the matrix exponential, together with the scaling and squaring and Taylor algorithms [2][3] by J. Burkardt. For using the phi functions, the code from [1] is needed. It can be downloaded, compiled and linked to the main library in the install phase. bla bla

All the implemented functions and the keywords needed to load the are given in Table *Implemented Methods*.

Table 1.1: Implemented Methods

Function	Variant	Matrix	fname	variant	Source
$f(\alpha A)$	Diagonalization	Symmetric	"USERF"	"SYM"	
$\exp(\alpha A)$	Taylor	General	"EXP"	"TAYLOR"	[2][3]
	Scaling and Squaring	General	"EXP"	"SASQ"	[2][3]
	Generalized Padè	General	"EXP"	"GENPADE"	[4]
	Chebyshev	Hessenberg	"EXP"	"CHBHES"	[4]
	Chebyshev	General	"EXP"	"CHBGEN"	[4]
	Chebyshev	Symmetric	"EXP"	"CHBSYM"	[4]
$\phi_k(\alpha A)$	Scaling and Squaring	Symmetric	"PHI"	"NONE"	[1]

1.1 Module

Description

This module contains the generic interfaces for the computation of the different matrix functions included in the library. The idea is that this modules computes, in a serial way, $y = f(\alpha A)x$.

Quick access

Types *unknown_type*

Routines *psfun_d_serial_apply_array()*, *psfun_d_serial_apply_sparse()*, *psfun_d_setinteger()*, *psfun_d_setpointer()*, *psfun_d_setreal()*, *psfun_d_setstring()*

Needed modules

- psb_base_mod
- scalesquare

Types

- **type** psfun_d_serial_mod/**unknown_type**

  Type fields

  - **% fname** *[character,optional/default='exp']*
  - **% padedegree** *[integer,optional/default=6]*
  - **% phiorder** *[integer,optional/default=1]*
  - **% scaling** *[real,optional/default=1.0_psb_dpk_]*
  - **% variant** *[character,optional/default='expokit']*

Variables

Subroutines and functions

subroutine psfun_d_serial_mod/**psfun_d_setstring**(*fun*, *what*, *val*, *info*)
Set function for setting options defined by a string

Parameters

- **fun** :: Function object
- **what** *[character,in]* :: String of option to set
- **val** *[character,in]* :: Value of the string
- **info** *[integer,out]* :: Output flag

Use psb_base_mod

subroutine psfun_d_serial_mod/**psfun_d_setreal**(*fun*, *what*, *val*, *info*)
Set function for setting options defined by a real

Parameters

- **fun** :: Function object
- **what** *[character,in]* :: String of option to set
- **val** *[real,in]* :: Real Value of the option
- **info** *[integer,out]* :: Output flag

Use psb_base_mod

subroutine psfun_d_serial_mod/**psfun_d_setinteger**(*fun*, *what*, *val*, *info*)
Set function for setting options defined by an integer

Parameters

- **fun** :: Function object
- **what** *[character,in]* :: String of option to set
- **val** *[integer,in]* :: Integer Value of the option
- **info** *[integer,out]* :: Output flag

Use psb_base_mod

subroutine psfun_d_serial_mod/**psfun_d_setpointer**(*fun*, *what*, *val*, *info*)
To set the function pointer inside the type

Parameters

- **fun** :: Function object
- **what** *[character,in]* :: String of option to set
- **val** :: Function to set
- **info** *[integer,out]* :: Output flag

Use psb_base_mod

subroutine psfun_d_serial_mod/**psfun_d_serial_apply_array**(*fun*, *a*, *y*, *x*, *info*)
This is the core of the function apply on a serial matrix to compute $y = f(\alpha*A)x$. It calls on the specific routines implementing the different functions. It is the function to modify if ones want to interface a new function that was not previously available or a new algorithm (variant) for an already existing function.

Parameters

- **fun** :: Function information
- **a** (,) *[real,in]* :: We need to work on a copy of a since the Lapack routine
- **y** (*) *[real,out]* :: Output vector
- **x** (*) *[real,in]* :: Input vector
- **info** *[integer,out]* :: Information on the output

Use psb_base_mod, scalesquare

subroutine psfun_d_serial_mod/**psfun_d_serial_apply_sparse**(*fun*, *a*, *y*, *x*, *info*)
This is the core of the function apply on a serial matrix to compute $y = f(\alpha * A)x$ when A is memorized in a sparse storage. In this case the routine converts it to a dense storage and then calls the array version of itself. That is the one implementing the different functions. It is the function to modify if ones want to interface a new function that was not previously available or a new algorithm (variant) for an already existing function.

Parameters

- **fun** :: Function information
- **a** *[psb_dspmat_type,inout]* :: Matrix
- **y** (*) *[real,out]* :: Output vector
- **x** (*) *[real,in]* :: Input vector
- **info** *[integer,out]* :: Information on the output

Use `psb_base_mod`

1.2 Bibliography

CHAPTER

TWO

KRYLOV MODULE

Description

The psfun_d_krylov_mod contains the generic call to a Krylov subspace method for the computation of $y = f(A)x$, for A large and sparse.

Quick access

Types *unknown_type*

Routines *psfun_d_parallel_apply()*, *psfun_d_setstring()*

Needed modules

- psb_base_mod
- *psfun_d_serial_mod*

Types

- **type** psfun_d_krylov_mod/**unknown_type**

  Type fields

  - **% kname** *[character,optional/default='arnoldi']*

Variables

Subroutines and functions

subroutine psfun_d_krylov_mod/**psfun_d_setstring**(*meth*, *what*, *val*, *info*)

Set function for setting options defined by a string

Parameters

- **meth** :: Krylov method object
- **what** *[character,in]* :: String of option to set
- **val** *[character,in]* :: Value of the string
- **info** *[integer,out]* :: Output flag

Use psb_base_mod

subroutine psfun_d_krylov_mod/**psfun_d_parallel_apply**(*meth*, *fun*, *a*, *desc_a*, *y*, *x*, *eps*, *info*[, *itmax*[, *itrace*[, *istop*[, *iter*[, *err*]]]]])

This is the generic function for applying every implemented Krylov method. The general iteration parameters (like the number of iteration, the stop criterion to be used, and the verbosity of the trace) can be passed directly to this routine. All the constitutive parameters of the actual method, and the information relative to the function are instead contained in the meth and fun objects. The Descriptor object :p psb_desc_type desc_a [in]: Descriptor for the sparse matrix

Parameters

- **meth** :: Krylov method object
- **fun** *[psfun_d_serial,inout]* :: Function object
- **a** *[psb_dspmat_type,in]* :: Distribute sparse matrix
- **y** *[psb_d_vect_type,inout]* :: Output vector
- **x** *[psb_d_vect_type,inout]* :: Input vector
- **eps** *[real,in]* :: Requested tolerance
- **info** *[integer,out]* :: Output flag
- **itmax** *[integer,in,]* :: Maximum number of iteration
- **itrace** *[integer,in,]* :: Trace for logoutput
- **istop** *[integer,in,]* :: Stop criterion
- **iter** *[integer,out,]* :: Number of iteration
- **err** *[real,out,]* :: Last estimate error

Use psb_base_mod, *psfun_d_serial_mod*

CHAPTER

THREE

LIBRARY USAGE EXAMPLES

3.1 Serial examples

program serialtest

Test program for the serial part of the library. This test program loads a matrix from file together with some options to test the serial computation of the matrix functions. Substantially, it test the interfacing with the library doing the serial part.

Use psb_base_mod, *psfun_d_serial_mod*, psb_util_mod (mm_mat_read(), mm_array_write())

3.2 Parallel examples

Polynomial Krylov method examples

program arnolditest

Test for the parallel computation of matrix function by means of the psfun_d_arnoldi function. It applies the classical Arnoldi orthogonalization algorithm on a distributed matrix.

Use psb_base_mod, *psfun_d_serial_mod*, *psfun_d_krylov_mod*, psb_util_mod

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Souji Koikari. Algorithm 894: On a Block Schur–Parlett Algorithm for ϕ-Functions Based on the Sep-Inverse Estimate. *ACM Trans. Math. Softw.*, April 2009. URL: https://doi.org/10.1145/1499096.1499101, doi:10.1145/1499096.1499101.

[2] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Rev.*, 20(4):801–836, 1978. URL: https://doi.org/10.1137/1020098, doi:10.1137/1020098.

[3] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45(1):3–49, 2003. URL: https://doi.org/10.1137/S00361445024180, doi:10.1137/S00361445024180.

[4] Roger B. Sidje. Expokit: A Software Package for Computing Matrix Exponentials. *ACM Trans. Math. Softw.*, 24(1):130–156, March 1998. URL: https://doi.org/10.1145/285861.285868, doi:10.1145/285861.285868.

FORTRAN MODULE INDEX

p

INDEX