PRIMME: PRECONDITIONED ITERATIVE MULTIMETHOD EIGENSOLVER
COPYRIGHT © 2005 ANDREAS STATHOPOULOS, JAMES R. MCCOMBS

Table Of Contents

LICENSE INFORMATION

Contact Information (reporting bugs or questions about functionality)

Andreas Stathopoulos	
College of William and Mary	E-mail: andreas@cs.wm.edu
Department of Computer Science	Phone: 757-221-3483
P.O. Box 8795	Fax: 757-221-1717
Williamsburg, VA 23187-8795	http://www.cs.wm.edu/~andreas

Purpose

PRIMME Version 1.2 (August 29, 2014)

Finds a number of eigenvalues and their corresponding eigenvectors of a real symmetric, or complex hermitian matrix A. Largest, smallest and interior eigenvalues are supported. Preconditioning can be used to accelarate convergence.

PRIMME is written in C, but a complete Fortran77 interface is provided.

Pronounced as "prime"

How to cite this code

To cite PRIMME, please cite paper [1].
More information on the algorithms and research that led to this software can be found in the rest of the papers. The work has been supported by a number of grants from the National Science Foundation.

[1] A. Stathopoulos and J. R. McCombs PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description, ACM Transaction on Mathematical Software Vol. 37, No. 2, (2010), 21:1--21:30.

[2] A. Stathopoulos, Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue, SIAM J. Sci. Comput., Vol. 29, No. 2, (2007), 481--514.

[3] A. Stathopoulos and J. R. McCombs, Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues, SIAM J. Sci. Comput., Vol. 29, No. 5, (2007), 2162-2188.

[4] J. R. McCombs and A. Stathopoulos, Iterative Validation of Eigensolvers: A Scheme for Improving the Reliability of Hermitian Eigenvalue Solvers SIAM J. Sci. Comput., Vol. 28, No. 6, (2006), 2337--2358.

[5] A. Stathopoulos, Locking issues for finding a large number of eigenvectors of hermitian matrices, Tech Report: WM-CS-2005-03, July, 2005.

Directory structure

PRIMME/

COPYING.txt	<- LGPL License
Make_flags	<- flags to be used by makefiles to compile library and tests
Link_flags	<- flags needed in making and linking the test programs
PRIMMESRC/	<- Directory with source code in the following subdirectories:
COMMONSRC/	<- Interface and common functions used by all precision versions
DSRC/	<- The source code for the double precision dprimme
ZSRC/	<- The source code for the double complex precision zprimme
DTEST/	<- dprimme sample C and F77 drivers, both seq and parallel
ZTEST/	<- zprimme sample C and F77 drivers, sequential only
libprimme.a	<- The PRIMME library (to be made)
makefile	<- makes the libraries, and the sequential/parallel tests
readme.html	<- this file
primmestyle.css	<- an html style file
readme.txt	<- this file in text format
doc.pdf	<- a printable version of this file

Making & Linking

Users must customize Make_flags to create the library.
Users may customize Link_flags to create the test programs.

Make_flags: has the flags and compilers used by the makefiles in PRIMMESRC to make the libprimme.a library (or the stand alone double or complex precision libraries, libdprimme.a, libzprimme.a). Make_flags is also used in DTEST, ZTEST dirs to build the sequential test executables. The parallel test is built with DTEST/Makefile_par using flags and compilers for parallelism.

At minimum, the user must specify the path where the PRIMME dir is located.

PRIMME uses the UNIX gettimeofday() utility. If not available, modify the PRIMMESRC/COMMONSRC/wtime.c

Link_flags: has paths for external libraries and linking loaders needed to link the executables of the test programs. To run PRIMME:

BLAS and LAPACK libraries should be available

Users must include primme.h in their C programs, or primme_f77.h in their fortran programs. For the double complex data struct see Complex.h.

These header files are located in PRIMMESRC/COMMONSRC/

makefile can perform the following functions:

make all	builds: lib depends seqs pars
make lib	builds libprimme.a in PRIMME/. Alternatively:
make libd	if only dprimme is of interest build libdprimme.a
make libz	if only zprimme is of interest build libzprimme.a
make depends	builds the dependencies files in (D)ZTEST/
make ddepends_seq	builds only the ddependencies_seq in DTEST
make zdepends_seq	builds only the zdependencies_seq in ZTEST
make ddepends_par	builds only the ddependencies_par in DTEST
make seqs	builds all sequential executables in (D)ZTEST
make seq_dprimme	builds only the seq real C executable in DTEST
make seqf77_dprimme	builds only the seq real F77 executable in DTEST
make seq_zprimme	builds only the seq herm C executable in ZTEST

make seqf77_zprimme	builds only the seq herm F77 executable in ZTEST
make pars	builds all the parallel executables in DTEST
make par_dprimme	currently the only parallel C executable in DTEST
make clean	removes all *.o, a.out, and core files from all dirs
make backup	makes a tar.gz dated archive of entire PRIMME directory

The dependencies in (D)ZTEST need not be built unless the test code has changed, or the d/zdependencies_* files have been deleted.

The sequential and parallel versions of d/zprimme (the front-ends to PRIMME) are the same, compiled with any sequential compiler. Parallel programs can just link with and call dprimme (see below).

In **DTEST/**
The driver_seq.c is the driver for the sequential test. It fairly well structured and reads all or any of the parameters from two configuration files. It should be the best starting point. Makefile_seq is used for this.

The provided sample preconditioner and matvec are from SPARSKIT:
See http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html

The driver_par.c is the driver for the parallel test. It follows closely the structure of driver_seq.c. It uses MPI for communication and performs matrix- vector multiplications and Sparse Approximate Inverse preconditioning using the ParaSails library. Makefile_par specifies parallel compilation/linking.

ParaSails is provided in DTEST/. For more information see:
http://www.llnl.gov/CASC/parasails/

The driver_f77seq.f is a sample f77 program made with Makefile_f77seq. It calls the Fortran interface provided by PRIMME. This interface includes a set of wrapper functions that are callable from Fortran to manipulate the structure and evoke the eigensolver. For a description of the F77 interface see at the end of this file.

In **ZTEST/**
We provide driver_seq.c and driver_f77seq.f as in the real case, with Makefile_seq and Makefile_f77seq respectively. Two different diagonal preconditioners can be used, but no ILUT or Parasails.

THE COMMENTS IN THE SAMPLE DRIVERS SHOW HOW TO RUN EXECUTABLES.

SYSTEMS WHERE THE CODE HAS BEEN TESTED

Primary development with GNU gcc, g++, and gfortran (Versions 4.2 and later)
Many users have reported builds on several other platforms/compilers

Compatible with LP64 BLAS and LAPACK libraries.
ILP64 possible with other compilers but not tested.

SUSE Linux 2.6.13-15.12-smp SMP (64 bit) dual core, dual Intel Xeon 3.73GHz
SUSE Linux 2.6.13-15.12-default (32 bit) Intel Pentium 4, 2.40GHz
CentOS Linux 2.6.9-22.ELsmp (64 bit) dual processor AMD Opteron 250
SunOS 5.9, quad processor Sun-Fire-280R, and several other UltraSparcs
AIX 5.2 IBM SP POWER 3+, 16-way SMP, 375 MHz nodes (seaborg at nersc.gov)
Darwin Kernel Version 8.8.0 PowerPC dual Mac G5, 2 GHz
Darwin Kernel Version 8.8.0 PowerPC Mac G4, 1.67 GHz
Macbook Pro, Intel i7, Darwin Kernel Version 12.5.0, 2.6 GHz

C Library Interface

To solve real symmetric standard eigenproblems call:

```
int dprimme(double *evals, double *evecs, double *resNorms, primme_params *primme);
```

To solve hermitian standard eigenproblems call:

```
int zprimme(double *evals, Complex_Z *evecs, double *resNorms, primme_params *primme);
```

The following interface functions are available in PRIMMESRC/COMMONSRC/primme_interface.c

```
void primme_initialize(primme_params *primme);

int primme_set_method(primme_preset_method method, primme_params *params);

void primme_display_params(primme_params primme);

void *primme_calloc(size_t nelem, size_t elsize, const char *target);

void *primme_valloc(size_t numBytes, const char *target);

void primme_Free(primme_params primme);
```

To view the primme data structure click here.

The following data type is used internally for double complex (available in PRIMMESRC/COMMONSRC/Complex.h):

```
typedef struct {
         double r, i;
} Complex_Z;
```

Running

To call dprimme follow the basic steps below (look also at DTEST/driver_seq.c).

- Include:

  ```
  #include "primme.h"
  ```

- Initialize a primme structure for default settings:

  ```
  primme_params primme;

  primme_initialize(&primme);
  ```

- Then, for a given eigenproblem, set one of the preset methods :

  ```
  ret = primme_set_method(method, &primme);
  ```

  or/and set some of the various structure members manually (see below and PRIMMESRC/COMMONSRC/primme.h)

- Then call dprimme:

  ```
  ret = dprimme(evals, evecs, rnorms, &primme);
  ```

`evals`	a double array of size primme.numEvals	OUTPUT
`evecs`	a double array of size primme.nLocal*primme.n	INPUT / OUTPUT
`rNorms`	a double array of size primme.numEvals	OUTPUT
`primme`	the above struct. Some members return values	INPUT / OUTPUT
`ret`	error return code	OUTPUT

- Before exiting or running primme again free the work arrays in primme:

  ```
  primme_Free(&primme);
  ```

REQUIRED PARAMETER SETTING

PRIMME requires the user to set at least the following members of the primme struct, as they define the problem to be solved:

```
n (the dimension of the matrix)
nLocal (only if it is parallel program)
void (*matrixMatvec)
  (void *x, void *y, int *blockSize, struct primme_params *primme);
```

In addition, most users would want to specify how many eigenpairs to find, and provide a preconditioner (if available), a mass matrix matvec function (if a generalized eigenproblem), and a globalSum operation (if parallel).

```
numEvals
void (*applyPreconditioner)
  (void *x, void *y, int *blockSize, struct primme_params *primme);
void (*massMatrixMatvec)
  (void *x, void *y, int *blockSize, struct primme_params *primme);
void (*globalSumDouble) (only if it is parallel program)
  (void *sendBuf, void *recvBuf, int *count, primme_params *primme);
```

It is useful to have set all these before calling primme_set_method. Also, if users have a preference on basisSize, blockSize, etc, they should also provide them into primme prior to the `primme_set_method()` call. This helps primme_set_method make the right choice on other parameters.

PRESET PARAMETER SETTING

To use one of the default methods call:

```
   ret = primme_set_method(method, &primme);
```

with method one of the following:

```
   primme_preset_method method;
```

```
typedef enum {
DYNAMIC,                     // Switches to the best method dynamically
DEFAULT_MIN_TIME,            // Currently set as JDQMR_ETol
DEFAULT_MIN_MATVECS,         // Currently set as GD_Olsen_plusK
Arnoldi,                     // Anoldi implemented a la Generalized Davidson
GD,                          // Generalized Davidson
GD_plusK,                    // GD+k with locally optimal restarting for k evals
GD_Olsen_plusK,              // GD+k, preconditioner applied to (r+deltaeps*x)
JD_Olsen_plusK,              // As above, only deltaeps computed as in JD
RQI,                         // (accelerated) Rayleigh Quotient Iteration
JDQR,                        // Jacobi-Davidson with const number of inner steps
JDQMR,                       // JDQMR adaptive stopping criterion for inner QMR
JDQMR_ETol,                  // JDQMR + stops after resid reduces by a 0.1 factor
SUBSPACE_ITERATION,          // Subspace iteration
LOBPCG_OrthoBasis,           // LOBPCG with orthogonal basis. maxBlockSize = numEvals
LOBPCG_OrthoBasis_Window     // LOBPCG with sliding window of maxBlockSize< numEvals
} primme_preset_method;
```

In the absense of preset values in the primme structure, primme_set_method() uses expertly tuned default values for all parameters so that users can set a method with minimum or no input.

Users can control any parameter before or after calling primme_set_method().

- Example: if primme.maxBasisSize is set before calling primme_set_method(), other parameters (restart size, maximum block size) are computed accordingly.
- Example: Only primme.maxBlockSize and primme.numEvals need be set before calling LOBPCG_OrthoBasis_Window, and only primme.numEvals if calling LOBPCG_OrthoBasis.
- Example: Expert users may decide to modify some choices made by primme_set_method(), e.g., increasing the maxBasisSize afterwards and before calling (z)dprimme().

See `primme_set_method` in PRIMMESRC/COMMONSRC/primme_interface.c for exact description of how each method sets the members of the primme structure.

Displaying the primme configuration

The user can call

```
   primme_display_params(primme);
```

to display all the settings of primme variables. This is done in a format similar to the input format required by our (D)ZTEST drivers. Note that in some cases, a couple of primme variables may change internally in `dprimme()`. Calling `primme_display_params()` after `dprimme()` shows the actual values used.

PRIMME_PARAMS STRUCTURE

```
typedef struct primme_params {
         // The user must input at least the following two arguments
         int n;
         void (*matrixMatvec)
           ( void *x, void *y, int *blockSize, struct primme_params *primme);

         // Preconditioner applied on block of vectors (if available)
         void (*applyPreconditioner)
            ( void *x, void *y, int *blockSize, struct primme_params *primme);
         // Matrix times a multivector for mass matrix B for generalized Ax = xBl
         void (*massMatrixMatvec)
            ( void *x, void *y, int *blockSize, struct primme_params *primme);
         // input for the following is only required for parallel programs
         int numProcs;
         int procID;
         int nLocal;
         void *commInfo;
         void (*globalSumDouble)
          (void *sendBuf, void *recvBuf, int *count, struct primme_params *primme);
         // Though primme_initialize will assign defaults, most users will set these
         int numEvals;
         primme_target target;
         int numTargetShifts; // For targeting interior epairs,
         double *targetShifts; // at least one shift must also be set
         /* Printing and reporting */
         FILE *outputFile;
         int printLevel;
         /* the following will be given default values depending on the method */
         int dynamicMethodSwitch;
         int locking;
         int initSize;
         int numOrthoConst;
         int maxBasisSize;
         int minRestartSize;
         int maxBlockSize;
         int maxMatvecs;
         int intWorkSize;
         long int realWorkSize;
         int iseed[4];
         int *intWork;
         void *realWork;
         double aNorm;
         double eps;
         void *matrix;
         void *preconditioner;
         double *ShiftsForPreconditioner;
         struct restarting_params restartingParams;
         struct correction_params correctionParams;
         struct primme_stats stats;
         struct stackTraceNode *stackTrace;
} primme_params;
```

Customized parameter setting

primme has the following members: (indicating also OUTPUT members)

The user must input at least the following arguments

```
int n;
```

This is the dimension of the matrix.

```
void (*matrixMatvec)
  (void *x, void *y, int *blockSize, struct primme_params *primme);
```

Block matrix-Multivector multiplication. y = A*x. x is one dimensional array containing the input blockSize vectors packed one after the other, each of size primme.nLocal. y contains the output blockSize vectors in the same fashion. BlockSize is by reference to facilitate a possible Fortran interface. x, y are void* arrays. Thus, the user must cast them before use:

```
xvec = (double *) x for dprimme
xvec = (Complex_Z *) x for zprimme
```

The following operators are for Preconditioning and the Mass Matrix in generalized eigenproblems

```
void (*applyPreconditioner)
  (void *x, void *y, int *blockSize, struct primme_params *primme);
```

Block preconditioner-multivector application. y = applyPreconditioner(x) BlockSize is by reference to facilitate a possible Fortran interface. x, y are void* arrays, that must be cast accordingly before use.

If this preconditioner is to be used, set also correctionParams.precondition to 1

```
void (*massMatrixMatvec)
  (void *x, void *y, int *blockSize, struct primme_params *primme);
```

Generalized eigenproblems not implemented in current version
This member is included for future compatibility

Block matrix-Multivector multiplication. y = B*x. B is the mass matrix in generalized eigenvalue problems. BlockSize is by reference to facilitate a possible Fortran interface. x, y are void* arrays, that must be cast accordingly before use.

Input for the following is only required for Parallel Programs

`int numProcs; [default = 1]`

Only needed if `pnumProcs == 1` to set the `nLocal` by default to n.
The user might possibly need this info in matvec or precond.

`int procID; [default = 0]`

ProcessID. Only `procID == 0` prints information.

`int nLocal;`

Number of local rows on this processor. `nLocal = n` if sequential.

`void *commInfo;`

A pointer to whatever parallel environment structures needed.
For example, with MPI, it could be set to point to the MPI communicator.
PRIMME does not use this. It is available for possible use in
matrixMatvec, applyPreconditioner, massMatrixMatvec and globalSumDouble.

```
void (*globalSumDouble)(double *sendBuf, double *recvBuf,
   int *count, primme_params *primme);
```

Global sum function for parallel programs. No need to set for sequential.
recvBuf(i) = sum_over_all_processors(sendBuf(i)), for all i=1,count
When MPI is used this is simply a wrapper to MPI_Allreduce()
primme is needed only for primme->commInfo (eg, MPI communicator)
Count is by reference to facilitate a possible Fortran interface.
The data type is always double (even for zprimme).

PRIMME_INITIALIZE WILL ASSIGN DEFAULTS FOR ALL THE REST OF PARAMETERS
YET MOST USERS WILL SET THESE:

`int numEvals; [default = 1]`

Number of eigenvalues wanted

`primme_target target [default = primme_smallest]`

Which eigenpairs to find. target can can be any of the following enum:

`primme_smallest`	Smallest algebraic eigenvalues. Target shifts ignored
`primme_largest`	Largest algebraic eigenvalues. Target shifts ignored
`primme_closest_geq`	Closest to, but greater or equal than a set of shifts
`primme_closest_leq`	Closest to, but less or equal than a set of shifts
`primme_closest_abs`	Closest in absolute value to a set of shifts

`int numTargetShifts; [default = 0]`

Number of shifts around which interior eigenvalues
will be targeted. Used only when interior eigenvalues are saught.

`double *targetShifts; [default = NULL]`

If numTargetShifts > 0, it should point to an array of shifts.
At least one shift is required.
Not used with primme_smallest or primme_largest.

Given shifts [tau1, tau2, ..., tauk] the code finds numEvals eigenpairs
according to the three modes as follows:
Find e1 closest to tau1 AND geq/leq or closest in abs value to tau1
Find e2 closest to tau2 AND geq/leq or closest in abs value to tau2

Find e(k:numEvals) closest to tauk AND geq/leq/abs to tauk

Notes:

- For code efficiency and robustness, the shifts should be ordered. Order taus in ascending (descending) order for shifts closer to the lower (higher) end of the spectrum.
- When tau_k is closer to the lower end of the spectrum of A, primme_closest_leq is not very robust. Use either geq or abs.
- Similarly, for tau_k in the higher end of the spectrum, primme_closest_geq is not very robust. Use either leq or abs.
- closest_leq and closest_geq are more efficient than closest_abs.
- For interior eigenvalues use larger maxBasisSize than usual.

`FILE *outputFile; [default = stdout]`

An optional output file that has been opened opened by the user.

`int printLevel; [default = 1]`

The level of message reporting from the code.

0	silent
1	print some error messages when these occur
2	Level 1 AND info about targeted eigenpairs when they converge #Converged $1 eval[$2]= $3 norm $4 Mvecs $5 Time $7 or with locking: #Lock epair[$1]= $3 norm $4 Mvecs $5 Time $7
3	Level 2 AND info about targeted eigenpairs every outer iteration OUT $6 conv $1 blk $8 MV $5 Sec $7 EV $3 lrl $4 Also, if method=DYNAMIC, show JDQMR/GDk ratio and current method
4	Level 3 AND info about targeted eigenpairs every inner iteration INN MV $5 Sec $7 Eval $3 Linlrl $9 EVlrl $4
5	Level 4 AND verbose info about certain choices of the algorithm

output key:

`$1:`	num of converged pairs up to now
`$2:`	The index of the pair currently converged
`$3:`	The eigenvalue
`$4:`	Its residual norm
`$5:`	The current number of matvecs
`$6:`	The current number of outer iterations
`$7:`	The current elapsed time
`$8:`	Index within the block of the targeted pair
`$9:`	QMR norm of the linear system residual

Convergence history for plotting may be produced simply by:

```
grep OUT outpufile | awk '{print $8" "$14}' > out
grep INN outpufile | awk '{print $3" "$11}' > inn
```

Then in Matlab:

```
plot(out(:,1),out(:,2),'bo');hold; plot(inn(:,1),inn(:,2),'r');
```

Attention: In parallel programs, printLevels 0 to 4 are only printed by procID = 0
However, printLevel 5 is printed by all processors.

`struct stackTraceNode *stackTrace; (OUTPUT)`

Struct with the following members. If an error occurs the function primme_PrintStackTrace(primme) prints the calling stack from top to the function that caused the error. Nothing to set.

```
int callingFunction;
int failedFunction;
int errorCode;
int lineNumber;
char fileName[PRIMME_MAX_NAME_LENGTH];
struct stackTraceNode *nextNode;
```

`double aNorm; [default = 0.0] (OUTPUT)`

An estimate of norm of the matrix A given by the user (usu Frobenious)

If aNorm > 0.0, convergence tolerance = primme.eps * primme.aNorm
(set aNorm = 1.0 to achieve exactly primme.eps)
If aNorm <= 0.0, convergence tolerance =
primme.eps * Computed_Estimate_of_A_norm,
the Computed_Estimate_of_A_norm = largest absolute Ritz value seen
ON RETURN, aNorm also is replaced with the Computed_Estimate_of_A_norm

`double eps; [default = 1e-12]`

Convergence is declared when the 2-norm of the residual satisfies:

```
||r|| < primme.eps * primme.aNorm
```

THE FOLLOWING WILL BE GIVEN DEFAULT VALUES DEPENDING ON THE METHOD

`int dynamicMethodSwitch; [default = 0] (OUTPUT)`

Setting the primme_method to DYNAMIC, sets dynamicMethodSwitch = 1
PRIMME alternates dynamically between DEFAULT_MIN_TIME (JDQMR_ETol)
and DEFAULT_MIN_MATVECS (GD+k), trying to identify the fastest method.

On exit, it holds a recommended method for future runs on this problem:

`dynamicMethodSwitch = -1`	use DEFAULT_MIN_MATVECS next time
`dynamicMethodSwitch = -2`	use DEFAULT_MIN_TIME next time
`dynamicMethodSwitch = -3`	Close call. Use again dynamic next time

Even for expert users we do not recommend setting dynamicMethodSwitch
by hand, but only through primme_set_method(DYNAMIC).
We obtain timings by the gettimeofday Unix utility. If a cheaper, more
accurate timer is available, modify the PRIMMESRC/COMMONSRC/wtime.c

`int locking; (OUTPUT)`

If set to 1, hard locking will be used (locking converged eigenvectors
out of the search basis). Otherwise the code will try to use soft
locking (a la ARPACK), when large enough minRestartSize is available.

`int initSize; [default = 0] (OUTPUT)`

On INPUT, the number of initial guesses provided in evecs array.
ON OUTPUT, the number of converged eigenpairs.
DURING execution, holds the current number of converged eigenpairs.
If in addition locking is used, these are accessible in evals & evecs.

`int numOrthoConst; [default = 0]`

Number of external orthogonalization constraints provided in the first
numOrthoConst vectors of evecs. THen eigenvectors are found orthogonal
to those constraints (equivalent to solving (I-YY')A(I-YY') for given Y)
This is a handy feature if some eigenvectors are already known, or
for finding some eigenvalues, exiting primme, and then calling it
again (possibly with different parameters) to find some more.

`int maxBasisSize;`

The maximum basis size allowed in the main iteration. This has memory
implications. The default depends on method.

`int minRestartSize;`

The code will try to keep at least as many Ritz vectors every time
it needs to restart after the maxBasisSize has been reached.
The default depends on maxBasisSize, blockSize and method.

`int maxBlockSize; [default = 1]`

The maximum block size the code will try to use. The user should set
this based on the architecture specifics of the target computer,
as well as any a priori knowledge of multiplicities. The code does
NOT require maxBlockSize > 1 to find multiple eigenvalues. For some
methods, keeping maxBlockSize = 1 yields the best overall performance.
NOTE: Inner iterations of QMR are not performed in a block fashion.
Every correction equation from a block is solved independently.

`int maxMatvecs; [default = INT_MAX]`

Maximum number of matrix vector multiplications (approximately equal to the number
of preconditioning operations) that the code is allowed to perform before it exits.

`int maxOuterIterations; [default = INT_MAX]`

Maximum number of outer iterations that the code is allowed to perform before it exits. NOTE: Currently we do not check this.

`int intWorkSize; (INPUT/OUTPUT)`

Size of the integer work array IN BYTES. The user provides it if the user provides also the work array. After a call to dprimme/zprimme with (NULL,NULL,NULL,&primme), intWorkSize has the size of integer workspace that will be required by the parameters set in primme.

`long int realWorkSize; (INPUT/OUTPUT)`

Size of the real/complex work array IN BYTES. The user provides it if the user provides also the work array. After a call to dprimme/zprimme with (NULL,NULL,NULL,&primme), realWorkSize has the size of real workspace that will be required by the parameters set in primme.

`int *intWork; [default = NULL] (INPUT/OUTPUT)`

pointer to an integer work array. If NULL, or if its size is not sufficient, the code will free *intWork and allocate its own workspace to match the space requirements of the requested method or the primme parameters.
On output, the first element shows if a Locking problem has occurred.
Using locking for large numEvals may, in some rare cases, cause some pairs to be practically converged, in the sense that their components are in the basis of evecs. If intWork[0] == 1, and if required, a Rayleigh Ritz on evecs will provide the accurate eigenvectors (see [4]).

`void *realWork; [default = NULL] (INPUT/OUTPUT)`

pointer to a void* work array. In ZPRIMME used both for Complex_Z and double work. If not given, or if its size is not sufficient the code will free *realWork and allocate its own workspace to match the space requirements of the requested method or the primme parameters.

`int iseed[4]; [default = (1 2 3 5)]`

The seeds needed by the Lapack d/zlarnv.f.

`void *matrix;`
`void *preconditioner;`

Unused pointers that the user can use to pass any required information in the matrix-vector and preconditioning operations. See test drivers.

`double *ShiftsForPreconditioner;`

Array provided by d/zprimme() holding the shifts to be used (if needed) in the preconditioning operation. For example if the block size is 3, there will be an array of three shifts in ShiftsForPreconditioner.
Then the user can invert a shifted preconditioner for each of the block vectors:
    (M-ShiftsForPreconditioner[i])^{-1} v_i
Classical Davidson (diagonal) preconditioning is an example of this.

Restarting parameters

`struct restarting_params restartingParams;`

stucture with the following members:

`primme_restartscheme scheme` [default = `primme_thick`]

possible values are:

`primme_thick`	Thick restarting. This is the most efficient and robust in the general case.
`primme_dtr`	Dynamic thick restarting. Helpful without preconditioning but it is expensive to implement.

`int maxPrevRetain` [default = 1]

number of approximations from previous iteration to be retained after restart. This is recurrence based restarting (see GD+1, LOBPCG, etc). If maxPrevRetain > 0, then the restart size will be: minRestartSize + maxPrevRetain.

Convergence Statistics

`struct primme_stats stats; (OUTPUT)`

Struct with the following members to report statistics back. Nothing has to be set. Can be checked also during execution, e.g., in the user provided Matvec or preconditioning function.

```
int numOuterIterations;
int numRestarts;
int numMatvecs;
double elapsedTime;
```

Parameters for the correction equation

```
struct correction_params correctionParams;
```

structure with the following members:

```
int precondition; [default = 0]
```

Set to 1 if preconditioning is to be performed. Make sure the applyPreconditioner is not NULL then!

```
int robustShifts;
```

Set to 1 to use robustShifting. It helps avoid stagnation and missconvergence some times.

```
int maxInnerIterations;
```

Number of inner QMR iterations:

= 0	No inner iterations (GD/JD)
= k	Perform at most k inner iterations per outer step (or if convergence < tol)
< 0	Perform at most the rest of allowed matvecs primme.maxMatvecs - primme.stats.numMatvecs so basically do not stop by number of iterations.

```
double relTolBase;
```

This is a legacy from classical JDQR. Inner QMR is iterated until linear system residual < relTolBase^(-OuterIterations). We recommend STRONGLY against its use.

```
primme_convergencetest convTest;
```

How to stop the inner QMR method.

`= primme_full_LTolerance`	// stop by iterations only
`= primme_decreasing_LTolerance`	// LinSysResidual < // relTolBase^(-outerIterations)
`= primme_adaptive`	// JDQMR adaptive (like // Notay's JDCG)
`= primme_adaptive_ETolerance`	// as in JDQMR adaptive but // stop also when // Eres_(innerIts) < Eres_(0) * 0.1

The last scheme simply stops inner iterations when 1 order of magnitude has been achieved for the eigenresidual (NOT the linear system residual)

```
struct JD_projectors projectors;
```

Set the following to 1:

`int LeftQ;`	if a projector with Q must be applied on the left
`int LeftX;`	if a projector with X must be applied on the left
`int RightQ;`	if a projector with Q must be applied on the right
`int RightX;`	if a projector with X must be applied on the right
`int SkewQ;`	if the Q right projector must be skew
`int SkewX;`	if the X right projector must be skew

$$\underbrace{(I-QQ')}_{Lq}\ \underbrace{(I-xx')}_{Lx}\ (A\text{-shift } I)\ \underbrace{(I-Kx(x'Kx)^{\wedge}(-1)x')}_{Rx\ Sx}\ \underbrace{(I-KQ(Q'K\,Q)^{\wedge}(-1)Q')}_{Rq\ Sq}\ K$$

see `setup_JD_projectors()` in inner_solve.c for more information.

Some notes on the inner QMR convergence tests:

The following two tests could be combined in all possible ways.
Eg: maxInnerIterations convTest

0	-	GD/JD
k	full_LTolerance	JDQR always k
k	decreasingTolerance	JDQR With Steinhaug(k)
k	adaptive	JDQMR(but no more than k)
< 0	full_LTolerance	Makes no sense! DO NOT DO!
< 0	decreasingTolerance	JDQR With Steinhaug
< 0	adaptive	JDQMR
< 0	adaptive_ETol	JDQMR with only 1 order improvement

etc...

Choice of GD projectors:

```
(I - (Kinv*x) x')(I - (Kinv*Q) Q') Kinv*r
```

RitzR,	RitzSkew	EvecR,	EvecSkew	
1	1	1	1	(this is almost never done)
1	1	0	0	Olsen's
1	0	0	0	Unnecessary. Ortho follows.
0	1	0	0	Does not exist
0	0	0	0	GD
0	0			approximate Olsen: Kinv(r-err*x)

Choice of JD projectors:

```
(I-QQ')(I-xx')(A-sI)(I - (Kinv*x) x')(I - (Kinv*Q) Q') Kinv*r
```

EvecL	RitzL	RitzR,	RitzSkew	EvecR,	EvecSkew	
1	1	1	1	1	1	
1	1	1	0	1	0	
1	1	1	1	1	0	
1	1	1	0	1	1	meaningless

...

Fortran Library Interface

The following functions are available for users to call from FORTRAN.
Notice the appendix _f77.

```
call dprimme_f77(double *evals, double *evecs, double *rnorms, primme_params **primme, int *ierr);

call zprimme_f77(double *evals, Complex_Z *evecs, double *rnorms, primme_params **primme, int *ierr);

call primme_initialize_f77(primme_params **primme);

call primme_free_f77(primme_params **primme);

call primme_display_params_f77(primme_params **primme);

call primme_printstacktrace_f77(primme_params **primme);

call primme_set_method_f77(primme_params **primme, int *method,int *ierr);

call primme_set_member_f77(primme_params **primme, int *label, void *ptr);

call primme_get_prec_shift_f77(primme_params *primme, int *i, double *shift);

call primme_get_member_f77(primme_params *primme, int *label, void *ptr);

call primmetop_get_member_f77(primme_params **primme, int *label, void *ptr);

call primmetop_get_prec_shift_f77(primme_params **primme, int *i, double *shift);
```

Running From Fortran

To call d/zprimme from Fortran, look at (D)ZTEST/driver_f77seq.c. Basic steps:

```
* Include "primme_f77.h"
* initialize a primme structure for default settings:

integer primme   (or integer*8 primme if running on a 64 bit operating system)

call primme_initialize_f77(primme)
```

Then, set the various structure members and possibly the method to be used. Individual primme members can be set by calling:

```
call primme_set_member_f77(primme, MEMBER_LABEL, variable)
```

where MEMBER_LABEL is the name of the primme member prepended with PRIMMEF77_ and replacing all dots (.) with underscores (_). See primme_f77.h Eg.:

```
call primme_set_member_f77(primme, PRIMMEF77_correctionParams_precondition, 1)
```

sets the primme->correctionParams.precondition = 1

Matrix-vector and preconditioning fortran function must be declared external and passed in the same way to primme:

```
external MV
...
call primme_set_member_f77(primme, PRIMMEF77_matrixMatvec, MV)
```

A preset method is chosen in a similar way:

```
   call primme_set_method_f77(primme, METHOD, RealMem)
```

where METHOD has the name of the preset method as above, prepended by PRIMMEF77_ All these are defined in primme_f77.

```
   call primme_set_method_f77(primme, PRIMMEF77_JDQMR_ETol, ierr)
```

Then call dprimme:

```
   call dprimme_f77(evals, evecs, rnorms, primme, ierr)
```

How to obtain the value of a member of the PRIMME structure depends on whether it is called from an F77 subroutine called directly by the driver, or by an F77 function invoked by PRIMME (such as matrixMatvec, applyPreconditioner, massMatrixMatvec or globalSumDouble)

From the driver: primme is really: primme_params **primme

```
   call primmetop_get_member_f77(primme, MEMBER_LABEL, ResultVariable)
```

From matrixMatvec, applyPreconditioner, massMatrixMatvec and globalSumDouble: primme is really: primme_params *primme, so a different function must be used:

```
   call primme_get_member_f77(primme, MEMBER_LABEL, ResultVariable)
```

In preconditioning, a PRIMME provided shift may be used. Assuming a block of blockSize, from which the i-th shift is needed, i = 1,2,3...,blockSize the user can call from inside the f77 applyPreconditioner() the following function:

```
   call primme_get_prec_shift_f77(primme, i, shift)
```

List of error codes returned by dprimme() / zprimme()

These can be found also in primme_d.c or primme_z.c

`0:`	Success
`1:`	Reporting only amount of required memory
`-1:`	Failure to allocate int or real workspace
`-2:`	Malloc failure in allocating a permutation integer array
`-3:`	main_iter() encountered problem. PRIMME has printed in STDERR the calling stack of the functions where the error occured
`-4:`	if (primme == NULL)
`-5:`	if (primme->n <= 0 ll primme->nLocal <= 0)
`-6:`	if (primme->numProcs < 1)
`-7:`	if (primme->matrixMatvec == NULL)
`-8:`	if (primme->applyPreconditioner == NULL && primme->correctionParams.precondition)
`-9:`	if (primme->globalSumDouble == NULL)
`-10:`	if (primme->numEvals > primme->n)
`-11:`	if (primme->numEvals < 0)
`-12:`	if (primme->eps > 0.0L && primme->eps < machine precision
`-13:`	if (primme->target not properly defined)
`-14:`	if (primme->target == primme_closest_geq/leq/abs (interior needed) && primme->numTargetShifts <= 0 (no shifts)
`-15:`	if (primme->target == primme_closest_geq/leq/abs (interior needed) && (primme->targetShifts == NULL (no shifts array)
`-16:`	if (primme->numOrthoConst < 0 ll primme->numOrthoConst >=primme->n (no free dimensions left)
`-17:`	if (primme->maxBasisSize < 2)
`-18:`	if (primme->minRestartSize <= 0)
`-19:`	if (primme->maxBlockSize <= 0)
`-20:`	if (primme->restartingParams.maxPrevRetain < 0)
`-21:`	if (primme->restartingParams.scheme != primme_thick or primme_dtr)
`-22:`	if (primme->initSize < 0)
`-23:`	if (!primme->locking && primme->initSize > primme->maxBasisSize)
`-24:`	if (primme->locking && primme->initSize > primme->numEvals)
`-25:`	if (primme->restartingParams.maxPrevRetain + primme->minRestartSize >= primme->maxBasisSize)
`-26:`	if (primme->minRestartSize >= primme->n)
`-27:`	if (primme->printLevel < 0 ll primme->printLevel > 5)
`-28:`	if (primme->correctionParams.convTest is not one of the primme_full_LTolerance, primme_decreasing_LTolerance, primme_adaptive_ETolerance, primme_adaptive)
`-29:`	if (primme->correctionParams.convTest == primme_decreasing_LTolerance primme->correctionParams.relTolBase <= 1.0L)
`-30:`	if evals == NULL (but not all evecs, evals and resNorms)
`-31:`	if evecs == NULL (but not all evecs, evals and resNorms)
`-32:`	if resNorms == NULL (but not all evecs, evals and resNorms)

HTML Prepared By: Kim Berry