Exploring the Potential of the PRIMME Eigensolver

Part II: Eigenvalue Problems

Andreas Stathopoulos[1]   Eloy Romero Alcalde[1]   Lingfei Wu[2]

[1]Computer Science Department, College of William & Mary, USA

[2]IBM Thomas J. Watson Research Center, Yorktown Heights, USA

CSE'17

Installation

Download:

- get tarball from github and extract `https://github.com/primme/primme/releases/latest`
- developer's version `git clone https://github.com/primme/primme`

Compile:

```
make            # build lib/libprimme.a
make matlab     # build lib/libprimme.a and Matlab's module
make octave     # build lib/libprimme.a and Octave's module
make python     # build lib/libprimme.a and Python's module
```

Installation

Set makefile variables, examples:

- `make CC=clang CFLAGS=-O2`
- `make CC=icc CFLAGS="-O2 -xHost"`

Alternatively, modify `Make_flags`.

BLAS/LAPACK related flags in `CFLAGS`:

`-DF77UNDERSCORE` Fortran function symbols like `dgemm_`, `dsyevx_`; if not, remove the flag from `Make_flags`

`-DPRIMME_BLASINT_SIZE=64` BLAS/LAPACK compiled with 64-bits integer (ILP64)

Example:

```
make CC=clang CFLAGS="-O2 -DPRIMME_BLASINT_SIZE=64"
```

Installation

Other `make` actions:

`make lib`	(default action) build static library `libprimme.a`
`make solib`	build dynamic library `libprimme.so/.dylib`
`make clean`	remove all object files (`*.o`)
`make test`	test `libprimme.a`
`make all_tests`	long test useful for developers

For testing you may need to set Fortran compiler and linking options for BLAS & LAPACK if they aren't in defaults' system paths. For instance

```
make test F77=gfortran \
          LDFLAGS="-L/my/blaslapack -llapack -lblas"
```

How to install BLAS & LAPACK

For Linux:

```
# Debian, Ubuntu
sudo apt-get install libblas-dev liblapack-dev
# Fedora, RHEL, Centos
sudo yum install blas-devel lapack-devel
# SuSE
sudo zypper install blas-devel lapack-devel
```

For Windows you can use binary packages from OpenBLAS (`bin/libopenblas.dll` includes BLAS & LAPACK).

Optimized BLAS:

- ATLAS `https://sourceforge.net/projects/math-atlas/`
- OpenBLAS `http://www.openblas.net/`
- Intel© MKL `https://software.intel.com/en-us/intel-mkl/`

Compile and Linking User Code

- For compiling, include `$PRIMME_DIR/include`
- For linking, link with PRIMME, LAPACK and BLAS, in that order; PRIMME library is in `$PRIMME_DIR/lib`

Eg to compile `ex_eigs_dseq.c` in `$PRIMME_DIR/examples`:

```
gcc -c ex_eigs_dseq.c -I$PRIMME_DIR/include
gcc -o exe ex_eigs_dsec.o -L$PRIMME_DIR/lib \
                              -lprimme -llapack -lblas
```

Working with IDE: Eclipse, XCode, Visual Studio...

For compiling PRIMME

1. Create a new project
2. Add PRIMME source files under `src`
3. Add `include` and `src/include` as include directories
4. Add the macro `F77UNDERSCORE` to the compiler line (eg `-DF77UNDERSCORE` for gcc/clang or `/DF77UNDERSCORE` for cl)
5. Add one of the following macros to compile for a particular datatype
   - `USE_DOUBLE`
   - `USE_DOUBLECOMPLEX`
   - `USE_FLOAT`
   - `USE_FLOATCOMPLEX`
6. Build

- Create separate projects for every architecture

Working with IDE: Eclipse, XCode, Visual Studio...

For using PRIMME in a project

- Add reference to the previously generated `libprimme.a/.dll`
- Add reference to BLAS/LAPACK libraries
- Add PRIMME `include` as include directory

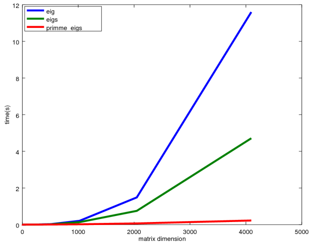
Matlab/Octave basic examples

```
ops.tol = 1e-5;
teig=[]; teigs=[]; tpeigs=[];
for i=2.^(1:12)
  tic;
  eig(diag(1:i));
  teig=[teig toc];

  tic;
  eigs(diag(1:i),1,'SA',ops);
  teig=[teig toc];

  tic;
  primme_eigs(diag(1:i),1,'SA',ops);
  tpeig=[tpeig toc];
end

plot(2.^(1:12), [teig' teigs' tpeigs']);
```



Not always primme_eigs is faster than eigs, but we try to!

C minimum example

```c
#include "primme.h" // PRIMME header file

// Declare and initialize PRIMME configuration struct
primme_params primme;
primme_initialize(&primme);

// Set the function that implements A*x
primme.matrixMatvec = MatrixMatvec;

// Set problem parameters
primme.n = 100;             // set problem dimension
primme.numEvals = 4;        // number of eigenpairs wanted
primme.eps = 1e-12;         // ||r|| <= eps * ||A||
primme.target = primme_smallest; // seek for smallest

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double)),
      *evecs = malloc(primme.n*primme.numEvals*sizeof(double));

dprimme(evals, evecs, rnorms, &primme); // do eigenstuff
```

C minimum example

```c
#include "primme.h" // PRIMME header file

// Declare and initialize PRIMME configuration struct
primme_params primme;
primme_initialize(&primme);

// Set the function that implements A*x
primme.matrixMatvec = MatrixMatvec;

// Set problem parameters
primme.n = 100;          // set problem dimension
primme.numEvals = 4;     // number of eigenpairs wanted
primme.eps = 1e-12;      // ‖r‖ ≤ eps ∗ ‖A‖
primme.target = primme_largest;   // seek for largest

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double)),
      *evecs = malloc(primme.n*primme.numEvals*sizeof(double));

dprimme(evals, evecs, rnorms, &primme); // do eigenstuff
```

C minimum example

```
#include "primme.h" // PRIMME header file

// Declare and initialize PRIMME configuration struct
primme_params primme;
primme_initialize(&primme);

// Set the function that implements A*x
primme.matrixMatvec = MatrixMatvec;

// Set problem parameters
primme.n = 100;            // set problem dimension
primme.numEvals = 4;       // number of eigenpairs wanted
primme.eps = 1e-12;        // ||r|| ≤ eps * ||A||
primme.target = primme_largest_abs; // seek for largest magn.

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double)),
      *evecs = malloc(primme.n*primme.numEvals*sizeof(double));

dprimme(evals, evecs, rnorms, &primme); // do eigenstuff
```

C minimum example

```c
#include "primme.h" // PRIMME header file

// Declare and initialize PRIMME configuration struct
primme_params primme;
primme_initialize(&primme);

// Set the function that implements A*x
primme.matrixMatvec = MatrixMatvec;

// Set problem parameters
primme.n = 100;          // set problem dimension
primme.numEvals = 4;     // number of eigenpairs wanted
primme.eps = 1e-12;      // ‖r‖ ≤ eps ∗ ‖A‖
// seek for closest to a point
double shift = 3.5;
primme.target = primme_closest_abs;     // both sides
primme.targetShifts = &shift;
primme.numTargetShifts = 1;

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double)),
```

C minimum example

```c
#include "primme.h" // PRIMME header file

// Declare and initialize PRIMME configuration struct
primme_params primme;
primme_initialize(&primme);

// Set the function that implements A*x
primme.matrixMatvec = MatrixMatvec;

// Set problem parameters
primme.n = 100;           // set problem dimension
primme.numEvals = 4;      // number of eigenpairs wanted
primme.eps = 1e-12;       // ‖r‖ ≤ eps ∗ ‖A‖
// seek for closest to a point
double shift = 3.5;
primme.target = primme_closest_leq; // on the left
primme.targetShifts = &shift;
primme.numTargetShifts = 1;

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
```

C minimum example

```c
#include "primme.h" // PRIMME header file

// Declare and initialize PRIMME configuration struct
primme_params primme;
primme_initialize(&primme);

// Set the function that implements A*x
primme.matrixMatvec = MatrixMatvec;

// Set problem parameters
primme.n = 100;           // set problem dimension
primme.numEvals = 4;      // number of eigenpairs wanted
primme.eps = 1e-12;       // ‖r‖ ≤ eps ∗ ‖A‖
// seek for closest to a point
double shift = 3.5;
primme.target = primme_closest_geq; // on the right
primme.targetShifts = &shift;
primme.numTargetShifts = 1;

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double))
```

C minimum example

```c
#include "primme.h" // PRIMME header file

// Declare and initialize PRIMME configuration struct
primme_params primme;
primme_initialize(&primme);

// Set the function that implements A*x
primme.matrixMatvec = MatrixMatvec;

// Set problem parameters
primme.n = 100;          // set problem dimension
primme.numEvals = 4;     // number of eigenpairs wanted
primme.eps = 1e-12;      // ||r|| <= eps * ||A||
primme.target = primme_smallest; // seek for smallest

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double)),
      *evecs = malloc(primme.n*primme.numEvals*sizeof(double));

dprimme(evals, evecs, rnorms, &primme); // do eigenstuff
```

C minimum example

```
...

// Set problem parameters
primme.n = 100;            // set problem dimension
primme.numEvals = 4;       // number of eigenpairs wanted
...

dprimme(evals, evecs, rnorms, &primme); // do eigenstuff

// Print values, residual norms, and the vectors
for (int i=0; i<primme.numEvals; i++) {
    printf("Pair %d, value %g residual norm %g\n",
           i, evals[i], rnorms[i]);
    printf("vector[%d] = ", i);
    for (int j=0; j<primme.n; j++)
        printf("%g ", evecs[j + primme.n*i]); // = evecs(j,i)
    printf("\n")
}

primme_free(&primme);      // free workarrays in PRIMME
```

User's Matrix-vector product

```
void MatrixMatvec(                    // Do y = A * x
  void *x, PRIMME_INT *ldx,           // input vectors and leading dimen
  void *y, PRIMME_INT *ldy,           // output vectors and lead. dim.
  int *numCols,                       // number of columns
  primme_params *primme,              // PRIMME configuration
  int *err                            // output flag error
) {

   for (int i=0; i<*numCols; i++) { // for every column
      double *x_i = (double*)x + (*ldx)*i; // = x(0,i)
      double *y_i = (double*)y + (*ldx)*i; // = y(0,i)

      // Do y_i = A * x_i
      ...
   }

   *err = 0; // All went ok
}
```

User's Matrix-vector product - Ex. diag(1:100)

```
void MatrixMatvec(                   // Do y = A * x
  void *x, PRIMME_INT *ldx,          // input vectors and leading dimen
  void *y, PRIMME_INT *ldy,          // output vectors and lead. dim.
  int *numCols,                      // number of columns
  primme_params *primme,             // PRIMME configuration
  int *err                           // output flag error
) {

   for (int i=0; i<*numCols; i++) { // for every column
      double *x_i = (double*)x + (*ldx)*i; // = x(0,i)
      double *y_i = (double*)y + (*ldx)*i; // = y(0,i)

      // Do y_i = diag(1:100) * x_i
      for (int j=0; i<100; j++) {
         y_i[j] = (double)(j+1) * x_i[j];
      }
   }

   *err = 0; // All went ok
}
```

Matlab/Octave basic examples

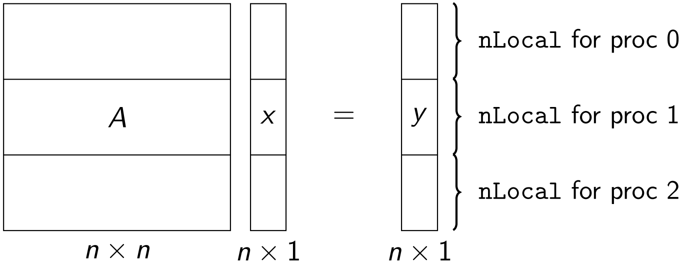
```
A = diag(1:100);

opts.tol = 1e-12; % norm(r) <= norm(A)*tol
d = primme_eigs(A, 10, 'LM', opts) % the 10 largest magnitude
d' =
  100.000   99.000   98.000   97.000   96.000   95.000
   94.000   93.000   92.000   91.000

d = primme_eigs(A, 3, 'SM', opts); % return the eigenvalues
d' =
   1.00000   2.00000   3.00000
```

Distributed matrix-vector product

- PRIMME assumes the matrix is distributed by rows
- Every process shoud set how many local rows have, `primme.nLocal`



Parallel example with PETSc

```
#include "primme.h" // PRIMME header file

// Declare and initialize PRIMME configuration struct
primme_params primme; primme_initialize(&primme);

// Set problem dimensions and user defined functions
Mat A = generateProblem();
PetscInt n, nLocal;
MatGetSize(A, &n, NULL); MatGetLocalSize(A, &nLocal, NULL);
primme.n = n;                 // set global size
primme.nLocal = nLocal;  // rows stored locally
primme.matrixMatvec = PetscMatrixMatvec; // A*x
primme.matrix = A;            // pass A to PetscMatrixMatvec
MPI_Comm_size(PETSC_COMM_WORLD, &primme.numProcs);
MPI_Comm_rank(PETSC_COMM_WORLD, &primme.procID);
primme.globalSumReal = PetscGlobalSumDouble;   // Allreduce

// Set problem parameters
primme.numEvals = 4;     // number of eigenpairs wanted
primme.eps = 1e-12;      // ||r|| <= eps * ||A||
primme.target = primme_smallest; // seek for smallest
```

Parallel example with PETSc

```
...
// Allocate eigenvalues, residuals and local part of vectors
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double)),
 *evecs = malloc(primme.nLocal*primme.numEvals*sizeof(double));

dprimme(evals, evecs, rnorms, &primme); // do eigenstuff

// Print values, residual norms, and the vectors
for (int i=0; i<primme.numEvals; i++) {
  if (primme.procID == 0) // All has same evals and rnorms
    printf("Pair %d, value %g residual norm %g\n",
       i, evals[i], rnorms[i]);
  /* Every process print its local part */
  printf("proc %d vector[%d] = ", primme.procID, i);
  for (int j=0; j<primme.nLocal; j++)
    printf("%g ", evecs[j + primme.nLocal*i]); // = evecs(j,i)
  printf("\n")
}

primme_free(&primme);  // free workarrays in PRIMME
```

PETSc Matrix-vector product

```
void PetscMatrixMatvec(          // Do y = A * x
  void *x, PRIMME_INT *ldx,      // local input vectors
  void *y, PRIMME_INT *ldy,      // local output vectors
  int *numCols,                  // number of columns
  primme_params *primme,         // PRIMME configuration
  int *err                       // output flag error
) {
  Mat *A = (Mat*)primme->matrix;

  // Create PETSc vectors
  Vec xv, yv; MatCreateVecs(*A, &xv, &yv);
  for (int i=0; i<*numCols; i++) {
    /* Assign the content of the PETSc vectors */
    VecPlaceArray(xv, ((double*)x) + *ldx*i); // xv = x(:,i)
    VecPlaceArray(yv, ((double*)y) + *ldy*i); // yv = y(:,i)
    MatMult(*A, xv, yv); // yvec = A*xvec
    VecResetArray(xvec); VecResetArray(yvec);
  }
  VecDestroy(&xvec); VecDestroy(&yvec);
  *err = 0;   // All went ok
}
```

Representation of vectors/dense matrices

PRIMME passes vectors to user defined functions

- the pointers `void* x` to the first element
- the leading dimension `int *ldx`, where the next column starts

BLAS/LAPACK uses this simple interface also.

Example, output evecs has leading dimension `primme.nLocal`:

```
void MV(void *x, PRIMME_INT *ldx, ...) {
   double *dx = (double*)x; // Cast to the proper datatype
   dx[i + (*ldx)*j]         // = x(i,j)
```

Example, output evecs has leading dimension `primme.nLocal`:

```
dprimme(evals, evecs, rnorms, &primme); // do eigenstuff
evecs[i+primme.n*j] // evecs(i,j)
```

Matlab/Octave Preconditioner

```
A = diag(1:1000)+diag(ones(999,1),-1)+diag(ones(999,1),1);

opts.tol = 1e-12; % norm(r) <= norm(A)*tol
[x,d,r,stats] = primme_eigs(A, 10, 99.5, opts) % the 10 closest
to 99.5
stats =
    numMatvecs = 6302
    elapsedTime =  7.9344
    estimateMinEVal =  0.35134
    estimateMaxEVal =  1000.6
    estimateAnorm =  1000.6

Pfun = @(x)(diag(A)-99.5).\x; % Jacobi preconditioner
[x,d,r,stats] = primme_eigs(A, 10, 99.5, opts, [], Pfun);
stats =
    numMatvecs = 159
    elapsedTime =  0.22002
    estimateMinEVal =  2.8562
    estimateMaxEVal =  998.73
    estimateAnorm =  998.73
```

Adv parameters – Preconditioner

```c
#include "primme.h" // PRIMME header file

// Declare and initialize PRIMME configuration struct
primme_params primme;
primme_initialize(&primme);

// Set the function that implements A*x
primme.matrixMatvec = MatrixMatvec;

// Set problem parameters
primme.n = 100;          // set problem dimension
primme.numEvals = 4;     // number of eigenpairs wanted
primme.eps = 1e-12;      // ‖r‖ ≤ eps ∗ ‖A‖
primme.target = primme_smallest; // seek for smallest

// Set preconditioner
primme.applyPreconditioner = ApplyPreconditioner;

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double)),
```

User's Preconditioner

```
void ApplyPrecontioner(          // Do y = M * x
  void *x, PRIMME_INT *ldx,      // input vectors and leading dimen
  void *y, PRIMME_INT *ldy,      // output vectors and lead. dim.
  int *numCols,                  // number of columns
  primme_params *primme,         // PRIMME configuration
  int *err                       // output flag error
) {

   for (int i=0; i<*numCols; i++) { // for every column
      double *x_i = (double*)x + (*ldx)*i; // = x(0,i)
      double *y_i = (double*)y + (*ldx)*i; // = y(0,i)

      // Do y_i = M * x_i
      ...
   }

   *err = 0; // All went ok
}
```

Matlab/Octave Initial Space

```
A = diag(1:1000)+diag(ones(999,1),-1)+diag(ones(999,1),1);

opts.tol = 1e-12; % norm(r) <= norm(A)*tol
% Compute the 10 closest eigenvalues to 99.5
[x,d,r,stats] = primme_eigs(A, 10, 99.5, opts);
stats =
    numMatvecs = 6302
    elapsedTime =  7.9344
    ...

v0 = eye(1000); v0 = v0(:,95:104);
opts.v0 = v0;
[x,d,r,stats] = primme_eigs(A, 10, 99.5, opts);
stats =
    numMatvecs = 4042
    elapsedTime =  5.2255
    ...
```

Adv parameters – Initial Space

```
...
// Set problem parameters
primme.n = 100;             // set problem dimension
primme.numEvals = 4;        // number of eigenpairs wanted
primme.eps = 1e-12;         // ||r|| ≤ eps * ||A||
primme.target = primme_smallest; // seek for smallest

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double)),
      *evecs = malloc(primme.n*primme.numEvals*sizeof(double));

// Set initial guesses
primme.initSize = 3;
for (int i=0; i<primme.initSize; i++)
    memcpy(&evecs[i*primme.n], initGuess[i],
            sizeof(double)*primme.n);

dprimme(evals, evecs, rnorms, &primme); // do eigenstuff
...
```

Adv parameters – Orthogonal Constrains

```
...
// Set problem parameters
primme.n = 100;           // set problem dimension
primme.numEvals = 4;      // number of eigenpairs wanted
primme.eps = 1e-12;       // ‖r‖ ≤ eps ∗ ‖A‖
primme.target = primme_smallest; // seek for smallest

// Allocate output eigenvalues, vectors and residuals
double*evals = malloc(primme.numEvals*sizeof(double)),
      *rnorms = malloc(primme.numEvals*sizeof(double)),
      *evecs = malloc(primme.n*primme.numEvals*sizeof(double));
...

// Set orthogonal constrains, after initial guesses
primme.numOrthoConst = 5;
for (int i=0; i<primme.numOrthoConst; i++)
    memcpy(&evecs[primme.initSize*primme.n + i*primme.n],
            initGuess[i], sizeof(double)*primme.n);

dprimme(evals, evecs, rnorms, &primme); // do eigenstuff
...
```

Method

```
primme_set_method(method, primme);    // Set method
```

DYNAMIC	switches dynamically to the best method
DEFAULT_MIN_TIME	best method for low-cost matrix-vector product
DEFAULT_MIN_MATVECS	best method for heavy matvec/preconditioner
GD_Olsen_plusK	GD+k with approximate Olsen precond.
JDQMR	Our block JDQMR method (similar to JDCG)
JDQMR_ETol	Slight, but efficient JDQMR modification
LOBPCG_OrthoBasis	equiv. to GD(nev,3*nev)+nev
LOBPCG_OrthoBasis_Window	equiv. to GD(block,3*block)+block nev>block

Method

```
primme_set_method(method, primme);    // Set method
```

Research methods for test

Arnoldi	Arnoldi not implemented efficiently
GD	classical block Generalized Davidson
JD_Olsen_plusK	GD+k, exact Olsen (two precond per step)
GD_plusK	GD+k block GD with recurrence restarting
RQI	Rayleigh Quotient Iteration. Also INVIT, but for INVIT provide OPTS.targetShifts
JDQR	Original block, Jacobi Davidson
SUBSPACE_ITERATION	equiv. to GD(block,2*block)

Check out our site

- https://github.com/primme/primme

When to use PRIMME

- Large-dimension, sparse matrix problems
- Few eigenpairs wanted
  - Otherwise you may prefer, LAPACK, ScaLAPACK, FEAST
- Hermitian problem
  - For generalized, non-Hermitian, polynomial eigenvalue problems you may use ARPACK, SLEPc, Anasazi (Trilinos), BLOPEX, FEAST, MAGMA...
- Single and double precision
  - For quadruple precision you may use SLEPc
- Shared and distributed memory
  - For GPUs you may use SLEPc, Anasazi, MAGMA, SPRAL
- Support C/C++, Fortran, Matlab/Octave, Python (NumPy), R
  - There's a binding of ARPACK for Julia and Haskell

Other Packages with Hermitian Davidson-type Methods

Software	Methods	Lang.	MPI	GPU	Precon.	Interior	GHEP	SVD	Bindings
PRIMME	Davidson-type	C	●		●	●	●	●	F77, M, Py, R
SLEPc	Dav, Krylov	C	●	●	●	●	●	●	F77, M, Py
(P)ARPACK	IR Arnoldi	F77	●			●	○	●	M, Py, Ju, R
Anasazi	Several methods	C++	●	●	●	○	●		Py
SciPy	ARPACK, LOBPCG	Python			●		●		
BLOPEX	LOBPCG	C++	●	●	●		●		
FEAST	CIRR	F90	●		●	●	●		
MAGMA	LOBPCG	C++		●	●		●		

PRIMME Strong Points

- BSD License, compatible with open source and commercial projects
- Single goal: Hermitian Eigenproblem
- Small size project, less than 10,000 lines of code
- Dependencies limited to BLAS & LAPACK
- Dense vector/matrices passed as Fortran (pointer, leading dimension)
- Advanced methods rarely available in other packages:
  - restarting $+$k
  - advanced stopping criterion for Jacobi correction equation