

FEAST Eigenvalue Solver v4.0

User Guide



<http://www.feast-solver.org>

Eric Polizzi's Research Lab.
Department of Electrical and Computer Engineering,
Department of Mathematics and Statistics,
University of Massachusetts, Amherst

References

If you are using FEAST, please consider citing one or more publications below in your work.

Main reference

E. Polizzi, *Density-Matrix-Based Algorithms for Solving Eigenvalue Problems*,
Phys. Rev. B. Vol. 79, 115112 (2009)

Math analysis

P. Tang, E. Polizzi, *FEAST as a Subspace Iteration EigenSolver Accelerated by Approximate Spectral Projection*;
SIAM Journal on Matrix Analysis and Applications (SIMAX) 35(2), 354-390 - (2014)

Non-Hermitian solver

J. Kestyn, E. Polizzi, P. T. P. Tang, *FEAST Eigensolver for Non-Hermitian Problems*,
SIAM Journal on Scientific Computing (SISC), 38-5, ppS772-S799 (2016);

Hermitian using Zolotarev quadrature

S. Güttel, E. Polizzi, P. T. P. Tang, G. Viaud, *Optimized Quadrature Rules and Load Balancing for the FEAST Eigenvalue Solver*,
SIAM Journal on Scientific Computing (SISC), 37 (4), pp2100-2122 (2015).

Eigenvalue count using stochastic estimates

E. Di Napoli, E. Polizzi, Y. Saad, *Efficient Estimation of Eigenvalue Counts in an Interval*,
Numerical Linear Algebra with Applications, V23, I4, pp674-692,(2016).

Polynomial Non-linear eigenvalue problem – Residual Inverse Iterations

B. Gavin, A. Miedlar, E. Polizzi, *FEAST Eigensolver for Nonlinear Eigenvalue Problems*
Journal of Computational Science, V. 27, 107, (2018)

IFEAST

B. Gavin, E. Polizzi, *Krylov eigenvalue strategy using the FEAST algorithm with inexact system solves*
Numerical Linear Algebra with Applications, vol 25, number 5, 20 pages (2018).

PFEAST

J. Kestyn, V. Kalantzis, E. Polizzi, Y. Saad, *PFEAST: A High Performance Sparse Eigenvalue Solver Using Distributed-Memory Linear Solvers*
Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM/IEEE Supercomputing Conference (SCÅŻ16), pp 16:1-16:12, (2016).

Contact

If you have any questions or feedback regarding FEAST, please send an-email to feastsolver@gmail.com.

FEAST algorithm and software team, collaborators and contributors

Code Developer/Contributors

Eric Polizzi (Lead)
James Kestyn (Non-Hermitian, PFEAST),
Brendan Gavin (Non-linear, IFEAST),
Braegan Spring (SPIKE banded solver, Hybrid solver–in progress),
Stefan Güttel (Zolotarev quadrature),
Julien Brenneck (GUI configurator, Quaternion–in progress).

Collaborators : Peter Tang, Yousef Saad, Agnieszka Miedlar, Edoardo Di Napoli, Ahmed Sameh

Acknowledgments

This work has been supported by Intel Corporation and by the National Science Foundation (NSF) under Grants #CCF-1510010, #SI2-SSE-1739423, and #CCF-1813480.

A. TABLE OF CONTENTS

A.	Table of Contents	3
1	Background	4
1.1	The FEAST Algorithm	4
1.2	The FEAST Solver	4
2	Installation and Setup: A Step by Step Procedure	6
2.1	Installation	6
2.2	Compilation	6
2.3	Linking FEAST	7
2.4	HelloWorld Example (F90, C, MPI-F90, MPI-C)	8
3	FEAST Interfaces	13
3.1	At a Glance	13
3.2	FEAST Hermitian	17
3.3	FEAST Non-Hermitian	21
3.4	FEAST Polynomial (quadratic, cubic, quartic, etc.)	24
3.5	IFEAST (FEAST w/o Factorization)	27
3.6	PFEAST and PIFEAST (MPI-solver)	29
4	Complement	33
4.1	Matrix storage	33
4.2	Search contour	33
4.3	Contour Customization	35
4.4	FEAST utility sparse drivers	37

1 Background

“The solution of the algebraic eigenvalue problem has for long had a particular fascination for me because it illustrates so well the difference between what might be termed classical mathematics and practical numerical analysis. The eigenvalue problem has a deceptively simple formulation and the background theory has been known for many years; yet the determination of accurate solutions presents a wide variety of challenging problems.”

J. H. Wilkinson- The Algebraic Eigenvalue Problem- 1965

The eigenvalue problem is ubiquitous in science and engineering applications. It can be encountered under different forms: Hermitian or non-Hermitian, and linear or non-linear. The eigenvalue problem has led to many challenging numerical questions and a central problem: how can we compute eigenvalues and eigenvectors in an efficient manner and how accurate are they?

The FEAST library package represents an unified framework for solving various family of eigenvalue problems and addressing the issues of numerical accuracy, robustness, performance and parallel scalability. Its originality lies with a new transformative numerical approach to the traditional eigenvalue algorithm design - the FEAST algorithm.

1.1 The FEAST Algorithm

The FEAST algorithm is a general purpose eigenvalue solver which takes its inspiration from the density-matrix representation and contour integration technique in quantum mechanics¹. The algorithm gathers key elements from complex analysis, numerical linear algebra and approximation theory, to construct an optimal subspace iteration technique making use of approximate spectral projectors². FEAST can be applied for solving both standard and generalized forms of the Hermitian or non-Hermitian problems (linear or non-linear), and it belongs to the family of contour integration eigensolvers. Once a given search interval is selected, FEAST’s main computational task consists of a numerical quadrature computation that involves solving independent linear systems along a complex contour, each with multiple right hand sides. A Rayleigh-Ritz procedure is then used to generate a reduced dense eigenvalue problem orders of magnitude smaller than the original one (the size of this reduced problem is of the order of the number of eigenpairs inside the search interval/contour). FEAST offers a set of appealing features: (i) Remarkable robustness with well-defined convergence rate; (ii) All multiplicities naturally captured; (iii) No explicit orthogonalization procedure on long vectors required in practice; (iv) Reusable subspace as initial guess when solving a series of eigenvalue problems; and (v) Efficient use of both blocked BLAS-3 operations and parallel resources for solving the linear systems with multiple right hand sides. FEAST can exploit a key strength of modern computer architectures, namely, multiple levels of parallelism. Natural parallelism appears at three different levels (L1, L2 or L3): (L1) search contours can be treated separately (no overlap), (L2) linear systems can be solved independently across the quadrature nodes of the complex contour, and (L3) each complex linear system with multiple right-hand-sides can be solved in parallel. Parallel resources can be placed at all three levels simultaneously in order to achieve scalability and optimal use of the computing platform. Within a parallel environment, the main numerical task can be reduced to the solution of a single linear system using direct or iterative parallel solvers.

1.2 The FEAST Solver

FEAST release dates with main features are listed below:

v1.0 (Sep. 2009): Hermitian problem (standard/generalized)

v2.0 (Mar. 2012): SMP+MPI+RCI interfaces

v2.1 (Feb. 2013): **Adoption by Intel-MKL**

v3.0 (Jun. 2015): Support for non-Hermitian

v4.0 (Feb. 2020): Residual inverse iterations - mixed precision - IFEAST (FEAST w/o factorization) - PFEAST (3 MPI levels) - Support for non-linear (polynomial) - Support for extreme eigenvalues (lowest/largest)

¹E. Polizzi, Phys. Rev. B. Vol. 79, 115112 (2009)

²P. Tang, E. Polizzi, SIMAX 35(2), 354–390 - (2014)

The FEAST package v2.1 has been featured as Intel-MKL’s principal HPC eigensolver since 2013³. The current version of the FEAST package (v4.0) released in Feb. 2020 represents a significant upgrade since the entire FEAST package has been re-coded to perform residual inverse iterations⁴. As a result, v4.0 is in average much faster than v2.1 or v3.0 ($\times 3 - 4$ using new default optimization parameters), and it became possible to implement new important features such as IFEAST (using Inexact Iterative solver) and Non-linear polynomial FEAST. Furthermore, v4.0 features PFEAST with its 3-MPI levels of parallelism.

FEAST is a comprehensive numerical library offering both simplicity and flexibility, and packaged around a “black-box” interface as depicted in Figure 1 for the Hermitian problem.

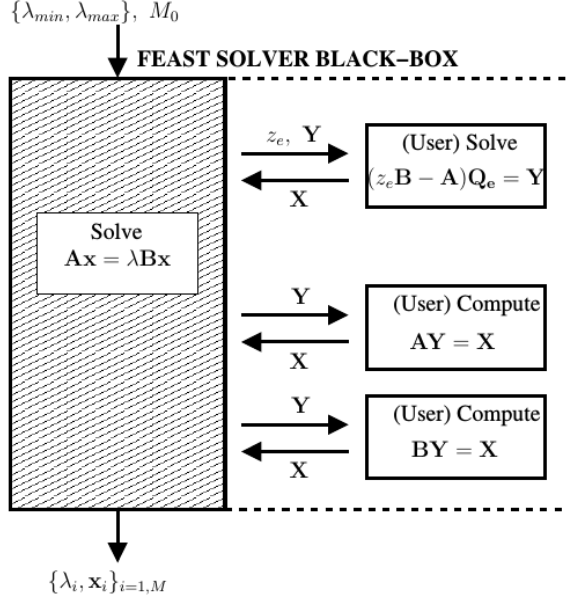


Figure 1: “Black-box” interface for the Hermitian problem. In normal mode, FEAST requires a search interval and a search subspace size M_0 . It includes features such as reverse communication interfaces (RCI) that are matrix format independent, and linear system solver independent, as well as ready to use driver interfaces for dense, banded and sparse systems. For the driver interfaces the “black-box” region extends then to the right dashed box, and only the system matrices are required as inputs from the users. The RCI interfaces represent the kernel of FEAST which can be customized by the users to allow maximum flexibility for their specific applications. Users have then the possibility to integrate their own linear system solvers (direct or iterative - with or without preconditioner) and handle their own matrix-vector multiplication procedure.

The current main features of the FEAST v4.0 package include:

- Standard or Generalized Hermitian and non-Hermitian eigenvalue problems (left/right eigenvectors and bi-orthonormal basis);
- Polynomial eigenvalue problems such as quadratic, cubic, quartic, etc. (left/right eigenvectors);
- Finding eigenpair within a search contour (normal mode); Finding extreme eigenvalues (lowest/largest) for sparse Hermitian systems;
- Real/Complex arithmetic and mixed precision (single precision operations leading to double precision final results);
- Two libraries: **SMP version** (one node), and **MPI version** (multi-nodes);
- Reverse communication interfaces (RCI).
- Driver interfaces for dense (using LAPACK), banded (using SPIKE), and sparse-CSR formats (using MKL-PARDISO);
- IFEAST- FEAST w/o factorization for sparse-CSR drivers (using BiCGStab);
- PFEAST- FEAST using 3 levels of MPI parallelism for HPC (MPI solvers includes MKL-Cluster-PARDISO and PBiCGStab); Sparse and RCI interfaces compatible with local row-distributed data.
- A set of flexible and useful practical options (quadrature rules, contour shapes, stopping criteria, initial guess, fast stochastic estimates for eigenvalue counts, etc.)
- Portability: FEAST routines can be called from any Fortran or C codes.
- FEAST interfaces only require (any optimized) LAPACK and BLAS packages.
- Large number of driver examples, utility routines, and documentation.

³<https://software.intel.com/en-us/articles/introduction-to-the-intel-mkl-extended-eigensolver>

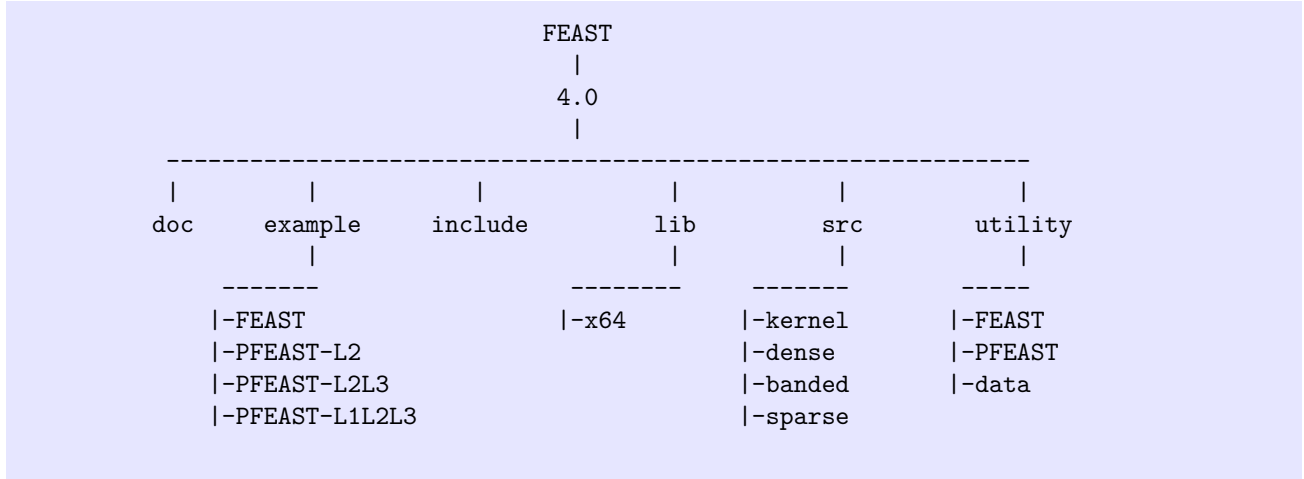
⁴B. Gavin, A. Miedlar, E. Polizzi, Journal of Computational Science, V. 27, 107, (2018); B. Gavin, E. Polizzi, in preparation (2020).

2 Installation and Setup: A Step by Step Procedure

2.1 Installation

Please follow the following steps (here for Linux/Unix systems):

1. Download the latest FEAST package version `feast_4.0.tgz` in <http://www.feast-solver.org>
2. Put the file in your preferred directory such as `$HOME` directory or (for example) `/opt/` directory if you have ROOT privilege.
3. Execute: `tar -xvzf feast_4.0.tgz` to create the following FEAST tree directory.



4. If `<FEAST directory>` denotes the package's main directory after installation, for example

`~/home/FEAST/4.0` or `/opt/FEAST/4.0`,

it is **not** mandatory but recommended to define the Shell variable `$FEASTROOT`, e.g.

```
export FEASTROOT=<FEAST directory>
```

or

```
set FEASTROOT=<FEAST directory>
```

respectively for the BASH or CSH shells. One of this command can be placed in the appropriate shell startup file in `$HOME` (i.e. `.bashrc` or `.cshrc`).

2.2 Compilation

Go to the directory `$FEASTROOT/src` and execute `make` to see all available options. The same FEAST source code is used for compiling FEAST-SMP (`libfeast`) and/or FEAST-MPI (`libpfeast`). The command is:

```
make ARCH=<arch> F90=<f90> MPI=<mpi> MKL=<mkl> {feast, pfeast}
```

where you can select the following options:

- `<arch>` : it is the name of the directory `$FEASTROOT/lib/<arch>` where the FEAST libraries will be located once compiled (you can use the name of your architecture). Default is **x64**
- `<f90>` : it is your own Fortran90 compiler (possible choices: `ifort`, `gfortran`, `pgf90`). Default is **ifort**
- `<mpi>` : (mandatory for compiling `libpfeast` only) it is your MPI library (possible choices: `impi`, `mpich`, `openmpi`). Defaults to **impi** (intel MPI)
- `<mkl>` : it enables Intel-MKL math library instructions (possible choices: `yes`, `no`). Default is **yes**.
 - if `<mkl>=yes`, at the linking stage, FEAST will have to be linked with Intel MKL.

- if `<mkl>=no`, at the linking stage, FEAST can be linked with any BLAS/LAPACK. Not using MKL will impact the behavior and performance of the FEAST sparse Driver interfaces: (i) it would not be possible to use MKL-PARDISO and cluster-MKL-PARDISO so the FEAST sparse interfaces will instead be calling IFEAST (BiCGStab); (ii) in-built sparse mat-vec routines (used by the IFEAST sparse interfaces) will be slower.

For example, if the above default options look fine with you, just use:

- **make feast**
to compile the FEAST-SMP library and create the file `libfeast.a` in `$FEASTROOT/lib/<arch>`
- **make pfeast**
to compile the FEAST-MPI library and create the file `libpfeast.a` in `$FEASTROOT/lib/<arch>`

Congratulations, FEAST is now successfully installed and compiled on your computer !!

2.3 Linking FEAST

In order to use the FEAST library for your main application code, you will then need to add the following instructions in your `Makefile`:

- *for the LIBRARY PATH:* `-L$FEASTROOT/lib/<arch>`
- *for the LIBRARY LINKS using FEAST-SMP:* `-lfeast`
using FEAST-MPI: `-lpfeast`
- *for the INCLUDE PATH (mandatory only for C codes):* `-I$(FEASTROOT)/include`

Remarks

- 1- If FEAST was compiled with the option `MKL=yes`, you must also link with the MKL libraries. Otherwise, you can link with any BLAS, LAPACK libraries.
- 2- If you use the FEAST banded interfaces, you need to install the SPIKE solver www.spike-solver.org. SPIKE must be compiled using the same Fortran compiler used for compiling FEAST.
- 3- For C codes, the user must include the following instructions in the header:

```
1 #include "feast.h"
2 #include "feast_dense.h" //for feast dense interfaces
3 #include "feast_banded.h" //for feast banded interfaces
4 #include "feast_sparse.h" //for feast sparse interfaces
```

Using PFEAST (MPI sparse linear system solver), you must use instead:

```
1 #include "pfeast.h"
2 #include "pfeast_sparse.h"
```

2.4 HelloWorld Example (F90, C, MPI-F90, MPI-C)

This example solves a 4-by-4 real symmetric standard eigenvalue system $\mathbf{Ax} = \lambda\mathbf{x}$ (using dense format) where

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}. \quad (1)$$

The four eigenvalue solutions are $\lambda = \{0, 2, 4, 4\}$. Let us suppose that one can specify a search interval (such as $[3, 5]$), a single call to the `dfeast_syev` subroutine should return the solutions associated with $\{4, 4\}$. The FEAST parameters need first to be set to their default values by a call to the `feastinit` subroutine. Below, we provide examples written in F90, C, MPI-F90, and MPI-C.

F90

A Fortran90 source code of `helloworld.f90` is provided below:

```

1 program helloworld
2   implicit none
3   !! 4x4 eigenvalue system
4   integer,parameter :: N=4
5   character(len=1) :: UPLO='F' ! 'L' or 'U' also fine
6   double precision,dimension(N*N) :: A=(/ 2.0d0, -1.0d0,-1.0d0, 0.0d0,&
7                                             &-1.0d0, 3.0d0,-1.0d0,-1.0d0,&
8                                             &-1.0d0, -1.0d0, 3.0d0,-1.0d0,&
9                                             & 0.0d0, -1.0d0,-1.0d0, 2.0d0/)
10  !! input parameters for FEAST
11  integer,dimension(64) :: fpm
12  integer :: M0=3 ! search subspace dimension
13  double precision :: Emin=3.0d0, Emax=5.0d0 ! search interval
14  !! output variables for FEAST
15  double precision,dimension(:),allocatable :: E, res
16  double precision,dimension(:,:),allocatable :: X
17  double precision :: epsout
18  integer :: loop,info,M,i
19
20  !!! Allocate memory for eigenvalues,eigenvectors,residual
21  allocate (E(M0),X(N,M0),res(M0))
22
23  !!!!!!!!!! FEAST
24  call feastinit(fpm)
25  fpm(1)=1 !! change from default value (print info on screen)
26  call dfeast_syev(UPLO,N,A,N,fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)
27
28  !!!!!!!!!! REPORT
29  if (info==0) then
30    print *, 'Solutions (Eigenvalues/Eigenvectors/Residuals)'
31    do i=1,M
32      print *, 'E=',E(i), 'X=',X(:,i), 'Res=',res(i)
33      print *, ''
34    enddo
35  endif
36
37 end program helloworld

```

To create the executable, compile and link the source program with the `feast` library, one can use (for example):

- `ifort -o helloworld helloworld.f90 -L$FEASTROOT/lib/<arch> -lfeast -mkl`

 if FEAST was compiled with `ifort` and MKL flag was set to 'yes'.

- ```
gfortran -o helloworld helloworld.f90 -L$FEASTROOT>/lib/<arch> -lfeast
-Wl,-start-group -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -Wl,-end-group -lgomp
-lpthread -lm -ldl
```

if FEAST was compiled with `gfortran` and MKL flag was set to 'yes'.

### Remarks:

1-Many other options are possible. For example, you can link with your own BLAS/LAPACK libraries, you can compile FEAST with `ifort` and compile the helloworld example using `gfortran` via extra flag options, or vice-versa, etc.

2- FEAST is using a linear solver that can be threaded (the LAPACK dense solver is used for the helloworld example). Using MKL, you can control the number of threads by setting up the value of `MKL_NUM_THREADS`. For example, in BASH shell:

```
export MKL_NUM_THREADS=<omp>
```

where `< omp >` represents the number of threads (cores).

A run of the resulting executable looks like:

```
./helloworld
```

and the output of the run should be:

```

***** FEAST v4.0 BEGIN *****

Routine DFEAST_SYEV
Solving AX=eX with A real symmetric
List of input parameters fpm(1:64)-- if different from default
 fpm(1)= 1

FEAST data
Emin
Emax
#Contour nodes
Quadrature rule
Ellipse ratio y/x
System solver
FEAST uses MKL?
Fact. stored?
Initial Guess
Size system
Size subspace

FEAST runs
#It | #Eig | Trace | Error-Trace | Max-Residual
0 | 2 | 7.9999999999999964E+00 | 1.0000000000000000E+00 | 2.2017475048923482E-08
1 | 2 | 7.9999999999999982E+00 | 3.5527136788005011E-16 | 1.0614112804180586E-15

==>FEAST has successfully converged with Residual tolerance <1E-12
FEAST outside it. 1
Eigenvalue found 2 from 3.9999999999999987E+00 to 4.0000000000000000E+00

FEAST-RCI timing
Fact. cases(10,20)
```

```

Solve cases(11,12)	0.0197
A*x cases(30,31)	0.0000
B*x cases(40,41)	0.0000
Misc. time	0.0004
Total time (s)	0.0210

***** FEAST- END*****

Solutions (Eigenvalues/Eigenvectors/Residuals)
E= 4.000000000000000 X= -0.409757405384329 4.544205370554897E-003
 0.814970605398106 -0.409757405384329 Res= 1.061411280418059E-015

E= 4.000000000000000 X= -0.286529001556041 0.866013481533371
-0.292955478421287 -0.286529001556041 Res= 6.481268641478987E-016
%\end{verbatim}

```

## C

Similarly to the F90 example, the corresponding C source code for the helloworld example (`helloworld.c`) is provided below:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "feast.h"
4 #include "feast_dense.h"
5 int main() {
6 /* 4x4 eigenvalue system */
7 int N=4;
8 char UPLO='F'; // 'L' and 'U' also fine
9 double A[16]={2.0, -1.0, -1.0, 0.0, -1.0, 3.0, -1.0, -1.0, -1.0, -1.0, 3.0, -1.0, 0.0, -1.0, -1.0, 2.0};
10 /* input parameters for FEAST */
11 int fpm[64];
12 int M0=3; //search subspace dimension
13 double Emin=3.0, Emax=5.0; // search interval
14 /* output variables for FEAST */
15 double *E, *res, *X;
16 double epsout;
17 int loop, info, M, i;
18
19 /* Allocate memory for eigenvalues.eigenvectors/residual */
20 E=calloc(M0, sizeof(double)); //eigenvalues
21 res=calloc(M0, sizeof(double)); //eigenvectors
22 X=calloc(N*M0, sizeof(double)); //residual
23
24 /* !!!!!!!!! FEAST !!!!!!!!! */
25 feastinit(fpm);
26 fpm[0]=1; /*change from default value */
27 dfeast_syev(&UPLO, &N, A, &LDA, fpm, &epsout, &loop, &Emin, &Emax, &M0, E, X, &M, res, &info);
28
29 /* !!!!!!!!! REPORT !!!!!!!!! */
30 if (info==0) {
31 printf("Solutions_\n");
32 for (i=0; i<=M-1; i=i+1){
33 printf("E=%15eX=%15eX=%15eX=%15eRes=%15e\n",
34 *(E+i), *(X+i*N), *(X+1+i*N), *(X+2+i*N), *(X+3+i*N), *(res+i));
35 }
36 }
37 return 0;
38 }

```

To create the executable, compile and link the source program with the `feast` library, one can use (for example):

- `icc -qopenmp -I$FEASTROOT/include -o helloworld helloworld.c -L$FEASTROOT/lib/x64 -lfeast -mkl -lirc -lifcore -lifcoremt`

if FEAST was compiled with ifort and MKL flag was set to 'yes'.

- `gcc -fopenmp -I$FEASTROOT/include -o helloworld helloworld.c -L$FEASTROOT/lib/x64 -lfeast -Wl,-start-group -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -Wl,-end-group -lgomp -lpthread -lm -ldl -lgfortran`

if FEAST was compiled with gfortran and MKL flag was set to 'yes'.

## MPI-F90

FEAST can be straightforwardly parallelized using MPI at level L2 (where the FEAST inner linear systems are automatically distributed among MPI processes). As a reminder level L1 corresponds to the parallelization of the search interval, and level L3 corresponds to the parallelization of each linear system using row-data-distribution and MPI solver. Examples using L1-L2-L3 (MPI-MPI-MPI) are discussed in the PFEAST Section 3.6).

You can create the file `phelloworld.f90` by cc-paste the content of `helloworld.f90` and by just adding a few lines at the beginning and at the end of the program, i.e.

```

1 !!!! add at the very beginning
2 include 'mpif.h'
3
4 !!!! add after variable declarations
5 integer :: code
6 call MPI_INIT(code)
7
8
9 !!!! add at the end
10 call MPI_FINALIZE(code)
```

Your program must be compiled using the same MPI implementation used to compile the FEAST-MPI library. Once compiled, your source program must now be linked with the `pfeast` library. You can use (for example):

- `mpiifort -o phelloworld phelloworld.f90 -L$FEASTROOT/lib/<arch> -lpfeast -mkl`

if FEAST was compiled with ifort, MKL flag was set to 'yes', and MPI was chosen to be 'impi' (intel mpi).

- `mpif90.mpich -fc=gfortran phelloworld.f90 -o phelloworld -L$FEASTROOT>/lib/<arch> -lpfeast -Wl,-start-group -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -Wl,-end-group -lgomp -lpthread -lm -ldl -lifcore`

if FEAST was compiled with gfortran, MKL flag was set to 'yes', and MPI was chosen to be 'mpich'.

A run of the resulting executable looks like:

```
mpirun -ppn 1 -n <np> ./phelloworld
```

where `< np >` represents the number of MPI processes (here we also choose 1 MPI process per compute node with the option `-ppn 1`)

## Remarks:

1-Scalability performances will be optimal here when the number of MPI processes `<np>` reaches the number of contour points (provided by the default value `fpm(2)=8` in this example).

2-Since FEAST is also threaded, make sure that your number of selected threads `<omp>` times the number of mpi processes `<np>` for a given compute node (i.e. `<omp>*<np>`) does not exceed the number of your physical cores.

## MPI-C

Similarly to the MPI-F90 example, You can create the file `phelloworld.c` by cc-paste the content of `helloworld.c` and by just adding a few lines at the beginning and at the end of the program, i.e.

```
1 !!!! add at the very beginning
2 #include <mpi.h>
3
4 !!!! change the argument list of the main function
5 int main(int argc, char **argv)
6
7 !!!! add after variable declarations
8 MPI_Init(&argc,&argv);
9
10 !!!! add at the end
11 MPI_Finalize();
```

Your program must be compiled using the same MPI implementation used to compile the FEAST-MPI library. Once compiled, your source program must now be linked with the `pfeast` library. You can use (for example):

- ```
mpiicc -qopenmp -I$FEASTROOT/include -o phelloworld phelloworld.c
-L$FEASTROOT/lib/x64 -lpfeast -mkl -lirc -lifcore -lifcoremt
```

if FEAST was compiled with `ifort`, MKL flag was set to 'yes', and MPI was chosen to be 'impi' (intel mpi).
- ```
mpicc -cc=gcc -fopenmp -I$FEASTROOT/include -o phelloworld phelloworld.c
-L$FEASTROOT/lib/x64 -lpfeast -Wl,-start-group -lmkl_gf_lp64 -lmkl_gnu_thread
-lmkl_core -Wl,-end-group -lgomp -lpthread -lm -ldl -lgfortran
```

if FEAST was compiled with `gfortran`, MKL flag was set to 'yes', and MPI was chosen to be 'mpich'.

### 3 FEAST Interfaces

#### 3.1 At a Glance

There are the two different types of interfaces available in the FEAST library:

##### Driver interfaces

- Optimal drivers acting on commonly used matrix data storage (dense, banded, sparse-CSR, row-distributed CSR).
- Use predefined linear system solvers: LAPACK (for dense), SPIKE (for banded), MKL-PARDISO (for sparse-CSR and FEAST), BiCGStab (for sparse-CSR and IFEAST), MKL-CLUSTER-PARDISO (for row-distributed CSR and PFEAST), PBiCGStab (for row-distributed CSR and PIFEAST).

##### Reverse communication interfaces (RCI)

- Constitute the kernel of FEAST, independent of the matrix data formats, so users can easily customize FEAST with their own explicit or implicit data format (or row-distributed data format).
- Mat-vec routines and direct/iterative linear system solvers must also be provided by the users.

Here is the complete list of all FEAST v4.0 interfaces (186 in total).

| Properties                                     | RCI interfaces             | Dense/ <a href="#">Banded</a> interfaces            | Sparse interfaces                    |
|------------------------------------------------|----------------------------|-----------------------------------------------------|--------------------------------------|
| <b>Linear</b> $AX = BX\Lambda$                 |                            |                                                     |                                      |
| Real Sym. $A = A^T, B$ spd                     | dfeast_src{ <i>x</i> }     | dfeast_{sy, <a href="#">sb</a> }{ev,gv}{ <i>x</i> } | {p}d{i}feast_scsr{ev,gv}{ <i>x</i> } |
| Complex Herm. $A = A^H, B$ hpd                 | zfeast_hrci{ <i>x</i> }    | zfeast_{he, <a href="#">hb</a> }{ev,gv}{ <i>x</i> } | {p}z{i}feast_hcsr{ev,gv}{ <i>x</i> } |
| Complex Sym. $A = A^T, B = B^T$                | zfeast_src{ <i>x</i> }     | zfeast_{sy, <a href="#">sb</a> }{ev,gv}{ <i>x</i> } | {p}z{i}feast_scsr{ev,gv}{ <i>x</i> } |
| Real General                                   | zfeast_grci{ <i>x</i> }    | dfeast_{ge, <a href="#">gb</a> }{ev,gv}{ <i>x</i> } | {p}d{i}feast_gcsr{ev,gv}{ <i>x</i> } |
| Complex General                                | zfeast_grci{ <i>x</i> }    | zfeast_{ge, <a href="#">gb</a> }{ev,gv}{ <i>x</i> } | {p}z{i}feast_gcsr{ev,gv}{ <i>x</i> } |
| <b>Polynomial</b> $\sum_i A_i X \Lambda^i = 0$ |                            |                                                     |                                      |
| Real Sym. $A_i = A_i^T$                        | zfeast_srcipev{ <i>x</i> } | dfeast_syev{ <i>x</i> }                             | {p}d{i}feast_scsrpev{ <i>x</i> }     |
| Complex Herm. $A_i = A_i^H$                    | zfeast_grcipev{ <i>x</i> } | zfeast_heev{ <i>x</i> }                             | {p}z{i}feast_hcsrpev{ <i>x</i> }     |
| Complex Sym. $A_i = A_i^T$                     | zfeast_srcipev{ <i>x</i> } | zfeast_syev{ <i>x</i> }                             | {p}z{i}feast_scsrpev{ <i>x</i> }     |
| Real General                                   | zfeast_grcipev{ <i>x</i> } | dfeast_geev{ <i>x</i> }                             | {p}d{i}feast_gcsrpev{ <i>x</i> }     |
| Complex General.                               | zfeast_grcipev{ <i>x</i> } | zfeast_geev{ <i>x</i> }                             | {p}z{i}feast_gcsrpev{ <i>x</i> }     |

where

- **dfeast** and **zfeast** stand for real double precision and complex double precision, respectively.
- {**ev,gv**} stands for either standard (i.e. B=I) or generalized eigenvalue problems.
- {**x**} is optional - stands for the expert FEAST version which enables customized quadrature nodes/weights.
- {**i**} is optional - stands for the IFEAST version of the sparse interfaces using inexact iterative solver.
- {**p**} is optional - stands for the PFEAST version of the sparse interfaces using distributed MPI solvers (direct or iterative).

In addition, all the input parameters for the FEAST algorithm are contained into an integer array of size 64 named here **fpm**. Prior calling the FEAST interfaces, this array needs to be initialized. There exists two FEAST initialization routines:

```

!!!! initialization for FEAST
feastinit(fpm)
!!!! initialization for PFEAST (needed if the sparse interfaces use a MPI solver)
pfeastinit(fpm,L1_comm_world,nL3)

```

All input FEAST parameters are then set to their default values. The detailed list of the **fpm** parameters is given in Table 1. Users can modify their values accordingly before calling the FEAST interfaces.

| <code>fpm(i)</code> F90<br><code>fpm[i-1]</code> C | Description                                                                                                                                                                                                                                        | Default values                                                                                |
|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| Runtime and algorithm options                      |                                                                                                                                                                                                                                                    |                                                                                               |
| <code>i=1</code>                                   | Print runtime comments<br>0: Off    1: On screen<br>n<0: Write/Append comments in the file <code>feast&lt;n .log</code>                                                                                                                            | 0                                                                                             |
| <code>i=2</code>                                   | #contour points for Hermitian FEAST (half-contour)<br>if <code>fpm(16)=0,2</code> , values permitted (1 to 20, 24, 32, 40, 48, 56)<br>if <code>fpm(16)=1</code> , all positive values permitted                                                    | 8 using FEAST<br>4 using IFEAST<br>3 using Stochastic <code>fpm(14)=2</code>                  |
| <code>i=3</code>                                   | Stopping convergence criteria in double precision (0 to 16)<br>$\epsilon = 10^{-fpm(3)}$                                                                                                                                                           | 12                                                                                            |
| <code>i=4</code>                                   | Maximum number of FEAST refinement loop allowed ( $\geq 0$ )                                                                                                                                                                                       | 20 using FEAST<br>50 using IFEAST                                                             |
| <code>i=5</code>                                   | Provide initial guess subspace (0: No; 1: Yes)                                                                                                                                                                                                     | 0                                                                                             |
| <code>i=6</code>                                   | Convergence criteria (for solutions in the search contour)<br>0: Using relative error on the trace <code>epsout</code> i.e. <code>epsout</code> < $\epsilon$<br>1: Using relative residual <code>res</code> i.e. $\max_i \text{res}(i) < \epsilon$ | 1                                                                                             |
| <code>i=8</code>                                   | #contour points for non-Herm./poly. FEAST (full-contour)<br>if <code>fpm(16)=0</code> , values permitted (2 to 40, 48, 64, 80, 96, 112)<br>if <code>fpm(16)=1</code> , all values permitted ( $>2$ )                                               | 16 using FEAST<br>8 using IFEAST<br>6 using Stochastic <code>fpm(14)=2</code>                 |
| <code>i=9</code>                                   | L2 communicator for PFEAST                                                                                                                                                                                                                         | set by call to <code>pfeastinit</code>                                                        |
| <code>i=10</code>                                  | Store linear system factorizations (0: No; 1: Yes).                                                                                                                                                                                                | 1 using all Driver interfaces<br>0 using RCI interfaces                                       |
| <code>i=14</code>                                  | 0: FEAST normal execution<br>1: Return subspace Q after 1 contour<br>2: Stochastic estimate of #eigenvalues inside search contour                                                                                                                  | 0                                                                                             |
| <code>i=15</code>                                  | #Contours for non-Hermitian or polynomial FEAST.<br>0: two-sided contour (compute right/left eigenvectors)<br>1: one-sided contour (compute only right eigenvectors)<br>2: one sided contour (left=right* eigenvectors)                            | 0 using non-sym. drivers<br>1 using Stochastic <code>fpm(14)=2</code><br>2 using sym. drivers |
| <code>i=16</code>                                  | Integration type (0: Gauss 1: Trapezoidal; 2: Zolotarev)<br>Remark: option 2 only for Hermitian                                                                                                                                                    | 0 for Hermitian FEAST<br>1 for non-Herm./poly. FEAST<br>1 for IFEAST                          |
| <code>i=18</code>                                  | Ellipse contour ratio 'vertical axis'/'horizontal axis' ( $\geq 0$ )<br><code>fpm(18)/100</code> = ratio                                                                                                                                           | 30 for Hermitian FEAST<br>100 for non-Herm./poly. FEAST<br>100 for IFEAST                     |
| <code>i=19</code>                                  | Ellipse rotation angle in degree from vertical axis [-180:180]<br>Remark: only for non-Hermitian                                                                                                                                                   | 0                                                                                             |
| <code>i=49</code>                                  | L3 communicator for PFEAST                                                                                                                                                                                                                         | set by call to <code>pfeastinit</code>                                                        |
| Driver interface options                           |                                                                                                                                                                                                                                                    |                                                                                               |
| <code>i=40</code>                                  | Search interval option for sparse Hermitian drivers<br>0: search interval provided by user<br>-1: search M0/2 lowest eigenvalues- return search interval<br>1: search M0/2 largest eigenvalues- return search interval                             | 0                                                                                             |
| <code>i=41</code>                                  | Matrix scaling for sparse drivers (0: No; 1: Yes).                                                                                                                                                                                                 | 1                                                                                             |
| <code>i=42</code>                                  | Mixed Precision for all drivers<br>0: use double precision linear system solvers<br>1: use single precision linear system solvers                                                                                                                  | 1                                                                                             |
| <code>i=43</code>                                  | Automatic switch from FEAST to IFEAST drivers<br>(0: <code>feast</code> ,1: <code>ifeast</code> )                                                                                                                                                  | 0                                                                                             |
| <code>i=45</code>                                  | Accuracy of BiCGStab in IFEAST $\mu = 10^{-fpm(45)}$                                                                                                                                                                                               | 1                                                                                             |
| <code>i=46</code>                                  | Maximum #iterations for BiCGStab in IFEAST                                                                                                                                                                                                         | 40                                                                                            |
| <code>i=60</code>                                  | <b>Output:</b> returns the total number of BicGstab iterations                                                                                                                                                                                     | N/A                                                                                           |
| All Others                                         | Reserved values and/or Undocumented options                                                                                                                                                                                                        | N/A                                                                                           |

Table 1: List FEAST parameters with default input values.

**Remark:** Using the C language, the components of the `fpm` array starts at 0 and stops at 63. Therefore, the components `fpm[j]` in C ( $j=0-63$ ) must correspond to the components `fpm(i)` in Fortran ( $i=1-64$ ) specified above (i.e. `fpm[i-1]=fpm(i)`).

Errors and warnings encountered during a run of the FEAST package are stored in an integer variable, **info**. If the value of the output **info** parameter is different than “0”, either an error or warning was encountered. The possible return values for the **info** parameter along with the error code descriptions, are given in Table 2.

| <b>info</b>  | Classification  | Description                                                                               |
|--------------|-----------------|-------------------------------------------------------------------------------------------|
| 202          | Error           | Problem with size of the system <b>N</b>                                                  |
| 201          | Error           | Problem with size of subspace <b>M0</b>                                                   |
| 200          | Error           | Problem with Emin, Emax or Emid, <b>r</b>                                                 |
| $(100 + i)$  | Error           | Problem with $i^{th}$ value of the input FEAST parameter (i.e <b>fpm(i)</b> )             |
| 7            | Warning         | The search for extreme eigenvalues has failed, search contour must be set by user         |
| 6            | Warning         | FEAST converges but subspace is not bi-orthonormal                                        |
| 5            | Warning         | Only stochastic estimation of #eigenvalues returned <b>fpm(14)=2</b>                      |
| 4            | Warning         | Only the subspace has been returned using <b>fpm(14)=1</b>                                |
| 3            | Warning         | Size of the subspace <b>M0</b> is too small ( <b>M0</b> <= <b>M</b> )                     |
| 2            | Warning         | No Convergence (#iteration loops> <b>fpm(4)</b> )                                         |
| 1            | Warning         | No Eigenvalue found in the search interval                                                |
| 0            | Successful exit |                                                                                           |
| -1           | Error           | Internal error conversion single/double                                                   |
| -2           | Error           | Internal error of the inner system solver in FEAST Driver interfaces                      |
| -3           | Error           | Internal error of the reduced eigenvalue solver                                           |
|              |                 | <i>Possible cause for Hermitian problem: matrix <b>B</b> may not be positive definite</i> |
| $-(100 + i)$ | Error           | Problem with the $i^{th}$ argument of the FEAST interface                                 |

Table 2: Return code descriptions for the parameter **info**.

### Quick Tutorial using FEAST Drivers

1. Identify your FEAST driver. Look at the corresponding section of this documentation to set up the argument lists.
2. Specify a search contour enclosing the wanted eigenvalues (normal FEAST mode). Alternatively, you can also use the Hermitian sparse drivers to search for the **M0/2** lowest (**fpm(40)=-1**) or largest (**fpm(40)=1**) eigenpairs (here **M0** must be set to two times the number of wanted eigenvalues).
3. In normal FEAST mode, specify the search subspace size **M0** as an overestimation of your estimated #eigenvalues **M** within the contour (typically **M0** ≥ 1.5**M**). If needed, user can take advantage of fast stochastic estimates for **M** within a particular contour using **fpm(14)=2**.
4. Change the **fpm** default options if needed and run the code.

### Tips

- The FEAST convergence rate depends on the choice of the search subspace size **M0**, the number of contour points, and the nature of the quadrature. To improve the convergence rate, you have the possibility to:
  - keep on increasing the number of quadrature nodes **fpm(2)** (**fpm(8)** for non-Hermitian/Polynomial).
  - keep on increasing **M0** for the Gauss-Legendre or Trapezoidal quadrature.
  - switch to Zolotarev quadrature for the Hermitian problem with **fpm(16)=2** (Zolotarev is ideally suited to deal with continuum spectra without the need to increase the subspace size **M0**~**M**).
- Although FEAST could be used to seek 1000’s of eigenpairs within a single search contour, the size of the search subspace **M0** is supposed to be much smaller than the size of the eigenvalue problem **N**. As a result, the arithmetic complexity would mainly depend on the inner system solve (i.e.  $O(NM_0)$  for narrow banded or sparse system solvers). If you are looking for a very large number of eigenvalues, it is recommended to consider multiple search intervals to be solved in parallel using PFEAST.
- FEAST v4.0 is using an inverse residual iteration algorithm which enables the linear systems to be solved with very low accuracy with no impact on the FEAST double precision convergence rate (!).

Consequently, all FEAST linear systems are solved in single precision by default (`fpm(42)=1`). Using the RCI interfaces, users can then plug in their own low accuracy (single precision or less) direct or iterative solver. Additionally, all the linear system factorizations can be kept in memory by (using `fpm(10)=1`) which improves performance but use more memory.

- Using the FEAST-SMP library, parallelism at the third level L3 (linear system solves) can only be achieved using the threading capabilities of the linear system solver and via the shell variable `MKL_NUM_THREADS` if Intel-MKL is used or the shell variable `OMP_NUM_THREADS` if SPIKE is used for the banded interfaces.
- Using the FEAST-MPI library, you can trivially parallelize the second level L2 (contour points) and keep on using the same FEAST/IFEAST driver interfaces with shared memory solver at L3. Scalability performances will be optimal when the number of MPI processes reaches the number of contour points (either `fpm(2)` for FEAST Hermitian or `fpm(8)` for FEAST non-Hermitian and Polynomial).
- The FEAST-MPI library also offers the possibility to use a MPI solver at level L3. This scheme is called PFEAST and it is detailed in Section 3.6.



### 3.2 FEAST Hermitian

```

!!!! Standard AX=EX - Real-Symmetric and Complex Hermitian
dfeast_sFev{x}({List-A},fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info,{Zne,Wne})
zfeast_hFev{x}({List-A},fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info,{Zne,Wne})
!!!! Generalized AX=EBX - Real-Symmetric and Complex Hermitian- (B is hpd)
dfeast_sFgv{x}({List-A},{List-B},fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info,{Zne,Wne})
zfeast_hFgv{x}({List-A},{List-B},fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info,{Zne,Wne})
!!!! RCI (format independent) - Real-Symmetric and Complex Hermitian
dfeast_src{i}{x}(ijob,N,Ze,work1,work2,Aq,Bq,fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info,{Zne,Wne})
zfeast_hrci{x}(ijob,N,Ze,work1,work2,Aq,Bq,fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info,{Zne,Wne})

```

We note the following:

- The Table below details the series of arguments in each **{List-A}**, and **{List-B}** that are specific to the type of matrix format represented above by **F** (as a placeholder).

|        | <b>F</b> | <b>List-A</b>           | <b>List-B</b>  |
|--------|----------|-------------------------|----------------|
| Dense  | {y,e}    | { UPL0, N, A, LDA }     | { B, LDB }     |
| Banded | b        | { UPL0, N, ka, A, LDA } | { kb, B, LDB } |
| Sparse | csr      | { UPL0, N, A, IA, JA }  | { B, IB, JB }  |

- Table 4 details the specific matrix-format arguments in **{List-A}** and **{List-B}**
- Table 3 details the common arguments in all the Hermitian FEAST interfaces above,
- Table 5 details the arguments for the Hermitian RCI interfaces (in red above).

|           | Type                                                                | I/O    | Description                                                                                                                                                                                      |
|-----------|---------------------------------------------------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| fpm       | integer(64)                                                         | in/out | FEAST input parameters (see Table 1)                                                                                                                                                             |
| epsout    | double real                                                         | out    | Trace relative error $ \text{trace}_k - \text{trace}_{k-1}  / \max( \text{Emin} ,  \text{Emax} )$                                                                                                |
| loop      | integer                                                             | out    | # of FEAST subspace iterations                                                                                                                                                                   |
| Emin,Emax | double real                                                         | in/out | Lower and Upper bounds of search interval<br>Remark: Output values if fpm(40)=+-1 for sparse drivers                                                                                             |
| M0        | integer                                                             | in/out | Search subspace dimension<br>On entry: initial guess $M0 \geq M$<br>On exit: new suitable M0 if guess too large<br>Remark: $M0=2*Wanted$ if fpm(40)=+-1 for sparse drivers                       |
| E         | double real(M0)                                                     | in/out | Eigenvalues<br>On entry: initial guess if fpm(5)=1 (previous FEAST run)<br>On exit: Eigenvalues solutions E(1:M)<br>Remark: the E(M+1:M0) values are outside [Emin,Emax]                         |
| X         | double real(N,M0) using dfeast<br>double complex(N,M0) using zfeast | in/out | Eigenvectors (N: size of the system)<br>On entry: initial guess if fpm(5)=1 (previous FEAST run)<br>On exit: Eigenvectors solutions X(1:N,1:M)<br>Remark: if fpm(14)=1, first Q subspace on exit |
| M         | integer                                                             | out    | #Eigenvalues found in [Emin,Emax]<br>#Estimated eigenvalues if fpm(14)=2                                                                                                                         |
| res       | double real(M0)                                                     | out    | Relative residual $\ Ax_i - \lambda_i Bx_i\ _2 / \ \alpha Bx_i\ _2$<br>with $\alpha = \max( \text{Emin} ,  \text{Emax} )$                                                                        |
| info      | integer                                                             | out    | Error handling (see Table 2 for all INFO codes)                                                                                                                                                  |
| Zne,Wne   | double complex(fpm(2))                                              | in     | Custom integration nodes and weights- Expert mode                                                                                                                                                |

Table 3: List of common arguments for the FEAST Hermitian Driver interfaces.

|            | Type                                                                                           | I/O | Description                                                                    |
|------------|------------------------------------------------------------------------------------------------|-----|--------------------------------------------------------------------------------|
| Common     |                                                                                                |     |                                                                                |
| UPLO       | character(len=1)                                                                               | in  | Matrix Storage ('F', 'L', 'U')<br>'F': Full; 'L': Lower; 'U': Upper            |
| N          | integer                                                                                        | in  | Size of the system                                                             |
| Dense      |                                                                                                |     |                                                                                |
| A          | double real(LDA,N)      using <b>dfeast</b><br>double complex(LDA,N)    using <b>zfeast</b>    | in  | Eigenvalue system (Stiffness) matrix                                           |
| LDA        | integer                                                                                        | in  | Leading dimension of A LDA>=N                                                  |
| B          | double real(LDB,N)      using <b>dfeast</b><br>double complex(LDA,N)    using <b>zfeast</b>    | in  | Eigenvalue system (Mass) matrix                                                |
| LDB        | integer                                                                                        | in  | Leading dimension of B LDB>=N                                                  |
| Banded     |                                                                                                |     |                                                                                |
| ka         | integer                                                                                        | in  | The number of sub or super-diagonals within the band of A.                     |
| A          | double real(LDA,N)      using <b>dfeast</b><br>double complex(LDA,N)    using <b>zfeast</b>    | in  | Eigenvalue system (Stiffness) matrix                                           |
| LDA        | integer                                                                                        | in  | Leading dimension of A LDA>=2*ka+1 if UPLO='F'<br>LDA>=ka+1 if UPLO='L' or 'U' |
| kb         | integer                                                                                        | in  | The number of sub or super-diagonals within the band of B.                     |
| B          | double real(LDB,N)      using <b>dfeast</b><br>double complex(LDB,N)    using <b>zfeast</b>    | in  | Eigenvalue system (Mass) matrix                                                |
| LDB        | integer                                                                                        | in  | Leading dimension of B LDB>=2*kb+1 if UPLO='F'<br>LDB>=kb+1 if UPLO='L' or 'U' |
| Sparse-csr |                                                                                                |     |                                                                                |
| A          | double real(IA(N+1)-1)    using <b>dfeast</b><br>double complex(IA(N+1)-1) using <b>zfeast</b> | in  | Eigenvalue system (Stiffness) matrix - CSR values                              |
| IA         | integer(N+1)                                                                                   | in  | Sparse CSR Row array of A.                                                     |
| JA         | integer(IA(N+1)-1)                                                                             | in  | Sparse CSR Column array of A.                                                  |
| B          | double real(IB(N+1)-1)    using <b>dfeast</b><br>double complex(IB(N+1)-1) using <b>zfeast</b> | in  | Eigenvalue system (Mass) matrix - CSR values                                   |
| IB         | integer(N+1)                                                                                   | in  | Sparse CSR Row array of B.                                                     |
| JB         | integer(IB(N+1)-1)                                                                             | in  | Sparse CSR Column array of B.                                                  |

Table 4: List of arguments that are matrix-format specific for the FEAST Driver interfaces. Applicable to Hermitian and Non-Hermitian Drivers.

|        | Type                                                                                        | I/O    | Description                                                                  |
|--------|---------------------------------------------------------------------------------------------|--------|------------------------------------------------------------------------------|
| ijob   | integer                                                                                     | in/out | On entry: ijob=-1 (initialization)<br>On exit: ID of the FEAST_RCI operation |
| N      | integer                                                                                     | in     | Size of the system                                                           |
| Ze     | double complex                                                                              | out    | Coordinate along the complex contour                                         |
| work1  | double real(N,M0)      using <b>dfeast</b><br>double complex(N,M0)    using <b>zfeast</b>   | in/out | Workspace                                                                    |
| work2  | double complex(N,M0)                                                                        | in/out | Workspace                                                                    |
| Aq, Bq | double real(M0,M0)      using <b>dfeast</b><br>double complex(M0,M0)    using <b>zfeast</b> | in/out | Workspace for the reduced eigenvalue problem                                 |

Table 5: List of arguments for the FEAST Hermitian RCI interfaces.

## Hermitian Driver Interfaces: Examples

Let us consider the following systems:

**System1** a “real symmetric” generalized eigenvalue problem  $\mathbf{Ax} = \lambda\mathbf{Bx}$ , where  $\mathbf{A}$  is real symmetric and  $\mathbf{B}$  is symmetric positive definite.  $\mathbf{A}$  and  $\mathbf{B}$  are of the size  $N = 1671$  and have the same sparsity pattern with number of non-zero elements  $NNZ = 11435$ .

**System2** a “complex Hermitian” standard eigenvalue problem  $\mathbf{Ax} = \lambda\mathbf{x}$ , where  $\mathbf{A}$  is complex Hermitian.  $\mathbf{A}$  is of size  $N = 600$  with number of non-zero elements  $NNZ = 2988$ .

The `$FEASTROOT/example/FEAST` directory provides Fortran and C implementation of these systems using both dense, banded and sparse-CSR storage. Here, the complete list of routines:

|        | System1                                                                                                               | System2                    |
|--------|-----------------------------------------------------------------------------------------------------------------------|----------------------------|
| dense  | {F90,C}dense_dfeast_sygv                                                                                              | {F90,C}dense_zfeast_heev   |
| banded | {F90,C}dense_dfeast_sbgv                                                                                              | {F90,C}dense_zfeast_hbev   |
| sparse | {F90,C}dense_dfeast_scsrgv<br>{F90,C}dense_dfeast_scsrgv_lowest*<br>(*using dfeast_scsrgv to compute few lowest eig.) | {F90,C}dense_zfeast_hcsrev |

The `$FEASTROOT/example/PFEAST-L2` directory provides the parallel implementation of all these routines using FEAST-MPI. The routine names are preceded by the letter P. The MPI parallelization operates only at the second level L2 where all the (fpm(2)) linear systems are distributed among the MPI processes.

## Hermitian RCI Interfaces

Using the FEAST\_RCI interfaces, the `ijob` parameter must first be initialized with the value `-1`. Once the RCI interface is called, the value of the `ijob` output parameter, if different than 0, is used to identify the FEAST operation that needs to be completed by the user. Users have then the possibility to customize their own matrix direct or iterative factorization and linear solve techniques as well as their own matrix multiplication routine.

Here is a general (F90) template example of RCI for solving real symmetric problem:

```

1 ijob=-1 ! initialization
2 do while (ijob/=0)
3 call dfeast_src1(ijob, N, Ze, work1, work2, Aq, Bq, fpm, epsout, loop, Emin, Emax, M0, E, X, M, res, info)
4 select case(ijob)
5 case(10) !!Factorize the complex matrix Az <=(ZeB-A) - or factorize a preconditioner of ZeB-A
6 !!REMARK: Az can be formed and factorized using single precision arithmetic
7 <<< user entry
8 case(11) !!Solve the linear system with fpm(23) rhs; Az * Qz=work2(1:N,1:fpm(23))
9 !!Result (in place) in work2 <= Qz(1:N,1:fpm(23))
10 !!REMARKS: -Solve can be performed in single precision
11 !! -Low accuracy iterative solver are ok
12 <<< user entry
13 case(30) !!Perform multiplication A * X(1:N,i:j) result in work1(1:N,i:j)
14 !! where i=fpm(24) and j=fpm(24)+fpm(25)-1
15 <<< user entry
16 case(40) !!Perform multiplication B * X(1:N,i:j) result in work1(1:N,i:j)
17 !! where i=fpm(24) and j=fpm(24)+fpm(25)-1
18 !!REMARK: user must set work1(1:N,i:j)=X(1:N,i:j) if B=I
19 <<< user entry
20 end select
21 end do
22

```

Here is a general (F90) template example of RCI for solving complex Hermitian problem:

```

1 ijob=-1 ! initialization
2 do while (ijob/=0)
3 call zfeast_hrci(ijob,N,Ze,work1,work2,Aq,Bq,fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)
4 select case(ijob)
5 case(10) !!Factorize the complex matrix Az <=(ZeB-A) - or factorize a preconditioner of ZeB-A
6 !!REMARK: Az can be formed and factorized using single precision arithmetic
7 <<< user entry
8 case(11) !!Solve the linear system with fpm(23) rhs; Az * Qz=work2(1:N,1:fpm(23))
9 !!Result (in place) in work2 <= Qz(1:N,1:fpm(23))
10 !!REMARKS: -Solve can be performed in single precision
11 !! -Low accuracy iterative solver are ok
12 <<< user entry
13 case(20) !![Optional: *only if* needed by case(21)]
14 !!Factorize the complex matrix Az^H
15 !!REMARKS: -The matrix Az from case(10) cannot be overwritten
16 !! -case(20) becomes obsolete if the solve in case(21) can be performed
17 !! by reusing the factorization in case(10)
18 <<< user entry
19 case(21) !!Solve the linear system with fpm(23) rhs; Az^H * Qz=work2(1:N,1:fpm(23))
20 !!Result (in place) in work2 <= Qz(1:N,1:fpm(23))
21 <<< user entry
22 case(30) !!Perform multiplication A * X(1:N,i:j) result in work1(1:N,i:j)
23 !! where i=fpm(24) and j=fpm(24)+fpm(25)-1
24 <<< user entry
25 case(40) !!Perform multiplication B * X(1:N,i:j) result in work1(1:N,i:j)
26 !! where i=fpm(24) and j=fpm(24)+fpm(25)-1
27 !!REMARK: user must set work1(1:N,i:j)=X(1:N,i:j) if B=I
28 <<< user entry
29 end select
30 end do
31

```

### 3.3 FEAST Non-Hermitian

```
!!!! Standard AX=EX - Complex Symmetric, Real General and Complex General
zfeast_sFev{x}({List-A},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
dfeast_gFev{x}({List-A},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
zfeast_gFev{x}({List-A},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
!!!! Generalized AX=EBX - Complex Symmetric (B also sym.), Real General and Complex General
zfeast_sFgv{x}({List-A},{List-B},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
dfeast_gFgv{x}({List-A},{List-B},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
zfeast_gFgv{x}({List-A},{List-B},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
!!!! RCI (format independent) - Complex Symmetric and Real/Complex General
zfeast_src{i}{x}(ijob,N,Ze,work1,work2,Aq,Bq,fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
zfeast_grci{x}(ijob,N,Ze,work1,work2,Aq,Bq,fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
```

We note the following:

- The Table below details the series of arguments in each **{List-A}**, and **{List-B}** that are specific to the type of matrix format represented above by **F** (as a placeholder).

|               | <b>F</b> | <b>List-A</b>         | <b>List-B</b>      |
|---------------|----------|-----------------------|--------------------|
| <b>Dense</b>  |          |                       |                    |
| Symmetric     | y        | {UPLO, N, A, LDA}     | {B, LDB}           |
| General       | e        | {N, A, LDA}           | {B, LDB}           |
| <b>Banded</b> |          |                       |                    |
| Symmetric     | b        | {UPLO, N, ka, A, LDA} | {kb, B, LDB}       |
| General       | b        | {N, kla, kua, A, LDA} | {klb, kub, B, LDB} |
| <b>Sparse</b> |          |                       |                    |
| Symmetric     | csr      | {UPLO, N, A, IA, JA}  | {B, IB, JB}        |
| General       | csr      | {N, A, IA, JA}        | {B, IB, JB}        |

- Similarly to the Hermitian case, Table 4 details the specific matrix-format arguments in **{List-A}** and **{List-B}**. For the banded drivers and the real/complex general cases, **kla** (resp. **klb**) represents the number of sub-diagonals for matrix **A** (resp. matrix **B**), and **kua** (resp. **kub**) the number of super-diagonals for matrix **A** (resp. matrix **B**).
- Table 7 details the common arguments in all the non-Hermitian FEAST interfaces above.
- Table 6 details the arguments for the non-Hermitian RCI interfaces (in red above).

|               | Type                                                                                                                         | I/O    | Description                                                                  |
|---------------|------------------------------------------------------------------------------------------------------------------------------|--------|------------------------------------------------------------------------------|
| <b>ijob</b>   | integer                                                                                                                      | in/out | On entry: ijob=-1 (initialization)<br>On exit: ID of the FEAST_RCI operation |
| <b>N</b>      | integer                                                                                                                      | in     | Size of the system                                                           |
| <b>Ze</b>     | double complex                                                                                                               | out    | Coordinate along the complex contour                                         |
| <b>work1</b>  | double complex(N,M0)<br>or<br>double complex(N,2*M0)<br>(if left vector calculated for non-sym.<br>interfaces and fpm(15)=0) | in/out | Workspace                                                                    |
| <b>work2</b>  | double complex(N,M0)                                                                                                         | in/out | Workspace                                                                    |
| <b>Aq, Bq</b> | double complex(M0,M0)                                                                                                        | in/out | Workspace for the reduced eigenvalue problem                                 |

Table 6: List of arguments for the FEAST RCI interfaces. Applicable to Non-Hermitian and Polynomial Drivers.

|                | Type                                                                                                                               | I/O    | Description                                                                                                                                                                                                                                                                                            |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>fpm</b>     | integer(64)                                                                                                                        | in/out | FEAST input parameters (see Table 1)                                                                                                                                                                                                                                                                   |
| <b>epsout</b>  | double real                                                                                                                        | out    | Trace relative error $ \text{trace}_k - \text{trace}_{k-1}  / \max( \text{Emid}  + r)$                                                                                                                                                                                                                 |
| <b>loop</b>    | integer                                                                                                                            | out    | # of FEAST subspace iterations                                                                                                                                                                                                                                                                         |
| <b>Emid</b>    | double complex                                                                                                                     | in     | Coordinate center of the contour ellipse                                                                                                                                                                                                                                                               |
| <b>r</b>       | double real                                                                                                                        | in     | Horizontal radius of the contour ellipse                                                                                                                                                                                                                                                               |
| <b>M0</b>      | integer                                                                                                                            | in/out | Search subspace dimension<br>On entry: initial guess $M0 \geq M$<br>On exit: new suitable $M0$ if guess too large                                                                                                                                                                                      |
| <b>E</b>       | double complex( $M0$ )                                                                                                             | in/out | Eigenvalues<br>On entry: initial guess if <b>fpm</b> (5)=1 (previous FEAST run)<br>On exit: Eigenvalues solutions <b>E</b> (1:M)<br>Remark: the <b>E</b> (M+1:M0) values are outside the contour                                                                                                       |
| <b>X</b>       | double complex(N,M0)<br>or<br>double complex(N,2*M0)<br>(if left vectors calculated for<br>non-sym. drivers and <b>fpm</b> (15)=0) | in/out | Eigenvectors (N: size of the system)<br>On entry: initial guess if <b>fpm</b> (5)=1 (previous FEAST run)<br>On exit: (right) Eigenvectors solutions <b>X</b> (1:N,1:M)<br>Remarks: -left vectors (if calculated) in <b>X</b> (1:N,M0+1:M0+M)<br>-if <b>fpm</b> (14)=1, first <b>Q</b> subspace on exit |
| <b>M</b>       | integer                                                                                                                            | out    | #Eigenvalues found inside contour<br>#Estimated eigenvalues if <b>fpm</b> (14)=2                                                                                                                                                                                                                       |
| <b>res</b>     | double complex( $M0$ )<br>or<br>double complex(2*M0)<br>(if left vectors calculated)                                               | out    | Relative residual <b>res</b> (1:M) (right); <b>res</b> (M0+1,M0+M) (left)<br>(right) $\ A x_i - \lambda_i B x_i\ _2 / \ \alpha B x_i\ _2$<br>(left) $\ A^H x_i - \lambda_i^* B^H x_i\ _2 / \ \alpha B^H x_i\ _2$<br>with $\alpha = \max( \text{Emid}  + r)$                                            |
| <b>info</b>    | integer                                                                                                                            | out    | Error handling (see Table 2 for all INFO codes)                                                                                                                                                                                                                                                        |
| <b>Zne,Wne</b> | double complex( <b>fpm</b> (8))                                                                                                    | in     | Custom integration nodes and weights- Expert mode                                                                                                                                                                                                                                                      |

Table 7: List of common arguments for the FEAST Driver interfaces. Applicable to Non-Hermitian and Polynomial Drivers.

### non-Hermitian Driver Interfaces: Examples

Let us consider the following systems:

**System3** a “real non-symmetric” generalized eigenvalue problem  $Ax = \lambda Bx$ , where **A** and **B** are real non-symmetric. **A** and **B** are of the size  $N = 1671$  and have the same sparsity pattern with number of non-zero elements  $NNZ = 13011$ .

**System4** a “complex symmetric” standard eigenvalue problem  $Ax = \lambda x$ , where **A** is complex symmetric. **A** is of size  $N = 801$  with number of non-zero elements  $NNZ = 24591$ .

The `$FEASTROOT/example/FEAST` directory provides Fortran and C implementation of these systems using both dense, banded and sparse-CSR storage. Here, the complete list of routines (System4 includes examples for using expert routines as well):

|        | System3                    | System4                       |
|--------|----------------------------|-------------------------------|
| dense  | {F90,C}dense_dfeast_gegv   | {F90,C}dense_zfeast_syev{x}   |
| banded | {F90,C}dense_dfeast_gbgv   | {F90,C}dense_zfeast_sbev{x}   |
| sparse | {F90,C}dense_dfeast_gcsrgv | {F90,C}dense_zfeast_scsrev{x} |

The `$FEASTROOT/example/PFEAST-L2` directory provides the parallel implementation of all these routines using FEAST-MPI. The routine names are preceded by the letter P. The MPI parallelization operates only at the second level L2 where all the (**fpm**(2)) linear systems are distributed among the MPI processes.

## non-Hermitian RCI Interfaces

The RCI template for solving the complex symmetric problem is the same than the one used for solving the real symmetric case in Section 3.2 (just replace `dfeast_src{xi}` by `zfeast_src{xi}`).

Here is a general (F90) template example of RCI for solving real/complex general problem:

```

1 ijob=-1 ! initialization
2 do while (ijob/=0)
3 call zfeast_grci(ijob,N,Ze,work1,work2,Aq,Bq,fpm,epsout,loop,Emid,r,M0,E,X,M,res,info)
4 select case(ijob)
5 case(10) !!Factorize the complex matrix Az <=(ZeB-A) - or factorize a preconditioner of ZeB-A
6 !!REMARK: Az can be formed and factorized using single precision arithmetic
7 <<< user entry
8 case(11) !!Solve the linear system with fpm(23) rhs; Az * Qz=work2(1:N,1:fpm(23))
9 !!Result (in place) in work2 <= Qz(1:N,1:fpm(23))
10 !!REMARKS: -Solve can be performed in single precision
11 !! -Low accuracy iterative solver are ok
12 <<< user entry
13 case(20) !![Optional: *only if* needed by case(21)]
14 !!Factorize the complex matrix Az^H
15 !!REMARKS: -The matrix Az from case(10) cannot be overwritten
16 !! -case(20) becomes obsolete if the solve in case(21) can be performed
17 !! by reusing the factorization in case(10)
18 <<< user entry
19 case(21) !!Solve the linear system with fpm(23) rhs; Az^H * Qz=work2(1:N,1:fpm(23))
20 !!Result (in place) in work2 <= Qz(1:N,1:fpm(23))
21 <<< user entry
22 case(30) !!Perform multiplication A * X(1:N,i:j) result in work1(1:N,i:j)
23 !! where i=fpm(24) and j=fpm(24)+fpm(25)-1
24 <<< user entry
25 case(31) !!Perform multiplication A^H * X(1:N,i:j) result in work1(1:N,i:j)
26 !! where i=fpm(34) and j=fpm(34)+fpm(35)-1
27 <<< user entry
28 case(40) !!Perform multiplication B * X(1:N,i:j) result in work1(1:N,i:j)
29 !! where i=fpm(24) and j=fpm(24)+fpm(25)-1
30 !!REMARK: user must set work1(1:N,i:j)=X(1:N,i:j) if B=I
31 <<< user entry
32 case(41) !!Perform multiplication B^H * X(1:N,i:j) result in work1(1:N,i:j)
33 !! where i=fpm(34) and j=fpm(34)+fpm(35)-1
34 !!REMARK: user must set work1(1:N,i:j)=X(1:N,i:j) if B=I
35 <<< user entry
36 end select
37 end do
38

```

### 3.4 FEAST Polynomial (quadratic, cubic, quartic, etc.)

Solving  $\sum_{i=0}^p \lambda^i A_i x = 0$

```
!!!! {Ai} Real-Sym., Complex Herm., Complex Sym., Real General and Complex General
dfeast_sFpev{x}({List-A},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
zfeast_hFpev{x}({List-A},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
zfeast_sFpev{x}({List-A},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
dfeast_gFpev{x}({List-A},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
zfeast_gFpev{x}({List-A},fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
!!!! RCI (format independent) - Real/Complex Symmetric and Hermitian/Real/Complex General
zfeast_srcipev{x}(ijob,p,N,Ze,work1,work2,Aq,Bq,fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
zfeast_grcipev{x}(ijob,p,N,Ze,work1,work2,Aq,Bq,fpm,epsout,loop,Emid,r,M0,E,X,M,res,info,{Zne,Wne})
```

We note the following:

- The Table below details the series of arguments in **{List-A}** that are specific to the type of matrix format represented above by **F** (as a placeholder). Remark: the banded format is not supported.

|                     | <b>F</b> | <b>List-A</b>           |
|---------------------|----------|-------------------------|
| <b>Dense</b>        |          |                         |
| Symmetric/Hermitian | {y,e}    | {UPLO, p, N, A, LDA}    |
| General             | e        | {p, N, A, LDA}          |
| <b>Sparse</b>       |          |                         |
| Symmetric/Hermitian | csr      | {UPLO, p, N, A, IA, JA} |
| General             | csr      | {p, N, A, IA, JA}       |

- Table 8 details the specific matrix-format arguments in **{List-A}**.
- The common arguments in all the polynomial FEAST interfaces above are identical to the ones given for the non-Hermitian case in Table 7.
- The argument for the RCI interfaces (in red above) are also identical to the ones given for the non-Hermitian case in Table 7. We note the addition of the argument integer **p** which stands for the degree of the polynomial (e.g. **p=2** fro quadratic, **p=3** for cubic etc.)

|                                                                                                       | Type                                                                                      | I/O | Description                                                         |
|-------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|-----|---------------------------------------------------------------------|
| <b>Common</b>                                                                                         |                                                                                           |     |                                                                     |
| UPLO                                                                                                  | character(len=1)                                                                          | in  | Matrix Storage ('F', 'L', 'U')<br>'F': Full; 'L': Lower; 'U': Upper |
| N                                                                                                     | integer                                                                                   | in  | Size of the system                                                  |
| p                                                                                                     | integer                                                                                   | in  | Degree of the polynomial                                            |
| <b>Dense</b>                                                                                          |                                                                                           |     |                                                                     |
| A                                                                                                     | double real(LDA,N,p+1)      using dfeast<br>double complex(LDA,N,p+1)    using zfeast     | in  | All system matrices A(:,i)                                          |
| LDA                                                                                                   | integer                                                                                   | in  | 1st Leading dimension of A LDA>=N                                   |
| <b>Sparse-csr - where nnz_max stands for the max of non-zero elements among all A sparse matrices</b> |                                                                                           |     |                                                                     |
| A                                                                                                     | double real(nnz_max,p+1)      using dfeast<br>double complex(nnz_max,p+1)    using zfeast | in  | All system matrices A(:,i) - CSR values                             |
| IA                                                                                                    | integer(N+1,p+1)                                                                          | in  | All sparse CSR Row array of A(:,i).                                 |
| JA                                                                                                    | integer(nnz_max,p+1)                                                                      | in  | All sparse CSR Column array of A(:,i).                              |

Table 8: List of arguments that are matrix-format specific for the FEAST Polynomial Driver interfaces. Remark:  $A(:,i) == A_{i-1}$  in Fortran.



## Polynomial Driver Interfaces: Examples

Let us consider the following system:

**System5** a quadratic eigenvalue problem  $(\mathbf{A}_2\lambda^2 + \mathbf{A}_1\lambda + \mathbf{A}_0)\mathbf{x} = \mathbf{0}$ , where  $\mathbf{A}_2, \mathbf{A}_1, \mathbf{A}_0$  are real symmetric. The size of the system is  $N = 1000$  with 2998 non-zero elements for  $A_0$  and  $A_1$ , and 1000 for  $A_2$ .

The \$FEASTROOT/example/FEAST directory provides Fortran and C implementation of this system using both dense and sparse-CSR storage. Here, the complete list of routines:

| System5 |                             |
|---------|-----------------------------|
| dense   | {F90,C}dense_dfeast_syev    |
| sparse  | {F90,C}dense_dfeast_scsrpev |

The \$FEASTROOT/example/PFEAST-L2 directory provides the parallel implementation of these routines using FEAST-MPI. The routine names are preceded by the letter P. The MPI parallelization operates only at the second level L2 where all the (fpm(2)) linear systems are distributed among the MPI processes.

## Polynomial RCI Interfaces

Here is a general (F90) template example of RCI for solving real/complex symmetric problem:

```

1 !!! Here your polynomial matrices are in stored A[i] (i=1,..,p+1) (p polynomial degree)
2 !!! All the matrices are real or complex symmetric
3 ijob=-1 ! initialization
4 do while (ijob/=0)
5 call zfeast_srcipev (ijob ,p,N,Ze ,work1 ,work2 ,Aq,Bq,fpm ,epsout , loop ,Emid,r ,M0,E,X,M, res , info)
6 select case (ijob)
7 case(10) !!Form and Factorize P(Ze)
8 !!Example for the quadratic problem: P(Ze)=A[3]*Ze**2+A[2]*Ze+A[1]
9 !!REMARK: P(Ze) can be formed and factorized using single precision arithmetic
10 <<< user entry
11 case(11) !!Solve the linear system with fpm(23) rhs; P(Ze)* Qz=work2(1:N,1:fpm(23))
12 !!Result (in place) in work2 <= Qz(1:N,1:fpm(23))
13 !!REMARKS: -Solve can be performed in single precision
14 !! -Low accuracy iterative solver are ok
15 <<< user entry
16 case(30) !!Perform multiplication A[fpm(57)] * X(1:N,i:j) result in work1(1:N,i:j)
17 !! where i=fpm(24) and j=fpm(24)+fpm(25)-1; fpm(57) take the values 1...p+1
18 <<< user entry
19 end select
20 end do
21

```

Here is a general (F90) template example of RCI for solving Hermitian or real/complex general problem:

```

1 !!! Here your polynomial matrices are in stored A[i] (i=1,..,p+1) (p polynomial degree)
2 !!! At least one matrix is not real/complex symmetric
3 ijob=-1 ! initialization
4 do while (ijob/=0)
5 call zfeast_grcipev (ijob ,p,N,Ze ,work1 ,work2 ,Aq,Bq,fpm ,epsout , loop ,Emid,r ,M0,E,X,M, res , info)
6 select case (ijob)
7 case(10) !!Form and Factorize P(Ze)
8 !!Example for the quadratic problem: P(Ze)=A[3]*Ze**2+A[2]*Ze+A[1]
9 !!REMARK: P(Ze) can be formed and factorized using single precision arithmetic
10 <<< user entry
11 case(11) !!Solve the linear system with fpm(23) rhs; P(Ze)* Qz=work2(1:N,1:fpm(23))
12 !!Result (in place) in work2 <= Qz(1:N,1:fpm(23))
13 !!REMARKS: -Solve can be performed in single precision
14 !! -Low accuracy iterative solver are ok
15 <<< user entry

```

```

16 case(20) !![Optional: *only if* needed by case(21)]
17 !!Factorize the complex matrix P(Ze)^H
18 !!REMARKS: -The factorization P(Ze) from case(10) cannot be overwritten
19 !! -case(20) becomes obsolete if the solve in case(21) can be performed
20 !! by reusing the factorization in case(10)
21 <<< user entry
22 case(21) !!Solve the linear system with fpm(23) rhs; P(Ze)^H * Qz=work2(1:N,1:fpm(23))
23 !!Result (in place) in work2 <= Qz(1:N,1:fpm(23))
24 <<< user entry
25 case(30) !!Perform multiplication A[fpm(57)] * X(1:N,i:j) result in work1(1:N,i:j)
26 !! where i=fpm(24) and j=fpm(24)+fpm(25)-1; fpm(57) will the values 1...p+1
27 <<< user entry
28 case(31) !!Perform multiplication A^H[fpm(57)] * X(1:N,i:j) result in work1(1:N,i:j)
29 !! where i=fpm(34) and j=fpm(34)+fpm(35)-1; fpm(57) will the values 1...p+1
30 <<< user entry
31 end select
32 end do
33

```

### 3.5 IFEAST (FEAST w/o Factorization)

IFEAST stands for using FEAST using inexact iterative solver for solving the linear systems (instead of using a direct solver). IFEAST only supports the sparse driver interfaces where the MKL-PARDISO solver is then replaced by a built-in BiCGstab solver. IFEAST is particularly effective if the sparse system matrix is very large (typically >1M) and/or the direct factorization becomes too expensive (both in time and memory). Two options are possible for calling IFEAST:

**Option1** In the naming convention of all FEAST sparse drivers, replace **feast** by **ifeast**.

**Option2** Keep the name of your FEAST sparse driver unchanged but use the new flag value **fpm(43)=1**.

As an example, let us re-work the helloworld example presented in Section 2.4. Here are the few lines that need to be changed in the code in order to make use of IFEAST:

```

1 !!!! the matrix A needs first to be defined in sparse CSR format;
2 !!!! add these lines in the variable declaration section of the program
3 integer , parameter :: NNZ
4 double precision , dimension (NNZ) :: sA=(/2.0d0,-1.0d0,-1.0d0,-1.0d0,3.0d0,-1.0d0,-1.0d0,&
5 &-1.0d0,-1.0d0,3.0d0,-1.0d0,-1.0d0,-1.0d0,2.0d0/)
6 integer , dimension (N+1) :: IA=(/1,4,8,12,15/)
7 integer , dimension (NNZ) :: JA=(/1,2,3,1,2,3,4,1,2,3,4,2,3,4/)
8
9
10 !!!! The direct call to IFEAST can be done as follow (option 1)
11 call feastinit(fpm)
12 fpm(1)=1 !! change from default value (print info on screen)
13 call difeast_scsrev (UPLO,N,sA,IA,JA,fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)
14
15 !!!! Alternatively, an indirect call to IFEAST is also possible (option 2)
16 call feastinit(fpm)
17 fpm(1)=1 !! change from default value (print info on screen)
18 fpm(43)=1 !! switch solver in FEAST sparse driver from MKL-PARDISO to BicGStab
19 call difeast_scsrev (UPLO,N,sA,IA,JA,fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)

```

Here is the output of the run:

```

***** FEAST v4.0 BEGIN *****

Routine DIFEAST_SCSREV
Solving AX=eX with A real symmetric
List of input parameters fpm(1:64)-- if different from default
 fpm(1)= 1

```

| FEAST data        |                        |
|-------------------|------------------------|
| Emin              | 3.0000000000000000E+00 |
| Emax              | 5.0000000000000000E+00 |
| #Contour nodes    | 4 (half-contour)       |
| Quadrature rule   | Trapezoidal            |
| Ellipse ratio y/x | 1.00                   |
| System solver     | BiCGstab               |
|                   | eps=1E-1; maxit= 40    |
|                   | Single precision       |
|                   | Matrix scaled          |
| FEAST uses MKL?   | Yes                    |
| Fact. stored?     | Yes                    |
| Initial Guess     | Random                 |
| Size system       | 4                      |
| Size subspace     | 3                      |

```

FEAST runs

```

```

#It | #Eig | Trace | Error-Trace | Max-Residual

#it 4; res min= 1.0501874385226984E-05; res max= 3.1005099299363792E-04
#it 3; res min= 1.8933478742837906E-02; res max= 9.8125219345092773E-02
#it 2; res min= 2.9427373781800270E-02; res max= 8.7035216391086578E-02
#it 2; res min= 1.8594998866319656E-02; res max= 4.0714181959629059E-02
0 2 7.9995148090616173E+00 1.0000000000000000E+00 8.7043125405406700E-03
#it 2; res min= 2.8199069201946259E-02; res max= 5.2359659224748611E-02
#it 2; res min= 6.4112566411495209E-02; res max= 8.7011836469173431E-02
#it 2; res min= 3.0766267329454422E-02; res max= 5.8814667165279388E-02
#it 2; res min= 1.4842565171420574E-02; res max= 1.5974638983607292E-02
1 2 7.9999999809553799E+00 9.7034378752525186E-05 5.5062636929826442E-05
#it 1; res min= 5.3539618849754333E-02; res max= 5.3563684225082397E-02
#it 1; res min= 8.7251774966716766E-02; res max= 8.7261073291301727E-02
#it 1; res min= 7.2112381458282471E-02; res max= 7.2119705379009247E-02
#it 1; res min= 6.5362729132175446E-02; res max= 6.5378636121749878E-02
2 2 7.9999999999999591E+00 3.8089158493903597E-09 7.6857681986636782E-08
#it 1; res min= 3.4910961985588074E-02; res max= 3.4911289811134338E-02
#it 1; res min= 8.4501713514328003E-02; res max= 8.4501914680004120E-02
#it 1; res min= 6.9185368716716766E-02; res max= 6.9185607135295868E-02
#it 1; res min= 6.2304046005010605E-02; res max= 6.2304504215717316E-02
3 2 8.0000000000000036E+00 8.8817841970012523E-15 2.5198258072819870E-12
#it 1; res min= 3.4680232405662537E-02; res max= 3.4876041114330292E-02
#it 1; res min= 8.4406383335590363E-02; res max= 8.4497839212417603E-02
#it 1; res min= 6.9181196391582489E-02; res max= 6.9283291697502136E-02
#it 1; res min= 6.2299706041812897E-02; res max= 6.2503643333911896E-02
4 2 8.0000000000000000E+00 7.1054273576010023E-16 3.0767402982137245E-16

==>FEAST has successfully converged with Residual tolerance <1E-12
FEAST outside it. 4
Inner BiCGstab it. 31
Eigenvalue found 2 from 3.999999999999991E+00 to 4.0000000000000009E+00

FEAST-RCI timing
Fact. cases(10,20)
Solve cases(11,12)
A*x cases(30,31)
B*x cases(40,41)
Misc. time
Total time (s)

***** FEAST- END*****

Solutions (Eigenvalues/Eigenvectors/Residuals)
E= 4.000000000000000 X= 0.353553390593291 -0.853553390593266
0.146446609406686 0.353553390593291 Res= 3.076740298213724E-016

E= 4.000000000000000 X= 0.353553390593257 0.146446609406767
-0.853553390593281 0.353553390593257 Res= 2.361858070262224E-016

```

### Some Remarks:

- IFEAST is using different default `fpm` parameters than FEAST. In particular, the trapezoidal rule is used along a half-circle with 4 contour integration points, as well as the BiCGStab iterative solver.
- Each FEAST loop reports the total number of BiCGstab iterations for each linear system solve needed to reach the accuracy defined in `fpm(45)`. The total number of BiCGstab iterations will be reported in `fpm(60)` (here 31, of course IFEAST is rather ineffective for such small system).
- You may have noticed that the values of the output eigenvectors for this example are different than the ones reported in Section 2.4. As a reminder, eigenvectors are not unique, both solutions are here orthonormal and correct (they span the same eigenvector subspace).

### 3.6 PFEAST and PIFEAST (MPI-solver)

In FEAST v4.0, the FEAST-MPI library enables the use of MPI linear system solvers at L3. As a result, the three level of parallelisms of FEAST (L1-L2-L3) can all support MPI (FEAST is internally using three MPI communicators). This MPI-MPI-MPI programming model is named PFEAST. PFEAST currently supports all the sparse drivers (for Hermitian/non-Hermitian/Polynomial problems) as well as all the RCI interfaces. Using RCI, an expert developer could straightforwardly customize PFEAST using highly-efficient application-specific MPI solvers such as: domain decompositions, or iterative/hybrid solvers with/without preconditioners.

Here some information about the use of PFEAST:

**Initialization** The `feastinit(fpm)` routine must be replaced by `pfeastinit(fpm, L1_comm_world, nL3)` which, in addition of setting up default `fpm` values, is going to initialize all MPI communicators.

- `L1_comm_world` represents your own defined MPI communicator for a given search contour (containing `nL1` total MPI processes), if only a single contour is used it must take the value `MPI_COMM_WORLD`.
- `nL3` is an (in/out) integer input that indicates the number of MPI processes you wish to use at level L3 (MPI system solver). The value of `nL3` will be reassigned to `nL1` if `nL1` is not a multiple of `nL3`. In addition, if `nL1 < nL3` then `nL3 = nL1` on exit. Ideally, `nL1/nL3` should be a multiple of a number of contour points (if it is equal to the number of contour points, then L2 is optimally used). Furthermore, L3 can also be threaded (MPI calling OpenMP on each local distributed system), make sure that your number of selected threads `<omp>` times `nL1` does not exceed the number of available physical cores of your cluster.
- This initialization routine is setting up the L3 communicator `fpm(49)` (n particular) that may be needed to distribute your matrix.

**Sparse Drivers** In the naming convention of the FEAST sparse drivers, all routine names must be preceded by the letter `p`. In particular, you must replace `z{i}feast` by `pz{i}feast`, or `d{i}feast` by `pd{i}feast`. We actually use the names PFEAST and PIFEAST to indicate the MPI version of the FEAST and IFEAST interfaces, respectively. PFEAST is using MKL-Cluster-PARDISO and PIFEAST is using a built-in MPI-BiCGstab iterative solver (PBiCGStab). PIFEAST is also using its own built-in highly efficient MPI sparse mat-vec library.

PFEAST/PIFEAST drivers allow two options for the row distribution of matrices and solution vectors:

- global and common to all L3-MPI processes (The row-distribution will then take place internally). The argument list for all interfaces stay unchanged.
- locally row distributed among all L3-MPI processes. The user is responsible for distributing the data using the `fpm(49)` L3 communicator. The argument list for all interfaces stay mostly unchanged but the size of matrix/vectors `N` which must now be local.

Furthermore, PFEAST/PIFEAST will automatically detect which option above you are using!

**RCI Interfaces** The names of the RCI interfaces do not change (the letter `p` is not needed). In the argument list, only the size `N` must be changed to its local value (i.e. local number of rows for the row-distributed vector and work arrays).

#### PFEAST with global L3 distribution: Examples

The `$FEASTROOT/example/PFEAST-L2L3` directory provides Fortran and C implementation of System1 to System5 examples (discussed previously). Here is the complete list of the PFEAST routines that are all using global sparse CSR-storage.

|         | Type                      | Routine                        |
|---------|---------------------------|--------------------------------|
| System1 | Real Sym. Generalized     | P{F90,C}sparse_pdfeast_scsrgv  |
| System2 | Complex Herm. Standard    | P{F90,C}sparse_pzfeast_hcsrev  |
| System3 | Real non-Sym. Generalized | P{F90,C}sparse_pdfeast_gcsrgv  |
| System4 | Complex Sym. Standard     | P{F90,C}sparse_pzfeast_scsrev  |
| System5 | Real Sym. Quadratic       | P{F90,C}sparse_pdfeast_scsrpev |

### PFEAST with local L3 distribution: Example

As an example, let us re-work the helloworld example presented in Section 2.4 using 2 MPI processes to distributed the  $4 \times 4$  matrix. We obtain (for example):

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix} = \begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{pmatrix}$$

A Fortran90 source code example is provided below:

```

1 program helloworld_pfeast_local
2 implicit none
3 include 'mpif.h'
4 !! 4x4 global eigenvalue system == two 2x4 local matrices
5 integer,parameter :: Nloc=2, NNZloc=7
6 character(len=1) :: UPLO='F'
7 double precision,dimension(NNZloc) :: Aloc
8 integer,dimension(Nloc+1) :: IAloc
9 integer,dimension(NNZloc) :: JAloc
10 !! input parameters for FEAST
11 integer,dimension(64) :: fpm
12 integer :: M0=3 ! search subspace dimension
13 double precision :: Emin=3.0d0, Emax=5.0d0 ! search interval
14 !! output variables for FEAST
15 double precision,dimension(:),allocatable :: E, res
16 double precision,dimension(:,:),allocatable :: X
17 double precision :: epsout
18 integer :: nL3,rank3,loop,info,M,i
19 !!! MPI
20 integer :: code
21 call MPI_INIT(code)
22
23 !!! Allocate memory for eigenvalues,eigenvectors,residual
24 allocate(E(M0),X(Nloc,M0),res(M0))
25
26 !!!!!!!!!!! INITIALIZE PFEAST and DISTRIBUTE MATRIX
27 nL3=2
28 call pfeastinit(fpm,MPI_COMM_WORLD,nL3)
29
30 call MPI_COMM_RANK(fpm(49),rank3,code) !! find rank of new L3 communicator
31 if (rank3==0) then
32 Aloc=(/2.0d0,-1.0d0,-1.0d0,3.0d0,-1.0d0,-1.0d0/)
33 IAloc=(/1,4,8/)
34 JAloc=(/1,2,3,1,2,3,4/)
35 elseif (rank3==1) then
36 Aloc=(/-1.0d0,-1.0d0,3.0d0,-1.0d0,-1.0d0,-1.0d0,2.0d0/)
37 IAloc=(/1,5,8/)
38 JAloc=(/1,2,3,4,2,3,4/)
39 endif
40
41 !!!!!!!!!!! PFEAST
42 fpm(1)=1 !! change from default value (print info on screen)
43 call pdfeast_scsrev(UPLO,Nloc,Aloc,IAloc,JAloc,fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)
44
45 !!!!!!!!!!! REPORT
46 if (info==0) then
47 print *, 'Solutions (Eigenvalues/Eigenvectors/Residuals) at rank L3',rank3
48 do i=1,M
49 print *, 'Eigenvalue',i
50 print *, 'E=',E(i), 'X=',X(:,i), 'Res=',res(i)
51 enddo
52 endif
53
54 end program helloworld_pfeast_local

```

Your program must be compiled using the same MPI implementation used to compile the FEAST-MPI library. Once compiled, your source program must now be linked with the `pfeast` library. You can use (for example):

- `mpiifort -o helloworld_pfeast_local helloworld_pfeast_local.f90 -L$FEASTROOT>/lib/<arch> -lpfeast -mkl=parallel -lmkl_blacs_intelmpi_lp64 -liomp5 -lpthread -lm -ldl`

if FEAST was compiled with `ifort`, MKL flag was set to 'yes', and MPI was chosen to be 'impi' (intel mpi).

- `mpif90.mpicx -fc=gfortran -o helloworld_pfeast_local helloworld_pfeast_local.f90 -L$FEASTROOT>/lib/<arch> -lpfeast -Wl,-no-as-needed -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -lmkl_blacs_intelmpi_lp64 -lgomp -lpthread -lm -ldl`

if FEAST was compiled with `gfortran`, MKL flag was set to 'yes', and MPI was chosen to be 'mpich'.

A run of the resulting executable looks like:

```
mpirun -n 2 ./helloworld_pfeast_local
```

and the output of the run should be:

```

***** FEAST v4.0 BEGIN *****

Routine PDFEAST_SCSREV
Solving AX=eX with A real symmetric
#MPI (total=L2*L3) 2= 1* 2
List of input parameters fpm(1:64)-- if different from default
 fpm(1)= 1

FEAST data
Emin
Emax
#Contour nodes
Quadrature rule
Ellipse ratio y/x
System solver
FEAST uses MKL?
Fact. stored?
Initial Guess
Size system
Size subspace

FEAST runs
#It | #Eig | Trace | Error-Trace | Max-Residual

0 | 2 | 7.9999999999999947E+00 | 1.0000000000000000E+00 | 1.2829791863202860E-08
1 | 2 | 8.0000000000000000E+00 | 1.0658141036401502E-15 | 6.6022321739723205E-16

==>FEAST has successfully converged with Residual tolerance <1E-12
FEAST outside it. 1
Eigenvalue found 2 from 3.9999999999999996E+00 to 4.0000000000000000E+00

FEAST-RCI timing
Fact. cases(10,20)
Solve cases(11,12)
A*x cases(30,31)
```

```

| B*x cases(40,41)| 0.0000 |
| Misc. time | 0.0005 |
| Total time (s) | 0.0088 |
|-----|

***** FEAST- END*****

Solutions (Eigenvalues/Eigenvectors/Residuals) at rank L3 0
Eigenvalue 1
E= 4.000000000000000 X= 0.358971274554087 -0.851189974005894 Res= 2.784560286672102E-016
Eigenvalue 2
E= 4.000000000000000 X= -0.348051180209196 -0.159610864767551 Res= 6.602232173972321E-016
Solutions (Eigenvalues/Eigenvectors/Residuals) at rank L3 1
Eigenvalue 1
E= 4.000000000000000 X= 0.133247424897719 0.358971274554087 Res= 2.784560286672102E-016
Eigenvalue 2
E= 4.000000000000000 X= 0.855713225185942 -0.348051180209196 Res= 6.602232173972321E-016

```

### PFEAST using 3 levels of parallelism: Example

Three levels of parallelism means that L1 is active and multiple search contours can be used simultaneously. FEAST v4.0 does not offer automatic partitioning of the overall eigenvalue spectrum, it is then up to the users to guess it. Users could take advantage of fast stochastic estimates with the flag `fpm(14)=2`. Once the eigenvalue spectrum partitioned, a single call to PFEAST will account for all L1-L2-L3 MPI parallelism.

The `$FEASTROOT/example/PFEAST-L1L2L3` directory provides Fortran and C implementation of the System2 example (discussed previously). It uses two search intervals. The name of the routines are:

| System2 |                               |
|---------|-------------------------------|
| sparse  | 3P{F90,C}dense_pzfeast_hcsrev |

**Remark:** Since multiple search intervals are involved, the option `fpm(1)=1` (printing FEAST info on screen), may provide a bit confusing results to read. You can easily change this flag value using `fpm(1)=-i` with `i` is associated with the `rank i-1` of the L1 MPI Communicator. Each search contour will then print all its FEAST results into separate files named `feast{i}.log`.



## 4 Complement

### 4.1 Matrix storage

Let us consider the following matrix  $\mathbf{A}$  (as an example):

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix} \quad (2)$$

If the matrix presents some particular properties such as Hermitian ( $a_{ij} = a_{ji}^*$  for  $i \neq j$ ) or symmetric ( $a_{ij} = a_{ji}$ ), only half of the matrix elements need to be defined. Using the FEAST Driver interfaces, this matrix could be stored in dense, banded or sparse-CSR format as follows:

**Dense** The matrix is stored in a two dimensional array in a straightforward fashion. Using the options `UPL0='L'` or `UPL0='U'`, the lower triangular or upper triangular part respectively, do not need to be referenced.

**Banded** The matrix is also stored in a two dimensional array following the banded LAPACK-type storage:

$$\mathbf{A} = \begin{pmatrix} * & a_{12} & a_{23} & a_{34} \\ a_{11} & a_{22} & a_{33} & a_{44} \\ a_{21} & a_{32} & a_{43} & * \end{pmatrix}$$

In contrast to LAPACK, no extra-storage space is necessary since  $\text{LDA} \geq \text{k1} + \text{ku} + 1$  if `UPL0='F'` (LAPACK banded storage would require  $\text{LDA} \geq 2 * \text{k1} + \text{ku} + 1$ ). For this example, the number of subdiagonals  $\text{k1}$  and superdiagonals  $\text{ku}$  are both equal to 1. Using the option `UPL0='L'` or `UPL0='U'`, the rows respectively above or below the diagonal elements row, do not need to be referenced (or stored).

**Sparse-CSR** The non-zero elements of the matrix are stored using a set of one dimensional arrays ( $\mathbf{A}$ ,  $\mathbf{IA}$ ,  $\mathbf{JA}$ ) following the definition of the CSR (Compressed Sparse Row) format

$$\begin{aligned} \mathbf{A} &= (a_{11}, a_{12}, a_{21}, a_{22}, a_{23}, a_{32}, a_{33}, a_{34}, a_{43}, a_{44}) \\ \mathbf{IA} &= (1, 3, 6, 9, 11) \\ \mathbf{JA} &= (1, 2, 1, 2, 3, 2, 3, 4, 3, 4) \end{aligned}$$

Using the option `UPL0='L'` or `UPL0='U'`, one would get respectively

$$\begin{aligned} \mathbf{A} &= (a_{11}, a_{21}, a_{22}, a_{32}, a_{33}, a_{43}, a_{44}) & \mathbf{A} &= (a_{11}, a_{12}, a_{22}, a_{23}, a_{33}, a_{34}, a_{44}) \\ \mathbf{IA} &= (1, 2, 4, 6, 8) & \mathbf{IA} &= (1, 3, 5, 7, 8) \\ \mathbf{JA} &= (1, 1, 2, 2, 3, 3, 4) & \mathbf{JA} &= (1, 2, 2, 3, 3, 4, 4) \end{aligned}$$

### 4.2 Search contour

Figure 2 summarizes the different search contour options possible for both the Hermitian and non-Hermitian (including Polynomial) FEAST algorithms.

For the Hermitian case, the user must then specify a 1-dimensional real-valued search interval  $[E_{min}, E_{max}]$ . These two points are used to define a circular or ellipsoid contour  $\mathcal{C}$  centered on the real axis, and along which the complex integration nodes are generated. The choice of a particular quadrature rule will lead to a different set of relative positions for the nodes and associated quadrature weights. Since the eigenvalues are real, it is convenient to select a symmetric contour with the real axis ( $\mathcal{C} = \mathcal{C}^*$ ) since it only requires to operate the quadrature on the half-contour (e.g. upper half).

With a non-Hermitian/Polynomial problem, it is necessary to specify a 2-dimensional search interval that surrounds the wanted complex eigenvalues. Circular or ellipsoid contours can also be used and they can be generated using standard options included into FEAST v4.0. These are defined by a complex midpoint  $E_{mid}$

and a radius  $r$  for a circle (for an ellipse the ratio between the horizontal axis  $2r$  and vertical axis can also be specified, as well as an angle of rotation). A “Custom Contour” feature is also supported that can use arbitrary quadrature nodes and weights (provided by the users).

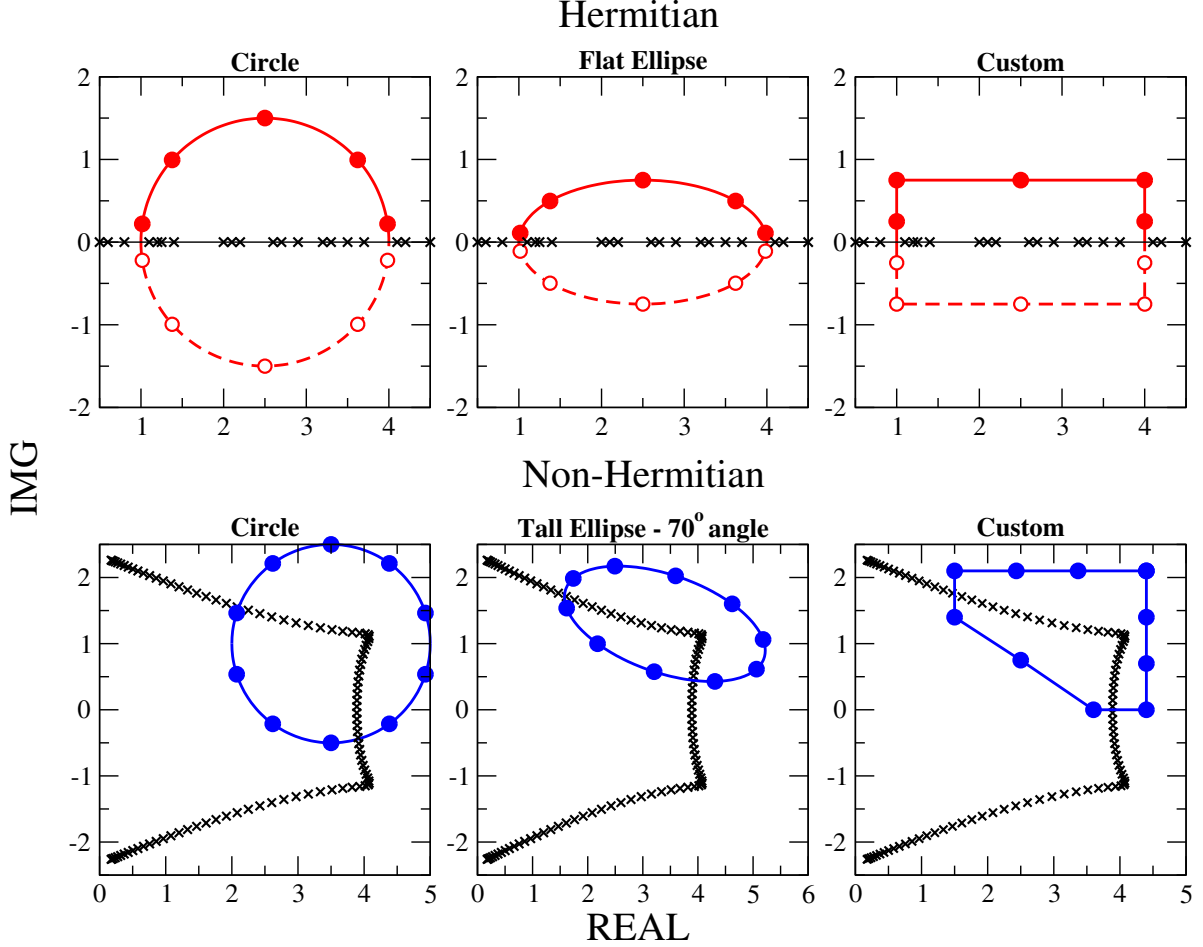


Figure 2: Various search contour examples for the Hermitian and the non-Hermitian/Polynomial FEAST algorithms. Both algorithms feature standard ellipsoid contour options and the possibility to define custom arbitrary shapes. In the Hermitian case, the contour is symmetric with the real axis and only the nodes in the upper-half may be generated. In the non-Hermitian/Polynomial case, a full contour is needed to enclose the wanted complex eigenvalues. Some data used to generate these plots:

**Hermitian case:**  $\text{fpm}(2)=5$  for all,  $[E_{\min}, E_{\max}] = [1, 4]$ ,  $r = 1.5$  for all;  $\text{fpm}(18)=50$  for the flat ellipse; and expert routine for the custom contour

**Non-Hermitian/Polynomial case:**  $\text{fpm}(8)=10$  for all;  $E_{\text{mid}} = 3.5 + i$  and  $r = 1.5$  for circle;  $E_{\text{mid}} = 3.4 + 1.3i$ ,  $r = 0.75$ ,  $\text{fpm}(18)=200$ ,  $\text{fpm}(19)=70$  for tall rotated ellipse; and expert routine for the custom contour

The FEAST package provides a couple of utility routines that can return the integration nodes and weight used by the FEAST interfaces:

`zfeast_contour(Emin, Emax, fpm2, fpm16, fpm18, Zne, Wne)`

Returns FEAST integration nodes and weights for a half-contour defined by  $E_{\min}$  and  $E_{\max}$ . To be used with Hermitian FEAST interfaces.

`zfeast_gcontour(Emid, r, fpm8, fpm16, fpm18, fpm19, Zne, Wne)`

- Returns FEAST integration nodes and weights for a full contour defined by  $E_{\text{mid}}$  and  $r$ . To be used with non-Hermitian and polynomial FEAST interfaces.

The description of the arguments list for these routines is given in Table 9 and Table 10.

|                   | Type           | I/O | Description                                            |
|-------------------|----------------|-----|--------------------------------------------------------|
| <b>Emin, Emax</b> | double real    | in  | Lower and Upper bounds of search interval              |
| <b>fpm2</b>       | integer        | in  | Value of <b>fpm(2)</b> - #contour point (half-contour) |
| <b>fpm16</b>      | integer        | in  | Value of <b>fpm(16)</b> - Integration type             |
| <b>fpm18</b>      | integer        | in  | Value of <b>fpm(18)</b> - Ellipse definition           |
| <b>Zne</b>        | double complex | out | Integration nodes                                      |
| <b>Wne</b>        | double complex | out | Integration weights                                    |

Table 9: List of arguments for **zfeast\_contour**.

|              | Type           | I/O | Description                                            |
|--------------|----------------|-----|--------------------------------------------------------|
| <b>Emid</b>  | double complex | in  | Coordinate center of the contour ellipse               |
| <b>r</b>     | double real    | in  | Horizontal radius of the contour ellipse               |
| <b>fpm8</b>  | integer        | in  | Value of <b>fpm(8)</b> - #contour point (full-contour) |
| <b>fpm16</b> | integer        | in  | Value of <b>fpm(16)</b> - Integration type             |
| <b>fpm18</b> | integer        | in  | Value of <b>fpm(18)</b> - Ellipse definition           |
| <b>fpm19</b> | integer        | in  | Value of <b>fpm(19)</b> - Ellipse rotation angle       |
| <b>Zne</b>   | double complex | out | Integration nodes                                      |
| <b>Wne</b>   | double complex | out | Integration weights                                    |

Table 10: List of arguments for **zfeast\_gcontour**.

### 4.3 Contour Customization

The Custom Contour feature grants the flexibility to target specific eigenvalues in a complex plane. This feature must be used with “Expert” routines that take two additional arguments containing the complex integration nodes and weights. Custom contours can be employed by following three simple steps:

1. Define a contour (half-contour that encloses  $[\lambda_{min}, \lambda_{max}]$  for the Hermitian problem, or full contour for the non-Hermitian/Polynomial problem),
2. Calculate corresponding integration nodes and weights, and
3. Call “Expert” FEAST routine (either Driver or RCI interfaces by adding a **x** at the end of the routine name).

Furthermore, the FEAST package provides a utility routine **zfeast\_customcontour** that can assist the user to extract nodes and weights from a custom design arbitrary geometry in the complex plane (full-contour). Users must only define the geometry of their contour. The contour can be comprised of line segments and half ellipses. Two important points to note: (i) the actual contour will end up being a polygon defined by the integration points along the path, and (ii) only convex contours may be used. A geometry that contains **P** contour parts/pieces is defined using three arrays **Zedge**, **Tedge**, and **Nedge**. The interface is defined below and the description of the arguments list is given in Table 11.

**zfeast\_customcontour(Nc,P,Nedge,Tedge,Zedge,Zne,Wne)**

As an example, the following code will generate the corresponding complex contour.

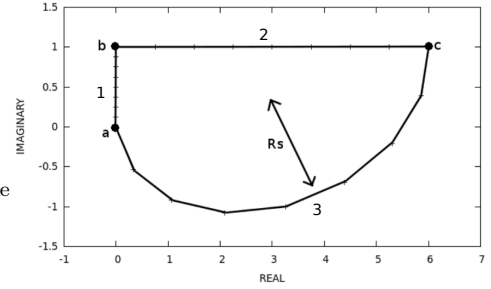
|              | Type           | I/O | Description                                                                                                                                                                                                                                                                  |
|--------------|----------------|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Nc</b>    | integer        | in  | The total number of integration nodes, should be equal to <b>SUM(Nedge(1:P))</b>                                                                                                                                                                                             |
| <b>P</b>     | integer        | in  | Number of contour parts/pieces that make up the contour                                                                                                                                                                                                                      |
| <b>Zedge</b> | integer(P)     | in  | Complex endpoints of each contour piece<br>Remark: * endpoints positioned in clockwise direction<br>* the $k^{th}$ piece is [Zedge(k), Zedge(k + 1)]<br>* last piece is [Zedge(P), Zedge(1)]                                                                                 |
| <b>Tedge</b> | integer(P)     | in  | The type of each contour piece:<br>*If <b>Tedge(k)</b> =0, $k^{th}$ piece is a line<br>*If <b>Tedge(k)</b> >0, $k^{th}$ piece is a (convex) half-ellipse<br>with <b>Tedge(k)/100</b> = ratio $a/b$ and $a$ primary radius from the endpoints<br>Remark: 100 is a half-circle |
| <b>Nedge</b> | integer(P)     | in  | #integration intervals to consider for each piece<br>define the accuracy of the trapezoidal rule by piece for FEAST                                                                                                                                                          |
| <b>Zne</b>   | double complex | out | Custom integration nodes for FEAST                                                                                                                                                                                                                                           |
| <b>Wne</b>   | double complex | out | Custom integration weights for FEAST                                                                                                                                                                                                                                         |

Table 11: List of arguments for **zfeast\_customcontour**.

```

1 P = 3 ! number of pieces that make up the contour
2 allocate(Zedge(1:P), Nedge(1:P), Tedge(1:P))
3 Zedge = ((0.0d0,0.0d0),(0.0d0,1.0d0),(6.0d0,1.0d0)/)
4 Tedge(:) = (/0,0,50/) ! (line)--(line)--(half-circle)
5 Nedge(:) = (/8,8,8/) ! integration intervals by piece
6 Nc = sum(Nedge(1:P)) ! #contour points (here 24)
7 allocate(Zne(1:Nc), Wne(1:Nc))
8 call zfeast_customcontour(Nc,P,Nedge,Tedge,Zedge,Zne,Wne)
9 ! (Zne, Wne) are now defined and ready to use for FEAST

```



The **\$FEASTROOT/example/FEAST** directory provides Fortran and C implementation of the expert FEAST routines using a custom contour. It is applied on the System4 example using both dense, banded and sparse-CSR storage. Here, the complete list of routines:

| System4 |                             |
|---------|-----------------------------|
| dense   | {F90,C}dense_zfeast_syevx   |
| banded  | {F90,C}dense_zfeast_sbevz   |
| sparse  | {F90,C}dense_zfeast_scsrevx |

#### 4.4 FEAST utility sparse drivers

If a sparse matrix can be provided by the user in coordinate/matrix market format, the `$FEASTROOT/utility` directory offers a quick way to test all the FEAST parameter options and the efficiency/reliability/timing of the FEAST SPARSE driver interfaces. Two general drivers are provided for FEAST/IFEAST and PFEAST-/PIFEAST, named `driver_feast_sparse` or `driver_pfeast_sparse` in their respective subdirectories. The command “>make all” should compile the drivers.

If we denote `mytest` a generic name for the user’s eigenvalue system test  $\mathbf{Ax} = \lambda\mathbf{x}$  or  $\mathbf{Ax} = \lambda\mathbf{Bx}$ . You will need to create the following three files:

- `mytest.mtx` should contain the matrix  $\mathbf{A}$  in coordinate format; As a reminder, the coordinate format is defined row by row as

|      |      |              |             |
|------|------|--------------|-------------|
| N    | N    | NNz          |             |
| :    | :    | :            | :           |
| i    | j    | real(valj)   | img(valj)   |
| :    | :    | :            | :           |
| :    | :    | :            | :           |
| :    | :    | :            | :           |
| iNNZ | jNNZ | real(valNNZ) | img(valNNZ) |

with N: size of matrix, and NNZ: number of non-zero elements.

- `mytestB.mtx` should contain the matrix  $\mathbf{B}$  (if any) in coordinate format;
- `mytest.in` should contain the search interval, selected FEAST parameters, etc. The following `.in` file is given as a template example (here for solving a standard eigenvalue problem in double precision):

```
s ! s: symmetric, h: hermitian, g: general
g ! e=standard or g=generalized eigenvalue problem
d ! (d,z) precision i.e (double real, double complex)
F ! UPLO (L: lower, U: upper, F: full) for the coordinate format of matrices
0.18d0 ! Emin
1.00d0 ! Emax
25 ! M0 search subspace (M0>=M)
2 !!!!!!!! How many changes from default fpm(1,64) (use 1-64 indexing)
1 1 !fpm(1)=1 !example comments on/off (0,1)
2 4 !fpm(2)=4 !number of contour points
```

You may change any of the above options to fit your needs. For example, you could add as many `fpm` FEAST parameters as you wish. You can also use the flag `fpm(43)=1` to switch to IFEAST. In addition, the L or U options for UPLO give you the possibility to provide only the lower or upper triangular part of the matrices `mytest.mtx` and `mytestB.mtx` in coordinate format.

Finally results and timing can be obtained by running the FEAST sparse driver:

```
./driver_feast_sparse <PATH_TO_MYTEST>/mytest
```

In addition, all the eigenvalue solutions will be locally saved in the file `eig.out`.

For the PFEAST sparse drivers, a run would look like (other options could be applied):

```
mpirun -env MKL_NUM_THREADS <omp> -ppn 1 -n <nL1> ./driver_pfeast_sparse
 <PATH_TO_MYTEST>/mytest nL3
```

where `<nL1>` represents the total number of MPI processes to use for a single contour, and `nL3` is the number of MPI processes used for solving the linear systems. As a reminder, L3 uses MKL-Cluster-PARDISO with PFEAST, and PBicGStab with PIFEAST. The level L3 can also be threaded by setting

the shell variable `MKL_NUM_THREADS` equal to the desired number of threads. Make sure that `<omp>*<nL1>` does not exceed the number of physical cores. Several combinations of `<nL1>`, `<nL3>` and `<omp>` are possible depending also on the value of the `-ppn` directive.

In order to illustrate a direct use of the utility drivers, several examples are provided in the directory `$FEASTROOT/utility/data` summarized in Table 12.

|            | Real | Complex | Symmetric | Hermitian | General | Standard | Generalized |
|------------|------|---------|-----------|-----------|---------|----------|-------------|
| helloworld | X    |         | X         |           |         | X        |             |
| system1    | X    |         | X         |           |         |          | X           |
| system2    |      | X       |           | X         |         | X        |             |
| system3    | X    |         |           |           | X       |          | X           |
| system4    |      | X       | X         |           |         | X        |             |
| cnt        | X    |         | X         |           |         |          | X           |
| co         | X    |         | X         |           |         |          | X           |
| c6h6       | X    |         | X         |           |         |          | X           |
| Na5        | X    |         | X         |           |         | X        |             |
| grcar      | X    |         |           |           | X       | X        |             |
| qc324      |      | X       | X         |           |         | X        |             |
| bcsstk11   | X    |         | X         |           |         |          | X           |

Table 12: List of system matrices provided in the `$FEASTROOT/utility/data` directory. System 1 to 4 corresponds to the matrices used in the `example` directory.

To run a specific test, you can execute using the FEAST driver (for example):

```
./driver_feast_sparse ../data/cnt
```

or using the PFEAST driver (for example):

```
mpirun -env MKL_NUM_THREADS 2 -n 4 ./driver_pfeast_sparse ../data/cnt 2
```

Here we use 8 total compute cores and `nL1=4` MPI, `nL2=nL1/nL3=2` MPI, `nL3=2` MPI, `omp=2` threads.