High-Performance Numerical Library for Solving Eigenvalue Problems

FEAST Eigenvalue Solver v2.1

User Guide

http://www.ecs.umass.edu/∼polizzi/feast

Eric Polizzi

Department of Electrical and Computer Engineering,
University of Massachusetts, Amherst

References

If you are using FEAST, please cite the following publication:

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

Contact

If you have any questions or feedback regarding FEAST, please contact the author by sending an-email to **feastsolver@gmail.com**.

Acknowledgments

The author thanks helpful technical discussions and inputs from Dr. Sergey Kuznetsov and Dr. Peter Tang. This work has been partially supported by Intel Corporation and by the National Science Foundation (NSF) under Grant #ECCS 0846457.

Contents

Chapter 1

Updates/Upgrades Summary

If you are a FEAST's first time user, you can skip this section. For each release, a list of updates/upgrades can be found in the current file `$FEASTROOT/UPDATE`. Here we propose to summarize the most important ones.

1.1 From v2.0 to v2.1

- No new major functionalities were added in 2.1, however it is using a more stable version of the algorithm. In some rare cases, it has been found that the basic FEAST algorithm may return few spurious solutions within the search interval. Although, those can be flagged a posteriori by computing the residual, an efficient and on-the-fly solution has been implemented in 2.1 based on a new in-depth numerical analysis of FEAST which can now be found in:

  "Subspace iteration with Approximate Spectral Projection",

  P. Tang, E. Polizzi, http://arxiv.org/abs/1302.0432, 2013

- Two types of convergence criteria are now possible: using the trace of the eigenvalues or the relative residual of the eigenpairs. Consequently:
  - the definition of the flag `fpm(6)` has changed- it is now associated with the type of convergence criteria.
  - the description of the argument "epsout" (error on the trace) and "res" (residual value) have also been modified.
- Finally, FEAST v2.1 (SMP version only) has been directly integrated within Intel-MKL v11.2.

1.2 From v1.0 to v2.0

- **FEAST-MPI** is now also available enabling up to three levels of parallelism: MPI for the user defined search interval, MPI for the contour points (i.e. shifts), OpenMP (shared-memory) for the linear system solver.
- **ATTENTION: v2.0 and v1.0 are not backward compatible**:
  - The linking options have changed and it is now mandatory to link the FEAST kernel (now named `-lfeast` or `-lpfeast`) along with other FEAST predefined interfaces library. Also new include files and include PATH are mandatory for `C` codes.
  - **All the interface names have changed** to anticipate future additional features. Since the arguments list has not changed, the migration from v1.0 to v2.0 should be straightforward using Table 1.1.
  - Definition of "case" while using FEAST-RCI interfaces has changed (see Figure 1.1)
- New options available for source code compilation and compatibilities. Two FEAST libraries available (using the same source code): -lfeast, etc. for SMP, and -lpfeast, etc. using MPI.
- Names of the drivers in `/examples` have changed (now consistent with the new FEAST interface names). New Fortran-MPI and C-MPI directories have been added for `/examples` and `/utility`. MPI examples also include new `3pdriver_zfeast_...` drivers which enable three levels parallelism.
- The FEAST-sparse interfaces need to be linked with MKL-PARDISO v10.3 (update 5 and beyond) which features the transpose option for PARDISO. As a result, the complex Hermitian problem would be solved up to x2 times faster (as compared to v1.0).
- FEAST PARAMETERS- Definition of `fpm` (==`feastparam`) updated for parameters:
  - `fpm(9)` (new for MPI option) contains the user defined MPI communicators for a given search interval (MPI_COMM_WORLD by default)
  - `fpm(10)` is removed
  - `fpm(11)` is removed – simplify transpose option in the RCI calls
  - `fpm(14)` is added – possibility to return only subspace Q size M0 after 1 contour

```
 - case(10), FACT of zB-A
 - case(11), SOLVE of (zB-A)*x=work
 - case(20), FACT of (zB-A)^T (* optional if case 21 can be handled by user
                                 using directly FACT of zB-A,
                               * otherwise the user is responsible to create
                                 a new matrix (zB-A)^T and then factorize it)
 - case(21), SOLVE of (zB-A)^T*x=work

 - case(30), A*x multiplication from column of going from fpm(24) to
                                       fpm(24)+fpm(25)-1 --(size fpm(25))
 - case(40), B*x multiplication from column of going from fpm(24) to
                                       fpm(24)+fpm(25)-1 --(size fpm(25))
```

Figure 1.1: Quick overview for the changes done in FEAST-RCI

v1.0	**v2.0**
RCI- interfaces	
{s,d}feast_rci	{s,d}feast_srci
{c,z}feast_rci	{c,z}feast_hrci
DENSE- interfaces	
{s,d}feast_dst	{s,d}feast_syev
{s,d}feast_dge	{s,d}feast_sygv
{c,z}feast_dst	{c,z}feast_heev
{c,z}feast_dge	{c,z}feast_hegv
BANDED- interfaces	
{s,d}feast_bst	{s,d}feast_sbev
{s,d}feast_bge	{s,d}feast_sbgv
{c,z}feast_bst	{c,z}feast_hbev
{c,z}feast_bge	{c,z}feast_hbgv
SPARSE- interfaces	
{s,d}feast_sst	{s,d}feast_scsrev
{s,d}feast_sge	{s,d}feast_scsrgv
{c,z}feast_sst	{c,z}feast_hcsrev
{c,z}feast_sge	{c,z}feast_hcsrgv

Table 1.1: New names for the FEAST interfaces from v1.0 to v2.0 (argument list of each interface is unchanged)

Chapter 2

Preliminary

2.1 The FEAST Algorithm

The FEAST solver package is a free high-performance numerical library for solving the standard, $\mathbf{Ax} = \lambda\mathbf{x}$, or generalized, $\mathbf{Ax} = \lambda\mathbf{Bx}$, eigenvalue problem, and obtaining all the eigenvalues λ and eigenvectors $\mathbf{x}$ within a given search interval $[\lambda_{min}, \lambda_{max}]$. It is based on an innovative fast and stable numerical algorithm presented in [1] – named the FEAST algorithm – which deviates fundamentally from the traditional Krylov subspace iteration based techniques (Arnoldi and Lanczos algorithms) or other Davidson-Jacobi techniques. The FEAST algorithm takes its inspiration from the density-matrix representation and contour integration technique in quantum mechanics. It is free from orthogonalization procedures, and its main computational tasks consist of solving very few inner independent linear systems with multiple right-hand sides and one reduced eigenvalue problem orders of magnitude smaller than the original one. The FEAST algorithm combines simplicity and efficiency and offers many important capabilities for achieving **high performance, robustness, accuracy, and scalability** on parallel architectures. This algorithm is expected to significantly augment numerical performances in large-scale modern applications. An in-depth numerical analysis of the algorithm can now be found in [2].

Some of the important capabilities and properties of the FEAST algorithm described in [1] and [2] can be briefly outlined as follows:

- Fast convergence - FEAST converges in 2-3 iterations with very high accuracy
- Naturally captures all multiplicities
- No-(explicit) orthogonalization procedure
- Reusable subspace - can benefit from suitable initial guess for solving series of eigenvalue problems that are close one another
- Ideally suited for large sparse systems- allows the use of iterative methods
- Three levels of parallelism

Denoting N the size of the system, $[\lambda_{\rm min}, \lambda_{\rm max}]$ the search interval, $\rm N_e$ the number of points for the Gauss-Legendre quadrature, $\rm M_0$ the size of the working subspace (where $\rm M_0$ is chosen greater that the number of eigenvalues M in the search interval), the basic pseudocode for the FEAST algorithm is given in Figure 2.1 in the case of real symmetric and complex Hermitian generalized eigenvalue problems. The performances of the basic FEAST algorithm will depend on a trade off between the choices of the number of Gauss quadrature points $\rm N_e$, the size of the subspace $\rm M_0$, and the number of outer refinement loops to reach the desired accuracy. *Using* $\rm M_0 \geq 1.5M$, $\rm Ne = 8$ *to* 16*, and with* ~ 2 *refinement loops, FEAST is most likely to produce a relative residual equal or smaller than* 10^{-10} *seeking up to* 1000 *eigenpairs in a given interval.*

```
1- Select $\mathrm{M}_0 > \mathrm{M}$ random vectors $\mathbf{Y}_{\mathrm{N}\times\mathrm{M}_0} \in \mathbb{R}^{\mathrm{N}\times\mathrm{M}_0}$
2- Set $\mathbf{Q} = 0$ with $\mathbf{Q} \in \mathbb{R}^{\mathrm{N}\times\mathrm{M}_0}$; $r = (\lambda_{\max} - \lambda_{\min})/2$;
   For $e = 1, \dots \mathrm{N_e}$
   compute $\theta_e = -(\pi/2)(x_e - 1)$,
   compute $\mathrm{Z_e} = (\lambda_{\max} + \lambda_{\min})/2 + r\exp(\imath\theta_e)$,
   solve $(\mathrm{Z_e}\mathbf{B} - \mathbf{A})\mathbf{Q_e} = \mathbf{Y}$ to obtain $\mathbf{Q_e} \in \mathbb{C}^{\mathrm{N}\times\mathrm{M}_0}$
   compute $\mathbf{Q} = \mathbf{Q} - (\omega_e/2)\Re\{r\exp(\imath\theta_e)\ \mathbf{Q_e}\}$
   End
3- Form $\mathbf{A_Q}_{\mathrm{M}_0\times\mathrm{M}_0} = \mathbf{Q^T A Q}$ and $\mathbf{B_Q}_{\mathrm{M}_0\times\mathrm{M}_0} = \mathbf{Q^T B Q}$
    rescale value of $\mathrm{M}_0$ if $\mathbf{B_Q}$ is not spd.
4- Solve $\mathbf{A_Q\Phi} = \epsilon\mathbf{B_Q\Phi}$ to obtain the $\mathrm{M}_0$ eigenvalue $\epsilon_\mathbf{m}$,
   and eigenvectors $\mathbf{\Phi}_{\mathrm{M}_0\times\mathrm{M}_0} \in \mathbb{R}^{\mathrm{M}_0\times\mathrm{M}_0}$
5- Set $\lambda_m = \epsilon_m$ and compute $\mathbf{X}_{\mathrm{N}\times\mathrm{M}_0} = \mathbf{Q}_{\mathrm{N}\times\mathrm{M}_0}\mathbf{\Phi}_{\mathrm{M}_0\times\mathrm{M}_0}$
   If $\lambda_\mathrm{m} \in [\lambda_{\min}, \lambda_{\max}]$, $\lambda_\mathrm{m}$ is an eigenvalue solution
   and its eigenvector is $\mathbf{X_m}$ (the $\mathrm{m^{th}}$ column of $\mathbf{X}$).
6- Check convergence for the trace of the eigenvalues $\lambda_m$
  If iterative refinement is needed, compute $\mathbf{Y} = \mathbf{BX}$
  and go back to step 2
```

```
1- Select $\mathrm{M}_0 > \mathrm{M}$ random vectors $\mathbf{Y}_{\mathrm{N}\times\mathrm{M}_0} \in \mathbb{C}^{\mathrm{N}\times\mathrm{M}_0}$
2- Set $\mathbf{Q} = 0$ with $\mathbf{Q} \in \mathbb{C}^{\mathrm{N}\times\mathrm{M}_0}$; $r = (\lambda_{\max} - \lambda_{\min})/2$;
   For $e = 1, \dots \mathrm{N_e}$
   compute $\theta_e = -(\pi/2)(x_e - 1)$,
   compute $\mathrm{Z_e} = (\lambda_{\max} + \lambda_{\min})/2 + r\exp(\imath\theta_e)$,
   solve $(\mathrm{Z_e}\mathbf{B} - \mathbf{A})\mathbf{Q_e} = \mathbf{Y}$ to obtain $\mathbf{Q_e} \in \mathbb{C}^{\mathrm{N}\times\mathrm{M}_0}$
   solve $(\mathrm{Z_e}\mathbf{B} - \mathbf{A})^\dagger\hat{\mathbf{Q}}_\mathbf{e} = \mathbf{Y}$ to obtain $\hat{\mathbf{Q}}_\mathbf{e} \in \mathbb{C}^{\mathrm{N}\times\mathrm{M}_0}$
    $\mathbf{Q} = \mathbf{Q} - (\omega_e/4)r\left(\exp(\imath\theta_e)\ \mathbf{Q_e} + \exp(-\imath\theta_e)\ \hat{\mathbf{Q}}_\mathbf{e}\right)$
   End
3- Form $\mathbf{A_Q}_{\mathrm{M}_0\times\mathrm{M}_0} = \mathbf{Q^T A Q}$ and $\mathbf{B_Q}_{\mathrm{M}_0\times\mathrm{M}_0} = \mathbf{Q^T B Q}$
    rescale value of $\mathrm{M}_0$ if $\mathbf{B_Q}$ is not hpd.
4- Solve $\mathbf{A_Q\Phi} = \epsilon\mathbf{B_Q\Phi}$ to obtain the $\mathrm{M}_0$ eigenvalue $\epsilon_\mathbf{m}$,
   and eigenvectors $\mathbf{\Phi}_{\mathrm{M}_0\times\mathrm{M}_0} \in \mathbb{C}^{\mathrm{M}_0\times\mathrm{M}_0}$
5- Set $\lambda_m = \epsilon_m$ and compute $\mathbf{X}_{\mathrm{N}\times\mathrm{M}_0} = \mathbf{Q}_{\mathrm{N}\times\mathrm{M}_0}\mathbf{\Phi}_{\mathrm{M}_0\times\mathrm{M}_0}$
   If $\lambda_\mathrm{m} \in [\lambda_{\min}, \lambda_{\max}]$, $\lambda_\mathrm{m}$ is an eigenvalue solution
   and its eigenvector is $\mathbf{X_m}$ (the $\mathrm{m^{th}}$ column of $\mathbf{X}$).
6- Check convergence for the trace of the eigenvalues $\lambda_m$
  If iterative refinement is needed, compute $\mathbf{Y} = \mathbf{BX}$
  and go back to step 2
```

Figure 2.1: FEAST basic pseudocode for solving the generalized eigenvalue problem $\mathbf{Ax} = \lambda\mathbf{Bx}$ and obtaining all the M eigenpairs within a given interval $[\lambda_{\min}, \lambda_{\max}]$. On the left, $\mathbf{A}$ is real symmetric and $\mathbf{B}$ is symmetric positive definite (spd), on the right: $\mathbf{A}$ is complex Hermitian, $\mathbf{B}$ is Hermitian positive definite (hpd). To perform the numerical integration, we have been using $\mathrm{N_e}$-point Gauss-Legendre quadrature with x_e the e^{th} Gauss node associated with the weight ω_e. For example, for the case $\mathrm{N_e} = 8$, one can use:
$(x_1, \omega_1) = (0.183434642495649, 0.362683783378361)$,
$(x_3, \omega_3) = (0.525532409916328, 0.313706645877887)$,
$(x_5, \omega_5) = (0.796666477413626, 0.222381034453374)$,
$(x_7, \omega_7) = (0.960289856497536, 0.101228536290376)$, and $(x_{2i}, \omega_{2i})_{i=1,\dots,4} = (-x_{2i-1}, \omega_{2i-1})$

2.2 The FEAST Solver Package version v2.1

This general purpose FEAST solver package includes both reverse communication interfaces and ready to use predefined interfaces (i.e. drivers) for dense, banded and sparse systems. It includes double and single precision arithmetic and can be called from Fortran (77,90), or C codes, with or without an MPI environment.

2.2.1 Current Features

The current version of the package focuses on solving the symmetric eigenvalue problems (real symmetric or complex Hermitian systems) on both shared-memory (SMP) architectures and distributed memory environment (using MPI).

The current features of the package are summarized as follows:

- Solving $\mathbf{Ax} = \lambda\mathbf{x}$ or $\mathbf{Ax} = \lambda\mathbf{Bx}$, where $\mathbf{A}$ is real symmetric or complex Hermitian, $\mathbf{B}$ is symmetric or Hermitian positive definite
- Two libraries: **SMP version** (one node), and **MPI version** (multi-nodes).
- Real/Complex and Single/Double precisions
- All FEAST interfaces compatible with Fortran (77 and 90) and C. All the FEAST interfaces require (any) BLAS and LAPACK packages.
- Source code and pre-compiled libraries provided for common architectures (x64).
- Reverse communication interfaces (RCI): **Maximum flexibility for application specific**. *Those are matrix format independent, inner system solver independent, so users must provide their own linear system solvers (direct or iterative) and mat-vec utility routines.*
- Predefined driver interfaces for dense, banded, and sparse (CSR) formats: **Less flexibility but easy to use ("plug and play"):**
  - FEAST_DENSE interfaces requires the LAPACK library. *In the SMP version the DENSE interfaces are not intended for performances, however, scalability can be obtained using the MPI version.*
  - FEAST_BANDED interfaces use the SPIKE[1] banded primitives (included)
  - FEAST_SPARSE interfaces requires the Intel MKL-PARDISO solver.
- Examples and documentation included,
- FEAST utility drivers for sparse systems included: users can provide directly their sparse systems for quick testing, timing, etc.

2.2.2 FEAST and Dependencies

As specified above:

- the libraries LAPACK/BLAS (not provided) are needed for all FEAST interfaces,
- the library MKL-PARDISO **from v10.3 -update 5- and beyond** (not provided) is needed for the predefined FEAST_SPARSE interfaces.

Remark: Although it could be possible to develop a large collection of FEAST drivers that can be linked with all the main linear system solver packages, we are rather focusing our efforts on the development of highly efficient, functional and flexible FEAST RCI interfaces which are placed on top of the computational hierarchy (see Figure 2.2). Within the FEAST RCI interfaces, maximum flexibility is indeed available to the users for choosing their preferred and/or application specific direct linear system method or iterative method with (customized or generic) preconditioner.

[1] *http://software.intel.com/en-us/articles/intel-adaptive-spike-based-solver/*

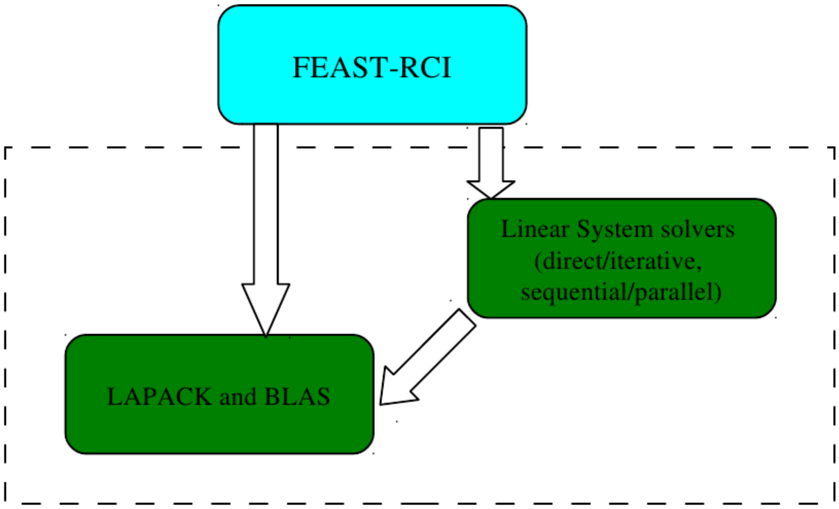
FEAST-RCI
Linear System solvers (direct/iterative, sequential/parallel)
LAPACK and BLAS

Figure 2.2: The FEAST-RCI interfaces at the top of a hierarchy of computational modules. FEAST-RCI is designed to be completely independent of the matrix formats, the linear system solvers and matrix-vector multiplication routines.

2.2.3 FEAST and Parallelism

Efficient parallel implementations for FEAST can be addressed at three different levels (represented also in Figure 2.3):

1. many search intervals can be run independently (no overlap),

2. each linear system can be solved independently (e.g. simultaneously on different compute nodes),

3. the linear systems can be solved in parallel (the multiple right sides can be parallelized as well).

Depending on the parallel architecture at hand, the local memory of a given node, and the properties of the matrices of the eigenvalue problems, one may preferably select one parallel option among the others or just take advantage of a combination of those. Ultimately, one can show that if enough parallel computing power is available at hand, the main computational cost of FEAST for solving the eigenvalue problem can be reduced to solving only one linear system.

Using the FEAST-SMP version

For a given search interval, parallelism (via shared memory programming) is not explicitly implemented in FEAST i.e. the inner linear systems are solved one after another within one node (avoid the fight for resources/save memory). Therefore, parallelism can be only achieved if the inner system solver and the mat-vec routines are threaded. Using the FEAST drivers, in particular, parallelism is implicitly obtained within the shared memory version of BLAS, LAPACK or MKL-PARDISO. If FEAST is linked with the INTEL-MKL library, the shell variable `MKL_NUM_THREADS` can be used for setting automatically the number of threads (cores) for both BLAS, LAPACK and MKL-PARDISO. In the general case, the user is responsible for activated the threaded capabilities of their BLAS, LAPACK libraries and their own linear systems solvers for the FEAST-RCI users (most likely using the shell variable `OMP_NUM_THREADS`).

Remark: If memory resource is not an issue (in particular for small systems), the second level parallelism of FEAST could potentially be implemented within SMP where the linear systems along

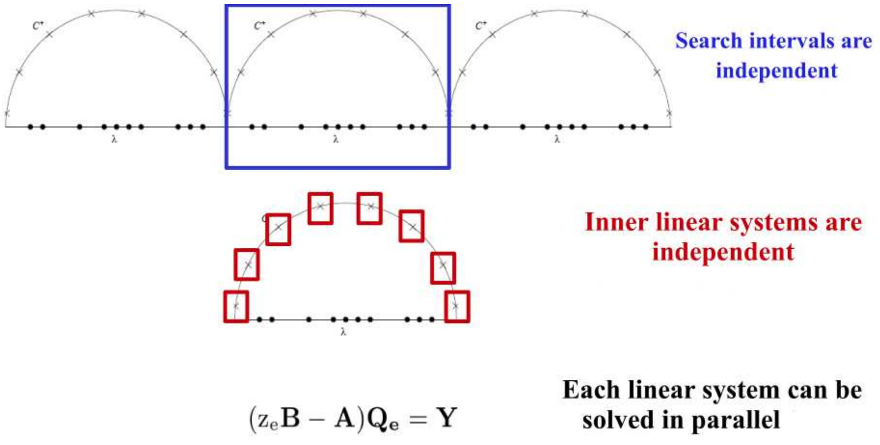
Search intervals are independent
Inner linear systems are independent
Each linear system can be solved in parallel
$(z_e\mathbf{B} - \mathbf{A})\mathbf{Q_e} = \mathbf{Y}$
λ
λ
λ
λ

Figure 2.3: Three-levels of parallelism possible in FEAST. Currently available: MPI- calling - MPI - calling - OpenMP (i.e shared memory linear system solver and mat-vec routine)

the contour are kept in memory. This option is not currently offered as it would require nested Open-MP loops that are difficult to generalize via RCI interfaces independently of the BLAS/LAPACK libraries. Nevertheless, it is important to mention that the scalability performances for the FEAST dense interface could potentially become competitive as compared to the LAPACK eigenvalue routines for a given search interval and on multicore systems (including large number of cores). To fix the ideas, similar scalability performances can currently be obtained if FEAST-MPI is used on a single node.

Using the FEAST-MPI version

The second-level of parallelism option is now addressed explicitly. This is accomplished in a trivial fashion by sending off the different linear systems (which can be solved independently for the points along the complex contour) along the compute nodes. Although, FEAST can run on any numbers of MPI processors, there will be a peak of performance if the number of MPI processes is equal to the number of contour points. Indeed, the MPI implementation is not yet provided as an option for the third level of parallelism (system solver level) and FEAST still needs to call a shared-memory solver. However, it is important to note that a MPI-level management has been added to allow easy parallelism of the search interval using a coarser level of MPI (i.e. first level parallelism). For this case, a new flag `fpm(9)` has been added as the only new input required by FEAST. This flag can be set equal to a new local communicator variable "MPI_COMM_WORLD" which contains the selected user's MPI processes for a given search interval. If only one search interval is used, this new flag is set by default to the global "MPI_COMM_WORLD" value in the FEAST initialization step. Another important point to mention is that the names of the FEAST interfaces (RCI and predefined drivers) stay completely unchanged (source code for FEAST-SMP and FEAST-MPI is identical– only few MPI directives have been added to the FEAST RCI interfaces which are activated using compiler directives). From the end user perspective, interfaces and arguments list stay unchanged and it is the `-lpfeast, etc.` library (instead of `-lfeast, etc.`) that needs to be linked within an MPI environment.

2.3 Installation and Setup: A Step by Step Procedure

In this section, we address the following question: How should you install and link the FEAST library?

2.3.1 Installation- Compilation

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

1. Download the latest FEAST version in **http://www.ecs.umass.edu/∼polizzi/feast**, for example, let us call this package `feast_2.1.tgz`.

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 -xzvf feast_2.1.tgz**; Figure 2.4 represents the main `FEAST` tree directory being created.

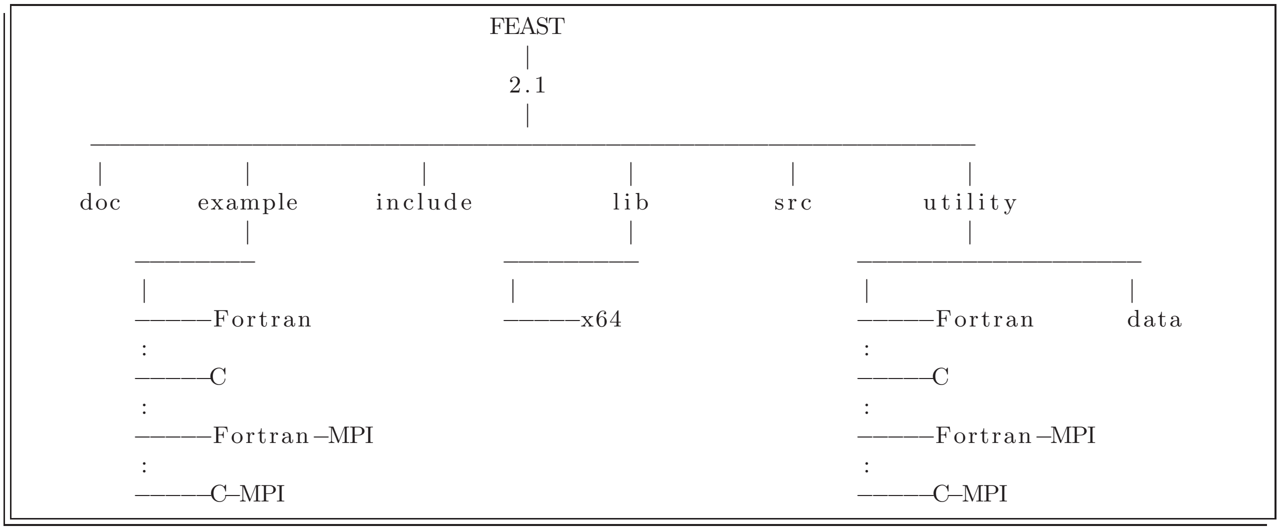

Figure 2.4: Main FEAST tree directory

4. let us denote <`FEAST` *directory*> the package's main directory after installation. For example, it could be

   ∼`/home/FEAST/2.1`   or   `/opt/FEAST/2.1`.

   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`).

5. The FEAST pre-compiled libraries can be found at

   `$FEASTROOT/lib/`<*arch*>

   where <*arch*> denotes the computer architecture. Currently, the following architectures are provided:

- `x64` for common 64 bits architectures (e.g. Intel em64t: Nehalem, Xeon, Pentium, Centrino etc.; amd64),

It should be noted that although FEAST is written in F90, the pre-compiled libraries are free from Fortran90 runtime dependencies (i.e. they can be called from any `Fortran` or `C` codes without compatibility issues). For the FEAST-MPI, the precompiled library include three versions compatible Intel-MPI, MPICH2 and OpenMPI. If your current architecture is listed above, you can proceed directly to step **7**, if not, you will need to compile the FEAST library in the next step. You would also need to compile FEAST-MPI if you are using a different MPI implementation that the one proposed here.

6. *Compilation of the FEAST library source code*:
   - Go to the directory `$FEASTROOT/src`
   - **Edit the `make.inc` file, and follow the directions.** Depending on your options, you would need to change appropriately the name/path of the `Fortran90` or/and `C Compilers` and optionally `MPI`.

     Two main options are possible:

     1- FEAST is written in `Fortran90` so direct compilation is possible using any `Fortran90` compilers (tested with `ifort` and `gfortran`). If this option is selected, users must then be aware of runtime dependency problems. For example, if the FEAST library is compiled using `ifort` but the user code is compiled using `gfortran` or `C` then the flag `-lifcoremt` should be added to this latter; In contrast, if the FEAST library is compiled using `gfortran` but the user code is compiled using `ifort` or `C`, the flag `-lgfortran` should be used instead.

     2- Compilation free from Fortran90 runtime dependencies (i.e. some low-level Fortran intrinsic functions are replaced by C ones). This is the best option since once compiled, the library could be called from any `Fortran` or `C` codes removing compatibility issues. This compilation can be performed using any `C` compilers (`gcc` for example), but it currently does require the use of the Intel Fortran Compiler.

     The same source codes are used for compiling FEAST-SMP and/or FEAST-MPI. For this latter, the MPI instructions are then activated by compiler directives (a flag `<-DMPI>` is added). The user is also free to choose any MPI implementations (tested with Intel-MPI, MPICH2 and OpenMPI).
   - For creating the FEAST-SMP: Execute:
     make ARCH=<*arch*> LIB=feast all
     where <*arch*> is your selected name for your architecture; your FEAST library including:
     `libfeast_sparse.a`
     `libfeast_banded.a`
     `libfeast_dense.a`
     `libfeast.a`
     will then be created in `$FEASTROOT/lib/`<*arch*> .
   - Alternatively, for creating the FEAST-MPI library: Execute:
     make ARCH=<*arch*> LIB=pfeast all to obtain:
     `libpfeast_sparse.a`
     `libpfeast_banded.a`
     `libpfeast_dense.a`
     `libpfeast.a`
     You may want to rename these libraries with a particular extension name associated with your MPI compilation.
7. Congratulations, FEAST is now installed successfully on your computer !!

2.3.2 Linking FEAST

In order to use the FEAST library for your `F77`, `F90`, `C` or `MPI` application, 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:* **(examples)**
  `-lfeast` (FEAST kernel alone - Reverse Communication Interfaces)
  `-lfeast_dense -lfeast` (FEAST dense interfaces)
  `-lfeast_banded -lfeast` (FEAST banded interfaces)
  `-lfeast_sparse -lfeast` (FEAST sparse interfaces)
  `-lfeast_sparse -lfeast_banded -lfeast` (FEAST sparse and banded interfaces)
- *for the LIBRARY LINKS using FEAST-MPI:***(examples)**
  `-lpfeast<ext>` (FEAST kernel alone - Reverse Communication Interfaces)
  `-lpfeast_dense<ext> -lpfeast<ext>` (FEAST dense interfaces)
  `-lpfeast_banded<ext> -lpfeast<ext>` (FEAST banded interfaces)
  `-lpfeast_sparse<ext> -lpfeast<ext>` (FEAST sparse interfaces)
  `-lpfeast_sparse<ext> -lpfeast_banded<ext> -lpfeast<ext>` (FEAST sparse and banded interfaces)
  where, in the precompiled library, `<ext>` is the extension name associated with `_impi`, `_mpich2` or `_openmpi` respectively for Intel MPI, MPICH2 and OpenMPI.

  In order to illustrate how should one use the above FEAST library links including dependencies, let us call (for example) `-lmklpardiso`, `-llapack`, `-lblas` respectively the library links to the MKL-PARDISO, LAPACK and BLAS packages. The complete library links with dependencies are then given for FEAST-SMP by **(examples)**:
  `-lfeast -llapack -lblas`
  `-lfeast_dense -lfeast -llapack -lblas`
  `-lfeast_banded -lfeast -llapack -lblas`
  `-lfeast_sparse -lfeast -lmklpardiso -llapack -lblas`
  Similarly, for FEAST-MPI, it comes **(examples)**:
  `-lpfeast<ext> -llapack -lblas`
  `-lpfeast_dense<ext> -lpfeast -llapack -lblas`
  `-lpfeast_banded<ext> -lpfeast -llapack -lblas`
  `-lpfeast_sparse<ext> -lpfeast -lmklpardiso -llapack -lblas`
- *for the INCLUDE PATH:* `-I/$(FEASTROOT)/include`
  It is mandatory only for `C` codes. Additionally, instructions need to be added in the header `C` file **(all that apply)**:

```
#include "feast.h"
#include "feast_sparse.h"
#include "feast_banded.h"
#include "feast_dense.h"
```

2.4 A simple "Hello World" Example (F90, C, MPI-F90, MPI-C)

This example solves a 2-by-2 dense standard eigenvalue system $\mathbf{Ax} = \lambda \mathbf{x}$ where

$$\mathbf{A} = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \tag{2.1}$$

and the two eigenvalue solutions are known to be $\lambda_1 = 1$, $\lambda_2 = 3$ which can be associated respectively with the orthonormal eigenvectors $\pm(\sqrt{2}/2, \sqrt{2}/2)$ and $\pm(\sqrt{2}/2, -\sqrt{2}/2)$.

Let us suppose that one can specify a search interval, a single call to the `DFEAST_SYEV` subroutine solves then this dense standard eigenvalue system in double precision. Also, the FEAST parameters can be set to their default values by a call to the `FEASTINIT` subroutine.

F90

The Fortran90 source code of `helloworld.f90` is listed in Figure 2.5. To create the executable, compile and link the source program with the FEAST v2.1 library, one can use:

```
 ifort helloworld.f90 -o helloworld \
-L<FEAST directory>/lib/<arch> -lfeast_dense -lfeast \
-Wl,--start-group -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -Wl,--end-group \
-liomp5 -lpthread
```

here, we assumed that: (i) the FORTRAN compiler is `ifort` the Intel's one, (ii) the BLAS and LAPACK libraries are obtained via Intel MKL v10.x, (iii) the FEAST package has been installed in a directory called `<FEAST` *directory*`>`, (iv) the user architecture is <*arch*> (x64, ia64, etc.), (iv) .

A run of the resulting executable looks like

```
> ./helloworld
```

and the output of the run appears in Figure 2.6.

C

Similarly to the F90 example, the corresponding C source code of the `helloworld` example (`helloworld.c`) is listed in Figure 2.7. The executable can now be created using the *gcc* compiler (for example), along with the `-lm` library:

```
 gcc helloworld.c -o helloworld \
-I<FEAST directory>/include -L<FEAST directory>/lib/<arch> -lfeast_dense -lfeast \
-Wl,--start-group -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -Wl,--end-group \
-liomp5 -lpthread -lm
```

here we assumed that FEAST has been compiled without runtime dependencies. In contrast, if the FEAST library was compiled using `ifort` alone then the flag `-lifcoremt` should be added above; In turn, if the FEAST library was compiled using `gfortran` alone, it is the flag `-lgfortran` that should be added instead.

MPI-F90

Similarly to the F90 example, the corresponding MPI-F90 source code of the `helloworld` example (`phelloworld.f90`) is listed in Figure 2.8. The executable can now be created using `mpif90` (for example):

```
 mpif90 -f90=ifort phelloworld.f90 -o phelloworld \
-L<FEAST directory>/lib/<arch> -lpfeast_dense_impi -lpfeast_impi \
-Wl,--start-group -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -Wl,--end-group \
-liomp5 -lpthread
```

where we assumed that: (i) the Intel Fortran compiler is used, (ii) the FEAST-MPI library has been compiled using the same MPI implementation (Intel MPI here).

A run of the resulting executable looks like

```
> mpirun − ppn 1 − n < x > ./phelloworld
```

where < x > represents the number of nodes.

MPI-C

Similarly to the MPI-F90 example, the corresponding MPI-C source code of the `helloworld` example (`phelloworld.c`) is listed in Figure 2.9. The executable can now be created using `mpicc` (for example):

```
mpicc -cc=gcc helloworld.f90 -o helloworld \
-L<FEAST directory>/lib/<arch> -lpfeast_dense_impi -lpfeast_impi \
-Wl,--start-group -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -Wl,--end-group \
-liomp5 -lpthread -lm
```

where we assumed that: (i) the gnu `C` compiler is used, (ii) the FEAST-MPI library has been compiled using the same MPI implementation (Intel MPI here), (iii) FEAST has been compiled without runtime dependencies (otherwise see comments in `C` example section).

```
program helloworld
  implicit none
  !! eigenvalue system
  integer,parameter :: N=2, LDA=2
  character(len=1),parameter :: UPLO='F' ! 'L' or 'U' also fine
  double precision,dimension(N*N) :: A=(/2.0d0,-1.0d0,-1.0d0,2.0d0/)
  double precision :: Emin=-5.0d0, Emax=5.0d0 ! search interval
  integer :: M0=2 ! (Initial) subspace dimension
  !! input parameters for FEAST
  integer,dimension(64) :: feastparam
  !! output variables for FEAST
  double precision,dimension(:),allocatable :: E, res
  double precision,dimension(:,:),allocatable :: X
  double precision :: epsout
  integer :: i,loop,info,M

!!! Allocate memory for eigenvalues.eigenvectors/residual
  allocate(E(M0))
  allocate(res(M0))
  allocate(X(N,M0))

!!!!!!!!!! FEAST
  call FEASTINIT(feastparam)
  feastparam(1)=1 !! change from default value
  call DFEAST_SYEV(UPLO,N,A,LDA,feastparam,epsout,loop,Emin,Emax,M0,&
                 & E,X,M,res,info)

!!!!!!!!!! REPORT
  print *,'FEAST OUTPUT INFO',info
  if (info==0) then
     print *,'*************************************************'
     print *,'************** REPORT ***************************'
     print *,'*************************************************'
     print *,'# Search interval [Emin,Emax]',Emin,Emax
     print *,'# mode found/subspace',M,M0
     print *,'# iterations',loop
     print *,'TRACE',sum(E(1:M))
     print *,'Relative error on the Trace',epsout
     print *,'Eigenvalues/Residuals'
     do i=1,M
        print *,i,E(i),res(i)
     enddo
     print *,'Eigenvectors'
     do i=1,M
        print *,i,"(",X(1,i),X(2,i),")"
     enddo
  endif

end program helloworld
```

Figure 2.5: A very simple F90 “helloworld” example. This code can be found in `<FEAST` *directory*`>` `/example/Fortran/1_dense`.

```
***********************************************
*********** FEAST- BEGIN *********************
***********************************************
Routine DFEAST_S{}{}
List of input parameters fpm(1:64)-- if different from default
   fpm(1)=1
Search interval [-5.000000000000000e+00; 5.000000000000000e+00]
Size subspace    2
#Loop  | #Eig  |       Trace              |      Error-Trace          |    Max-Residual
0         2        4.000000000000001e+00   1.000000000000000e+00   1.067662871904967e-15
1         2        4.000000000000000e+00   1.776356839400251e-16   3.140184917367550e-17
==>FEAST has successfully converged (to desired tolerance)
***********************************************
*********** FEAST- END************************
***********************************************

 FEAST OUTPUT INFO              0
 ************************************************
 ************** REPORT **************************
 ************************************************
 # Search interval [Emin,Emax]    -5.00000000000000           5.00000000000000
 # mode found/subspace              2             2
 # iterations              1
 TRACE    4.00000000000000
 Relative error on the Trace  1.776356839400251E-016
 Eigenvalues/Residuals
            1     1.00000000000000          3.140184917367550E-017
            2     3.00000000000000          0.000000000000000E+000
 Eigenvectors
            1  (  -0.707106781186548        -0.707106781186548        )
            2  (  -0.707106781186548         0.707106781186548        )
```

Figure 2.6: Output results for the “helloworld” example.

```
#include <stdio.h>
#include <stdlib.h>

#include ”feast.h”
#include ”feast_dense.h”
int main() {
  /*  eigenvalue system  */
  int   N=2,LDA=2;
  char  UPLO=’F’; // ’L’ and ’U’ also fine
  double A[2][2]={2.0,-1.0,-1.0,2.0};
  double Emin=-5.0, Emax=5.0;
  int M0=2; //size initial subspace
  /* input parameters for FEAST */
  int feastparam[64];
  /* output variables for FEAST */
  double *E, *res, *X;
  double  epsout,trace;
  int i,loop,info,M;

  /* Allocate memory for eigenvalues.eigenvectors/residual */
  E=calloc(M0,sizeof(double));  //eigenvalues
  res=calloc(M0,sizeof(double));//eigenvectors
  X=calloc(N*M0,sizeof(double));//residual (if needed)

  /* !!!!!!!!!! FEAST !!!!!!!!!*/
  FEASTINIT(feastparam);
  feastparam[0]=1;  /*change from default value */
  DFEAST_SYEV(&UPLO,&N,&A,&LDA,feastparam,&epsout,&loop,&Emin,&Emax,&M0,\
             E,X,&M,res,&info);

  /*!!!!!!!!!! REPORT !!!!!!!!!*/
  printf(”FEAST OUTPUT INFO %d\n”,info);
  if (info==0) {
    printf(”*************************************************\n”);
    printf(”************** REPORT ***************************\n”);
    printf(”*************************************************\n”);
    printf(”# Search interval [Emin,Emax] %.15e %.15e\n”,Emin,Emax);
    printf(”# mode found/subspace %d %d \n”,M,M0);
    printf(”# iterations %d \n”,loop);
    trace=0.0;
    for (i=0;i<=M-1;i=i+1){
      trace=trace+*(E+i);}
    printf(”TRACE %.15e\n”, trace);
    printf(”Relative error on the Trace %.15e\n”,epsout );
    printf(”Eigenvalues/Residuals\n”);
    for (i=0;i<=M-1;i=i+1){
      printf(”   %d %.15e %.15e\n”,i,*(E+i),*(res+i));
    }
    printf(”Eigenvectors\n”);
    for (i=0;i<=M-1;i=i+1){
      printf(”   %d (%.15e, %.15e)\n”,i,*(X+i*M),*(X+1+i*M));
    }
}
  return 0;}
```

Figure 2.7: A very simple C “helloworld” example. This code can be found in `<FEAST` *directory*`>` `/example/C/1_dense`.

```
program phelloworld
  implicit none
include 'mpif.h'      !!!!!!!!!! MPI
  !! eigenvalue system
  integer,parameter :: N=2,LDA=2
  character(len=1),parameter :: UPLO='F' ! 'L' or 'U' also fine
  double precision,dimension(N*N) :: A=(/2.0d0,-1.0d0,-1.0d0,2.0d0/)
  double precision :: Emin=-5.0d0, Emax=5.0d0 ! search interval
  integer :: M0=2 ! (Initial) subspace dimension
  !! input parameters for FEAST
  integer,dimension(64) :: feastparam
  !! output variables for FEAST
  double precision,dimension(:),allocatable :: E, res
  double precision,dimension(:,:),allocatable :: X
  double precision :: epsout
  integer :: i,loop,info,M
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! MPI!!!!!!!!!!!!!!!!!!!!
  integer :: code,rank,nb_procs
  call MPI_INIT(code)
  call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code)
  call MPI_COMM_RANK(MPI_COMM_WORLD,rank,code)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! Allocate memory for eigenvalues.eigenvectors/residual
  allocate(E(M0))
  allocate(res(M0))
  allocate(X(N,M0))

!!!!!!!!!! FEAST
  call FEASTINIT(feastparam)
  feastparam(1)=1 !! change from default value
  call DFEAST_SYEV(UPLO,N,A,LDA,feastparam,epsout,loop,Emin,Emax,M0,&
                  &E,X,M,res,info)
!!!!!!!!!! REPORT
if (rank==0) then
  print *,'FEAST OUTPUT INFO',info
  if (info==0) then
     print *,'*************************************************'
     print *,'************** REPORT ***************************'
     print *,'*************************************************'
     print *,'# processors',nb_procs
     print *,'# Search interval [Emin,Emax]',Emin,Emax
     print *,'# mode found/subspace',M,M0
     print *,'# iterations',loop
     print *,'TRACE',sum(E(1:M))
     print *,'Relative error on the Trace',epsout
     print *,'Eigenvalues/Residuals'
     do i=1,M
        print *,i,E(i),res(i)
     enddo
     print *,'Eigenvectors'
     do i=1,M
        print *,i,"(",X(1,i),X(2,i),")"
     enddo
  endif
endif
call MPI_FINALIZE(code) !!!!!! MPI
end program phelloworld
```

Figure 2.8: A very simple MPI-F90 “helloworld” example. This code can be found in `<FEAST` *directory*`> /example/Fortran-MPI/1_dense`.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include ”feast.h”
#include ”feast_dense.h”
int main(int argc, char **argv) {
  /*  eigenvalue system  */
  int    N=2,LDA=2;
  char  UPLO=’F’; // ’L’ and ’U’ also fine
  double A[4]={2.0,-1.0,-1.0,2.0};
  double Emin=-5.0, Emax=5.0;
  int M0=2; //size initial subspace
  /* input parameters for FEAST */
  int feastparam[64];
  /* output variables for FEAST */
  double *E, *res, *X;
  double  epsout,trace;
  int i,loop,info,M;
/*********** MPI ****************************/
int rank,numprocs;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/*********************************************/
  /* Allocate memory for eigenvalues.eigenvectors/residual */
  E=calloc(M0,sizeof(double));  //eigenvalues
  res=calloc(M0,sizeof(double));//eigenvectors
  X=calloc(N*M0,sizeof(double));//residual (if needed)

  /* !!!!!!!!!! FEAST !!!!!!!!!*/
  feastinit(feastparam);
  feastparam[0]=1;  /*change from default value */
  dfeast_syev(&UPLO,&N,A,&LDA,feastparam,&epsout,&loop,&Emin,&Emax,&M0,\
              E,X,&M,res,&info);
  /*!!!!!!!!!! REPORT !!!!!!!!!*/
if (rank==0) {  printf(”FEAST OUTPUT INFO %d\n”,info);
  if (info==0) {
    printf(”*************************************************\n”);
    printf(”************** REPORT ***************************\n”);
    printf(”*************************************************\n”);
    printf(”# of processors %d \n”,numprocs);
    printf(”# Search interval [Emin,Emax] %.15e %.15e\n”,Emin,Emax);
    printf(”# mode found/subspace %d %d \n”,M,M0);
    printf(”# iterations %d \n”,loop);
    trace=0.0;
    for (i=0;i<=M-1;i=i+1){
      trace=trace+*(E+i); }
    printf(”TRACE %.15e\n”, trace);
    printf(”Relative error on the Trace %.15e\n”,epsout );
    printf(”Eigenvalues/Residuals\n”);
    for (i=0;i<=M-1;i=i+1){
      printf(”   %d %.15e %.15e\n”,i,*(E+i),*(res+i));}
    printf(”Eigenvectors\n”);
    for (i=0;i<=M-1;i=i+1){
      printf(”   %d (%.15e, %.15e)\n”,i,*(X+i*M),*(X+1+i*M));}   }}
MPI_Finalize(); /************ MPI ***************/
  return 0;}
```

Figure 2.9: A very simple MPI-C “helloworld” example. This code can be found in `<FEAST` *directory*`>` `/example/C-MPI/1_dense`.

2.5 FEAST: General use

Here, we propose to briefly present to the FEAST users a list of specifications (i.e. what is needed from users), expectations (i.e. what users should expect from FEAST), and directions for achieving performances (i.e. including parallel scalability and current limitations).

2.5.1 Single search interval and FEAST-SMP

Specifications: (i.e what is needed)

- the search interval and the size of the subspace M_0 (overestimation of the number of eigenvalues M within);
- the system matrix in dense, banded or sparse-CSR format if FEAST predefined interfaces are used, or a high-performance complex direct or iterative system solver and matrix-vector multiplication routine if FEAST RCI interfaces are used instead.

Expectations: (i.e what to expect)

- fast convergence to very high accuracy seeking up to 1000's eigenpairs (in 2-4 iterations using $\mathrm{M}_0 \geq 1.5\mathrm{M}$, and starting with $\mathrm{Ne} = 8$ or at most using $\mathrm{Ne} = 16$ contour points);
- an extremely robust approach (no known failed case or convergence problem reported to date).

Directions for achieving performances:

- M_0 should be much smaller than the size of the eigenvalue problem, then the arithmetic complexity should mainly depend on the inner system solver (i.e. O(NM) for narrow banded or sparse system);
- parallel scalability performances at the third level of parallelism depends on the shared memory capabilities of the inner system solver (i.e. via the shell variable `MKL_NUM_THREADS` if Intel-MKL is used);
- if M_0 increases significantly for a given search interval, the complexity $\mathrm{O}(\mathrm{M}_0^3)$ for solving the reduced system would become more significant (typically, if $\mathrm{M}_0 > 2000$). In this case it is recommended to consider multiple search intervals to be performed in parallel. For example, if 10^4 eigenpairs of a very large system are needed, many search intervals could be used simultaneously to decrease the size of the reduced dense generalized eigenproblem (e.g. if 10 intervals are selected the size of each reduced problem would then be $\sim 10^3$);
- For very large sparse and challenging systems, it is strongly recommended for expert application users to make use of the FEAST-RCI with customized highly-efficient iterative system solvers and preconditioners;
- For banded systems and FEAST-banded interfaces, the parallel scalability performances are currently limited (as the SPIKE banded primitives are only threaded via BLAS), however, an OpenMP SPIKE-banded solver is currently under development (see [3] for preliminary performances).

2.5.2 Single search interval and FEAST-MPI

Specifications: (i.e what is needed)

- Same general specification than for the FEAST-SMP;

- MPI environment application code and link to the FEAST-MPI library.

Expectations: (i.e what to expect)

- Same general expectation than for FEAST-SMP;
- ideally, a linear scalability with the number of MPI processes up to Ne the number of contour points (so up to 8 nodes by default). If the number of MPI processes is exactly equal to Ne, the factorization of the system matrices is kept in memory along the FEAST iterations and "superlinear scalability" can then be expected.

Directions for achieving performances:

- Same general directions than for FEAST-SMP;
- For a given search interval, the second and third level of parallelism is then easily achieved by MPI (along the contour points) calling OpenMP (i.e shared memory linear system solver). Among the two levels of parallelism offered here, there is a trade-off between the choice of the number of MPI processes and threads by MPI process. For example let us consider: (i) the use of FEAST-SPARSE interface, (ii) 8 contour points (i.e. 8 linear systems to solve), and (iii) a cluster of 4 physical nodes with 8 cores/threads by nodes; the two following options (among many others) should be considered to run the MPI code `myprog`:

  ```
  > mpirun − genv MKL_NUM_THREADS 8 − ppn 1 − n 4 ./myprog
  ```

  where 8 threads can be used on each node, and where each physical node ends up solving consecutively two linear systems using 8 threads. This option saves memory.

  ```
  > mpirun − genv MKL_NUM_THREADS 4 − ppn 2 − n 8 ./myprog
  ```

  where 4 threads can be used on each node, and where the MPI processes end up solving simultaneously the 8 linear systems using 4 threads. This option should provide better performance but it is more demanding in memory (two linear systems are stored on each node).

  In contrast if 8 physical nodes are available, the best possible option for 8 contour points becomes:

  ```
  > mpirun − genv MKL_NUM_THREADS 8 − ppn 1 − n 8 ./myprog
  ```

  where all the 64 cores are then used.
- If more physical nodes than contour points are available, scalability cannot be achieved at the second level parallelism anymore, but multiple search intervals could then be considered (i.e. first level of parallelism).

2.5.3 Multiple search intervals and FEAST-MPI

Specifications: (i.e what is needed)

- Same general specification than for the FEAST-MPI using a single search interval;
- a new flag `fpm(9)` can easily by defined by the user to set up a local MPI communicator associated to different cluster of nodes for each search interval (which is set to `MPI_COMM_WORLD` by default). An example on how one can proceed for two search intervals is given in Figure 2.10. This can easily be generalized to any numbers of search intervals.

Expectations: (i.e what to expect)

- Same general expectation than for FEAST-MPI using a single search interval;
- Perfect parallelism using multiple neighboring search intervals without overlap (overall orthogonality should also be preserved).

Directions for achieving performances:

- Same general directions than for FEAST-MPI using a single search interval;
- Additionally, among the different levels of parallelism offered by FEAST-MPI, there will be a trade-off between the number of intervals to consider and the number of eigenpairs that can be handled by intervals;
- In practical applications, the users should have a good apriori estimates of the distribution of the overall eigenvalue spectrum in order to make efficiently use of the first level of parallelism (i.e. in order to specified the search intervals). However, it is also possible to envision runtime automatic strategies (see example in [4]);
- Future developments of FEAST will be concerned with MPI implementation for the linear systems at the third level of parallelism.

```
.
.
.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! MPI!!!!!!!!!!!!!!!!!!!!
  integer :: code,rank,lrank,nb_procs,lnb_procs,color,key
  integer :: NEW_COMM_WORLD
  call MPI_INIT(code)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, nb_procs,code)
  call MPI_COMM_RANK(MPI_COMM_WORLD, rank,code)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
.
.
.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!! Definition of the two intervals
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
if (rank<=nb_procs/2-1) then
color=1 ! first interval
else
color=2 ! second interval
endif
!!!!!! create new_mpi_comm_world communicator
key=0
call MPI_COMM_SPLIT(MPI_COMM_WORLD, color,key,NEW_COMM_WORLD, code)
call MPI_COMM_RANK(NEW_COMM_WORLD, lrank,code) ! local rank

!!!!! search interval [Emin,Emax] including M eigenpairs (example)
if (color==1) then !! 1st interval
 Emin=-0.35d0
 Emax= 0.0d0
 M0=40
elseif(color==2) then !! 2nd interval
 Emin= 0.0d0
 Emax= 0.23d0
 M0=40
endif

!!!!!!!!!!!!! ALLOCATE VARIABLE
  allocate(e(1:M0))      ! Eigenvalue
  allocate(X(1:n,1:M0)) ! Eigenvectors
  allocate(res(1:M0))    ! Residual (if needed)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!  FEAST (for the two search intervals)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  call feastinit(fpm) !! by default fpm(9)=MPI_COMM_WORLD
  fpm(9)=NEW_COMM_WORLD !! Local for a given interval
!!! call FEAST interfaces as usual (feast_dense example here)
  call zfeast_heev(UPLO,N,A,N,fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)
.
.
.
```

Figure 2.10: A simple MPI-F90 example procedure using two search intervals and the flag `fpm(9)`. In this example, if the code is executed using `n` MPI-processes, the first search interval would be using `n/2` processes while `n-n/2` will be used by the second search interval. Driver examples that include three levels of parallelism can be found in all subdirectories of `<FEAST` *directory*`> /example/Fortran-MPI/` and `<FEAST` *directory*`> /example/C-MPI/` (see FEAST application section for more details).

Chapter 3

FEAST Interfaces

3.1 Basics

3.1.1 Definition

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

1. The reverse communication interfaces (RCI):

   TFEAST_SRCI and **T**FEAST_HRCI,

   respectively for symmetric and hermitian problems. These interfaces constitute the kernel of FEAST. They are matrix free format (the interfaces are independent of the matrix data formats), users can then define their own explicit or implicit data format. Mat-vec routines and direct/iterative linear system solvers must also be provided by the users.

2. The predefined interfaces:

   TFEAST_$\{X\}$**YY**

   These interfaces can be considered as predefined optimized drivers for **T**FEAST_{S,R}RCI that act on commonly used matrix data storage (dense, banded and sparse-CSR), using predefined mat-vec routines and preselected inner linear system solvers.

For theses interfaces, we note:

- **T**: the data type of matrix **A** (and matrix **B** if any) defined as follows:

Value of **T**	Type of matrices
S	real single precision
D	real double precision
C	complex single precision
Z	complex double precision

- $\{X\}$: the type of problem {Symmetric, Hermitian} and the type of matrix format defined as follows:

Value of $\{X\}$	SY	HE	SB	HB	SCSR	HCSR
Problem Type	Symmetric	Hermitian	Symmetric	Hermitian	Symmetric	Hermitian
Matrix format	dense		banded		sparse	
Linear solver	LAPACK		SPIKE primitives		MKL-PARDISO	

- **YY**: the eigenvalue computation – either EV or GV respectively for standard, and generalized eigenvalue problem.

The different FEAST interface names and combinations are summarized in Table 3.1.

RCI- interfaces
{s,d}feast_srci {c,z}feast_hrci
DENSE- interfaces

{s,d}feast_sy{ev,gv}
{c,z}feast_he{ev,gv}

BANDED- interfaces

{s,d}feast_sb{ev,gv}
{c,z}feast_hb{ev,gv}

SPARSE- interfaces

{s,d}feast_scsr{ev,gv}
{c,z}feast_hcsr{ev,gv}

Table 3.1: List of all FEAST interfaces available in FEAST version v2.1

3.1.2 Common Declarations

The arguments list for the FEAST interfaces are defined as follows:

TFEAST_{*extension*} (**{list}**, fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)

where **{list}** denotes a series of arguments that are specific to each interfaces and will be presented in the next sections. The rest of the arguments are common to both RCI and predefined FEAST interfaces and their definition is given in Table 3.2.

3.1.3 Input FEAST parameters

In the common argument list, 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 using the routine `FEASTINIT` as follows (Fortran notation):

```
CALL FEASTINIT(fpm)
```

All input FEAST parameters are then set to their default values. The detailed list of these parameters is given in Table 3.3.

3.1.4 Output FEAST `info` details

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 3.4.

Component	Type (Fortran)	Input/Output	Description
`fpm`	integer(64)	input	FEAST input parameters (see Table 3.3) The size should be at least 64
`epsout`	real *if* **T**=`S, C` *or* double precision *if* **T**=`D, Z`	output	Relative error on the trace $\lvert \texttt{trace}_k - \texttt{trace}_{k-1}\rvert / \max(\lvert\texttt{Emin}\rvert, \lvert\texttt{Emax}\rvert)$
`loop`	integer	output	# of refinement loop executed
`Emin`	real *if* **T**=`S, C` *or* double precision *if* **T**=`D, Z`	input	Lower bound search interval
`Emax`	real *if* **T**=`S, C` *or* double precision *if* **T**=`D, Z`	input	Upper bound search interval
`M0`	integer	input/output	Subspace dimension On entry: initial guess (`M0>M`) On exit: new suitable `M0` if guess too large
`E`	real(`M0`) *if* **T**=`S, C` *or* double precision(`M0`) *if* **T**=`D, Z`	output	Eigenvalues the first `M` values are in the search interval the others `M0-M` values are outside
`X`	*Same type than* **T** *with 2D dimension* (`N,M0`)	input/output	Eigenvectors (`N`: size of the system) On entry: guess subspace if fpm(5)=1 On exit: eigenvectors solutions (same order as in `E`) (Rq: if fpm(14)=1, first **Q** subspace on exit)
`M`	integer	output	# eigenvalues found between `[Emin,Emax]`
`res`	real(`M0`) *if* **T**=`S, C` *or* double precision(`M0`) *if* **T**=`D, Z`	output	Relative residual $\lvert\lvert \mathbf{Ax_i} - \lambda_\mathbf{i}\mathbf{Bx_i}\rvert\rvert_\mathbf{1} / \lvert\lvert \max(\lvert\texttt{Emin}\rvert, \lvert\texttt{Emax}\rvert)\mathbf{Bx_i}\rvert\rvert_\mathbf{1}$ (same ordering as in `E`)
`info`	integer	output	Error handling (if =0: successful exit) (see Table 3.4 for all INFO return codes)

Table 3.2: List of arguments common for the RCI and predefined FEAST interfaces.
Remark 1: To maximize performances users may choose `M0` 1.5 to 2 times greater than the number of eigenvalues estimated in the search interval `[Emin,Emax]`. In case where the initial guess `M0` is chosen too small, FEAST will exit and return the `info` code 3. In case where the initial guess `M0` it too large, FEAST will automatically return a smaller and suitable size subspace.
Remark 2: the arrays `E`, `X` and `res` return the eigenpairs and associated residuals. The solutions within the intervals are contained in the first `M` components of the arrays. Note for expert use: the solutions that are directly outside the intervals can also be found with less accuracy in the other `M0-M` components (i.e. from element `M+1` to `M0`). In addition where spurious solutions may be found in the processing of the FEAST algorithm, those are put at the end of the arrays `E` and `X` and are flagged with the value -1 in the array `res`.

`fpm(i)` Fortran `fpm[i-1]` C	Description	Default value
`i=1`	Print runtime comments on screen (0: No; 1: Yes)	0
`i=2`	# of contour points (3,4,5,6,8,10,12,16,20,24,32,40,48)	8
`i=3`	Stopping convergence criteria for double precision ($\epsilon = 10^{-\texttt{fpm(3)}}$)	12
`i=4`	Maximum number of FEAST refinement loop allowed (≥ 0)	20
`i=5`	Provide initial guess subspace (0: No; 1: Yes)	0
`i=6`	Convergence criteria (for the eigenpairs in the search interval) 0: Using relative error on the trace `epsout` i.e. `epsout`$< \epsilon$ 1: Using relative residual `res` i.e. $\max_i$ `res(i)` $< \epsilon$	0
`i=7`	Stopping convergence criteria for single precision ($\epsilon = 10^{-\texttt{fpm(7)}}$)	5
`i=9`	User defined MPI communicator for a given search interval	MPI_COMM_WORLD
`i=14`	Return only subspace Q after 1 contour (0: No, 1: Yes)	0
`i=30-63`	unused	
`All Others`	Reserved value	N/A

Table 3.3: List of input FEAST parameters and default values obtained with the routine `FEASTINIT`.
Remark 1: 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)`).
Remark 2: If FEAST-SMP is used along with the MKL library, the shell variable MKL_NUM_THREADS can be used for setting the number of threads for all FEAST interfaces.
Remark 3: If FEAST-MPI is used, the user can have the option to modify `fpm(9)` to enable the first level of parallelism (i.e. runs many search interval in parallel, each one associated with a cluster of nodes).

`info`	Classification	Description
202	Error	Problem with size of the system `N` (`N<=0`)
201	Error	Problem with size of subspace `M0` (`M0>N or M0<=0`)
200	Error	Problem with Emin,Emax (`Emin>=Emax`)
$(100 + i)$	Error	Problem with i^{th} value of the input FEAST parameter (i.e `fpm(i)`)
4	Warning	Only the subspace has been returned using `fpm(14)=1`
3	Warning	Size of the subspace `M0` is too small (`M0<=M`)
2	Warning	No Convergence (#iteration loops>`fpm(4)`)
1	Warning	No Eigenvalue found in the search interval
0	Successful exit	
-1	Error	Internal error for allocation memory
-2	Error	Internal error of the inner system solver
-3	Error	Internal error of the reduced eigenvalue solver *Possible cause: matrix* **B** *may not be positive definite*
$-(100 + i)$	Error	Problem with the i^{th} argument of the FEAST interface

Table 3.4: Return code descriptions for the parameter `info`.
Remark: In some extreme cases the return value `info=1` may indicate that FEAST has failed to find the eigenvalues in the search interval. This situation would appear only if a very large search interval is used to locate a small and isolated cluster of eigenvalues (i.e. in case the dimension of the search interval is many orders of magnitude off-scaling). For this case, it is then either recommended to increase the number of contour points `fpm(2)` or simply rescale more appropriately the search interval.

3.2 FEAST_RCI interfaces

If you are not familiar with reverse communication mechanisms and/or your application does not require specific linear system solvers or matrix storage, you may want to skip this section and go directly to the section 3.3 on predefined interfaces.

3.2.1 Specific declarations

The arguments list for the FEAST_RCI interfaces is defined as follows:

TFEAST_{S,H}RCI (**{list}**, fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)

with **{list}**=={ijob,N,Ze,work1,work2,Aq,Bq}

The definition of the series of arguments in **{list}** is given in Table 3.5.

Component	Type (Fortran)	Input/ Output	Description
ijob	integer	in/out	ID of the FEAST_RCI operation On entry: ijob=-1 (initialization) On exit: ijob=0,10,11,20,21,30,40
N	integer	in	Size of the system
Ze	complex *if* **T**=S, C complex*16 *if* **T**=D, Z	out	Coordinate along the complex contour
work1	*Same type than* **T** *with 2D dimension* (N,M0)	in/out	Workspace
work2	complex(N,M0) *if* **T**=S, C complex*16(N,M0) *if* **T**=D, Z	in/out	Workspace
Aq *or* Bq	*Same type than* **T** *with 2D dimension* (M0,M0)	in/out	Workspace for the reduced eigenvalue problem

Table 3.5: Definition of arguments specific for the **T**FEAST_RCI interfaces.

3.2.2 RCI mechanism

The FEAST_RCI interfaces can be used to solve standard or generalized eigenvalue problems, and are independent of the format of the matrices. The reverse communication interface (RCI) mechanism is detailed in Figure 3.1 for the real symmetric problem and in Figure 3.2 for the Hermitian one. The `ijob` parameter is first 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 done. 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. It should be noted that if an iterative solver is used along with a preconditioner, the factorization of the preconditioner could be performed with `ijob=10` (and `ijob=20` if applicable) for a given value of `Ze`, and the associated iterative solve would then be performed with `ijob=11` (and `ijob=21` if applicable).

```
ijob=-1 ! initialization
do while (ijob/=0)
call TFEAST_SRCI(ijob,N,Ze,work1,work2,Aq,Bq,&
                 &fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)
 select case(ijob)
 case(10) !!Factorize the complex matrix (ZeB-A)
................ <<< user entry
 case(11) !!Solve the linear system (ZeB-A)y=work2(1:N,1:M0) result in work2
................ <<< user entry
 case(30) !!Perform multiplication A*X(1:N,i:j) result in work1(1:N,i:j)
          !! where i=fpm(24) and j=fpm(24)+fpm(25)-1
................ <<< user entry
 case(40) !!Perform multiplication B*X(1:N,i:j) result in work1(1:N,i:j)
          !! where i=fpm(24) and j=fpm(24)+fpm(25)-1
................ <<< user entry
 end select
end do
```

Figure 3.1: The FEAST reverse communication interface mechanism for real symmetric problem. The **T** option in **T**`FEAST_SRCI` should be replaced by either `S` or `D` depending of the matrix data type in single or double precision.

```
ijob=-1 ! initialization
do while (ijob/=0)
call TFEAST_HRCI(ijob,N,Ze,work1,work2,Aq,Bq,&
                 &fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)
 select case(ijob)
 case(10) !!Factorize the complex matrix (ZeB-A)
................ <<< user entry
 case(11) !!Solve the linear system (ZeB-A)y=work2(1:N,1:M0) result in work2
................ <<< user entry
 case(20) !!Factorize (if needed by case(21)) the complex matrix (ZeB-A)^H
          !!ATTENTION: This option would need additional memory storage
          !! (i.e. the resulting matrix from case(10) cannot be overwritten)
................ <<< user entry
 case(21) !!Solve the linear system (ZeB-A)^Hy=work2(1:N,1:M0) result in work2
          !!REMARK: case(20) becomes obsolete if this solve can be performed
          !!        using the factorization in case(10)
................ <<< user entry
 case(30) !!Perform multiplication A*X(1:N,i:j) result in work1(1:N,i:j)
          !! where i=fpm(24) and j=fpm(24)+fpm(25)-1
................ <<< user entry
 case(40) !!Perform multiplication B*X(1:N,i:j) result in work1(1:N,i:j)
          !! were i=fpm(24) and j=fpm(24)+fpm(25)-1
................  <<< user entry
 end select
end do
```

Figure 3.2: The FEAST reverse communication interface mechanism for complex hermitian problem. The **T** option in **T**`FEAST_HRCI` should be replaced by either `C` or `Z` depending of the matrix data type in single or double precision. It should be noted that if `case(20)` can be avoided, the performances are expected to be up to ×2 faster, and FEAST would be using twice less memory.

3.3 FEAST predefined interfaces

3.3.1 Specific declarations

The arguments list for the FEAST predefined interfaces is defined as follows:

TFEAST_*X***YY** (**{list}**, fpm,epsout,loop,Emin,Emax,M0,E,X,M,res,info)

where the series of arguments in **{list}** is specific to the values of **T**, *X* and **YY**, and is given in Table 3.6. The definition of the arguments is then given in Table 3.7.

{list}	Standard: **YY**=EV	Generalized: **YY**=GV
Dense: *X*=SY or HE	{UPLO,N,A,LDA}	{UPLO,N,A,LDA,B,LDB}
Banded: *X*=SB or HB	{UPLO,N,kla,A,LDA}	{UPLO,N,kla,A,LDA,klb,B,LDB}
Sparse: *X*=SCSR or HCSR	{UPLO,N,A,IA,JA}	{UPLO,N,A,IA,JA,B,IB,JB}

Table 3.6: List of arguments specific for the **T**FEAST_*X***YY** interfaces.

Component	Type (Fortran)	Input/ Output	Description
UPLO	character(len=1)	in	Matrix Storage ('F','L','U') 'F': Full; 'L': Lower; 'U': Upper
N	integer	in	Size of the system
kla	integer	in	The number of subdiagonals within the band of A.
klb	integer	in	The number of subdiagonals within the band of B.
A	*Same type than* **T** *with 2D dimension:* (LDA,N) *if Dense* *with 2D dimension:* (LDA,N) *if Banded* *with 1D dimension:* (IA(N+1)-1) *if Sparse*	in	Eigenvalue system (Stiffness) matrix
B	*Same type than* **T** *with 2D dimension:* (LDB,N) *if Dense* *with 2D dimension:* (LDB,N) *if Banded* *with 1D dimension:* (IB(N+1)-1) *if Sparse*	in	Eigenvalue system (Mass) matrix
LDA	integer	in	Leading dimension of A; LDA>=N *if Dense* LDA>=2kla+1 *if Banded;* UPLO='F' LDA>=kla+1 *if Banded;* UPLO/='F'
LDB	integer	in	Leading dimension of B; LDB>=N *if Dense* LDB>=2klb+1 *if Banded;* UPLO=='F' LDB>=klb+1 *if Banded;* UPLO/='F'
IA	integer(N+1)	in	Sparse CSR Row array of A.
JA	integer(IA(N+1)-1)	in	Sparse CSR Column array of A.
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 3.7: Definition of arguments specific for the **T**FEAST_*X***YY** interfaces.

3.3.2 Matrix storage

Let us consider a standard eigenvalue problem and the following (stiffness) matrix **A**:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix} \tag{3.1}$$

where $a_{ij} = a^*_{ji}$ for $i \neq j$ (i.e. $a_{ij} = a_{ji}$ if the matrix is real). Using the FEAST predefined interfaces, this matrix could be stored in dense, banded or sparse format as follows:

- Using the dense format, **A** is stored in a two dimensional array in a straightforward fashion. Using the options `UPLO='L'` or `UPLO='U'`, the lower triangular and upper triangular part respectively, do not need to be referenced.

- Using the banded format, **A** 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 `LDA>=2*kla+1` if `UPLO='F'` (LAPACK banded storage would require `LDA>=3*kla+1`). For this example, the number of subdiagonals or superdiagonals is `kla=1`. Using the option `UPLO='L'` or `UPLO='U'`, the `kla` rows respectively above or below the diagonal elements row, do not need to be referenced (or stored).

- Using the sparse storage, the non-zero elements of **A** are stored using a set of one dimensional arrays (`A,IA,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 `UPLO='L'` or `UPLO='U'`, one would get respectively

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

Finally, the (mass) matrix **B** that appears in generalized eigenvalue systems, should use the same family of storage format than the matrix **A**. It should be noted, however, that the bandwidth can be different for the banded format (`klb` can be different than `kla`), and the position of the non-zero elements can also be different for the sparse format (CSR coordinates `IB,JB` can be different than `IA,JA`).

Chapter 4

FEAST in action

4.1 Examples

The `$FEASTROOT/example` directory provides Fortran, Fortran-MPI, C and MPI-C examples for using the FEAST predefined interfaces. The Fortran examples are written in F90 but it should be straightforward for the user to transform then into F77 if needed (since FEAST uses F77 argument-type interfaces). Examples are limited at solving two particular types of eigenvalue problems:

Example 1 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.

Example 2 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.

Each problem is solved while considering either single or double precision arithmetic and either a dense, banded or sparse storage for the matrices.

The `$FEASTROOT/example/` directory contains the directories `Fortran, C, Fortran-MPI, C-MPI` which, in turn, contain similar subdirectories `1_dense`, `2_banded`, `3_sparse` with source code drivers for the above two examples in single and double precisions.

In order to compile and run the examples of the FEAST package, please follow the following steps:

1. Go to the directory `$FEASTROOT/example`
2. Edit the `make.inc` file and follow the directions to customize appropriately: (i) the name/path of your F90, C and MPI compilers (if you wish to compile and run the F90 examples alone, it is not necessary to specify the C compiler as well as MPI and vice-versa); (ii) the path `LOCLIBS` for the FEAST libraries and both paths and names in `LIBS` of the MKL-PARDISO, LAPACK and BLAS libraries (if you do not wish to compile the sparse examples there is no need to specify the path and name of the MKL-PARDISO library).

   By default, all the options in the `make.inc` assumes calling the FEAST library compiled with no-runtime dependency (add the appropriate flags `-lifcoremt, -lgfortran, etc.` otherwise). Additionally, `LIBS` in `make.inc` uses by default the Intel-MKL v10.x to link the required libraries.
3. By executing `make`, all the different Makefile options will be printed out including compiling alone, compiling and running, and cleaning. For example,

   ```
   >make allF
   ```

   compiles all Fortran examples, while

```
>make rallF
```

compiles and then run all Fortran examples.

4. If you wish to compile and/or run a given example, for a particular language and with a particular storage format, just go to one the corresponding subdirectories `1_dense`, `2_banded`, `3_sparse` of the directories `Fortran, C, Fortran-MPI, C-MPI`. You will find a local `Makefile` (using the same options defined above in the `make.inc` file) as well as the four source codes covering the **example 1** and **example 2** above in single and double precisions. The `1_dense` directories include also the "helloworld" source code examples presented in Section 2.4. The `Fortran-MPI` and `Fortran-C` directories contain one additional example using all three levels of parallelism for FEAST.

For the `Fortran` or `C` example directories, and denoting **x** either **f90** or **c**:

- The `1_dense` subdirectory contains:
  - `helloworld.x` (solve the helloworld example presented in the documentation)
  - `driver_{s,d}feast_syge.x` (solve example1 using **{S,D}**`FEAST_SYGE` interface)
  - `driver_{c,z}feast_heev.x` (solve example2 using **{C,Z}**`FEAST_HEEV` interface)
- The `2_banded` subdirectory contains:
  - `driver_{s,d}feast_sbgv.x` (solve example1 using **{S,D}**`FEAST_SBGV` interface)
  - `driver_{c,z}feast_hbev.x` (solve example2 using **{C,Z}**`FEAST_HBEV` interface)
- The `3_sparse` subdirectory contains:
  - `driver_{s,d}feast_scsrgv.x` (solve example1 using **{S,D}**`FEAST_SCSRGV` interface)
  - `driver_{c,z}feast_hcsrev.x` (solve example2 using **{C,Z}**`FEAST_HCSREV` interface)

For the `Fortran-MPI` or `C-MPI` example directories, and denoting **x** either **f90** or **c**:

- The `1_dense` subdirectory contains:
  - `phelloworld.x` (solve the helloworld example presented in the documentation)
  - `pdriver_{s,d}feast_syge.x` (solve example1 using **{S,D}**`FEAST_SYGE` interface)
  - `pdriver_{c,z}feast_heev.x` (solve example2 using **{C,Z}**`FEAST_HEEV` interface)
  - `3pdriver_{c,z}feast_heev.x` (solve example2 using **{C,Z}**`FEAST_HEEV` interface and two-search intervals)
- The `2_banded` subdirectory contains:
  - `pdriver_{s,d}feast_sbgv.x` (solve example1 using **{S,D}**`FEAST_SBGV` interface)
  - `pdriver_{c,z}feast_hbev.x` (solve example2 using **{C,Z}**`FEAST_HBEV` interface)
  - `3pdriver_{c,z}feast_hbev.x` (solve example2 using **{C,Z}**`FEAST_HBEV` interface and two-search intervals)
- The `3_sparse` subdirectory contains:
  - `pdriver_{s,d}feast_scsrgv.x` (solve example1 using **{S,D}**`FEAST_SCSRGV` interface)
  - `pdriver_{c,z}feast_hcsrev.x` (solve example2 using **{C,Z}**`FEAST_HCSREV` interface)
  - `3pdriver_{c,z}feast_hcsrev.x` (solve example2 using **{C,Z}**`FEAST_HCSREV` interface and two-search intervals)

4.2 FEAST utility sparse drivers

If a sparse matrix can be provided by the user in coordinate format, the `$FEASTROOT/utility` directory offers a quick way to test the efficiency/reliability of the FEAST SPARSE predefined interfaces using the MKL-PARDISO solver. A general driver is provided for `Fortran`, `C`, `Fortran-MPI` and `C-MPI`, named `driver_feast_sparse` or `pdriver_feast_sparse` in the respective subdirectories.

You will also find a local `Makefile` where compiler and libraries paths/names need to be edited and changed appropriately (the MKL-PARDISO solver is needed). The command "`>make all`" should compile the drivers.

Let us now 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.A` should contain the matrix **A** in coordinate format; As a reminder, the coordinate format is defined row by row as

```
  N       N       NNz
 i1      j1      val1
 i2      j2      val2
 i3      j3      val3
 .       .        .
 .       .        .
 iNNZ    jNNZ    valNNZ
```

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

- `mytest.B` should contain the matrix **B** (if any) in coordinate format;

- `mytest.in` should contain the search interval, some 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"tandard or "g"eneralized eigenvalue problem
d      !(s,d,c,z)precision i.e (single real,double real,complex,double complex)
F      ! UPLO (matrix elements provided:'F' Full,'L' Lower part,'U' upper part)
-5.0e0 ! Emin (lower bound search interval)
5.0e0  ! Emax (upper bound search interval)
20     ! M0 (size of the subspace)
!!!!FEASTPARAM (1,64) in Fortran; [0,63] in C
1   !feastparam(1)[0] !(0,1)
8   !feastparam(2)[1] !(3,4,5,6,8,10,12,16,20,24,32,40,48)
12  !feastparam(3)[2] if (s,c) precision, or feastparam(7)[6] if (d,z) precision
20  !feastparam(4)[3] !maximum #loop
1   !feastparam(6)[5] !(0,1)
```

  You may change any of the above options to fit your needs. It should be noted that the `UPLO` `L` or `U` options give you the possibility to provide only the lower or upper triangular part of the matrices `mytest.A` and `mytest.B` in coordinate format.

Finally results and timing can be obtained by running:

```
>$FEASTROOT/utility/Fortran/driver_feast_sparse <PATH_TO_MYTEST>/mytest
```

or

```
>$FEASTROOT/utility/C/driver_feast_sparse <PATH_TO_MYTEST>/mytest
```

respectively for the `Fortran` or `C` drivers.

For the Fortran-MPI and C-MPI drivers, a run would look like (other options could be applied including Ethernet fabric, etc.):

```
> mpirun − genv MKL_NUM_THREADS  < y > −ppn 1 − n < x >

$FEASTROOT/utility/Fortran/pdriver_feast_sparse < PATH_TO_MYTEST > /mytest,
```

where < x > represents the number of nodes at the second level of parallelism (along the contour– see Figure 2.3). As a reminder, the third level of parallelism for all drivers (for the linear system solver) is activated by the setting shell variable MKL_NUM_THREADS equal to the desired number of threads. In the example above for MPI, if the MKL_NUM_THREADS is set with value <y>, i.e. FEAST would run on <x> separate nodes, each one using <y> threads. Several combinations of <x> and <y> are possible depending also on the value of the -ppn directive.

In order to illustrate a direct use of the utility drivers, four examples are provided in the directory $FEASTROOT/utility/data with the following generic name:

- system1 (identical to example 1 presented in section 4.1),
- system2 (identical to example 2 presented in section 4.1),
- helloworld (helloworld example provided in this documentation, now in sparse format),
- cnt (a realistic example from Carbon Nanotube Nanoelectronics Simulation used in [1]),

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

```
>$FEASTROOT/utility/Fortran/driver_feast_sparse ../data/cnt
```

4.3 FEAST complements

This section points out the potentialities and features of the FEAST package in terms of numerical stability, robustness and scalability. More detailed information on the FEAST algorithm and simulation results with comparisons with the ARPACK package are provided in [1].

4.3.1 Accuracy and convergence

The performances of FEAST can depend on some tunings parameters such as the choices of the size of the subspace `M0`, the number of contour points `fpm(2)`, and the stopping criteria for the error on the trace (`fpm(3)` for double precision or `fpm(7)` for single precision).

In Table 4.1, we report the times and relative residual obtained by the FEAST_SPARSE interface for solving the `cnt` example provided in the directory `$FEASTROOT/utility/data`, and seeking up to `M=800` eigenpairs in a given search interval. The simulation runs are here restricted to a given node of a 8-cores Intel Clovertown system (16Gb,2.66GHz). In our experiments, the convergence criteria on the relative residual for FEAST is obtained when the relative error on the trace of the eigenvalues in the search interval is smaller or equal to 10^{-13} (i.e. `fpm(3)=13`). Table 4.2 shows the variation of the relative error on the trace (`epsout`) with the number of outer-iterative refinement for FEAST (`loop`). In these experiments, only 2 to 3 refinement loops suffice to obtain the small relative residuals for the different cases reported in Table 4.1. It should be noted that with only one loop, the trace on the eigenvalues is obtained with an accuracy of $\sim 10^{-5}$ or below.

cnt	FEAST	
N=12,450	Time(s)	Resid.
M=100	7.8	$4.5 * 10^{-10}$
M=200	14	$5.5 * 10^{-10}$
M=400	21	$1.8 * 10^{-10}$
M=800	58	$3.4 * 10^{-11}$

Table 4.1: FEAST timing with relative residuals $\max_i(||\mathbf{Ax_i} - \lambda_\mathbf{i}\mathbf{Bx_i}||_\mathbf{1}/||\mathbf{Ax_i}||_\mathbf{1})$, obtained for different search intervals. The size of the subspace has been chosen to be `M0=1.5M`, and the number of contour points for FEAST is `fpm`$(2) = 8$.

cnt	Relative error on the Trace		
N=12,450	1st loop	2nd loop	3rd loop
M=100	$3.0 * 10^{-6}$	$2.9 * 10^{-12}$	$1.0 * 10^{-15}$
M=200	$1.8 * 10^{-5}$	$4.8 * 10^{-12}$	$2.1 * 10^{-14}$
M=400	$2.4 * 10^{-8}$	$3.2 * 10^{-16}$	
M=800	$1.8 * 10^{-9}$	$4.3 * 10^{-16}$	

Table 4.2: Relative error on the trace `epsout` in function of the number of outer-iterative refinement `loop`. The convergence criteria is set to `fpm`$(3) = 13$ where the final relative residual on the eigenpairs is reported in Table 4.1.

4.3.2 Robustness of the contour integration technique

The simulation results in Table 4.1 demonstrate very good numerical stability for FEAST while the search interval keeps increasing and the number of contour points `fpm(2)=8` stays identical (i.e. the number of numerical operations stays the same for a given loop of FEAST with a fixed number of linear systems to solve). In addition, from Table 4.3, one can see how the robustness of the FEAST algorithm is affected while the number of contour points `fpm(2)` changes. In comparisons to the results reported in Table 4.1, 4 contour points did suffice to capture `M=100` eigenpairs with a relatively small residual and a faster simulation time, while the case of 16 contour points generated a smaller relative residual. One could also show that for a fixed number of contour point, increasing the size of subspace `M0` should also improve the convergence.

As a general rule, better accuracy (`epsout`) and better convergence (i.e. small value for `loop`) can be obtained while increasing the number of contour points `fpm(2)`. However, the simulation time would also increase since more linear systems need to be solved by loop. Therefore, the fastest time can be obtained while considering a trade off between the choices of `fpm(2)`, `M0` and the desired accuracy `fpm(3)` (for double precision).

cnt M = 100	FEAST Time(s)	 Resid.	 # loops
$N_e = 4$	7.0	$8.3 * 10^{-8}$	6
$N_e = 8$	7.8	$4.5 * 10^{-10}$	4
$N_e = 16$	10.2	$3.4 * 10^{-12}$	3

Table 4.3: Performance results obtained by FEAST seeking M = 100 eigenpairs for different values of fpm(2). The convergence is obtained when the error on the trace epsout is equal or smaller to 10^{-13}.

4.3.3 Complexity and multiplicities

We propose to create artificially new cnt systems called k(N,M) where the matrices **A** and **B** are repeated k times along the main diagonal (the new system matrix is block diagonal with k blocks). If we keep the same search interval used to obtain M=100 eigenpairs for k=1 (where the size of the matrices **A** and **B** is N), 100k eigenpairs must now be found for $k \geq 1$, where each one of them have the multiplicity k. In Table 4.4, we report the simulation times and relative residuals obtained using FEAST on these k(N,M) cnt systems. For the case 8(N,M), for example, the size of the new system matrix is 99, 600 and the first 100 eigenvalues have all the multiplicity 8 (so 800 eigenpairs are found in total). FEAST captures all multiplicities while the number of operations stays the same for all these cases (i.e. fpm(2) is fixed at 8). The simulation results show linear scalability performances with the size of the system and the number of eigenpairs (i.e. numerical complexity of O(NM)). In practice, the scalability of the FEAST will depend mainly on the scalability of the inner linear system solver.

cnt
N = 12, 450
M = 100
	FEAST
	
	Time(s)
		
		Resid.

(N, M)	7.8	$4.5 * 10^{-10}$
2(N, M)	27	$7.7 * 10^{-10}$
4(N, M)	109	$8.8 * 10^{-10}$
8(N, M)	523	$6.5 * 10^{-10}$

Table 4.4: Simulations times and relative residual obtained using the FEAST_SPARSE interface on the k(N,M) cnt systems which artificially reproduce k times the original cnt system. kM eigenpairs are found where each eigenvalue has a multiplicity of k.

4.3.4 Series of eigenproblems

We know that FEAST can re-use the computed subspace as suitable initial guess for performing iterative refinements. This capability can also be of benefit to modern applications in science and engineering where it is often necessary to solve a series of eigenvalue problems that are close one another. For example, let us consider many eigenvalue problems of the form $\mathbf{A_k x_k} = \lambda_\mathbf{k} \mathbf{B x_k}$ that need to be solved for different values of **k**. Using the FEAST package, after the first **k** point, the input parameter fpm(5) could take the value 1, and then the 2D input array **X** would contain the eigenvectors obtained at the point $\mathbf{k-1}$. Figure 4.1 illustrates a problem-type using the same search interval for the eigenvalues λ for different values of **k** where the subspace computed by FEAST at the point $\mathbf{k-1}$ is successively used as initial guess for the neighboring point **k**.

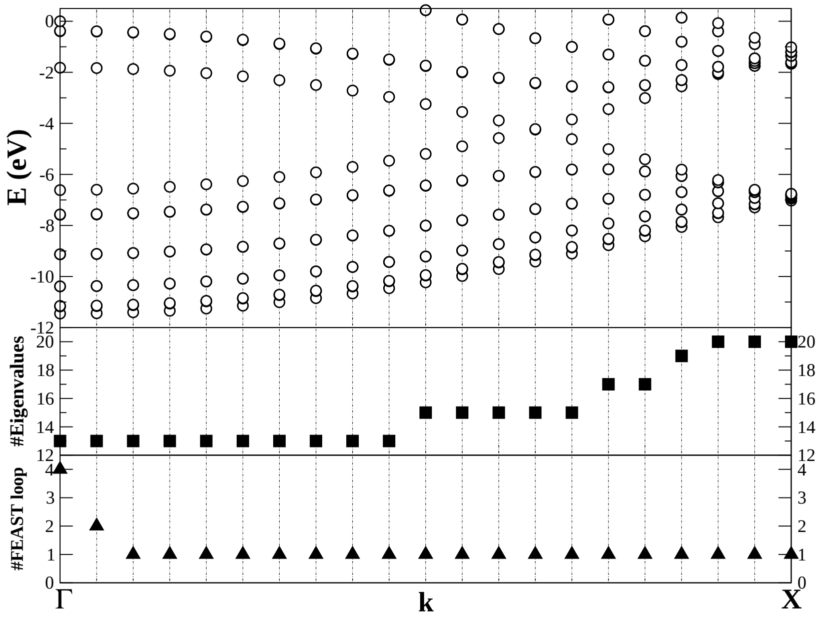

Figure 4.1: Bandstructure calculations of a (5,5) metallic carbon nanotube (CNT). The eigenvalue problems are solved successively for all the **k** points (from **Γ** to **X**), while the computed subspace of size `M0=25` at the point **k** is used as initial guess for the point $\mathbf{k+1}$. The number of eigenvalues found ranges from 13 to 20, and by the third **k** point, the FEAST convergence is obtained using only one refinement loop.

4.3.5 Iterative inner system solvers

Using the FEAST algorithm, the difficulty of solving an eigenvalue problem has been replaced by the difficulty of solving a linear system with multiple right-hand sides. For large sparse systems, this latter can be solved using either a direct system solver such as MKL-PARDISO (as predefined in the FEAST_SPARSE interface), or an iterative system solver with preconditioner (that can be customized by the user within the FEAST_RCI interface). It should be noted that the inner linear systems arising from standard eigenvalue solvers (using the shift-strategy), need often to be solved highly accurately via direct methods. Direct system solvers, however, are not always suited for addressing large-scale modern applications because of memory requirements.

In Figure 4.1, the inner linear systems in FEAST are actually solved using an iterative method with preconditioner where a modest relative residual of 10^{-3} is used. The convergence criteria for the relative error on the trace of the eigenvalues is chosen much smaller at 10^{-8} (`fpm(3)=8`), while the eigenvectors are expected to be obtained within the same (or a smaller) order of accuracy that the one used for the solutions of the inner systems (in the simulation results, however, the final relative residuals on the eigenpairs range from 10^{-3} to 10^{-5}). Finally, the resulting subspace could also be used as a very good initial guess for a one step more accurate refinement procedure (i.e. using more accurate relative residual for the inner linear systems).

Bibliography

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

[2] P. Tang, E. Polizzi*"Subspace iteration with Approximate Spectral Projection"*, http://arxiv.org/abs/1302.0432 (2013)

[3] K. Mendiratta, E. Polizzi, *A threaded SPIKE algorithm for solving general banded systems*, Parallel Computing, Vol. 37 I12, December, pp733-741 (2011).

[4] M. Galgon, L. Krämer, and B. Lang, *The FEAST algorithm for large sparse eigenvalue problems.* Proc. Appl. Math. Mech., 11(1):747-748, (2011).