EIGIFP: User Manual

JAMES H. MONEY[†] and QIANG YE[*]
UNIVERSITY OF KENTUCKY

1. EIGIFP CALLS

`eigifp` has a simple calling structure but also take several options. We will first discuss the basic calling structure of the program, working our way up to the more complex cases, and finally introduce all the advanced optional arguments a user can use to optimize performance.

The most basic call to `eigifp` is

$$>> [\texttt{Evalues}, \texttt{Evectors}] = \texttt{eigifp}(\texttt{A})$$

where A is a matrix in sparse format. This returns the smallest eigenvalue of A in `Evalues` and and its corresponding eigenvector in `Evectors`.

To compute the k smallest eigenpairs of the matrix A, one appends the value k to the above call

$$>> [\texttt{Evalues}, \texttt{Evectors}] = \texttt{eigifp}(\texttt{A}, \texttt{k})$$

where $k \geq 1$ is an integer. Then the returned results are a vector of k smallest eigenvalues `Evalues` and an $n \times k$ matrix of the corresponding eigenvectors `Evectors`.

Now, for solving the pencil eigenvalue problem of (A, B) which is $Ax = \lambda Bx$, one appends B after A in the above calls, namely,

$$>> [\texttt{Evalues}, \texttt{Evectors}] = \texttt{eigifp}(\texttt{A}, \texttt{B})$$

returns the smallest eigenpair and

$$>> [\texttt{Evalues}, \texttt{Evectors}] = \texttt{eigifp}(\texttt{A}, \texttt{B}, \texttt{k})$$

returns the k smallest eigenpairs as above.

In all cases, replacing A by $-A$ in the input argument and multiplying the output by -1 compute the largest eigenvalue correspondingly. For example,

$$>> [\texttt{Evalues}, \texttt{Evectors}] = \texttt{eigifp}(-\texttt{A}, \texttt{B}, \texttt{k}); \texttt{Evalues} = -\texttt{Evalues};$$

returns the k largest eigenpairs of (A, B). This can also be done by directly calling A, B with the option `opt.maxeig`.

`eigifp` also uses an option structure to provide extra information to the algorithm and to help improve performance. Users can pass a set of optional parameters via a structure in MATLAB. This is done by first setting the value in the structure, e.g.

$$>> \texttt{opt.initialvec} = \texttt{ones}(\texttt{n}, 1)$$

and then pass `opt` to `eigifp` by calling

$$>> [\texttt{Evalues}, \texttt{Evectors}] = \texttt{eigifp}(\texttt{A}, \texttt{B}, \texttt{k}, \texttt{opt})$$

One can pass less parameters, as long as the `opt` structure is the last parameter in the list.

The following discuss various optional parameters one can set.

1.1 Informational Options

Below is a list of the informational options `eigifp` takes:

`opt.size`	The dimension of A.
`opt.normA`	An estimate of the norm of A.
`opt.normB`	An estimate of the norm of B.
`opt.tolerance` (or `opt.tol`)	Set the termination threshold for the norm of the residual. Default value is given by (0??)
`opt.maxiteration` (or `opt.maxit`)	Set the maximal number of outer iterations to perform. Default value = 500.
`opt.disp`	Set to 0 to disable on-screen display of output and to any other numerical value to enable display. Default value = 1.
`opt.maxeig`	Set to 1 to compute the largest eigenvalue. Default value = 0.

The first three options are primarily used when the matrices are passed as functions (discussed in Sec.3.3 below). The first one provides the dimension of vectors and is required in the use of functions. The other two are optional, and estimate the norm of A and B. If they are not provided, then the norms are estimated by $\|Ax\|$ and $\|Bx\|$ for a random vector x.

The `tolerance` option allows the user to specify a tolerance different from the default value (0??). It can be set to be much larger (e.g. of $\mathcal{O}(\sqrt{\epsilon})$) if only eigenvalues are desired.

The `maxiteration` option lets the user specify how many outer iterations `eigifp` will perform before it terminates without convergence. The user can set a larger value for particularly difficult problems.

1.2 Performance Options

Below is a list of performance options one can specify to optimize the performance.

`opt.initialvec` (or `opt.V0`)	An $n \times k$ matrix whose j-th column is the initial guess for the j-th eigenvector.
`opt.inneriteration` (or `opt.innerit`)	Set the number of inner iteration (i.e. the dimension of the Krylov subspaces).
`opt.useprecon`	Setting this to 'NO' causes no preconditioning to happen. Otherwise, this is assumed to be the initial shift to be used for computing the preconditioner.

`opt.iluthresh` (or `opt.ilu`)	Set the threshold used in the incomplete LU factorization that is called for preconditioning. Setting it to 0 will lead to full factorization while setting it to 1 corresponds to incomplete factorization with no fill-in.
`opt.preconditioner` (or `opt.precon`)	Input a matrix or a function to be used as a user specified preconditioner.

The `initialvec` option allows the user to input initial eigenvector guesses and should be provided whenever they are available. If it is $n \times p$, then `initialvec(:,1)` is used as the initial vector for the first eigenvalue, `initialvec(:,2)` for the second, and so on.

The `inneriteration` option allows the user to set a fixed inner iteration. This will disable the adaptive setting by default. It can be used either to limit memory usage in the inner iteration or to improve convergence for more difficult problems. Thus, it should be set to a small value if `eigifp` runs out of memory but to a larger value when a very slow convergence is observed.

The `useprecon` option allows the user either to disable preconditioning totally when computing the incomplete factorization is not efficient, or to input an approximate eigenvalue to be used as a shift for computing a preconditioner. The latter setting saves the program to search for an initial approximation and will use the preconditioned iteration throughout.

The `iluthresh` option sets the threshold value for computing the incomplete LDL^T decomposition. Setting it to a smaller value leads to a better preconditioner which is more expensive to compute. The default value is 10^{-3}.

Finally, the `preconditioner` option allows the user to supply either a matrix or a function for the preconditioner. If it is a matrix, it should be the factor L; if it is a function, it should perform $L^{-T}L^{-1}x$ for an input x.

1.3 Matrices as Functions

In addition to passing A and B as sparse matrices, one can pass in the name of functions to call. The function should be of the form

$$>> \texttt{Ax} = \texttt{functionA(x)}$$

where `functionA` returns the result of $A * x$ as a vector. Similarly, one can provide a function call for B. In this case, the dimension of the matrices must be passed through `opt.size`. Then, one can call `eigifp` as

$$>> [\texttt{Evalues}, \texttt{Evectors}] = \texttt{eigifp}('\texttt{functionA}', \texttt{k}, \texttt{opt})$$

It is also desirable to pass the norms of the matrices through `opt.normA` and `opt.normB`.

2. EXAMPLES

In this section, we will present some examples of calling `eigifp` (version 2.1.1) and its outputs (with part of on-screen printout omitted) from execution. Below, A is the 7668×7668 matrix from discretizing the 2 dimensional Laplacian operator on a unit disc using a 5 point stencil. The boundary conditions are Dirichlet boundary conditions. We generate the sparse matrix in MATLAB and call `eigifp` as follows

```
>> A=delsq(numgrid('D',100));
>> eigenvalues=eigifp(A)
Computing Eigenvalue 1:
...
  ------------
  Eigenvalue 1 converged.
  # of multiplications by A (and B):      197.
  # of multiplications by preconditioner: 9.
-----
  CPU Time:3.250000.
eigenvalues =
    0.0023
```

For the pencil problem, we use a diagonal matrix for B.

```
>>B=spdiags([1:size(A,1)]',0,size(A,1),size(A,2));
>>eigenvalues=eigifp(A,B)
Computing Eigenvalue 1:
...
  ------------
  Eigenvalue 1 converged.
  # of multiplications by A (and B):      155.
  # of multiplications by preconditioner: 9.
-----
  CPU Time:3.990000.
eigenvalues =
   5.5653e-07
```

To compute the first three eigenpairs of (A, B), we use:

```
>>eigenvalues=eigifp(A,B,3)
Computing Eigenvalue 1:
...
  ------------
  Eigenvalue 1 converged.
  # of multiplications by A (and B):      159.
  # of multiplications by preconditioner: 9.
Computing Eigenvalue 2:
...
  ------------
  Eigenvalue 2 converged.
  # of multiplications by A (and B):      39.
  # of multiplications by preconditioner: 37.
Computing Eigenvalue 3:
...
  ------------
  Eigenvalue 3 converged.
  # of multiplications by A (and B):      18.
  # of multiplications by preconditioner: 16.
```

```
-----
  CPU Time:7.860000.
eigenvalues =
   1.0e-05 *
    0.0557
    0.1365
    0.1557
```

Now we present some examples using the options. The following specifies an initial guess for x_0 and a tolerance of 10^{-5}:

```
>>opt.v0=ones(size(A,1),1);
>>opt.tol=1e-5;
>>eigenvalues=eigifp(A,B,opt)
Computing Eigenvalue 1:
...
  ------------
  Eigenvalue 1 converged.
  # of multiplications by A (and B):      62.
  # of multiplications by preconditioner: 2.
-----
  CPU Time:2.030000.
eigenvalues =
   5.5653e-07
```

Next, we can disable preconditioning totally by setting the `useprecon` option.

```
>>opt.useprecon='NO';
>>eigenvalues=eigifp(A,B,opt)
Computing Eigenvalue 1:
...
  ------------
  Eigenvalue 1 converged.
  # of multiplications by A (and B):      148.
-----
  CPU Time:2.940000.
eigenvalues =
   5.5653e-07
```

On the other hand, since 0 is an approximate eigenvalue here, we can use it as a shift to start the preconditioned iterations by setting `opt.useprecon` to 0.

```
>>opt.useprecon=0;
>>eigenvalues=eigifp(A,B,opt)
Computing threshold ILU factorization using the shift provided.
Computing Eigenvalue 1:
...
  ------------
  Eigenvalue 1 converged.
  # of multiplications by A (and B):      7.
  # of multiplications by preconditioner: 5.
-----
  CPU Time:0.360000.
eigenvalues =
   5.5653e-07
```

We can also manually set a fixed value for the inner iterations:

```
>>opt.innerit=32;
>>eigenvalues=eigifp(A,B,opt)
Computing Eigenvalue 1:
...
  ------------
  Eigenvalue 1 converged.
  # of multiplications by A (and B):      202.
  # of multiplications by preconditioner: 8.
-----
  CPU Time:7.330000.
eigenvalues =
   5.5653e-07
```

Lastly, here is an example using a function $Afunc$ and $Bfunc$ to compute the two **largest** eigenvalues of (A,B). Here the functions use the matrices A and B respectively as they were defined above.

```
>> opt.maxeig=1;
>> opt.size=size(A,1);
>> opt.normA=norm(A,1);
>> opt.normB=norm(B,1);
>> eigenvalues=eigifp('Afunc','Bfunc',2,opt);
Computing the smallest eigenvalues of (-A,B) first.

Computing Eigenvalue 1:
...
  ------------
  Eigenvalue 1 converged.
  # of multiplications by A (and B):      1244.

Computing Eigenvalue 2:
...
  ------------
  Eigenvalue 2 converged.
  # of multiplications by A (and B):      1250.

-----
  CPU Time:105.520000.
  Negating the eigenvalues of (-A, B) => the largest eigenvalues of (A,B):
  4.239535e+00
  2.032777e+00
```

```
eigenvalues =
    4.2395
    2.0328
```