LBNL-42953

TRLAN User Guide

Edition 1.0 for TRLAN version 1.0

March 1999

Kesheng Wu
Horst Simon

This work was supported by the Director, Office of Science, Office of Laboratory Policy and Infrastructure Management, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

The development and testing of this software used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.

Table of Contents

1 Overview

TRLAN is a program designed to find a small number of extreme eigenvalues and their corresponding eigenvectors of a real symmetric matrix. Denote the matrix as A, the eigenvalue as λ, and the corresponding eigenvector as x, they are defined by the following equation,

$$Ax = \lambda x.$$

There are a number of different implementations of the Lanczos algorithm available[1]. Why another one? Our main motivation is to develop a specialized version that only target the case where one wants both eigenvalues and eigenvectors of a large real symmetric eigenvalue problems that can not use the shift-and-invert scheme. In this case the standard non-restarted Lanczos algorithm requires one to store a large number of Lanczos vectors which can cause storage problem and make each iteration of the method very expensive. The underlying algorithm of TRLAN is a dynamic thick-restart Lanczos algorithm. Like all restarted methods, the user can choose how many vectors can be generated at once. Typically, the user choose a moderate size so that all Lanczos vectors can be stored in core. This allows the restarted methods to execute efficiently. This implementation of the thick-restart Lanczos method also uses the latest restarting technique, it is very effective in reducing the time required to compute a desired solutions compared to similar restarted Lanczos schemes, e.g., *ARPACK*[2].

When solving most problems, the three most time-consuming procedures in the Lanczos method are the matrix-vector multiplication, re-orthogonalization and computation of the Ritz vectors. To make this package as small as possible, we have delegated the task of performing the matrix-vector multiplication to the user. This is a reasonable approach because there is simply too many possible ways of performing the operation and the user usually can construct a specialize version that is better than a generic matrix-vector multiplication routine. In addition, there are high quality matrix-vector multiplication routines available as parts of larger packages, for example, P_SPARSLIB, AZTEC, BLOCKSOLVE and PETSc, see Section 5.2 [operator interface], page 21, for details. To reduce the amount of the time spent in re-orthogonalization, we only perform re-orthogonalization if it is necessary. The Ritz vectors are computed as during the restarting process, in TRLAN, we only compute those that are determined to be needed. This reduces the number of Ritz vectors computed. To compute them efficiently, we call the `BLAS` library to perform the actual computation.

The program is implemented in `Fortran 90`. The main advantages of using `Fortran 90` compared to `Fortran 77` is that `Fortran 90` offers dynamic memory management which make it more flexible in terms of allocating temporary work arrays. If there is an array not used for other task, it can be passed into TRLAN, else the user can simple let TRLAN allocate its own work arrays. TRLAN internal allocate all the space it requires up front to avoid repeated call to allocated small pieces of workspace.

Similar to other languages, such as C/C++, `Fortran 90` offers data encapsulation which makes it convenient to pass a significant amount of information cleanly. TRLAN packages

[1] Many mathematical packages are available from NETLIB (`http://www.netlib.org/`) and ACM TOMS (`http://www.acm.org/toms/`).

[2] ARPACK can be found at `http://www.caam.rice.edu/software/ARPACK/`.

a large amount of information in a single object to reduce the size of the external user interface. See Chapter 4 [TRL_INFO module], page 7, for details. A significant advantage of using `Fortran` compared to C/C++ is the ease of using computational libraries such as `BLAS` and `LAPACK`. In fact, most numerical computations of TRLAN are performed using those library functions. Because most machines have vendor optimized `BLAS` and `LAPACK`, being able to effectively use them is crucial to the effectiveness of TRLAN package.

`Fortran 90` also provides some utility functions such as query function for the machine precision, random number generator and timing functions. They make the program more portable across different platforms.

Parts of this document contains details about the software package which may not be of interest to every user. Here are some advice on how to use this document. If you just want to get a feel of how TRLAN looks like in a program, take a look at Chapter 3 [Example], page 4 or the examples come with the software package. Chapter 3 contains a short example that uses mostly default parameters. To assert more control over TRLAN, see Chapter 4 [TRL_INFO module], page 7, and Chapter 5 [TRLAN interface], page 20. If you are somewhat puzzled about how to choose the parameters, see Chapter 6 [Parameters], page 24, for our recommendations. If you don't use `Fortran 90`, see Section 7.3 [other languages], page 27, for what to do. For most users, there is no need to read everything in Section 4.6 [elements], page 13, and Section 4.7 [error code], page 14. If you read the entire document and are still puzzled, contact the authors, see Section 7.5 [contacting authors], page 28.

2 Installation

The source code of the package is available at

`http://www.nersc.gov/~kewu/trlan.tar.gz`.

This document is distributed with the package and is also separately available at

`http://www.nersc.gov/~kewu/ps/trlan-ug.ps`.

The package may be unpacked by

```
tar -xzf trlan.tar.gz
```

If you `tar` program does not recognize flag `z`, you can unpack it in two steps

```
gunzip trlan.tar.gz
tar -xf trlan.tar
```

After this, the files in the package will be unpacked into a directory called '`TRLan`'.

To install the package, you will need a `Fortran 90` compiler, the `BLAS` and `LAPACK` libraries. On parallel machines, MPI is also needed. The compiler name and the options used are specified in the file called '`Make.inc`'. A number of examples are provided in the file for different machines. If your compiler name and library locations are same as one of the examples, you can uncomment the section, comment out the default values, and use the settings. If your compiler has a different name or the libraries are located at a different place, you will need to modify the file to refer to their correct values. The package may be compiled into one of the two library files `libtrlan.a` and `libtrlan_mpi.a` where the former is the sequential version of the package and the latter is the parallel version. To generate them go to '`TRLan`' and type

```
make libtrlan.a
```

or

```
make libtrlan_mpi.a
```

To compile the examples, go to the appropriate subdirectory in '`examples`'. If you are on a sequential or a shared memory computer and there is no subdirectory that matches your computer, the source code in the `SUN` directory can be used. The examples in '`T3E`' and '`psp`' can be run on parallel machines that support MPI. '`Makefile`' in the '`T3E`' and '`psp`' directories are only tested on a Cray T3E. They will need modification in order to be used elsewhere. The examples in directory '`psp`' also need a special supporting library called P_SPARSLIB, read the file '`README`' before try to use it.

There are three examples in most example directories (except '`psp`'), `simple`, `simple77` and `simplec`. These are three programs should be doing the same thing using three different languages. The executables can be generated by `make`

```
make simple simplec simple77
```

They should output the same eigenvalues, however the actual printout may differ slightly due floating-point round-off errors.

For further questions, consult the '`README`' files in the directories. To report errors in the installation procedure or suggest improvements, the authors can be reached at `kwu@lbl.gov` (Kesheng Wu) and `hdsimon@lbl.gov` (Horst Simon).

3 A small example

This is a simple example in Fortran 90. It is short because we have used a very simple matrix and used default parameters wherever possible. It uses MPI to handle data communication required by TRLAN. This example comes with the distribution of the source code in directory 'examples/T3E'. The name of the file is 's1.f90' and on T3E is can be compiled by make s1 which generates the executable s1.

```
!!! a really simple example of how to use TRLAN
Program simple
  Use trl_info
  Use trl_interface
  Implicit None
  Include 'mpif.h'
  ! local variable declaration
  Integer, Parameter :: nrow=100, lohi=-1, ned=5, maxlan=40, mev=10
  Double Precision :: eval(mev), evec(nrow, mev)
  Type(trl_info_t) :: info
  Integer :: i
  External diag_op  ! name of the matrix-vector multiplication routine
  Call MPI_INIT(i)  ! initialize MPI
  ! initialize info -- tell TRLAN to compute NED smallest eigenvalues
  Call trl_init_info(info, nrow, maxlan, lohi, ned)
  ! call TRLAN to compute the eigenvalues
  Call trlan(diag_op, info, nrow, mev, eval, evec, nrow)
  Call trl_print_info(info, nrow+nrow)
  If (info%my_pe .Eq. 0) Then
     write (6, FMT=100) (i, eval(i), i=1,info%nec)
  End If
100 Format('E(', I1, ') = ', 1PG25.17)
  Call MPI_finalize(i)
End Program simple
!!!
! a simple matrix-vector multiplications routine
! defines a diagonal matrix with value (1, 4, 9, 16, 25, 36, ...)
!!!
Subroutine diag_op(nrow, ncol, xin, ldx, yout, ldy)
  Implicit None
  Integer, Intent(in) :: nrow, ncol, ldx, ldy
  Double Precision, Dimension(ldx*ncol), Intent(in) :: xin
  Double Precision, Dimension(ldy*ncol), Intent(out) :: yout
  Include 'mpif.h'
  ! local variables
  Integer :: i, j, ioff, joff, doff
  Call MPI_COMM_RANK(MPI_COMM_WORLD, i, j)
  doff = nrow*i
  Do j = 1, ncol
     ioff = (j-1)*ldx
     joff = (j-1)*ldy
     Do i = 1, nrow
```

```
        yout(joff+i) = (doff+i)*(doff+i)*xin(ioff+i)
     End Do
  End Do
End Subroutine diag_op
```

There are two parts in this example, the main program and the matrix-vector multiplication subroutine. The main program sets up the `info` variable to carry information to and from TRLAN, calls TRLAN, and prints the information carried out in `info` and the eigenvalues computed. Here is a short explanation of the arguments to `trl_init_info`.

```
call trl_init_info(info,      ! the variable to be set
                   nrow=100,  ! there are 100 rows on each processor
                   maxlan=40, ! maximum Lanczos basis size is 40
                   lohi=-1,   ! compute the smallest eigenvalues
                   ned=5)     ! compute 5 eigenvalues
```

The calling sequence of TRLAN is fairly simple because all the gory details are hidden inside `info`. The following listing describes the information required by TRLAN to solve an eigenvalue problem.

```
call trlan(diag_op,           ! matrix-vector multiplication routine
           info,              ! what eigenvalues to compute, etc.
           nrow,              ! 100 rows on this processor
           mev,               ! number of eigenpairs can be stored in
                              ! eval and evec
           eval,              ! real(8) :: eval(mev)
                              ! array to store eigenvalue
           evec,              ! real(8) :: evec(lde,mev)
                              ! array to store the eigenvectors
           lde)               ! the leading dimension of evec
```

The content of `info` and the eigenvalues are printed separately. The content of `info` is printed by calling `trl_print_info` which accepts two arguments, `info` to be printed and the number of floating-point operations used for one matrix-vector multiplication on this processor. The second parameter is needed because the matrix-vector multiplication is user-supplied. The information is used to compute the speed of the matrix-vector multiplication and the speed of the whole program. It can be ignored, in which case `trl_print_info` will leave the related fields blank.

The short matrix-vector multiplication routine, `diag_op`, performs multiplication with a very simple matrix, `diag(1, 4, 9, ...)`. The example tries to find 5 smallest eigenvalues of this matrix, 1, 4, 9, 16, 25. The following is the output from a run on a T3E/900 located at the National Energy Research Supercomputing Center[1].

```
                                  1998/09/24 18:37:16.834 (-07:00)
 TRLAN execution summary (exit status = 0 ) on PE  0
Number of SMALLEST eigenpairs:          6 (computed),      5 (wanted)
Times the operator is applied:        847 (MAX:         2000 )
Problem size:                         100 (PE:   0),     400 (Global)
Convergence tolerance:          1.490E-08 (rel),   2.384E-03 (abs)
Maximum basis size:                    40
```

[1] Information about the National Energy Research Supercomputing Center can be found on the web at `http://www.nersc.gov/`.

```
Restarting scheme:                        0
Number of re-orthogonalizations         847
Number of (re)start loops:               36
Number of MPI processes:                  4
Number of eigenpairs locked:              0
OP(MATVEC):               9.35440E-03 sec,      1.81091E+01 MFLOPS
Re-Orthogonalization:     2.12016E-01 sec,      4.68742E+01 MFLOPS
Restarting:               2.36795E-01 sec,      2.02369E+01 MFLOPS
TRLAN on this PE:         5.29851E-01 sec,      3.00020E+01 MFLOPS
 -- Global summary --
             Overall     MATVEC    Re-orth    Restart
Time(ave) 5.2985E-01 9.4129E-03 2.1143E-01 2.3685E-01
Rate(tot) 1.2001E+02 7.1990E+01 1.8801E+02 8.0930E+01
E(1) =   0.99999999997742750
E(2) =    3.9999999999816311
E(3) =    8.9999999999916049
E(4) =    16.000000000026944
E(5) =    25.000000000089663
E(6) =    36.000000000367905
```

In short, to use TRLAN to find some extreme eigenvalues, the user defines a matrix-vector multiplication routine with the same interface as `diag_op`, calls `trl_init_info` to specify what eigenvalues to compute and calls `trlan` to perform the bulk of the computation. The remainder of this manual will explain the user interface and how to control TRLAN in more detail. How to Write a particular matrix-vector multiplication for an operator is beyond the scope of this manual. Some packages containing distributed matrix-vector multiplications routines are listed in Section 5.2 [operator interface], page 21.

4 TRL_INFO module

The example in previous chapter uses two modules, TRL_INFO and TRL_INTERFACE. As the name suggested, TRL_INTERFACE contains the user interface for accessing TRLAN. The module TRL_INFO only contains the definition of the `Fortran 90` derived type *TRL_INFO_T*. To make it easy to access, we have provided six access functions, `trl_init_info`, `trl_set_debug`, `trl_set_iguess`, `trl_set_checkpoint`, `trl_print_info`, and `trl_terse_info`. The first four are for manipulate input parameters to TRLAN and the last two are for printing the content of *TRL_INFO_T*. We will discuss these access functions in this chapter. The remaining interface functions are described in the next Chapter. The last two sections of this chapter may be skipped if the reader is only seeking information on how to use the package.

4.1 Initialization

This initialization routine is equivalent to a generator function in `C++`. It is intended to be called before any other TRLAN functions. Any parameter not explicitly set by the caller is set to its default value and all internal counters are set to zero. Its `Fortran 90 interface` block is as follows,

```
Subroutine trl_init_info(info, nrow, mxlan, lohi, ned, tol,&
     & trestart, maxmv, mpicom)
  Use trl_info
  Integer, Intent(in) :: lohi, mxlan, ned, nrow
  Integer, Intent(in), Optional :: maxmv, mpicom, trestart
  Real(8), Intent(in), Optional :: tol
  Type(TRL_INFO_T), Intent(out) :: info
End Subroutine trl_init_info
```

We have seen the mandatory arguments in the example, however, there are four optional arguments that were not used before. For completeness, we will give a short description of all arguments here.

`info`: The `Fortran 90` derived type that will carry the information to the `trlan` subroutine. It is set by this subroutine. Any prior content will be cleared.

`nrow`: The *local* problem size. The vectors are assumed to be distributed conformally, i.e., if 10 elements of a Lanczos vector are located on a processor, the same 10 elements of all other Lanczos vectors are located on the same processor. The variable `nrow` refers to the number of rows located on the current processor. It may vary from processor to processor.

`maxlan`: The maximum Lanczos basis size. This determines the maximum memory requirement of `trlan`. The restarted Lanczos algorithm will store up to `maxlan` Lanczos vectors and one (1) residual vector. An additional memory of size `maxlan*(maxlan+10)` is required to perform the Rayleigh-Ritz projection to compute the approximate solutions. Generally, the larger `maxlan` is, the fewer matrix-vector multiplications are needed. See Section 6.1 [basis size], page 24, for further discussion on this parameter.

`lohi`: This parameter indicates which end of the spectrum to compute. The Lanczos algorithm is only able to compute the extreme eigenvalues effectively. The choices are either to compute the smallest ones (`lohi` < 0), or the largest ones (`lohi` > 0), or whatever converges first (`lohi` = 0).

`ned`: The number of eigenvalues and eigenvectors desired.

The parameters, `nrow`, `maxlan`, `lohi`, and `ned`, are mandatory when calling `trl_init_info`. The following parameters are optional because a reasonable value can be determined by `trl_init_info`.

`tol`: The relative tolerance on the residual norms. The Lanczos algorithm computes the approximate solution to the eigenvalue problem. As more steps are taken, the solutions become more accurate. For symmetric eigenvalue problems, the residual norm is one of the most commonly used measure of the solution accuracy. If the approximate eigenvalue is λ, and the approximate eigenvector is x, the residual norm is defined to be $r = \|Ax - \lambda x\|$. In TRLAN, the convergence test is relative the norm of the matrix A. If

$$r < tol\|A\|,$$

then the approximate solution is considered converged. If this argument is not present, it is set to the square root of the unit round-off error. If the 8-byte IEEE floating-point arithmetic is used, this default value is roughly 1.49×10^{-8}.

`restart`: The flag to indicate which thick-restart scheme to use. In version 1.0 of TRLAN, there are five choices for this parameter, 1, 2, 3, 4, 5. If this parameter is not provided, the default choice is 0 which is treated same as 1 in the current implementation. See Section 6.3 [restarting scheme], page 25, for further discussion on this parameter.

`maxmv`: The maximum number of matrix-vector multiplications allowed. The purpose of this parameter is usually to make sure the program stop eventually in case of stagnation. The default value is `ned*ntot` where `ntot` the global problem size.

`mpicom`: The MPI communicator to be used by `trlan`. This parameter is only meaningful if MPI is used. If the sequential version is used, this variable is simply ignored internally. If MPI is used and this variable is not set, `trl_init_info` will duplicate `MPI_COMM_WORLD` and use the resulting communicator for its internal communication operations.

4.2 Setting debug parameters

There are cases we would like to monitor the progress of the restarted Lanczos algorithm. We can do this by setting a few logging parameters. Since the debug information may be voluminous, `trlan` writes them to files. Each MPI process will write its own debug information to a separate file. The name of the file and how much debug information to write is controlled by calling `trl_set_debug`.

```
Subroutine trl_set_debug(info, msglvl, iou, file)
  Use trl_info
```

```
    implicit none
    Type(TRL_INFO_T), intent(inout) :: info
    integer, intent(in) :: msglvl, iou
    Character*(*), Optional :: file
  End Subroutine trl_set_debug
```

`info`: The `TRL_INFO_T` type variable to be modified. The function `trl_init_info` should have been called before calling `trl_set_debug`.

`msglvl`: This parameter controls how much debug information to print. If it is zero or less, nothing is printed. When its value is between 1 and 10, the larger it is, the more information is printed. When it is larger than 10, it has the same effect as 10.

The function `trl_init_info` sets it to zero as the default value.

`iou`: The Fortran I/O unit number to be used when writing debug information. The user should choose an I/O unit not used for anything else during the time `trlan` is being used.

`Trl_init_info` sets it to 99 as the default value.

`file`: The leading part of the debug file names. The debug file names are form by appending the MPI processor rank to this string. In sequential environment, MPI processor rank is always zero (0). This is an optional argument to `trl_set_debug`. When it is not set, the corresponding element of `TRL_INFO_T` is not changed.

`Trl_init_info` sets this elements to '`TRL_LOG_`' by default.

4.3 Setting initial guess options

TRLAN program can either use a user-supplied initial guess, generate an arbitrary initial guess or read a set of checkpoint file to get starting vectors. The thick-restart Lanczos may start with arbitrary number of vectors, however, the starting vectors have to satisfy a strict relation. The simplest way to start the algorithm is to simply provide one starting vector. If the function `trl_set_iguess` is not called, the starting vector is set to [1, ..., 1] by default. The checkpoint option is implemented to enable a user to continue improve the accuracy of the solutions progressively.

```
  Subroutine trl_set_iguess(info, nec, iguess, oldcpf)
    Use trl_info
    Implicit None
    Type(TRL_INFO_T) :: info
    Integer, Intent(in) :: iguess, nec
    Character(*), Intent(in), Optional :: oldcpf
  End Subroutine trl_set_iguess
```

`info`: The `TRL_INFO_T` type variable to be modified. The function `trl_init_trl` should have been called before calling `trl_set_iguess`.

`nec`: The number of eigenvalues and eigenvectors already converged. If `nec` is greater than zero (0), the first `nec` columns of array `evec` should contain eigenvectors of the operator and the first `nec` elements of `eval` should contain the corresponding eigenvalues. This is designed to allow the user to return to `trlan` to compute more eigenvalues and eigenvectors.

`Trl_init_info` sets this value to zero to indicate no converged eigenvalues.

`iguess`: The parameter to indicate option for initial guess vector.

- `<1`: TRLAN will generate an arbitrary starting vector for the Lanczos algorithm. If it is zero (0), vector [1, 1, ..., 1] is used. When `iguess` is less than zero, a random perturbations will be added to this vector before it is taken as the starting vector.
- `1`: The user has supplied a starting vector. It will be used.
- `>1`: TRLAN will read a checkpoint file and use its content to start the Lanczos process. The idea of checkpoint is explained later.

`oldcpf`: The leading portion of the existing checkpointing file names. As with the log files, the checkpoint files are named by concatenating this leading portion and the MPI processor rank. The default value set by `trl_init_info` for this is 'TRL_CHECKPOINT_'.

NOTE: Reading the checkpoint files are done through I/O unit `cpio`. `Trl_init_info` sets this value to 98 by default. If I/O unit is used for another task already, use `trl_set_checkpoint` to set `cpio` to an unused I/O unit number.

4.4 Checkpointing

TRLAN has implemented a scheme of checkpointing to allow the user to stop and restart. The checkpoint files of the thick-restart Lanczos algorithm contains all the information necessary for it to continue the Lanczos iterations. To minimize the size of the files, the checkpoint files are written at the end of the restart process because the basis is the smallest in size at this point. For efficiency reasons, each MPI processor writes its own checkpoint file in FORTRAN unformatted form. These checkpoint files can only be read on the same type of machines and using the same number of MPI processors. The function `trl_set_iguess` controls whether the checkpoint files are read. The following function controls when to write the checkpoint files.

```
Subroutine trl_set_checkpoint(info, cpflag, cpio, file)
  Use trl_info
  Implicit None
  Type(TRL_INFO_T) :: info
  Integer, Intent(in) :: cpflag, cpio
  Character(*), Optional :: file
End Subroutine trl_set_checkpoint
```

`info`: The `TRL_INFO_T` type variable to be modified. The function `trl_init_trl` should have been called before calling `trl_set_checkpoint`.

`cpflag`: If this value is greater than zero, then TRLAN will write `cpflag` set of checkpointing files in `maxmv` iterations. If it is less or equal to zero, no checkpoint file is written. Checkpointing files are only written if TRLAN runs correctly. If `cpflag` is greater than zero, at least one set of checkpoint files is written when TRLAN completes successfully. To debug the program, turn on verbose printing by using `trl_set_debug`.

`Trl_init_info` sets this value to zero.

`cpio`: The FORTRAN I/O unit number to be used for writing checkpoint files. The value of `cpio` is set to 98 by default (`trl_init_info`).

`file`: The leading portion of the checkpoint files. The checkpoint file names are formed by concatenating the value of this variable and the MPI processor rank. If this argument is not present internally, the corresponding element of `TRL_INFO_T` is not modified.

`Trl_init_info` sets this variable to '`TRL_CHECKPOINT_`' by default.

4.5 Printing functions

Upon returning from `trlan`, the user may wish to exam the progress of `trlan`. One simple way to do this is to printout the content of `info`. There are two printing functions `trl_print_info` and `trl_terse_info`. Function `trl_print_info` is the one that printed the results in Chapter 3 [Example], page 4. The following is the same information printed using `trl_terse_info`.

NOTE: The eigenvalues are not stored in `info`. The following printout is from a different run of the same example, there is slight difference in time.

```
MAXLAN:      40, Restart:        0,   NED:  -    5,     NEC:       6
MATVEC:     847,  Reorth:      847, Nloop:      36, Nlocked:       0
Ttotal:0.535910,    T_op:0.008962, Torth:0.209496,  Tstart:0.238200
```

The `Fortran 90 interface` of the two subroutines are as follows.

```
Subroutine trl_print_info(info, mvop)
  Use trl_info
  Implicit None
  Type(TRL_INFO_T), Intent(in) :: info
  Integer, Intent(in) :: mvop
End Subroutine trl_print_info
Subroutine trl_terse_info(info, iou)
  Use trl_info
  Implicit None
  Type(TRL_INFO_T), Intent(in) :: info
  Integer, Intent(in) :: iou
End Subroutine trl_terse_info
```

`info`: The `TRL_INFO_T` variable to be printed. In addition of keeping track of how many eigenvalues have converged and how many are wanted. There are significant amount of information about how many matrix-vector multiplications have been used, how much time is used in various parts of the program, and so forth. The verbose version of the printing function will printout most of the recorded information that are deemed to be useful. The terse version only printout the 12 most important fields.

`mvop`: The number of floating-point operations performed on one processor during one matrix-vector multiplication. This information is used by the printing function `trl_print_info` to determine the speed of the matrix-vector multiplication and the speed of the overall eigenvalue program. If this information is not present, the relevant fields are left blank in the printout. The variable `info` contains timing information and floating-point operations performed inside the `trlan`. Since the matrix-vector multiplication is a user-supplied function, the user has to provide information about its complexity.

`iou:` The terse printing function `trl_terse_info` is allowed to print to any valid `FORTRAN` I/O unit. This is different from the verbose printing function where the printout is always sent to the I/O unit that is used for logging debugging information.

In addition to the differences mentioned already, the function `trl_print_info` requires every processor to participate but `trl_terse_info` can be called by each processor individually. Because of this reason, `trl_print_info` can provide global performance information but not `trl_terse_info`.

The verbose version of the printout is designed to be self-explanatory. The rate fields for floating-point operations are in MFLOPS. The rate of `Read` and `Write` refers to the speed of reading and writing checkpoint files, they are in MegaBytes per second. Since the simple example shown does not use checkpointing, no information regarding `Read` and `Write` is presented in the printout.

The heading for the terse version of the printout is bit cryptic. They are

`MAXLAN:` The maximum Lanczos basis size.

`Restart:` The flag of restarting scheme to be used, 0, ..., 5.

`NED:` Number of eigenvalues desired. It also contains an one-character sign which can be `+`, `-`, or `0` to indicate which end of the spectrum is being computed.

`NEC:` Number of eigenvalues converged.

`MATVEC:` number of times the operator has been applied, i.e., the number of matrix-vector multiplications, also the number of iterations.

`Reorth:` Number of times re-orthogonalization has been applied. Each time the Gram-Schmidt procedure is called, this counter is incremented by one.

`Nloop:` Number of outer/restarted iterations.

`Nlocked:` Number of Ritz pairs that have extremely small residual norms (`< epsilon || A ||`). Because the residual norms are so small, we lock the Ritz pairs to reduce the among of arithmetic operations needed in Rayleigh-Ritz projection.

`Ttotal:` The total time (seconds) used by TRLAN.

`T_op:` The time (seconds) spent in performing matrix-vector multiplications.

`Torth:` The time (seconds) spent in performing re-orthogonalizations.

`Tstart:` The time (seconds) spent in restarting including performing Rayleigh-Ritz projections.

4.6 The elements of TRL_INFO_T

In some instances, it might be necessary to directly access the status information in `info` rather than print out the information. Whether `trlan` terminated because of some kind of error, the two elements of `TRL_INFO_T` that are most like to be useful after returning from `trlan` are `stat` and `nec`, where the first one is the error flag of `trlan` and the second one indicates how many eigenvalues and eigenvectors have converged.

The input parameters to TRLAN have appeared in the calling sequence of `trl_init_info`, `trl_set_debug`, `trl_set_iguess` and `trl_set_checkpoint` are described earlier in this chapter. The following are elements of `TRL_INFO_T` are counters set by `trlan`. To the user, they are part of the output from TRLAN.

`my_pe`
`npes` The PE number and the number of PEs.

`ntot` The global size of the problem. `ntot` $= \sum_{allPEs}$ `nrow`.

`matvec` The number of matrix-vector multiplications used by the restarted Lanczos algorithm. It will not significantly exceed the value of `maxmv`.

`nloop` The number of restarting loops, i.e., the number of times `trlan` has reached the maximum size and restarted.

`north` The number of times the Gram-Schmidt process is invoked to perform re-orthogonalization.

`nrand` The number of times `trlan` has generated random vectors in attempt to produce an vector that is orthogonal to the current basis vectors.

`locked` The number of eigenpairs that has extremely small residual norms (`< epsilon || A ||`). The accuracies of these eigenpairs can not be further improve by more Rayleigh-Ritz projection, therefore they are *locked* to reduce arithmetic operations, they are only used to perform re-orthogonalization but nothing else. TRLAN locks not only the wanted eigenpairs with small residual norm, it also locks unwanted ones depending the restarting strategy. The rational is that if they converge quickly, it is not good idea to throw them away because they will reappear in the next basis built.

`clk_rate` The clock rate of as measured by `Fortran 90` intrinsic function `system_clock`.

`clk_max` The maximum clock ticks before it rolls over.

`clk_tot tick_t clk_op tick_o clk_orth tick_h clk_res tick_r`
These set of integer and floating-point variables are used to keep track of time spend in performing matrix-vector multiplication (`clk_op`, `tick_o`), re-orthogonalization (`clk_orth`, `tick_h`), restarting (`clk_res`, `tick_r`), and the whole `trlan` (`clk_tot`, `tick_t`). The four integer counters, `clk_op`, `clk_orth`, `clk_res`, and `clk_tot`, are output from `Fortran 90` intrinsic function `system_clock`. Since it is likely the integer counters will overflow in some cases, each of them has a floating-point counterpart. If they become larger than a quarter of the maximum counter value `clk_max`, they are added to the floating-point counters and reset to zero.

`flop rflp flop_h rflp_h flop_r rflp_r`
: These set of counters are for counting the number of floating-point operations used by TRLAN. `Flop` and `rflp` are for counting the total floating pointer operations excluding those used by matrix-vector multiplications. `Flop_h` and `rflp_h` are for counting the re-orthogonalization procedure. `Flop_r` and `rflp_r` count operations used in restarting. The integer counters are used initially until they become larger than `clk_max/4`. Once they become too large, their values are added to the corresponding floating-point counter and they are reset to zero. Since the matrix-vector multiplication routine is supplied by the user, TRLAN can not account for the floating-point operations used in that procedure.

`crat tmv tres trgt`
: This set of variables are used to track the convergence factor of the eigenvalue method. The variable `crat` is the convergence factor. It is measured after `tmv` number of matrix-vector multiplications are used. The value of the target at `tmv` was `tres`. The convergence factor is updated as follows. Among the Ritz values, search for the one that is closest to `trgt`. This Ritz value is regarded as the updated version of the previous target Ritz value. Let `res` be the residual norm of the Ritz value, the convergence factor is computed as `crat` = Exp(Log(`res / tres`) / (`matvec - tmv`)).

  After `crat` is updated, the current target value is identified as the first Ritz value that is not converged yet. The value of `tmv` and the corresponding residual norm `tres` are recorded.

`anrm` The estimated norm of the matrix. This is the largest absolute value of any Ritz value ever computed by TRLAN. After a number of steps, this is a good estimate of the matrix 2-norm. This variable is primarily used in the convergence test. If the user specifies a tolerance `tol`, all the Ritz pairs with residual norm less than `tol * anrm` are considered converged.

`stat` The error flag. The next section describes the meaning of the error codes in detail.

4.7 TRLAN error code

This section lists all error numbers defined in TRLAN and discusses possible remedies to the errors. Currently (TRLAN version 1.0), defines the follow error numbers,

`0` No error.

  It is possible that `trlan` has not computed all wanted eigenpairs, check the value of `nec` to see exactly how many wanted eigenpairs have converged. In case you have not computed all wanted eigenpairs, the possible solutions are:

  - If checkpoint files were written, restart with the checkpoint files.
  - If no checkpoint files were written, make a linear combination of the approximate eigenvectors and use the resulting vector as the initial guess. In addition, make sure to set appropriate options with `trl_set_checkpoint` to generate checkpoint files for future use.

- Increase the maximum basis size `maxlan`, See Section 6.1 [basis size], page 24 for more details.
- Increase the maximum number of iterations allowed `maxmv`, See Section 6.4 [maximum iterations], page 25 for more details.
- Use a different restarting strategy, See Section 6.3 [restarting scheme], page 25 for more details.

`-1` The internal record of local problem size (`nloc`) does not match the value of `nrow` used when calling `trlan`. Most likely the user has used the `info` variable defined for a different problem.

SOLUTION: Make sure `trl_init_info` is called before `trlan` and the arguments to both functions are correct for the intended eigenvalue problem.

`-2` The leading dimension of the eigenvector array `evec` is smaller than local problem size, `lde < nrow`. There isn't enough space in `evec` to store the eigenvectors correctly.

SOLUTION: Allocate the array `evec` with leading dimension larger or equal to `nrow`.

`-3` The array size of `eval` is too small to store the eigenvalues, `mev < info%ned`. There isn't enough columns in `evec` either.

SOLUTION: Increase the size of array `eval` and increase the number of columns in `evec`.

`-4` TRLAN failed to allocate space for storing the projection matrix, etc.. The size of this work array (internally called `misc`) is `maxlan * (maxlan + 10)`. TRLAN tries to allocate its own workspace if the user has not provided enough workspace to store the Lanczos basis vectors and the projection matrix, et al.

SOLUTION:

- If there is addition workspace not used, given TRLAN more workspace.
- Decrease the size of `maxlan`. This will decrease the among of workspace required.
- If you have control over the swap file/partition size, increase it will also solve this problem.

`-5` TRLAN failed to allocate space to store the Lanczos vectors. The workspace (internally called `base`) size required here is `(maxlan + 1 - mev) * nrow`.

SOLUTION: See solutions for error code `-4`.

`-11` TRLAN does not have enough workspace to perform Gram-Schmidt procedure which is used to perform re-orthogonalization. This should not happen unless the caller directly calls lower level routine `trlanczos` with insufficient workspace.

SOLUTION: Increase workspace provided.

`-12` TRLAN does not have enough workspace to compute eigenvalues of a symmetric tridiagonal matrix. This should not happen unless the caller directly uses the lower level routine `trlanczos` with insufficient workspace.

SOLUTION: Increase workspace provided.

`-101` The orthogonalization routine of `TRLAN` does not have enough workspace. This should not happen unless `trl_orth` is directly called outside of TRLAN with insufficient workspace.

SOLUTION: Increase workspace provided.

`-102` The norm of the residual vector of Lanczos iterations is not a set of valid finite floating-point numbers. Unless the operator norm is exceeding large, say large than `1E160`, this error code should not be generated. If it is, normally it is an indication of other errors. For example, the workspace given by the user is not as large as the user indicated to `trlan`, the array `evec` is actually smaller than `(lde, mev)`, the first `nec` columns of `evec` are not orthonormal vectors on input, or you have encounter a bug in TRLAN or one of the libraries used by TRLAN.

SOLUTION:

- Make sure the workspace array given to `trlan` is as large as claimed, i.e., the actual size of `wrk` is at least as large as `lwrk`. If `wrk` is present but not `lwrk`, the actual size of `wrk` should be no less than `mev`.
- Make sure there is enough space to store the eigenvalues and eigenvectors even if you think `trlan` is not going to compute all the eigenvectors because `trlan` uses the space in `evec` to store the Lanczos vectors during its computations.
- If the initial `nec` is not zero, make sure that the known eigenvectors are stored in the first `nec` columns of `evec` and the eigenvalues in the first `nec` elements of `eval`.

The above solution should be considered applicable to all the following error conditions. If you have checked everything suggested here, then you may have found an error in TRLAN program. Report the problem to the authors.

`-111` Insufficient workspace to one of a lower level routine used by TRLAN to reduce the arrowhead of the projection matrix into tridiagonal matrix. This should not happen unless the user directly calls the low level routine.

SOLUTION: Report the problem to the authors.

`-112` TRLAN has failed to generate an orthogonal transformation to reduce the projection matrix into a tridiagonal matrix, i.e., `LAPACK` routine `dsytrd/ssytrd` has failed. This is extremely unlikely to happen.

SOLUTION: Check to make sure you have a correct version of the `LAPACK`. If your `LAPACK` is installed correctly, see suggestions for error `-102`.

`-113` TRLAN has failed to apply the orthogonal transformation to reduce the projection matrix into a tridiagonal matrix, i.e., `LAPACK` routine `dorgtr/sorgtr` has failed. This is extremely unlikely to happen.

SOLUTION: See solutions to error `-102`.

`-121` Insufficient workspace to compute the eigenvalues of a tridiagonal matrix. This error should not occur if the size of workspace `wrk` passed to `trlan` is no less than `lwrk`.

SOLUTION: See solutions to error `-102`.

-122 TRLAN has failed to computed the eigenvalues of a tridiagonal matrix. This is extremely unlikely to happen.

SOLUTION: See solutions to error -102.

-131 Insufficient workspace to compute the eigenvectors of a tridiagonal matrix. This error should not occur if workspace of correct size was provided to TRLAN.

SOLUTION: See solution to error -102.

-132 TRLAN has failed to compute the eigenvectors of the projection matrix, more specifically, LAPACK routine dstein/sstein has failed. Normally, if this happens, TRLAN will switch to a different method of computing the eigenvectors. It is very unlikely the user will see this error flag. If it does show up, it might be an indication of error in the program.

SOLUTION: See solutions to error -102.

-141 Insufficient workspace to compute the eigenvectors of a tridiagonal matrix. This error should not occur if workspace of correct size was provided to TRLAN.

SOLUTION: See solutions to error -102.

-142 TRLAN has failed because LAPACK routine dsyev/ssyev has failed to compute the eigenvalues and eigenvectors of the projection matrix.

SOLUTION: Check to make sure LAPACK is installed correctly. See solution to error -102.

-143
-144 TRLAN is unable to match the Ritz values selected to be saved with the eigenvalue found by dsyev/ssyev.

SOLUTION: See solutions to error -102.

-201 The Gram-Schmidt procedure is called with insufficient workspace.

SOLUTION: Increase the workspace size. If you did not call trl_cgs directly, make sure workspace size lwrk matches the actual size of wrk when calling trlan.

-202
-203 The Gram-Schmidt process has failed to orthogonalize the given vector to the current basis vectors. This is unlikely to happen. If it does, it indicates two possibly source of problem, the current basis vectors are not orthogonal, or the random vectors generated by TRLAN fall in the space spanned by the current Lanczos vectors.

SOLUTION: One possible solution to this problem is to call FORTRAN 90 random number generator to set each processor with a different seed value. For example, the following code segment will cause random_number to generate different random numbers on each processor the next time it is used and it also produces an random vector as initial guess for the Lanczos iterations.

```
call random_number(evec(1:nrow,1))
do i = 1, info%my_pe
   call random_number(evec(1:nrow,1))
end do
```

If reseting the seed of the random number generator does not fix the problem, then there might be a more serious problem.

See also solutions to error `-102`.

`-204` The vector norm after orthogonalization is not a valid floating-point number.

SOLUTION: See solutions to error `-102`.

`-211` The leading dimension of the arrays are not large enough for storing the vectors in a checkpoint file. This error should have been caught earlier as error `-2` unless the checkpoint files are not for the same problem or not produced with the same number of processors.

SOLUTION: Make sure the checkpoint files are generated on the same type of machines and for the same problem using the same number of processors.

`-212` Unable to open checkpoint files to read.

SOLUTION: Make sure the checkpoint files exist, the names are correct, and the I/O unit number `cpio` is not used for something else already.

`-213` The array size stored in checkpoint file is different from passed in by user. The checkpoint file is probably for a different problem or was generated with different number of processors.

SOLUTION: Make sure the checkpoint files are generated for the same problem and using the same number of processors.

`-214` There are more vectors stored in the checkpoint file than `maxlan`.

SOLUTION: Increase the size of `maxlan`.

`-215` Error was encountered while reading the content of the checkpoint files.

SOLUTION: Make sure the checkpoint files are generated for the same problem on the same type of machines.

`-216` Error was encountered while trying to close the checkpoint file after completed reading.

SOLUTION: This error probably can be ignored. Consult you system administrator.

`-221` Unable to open checkpoint files for writing.

SOLUTION: Make sure the I/O unit number specified to be used for writing checkpoint files are not used for other tasks and make sure you have permission to write files in the location where the program is running.

`-222` Error was encountered while writing the checkpoint files.

SOLUTION: Make sure there is enough space on the disk to store the checkpoint files. If there is `k` Lanczos vectors to be written out, the number of bytes is `8*(2+2*k+nrow*(k+1))` for each processor. The maximum value for `k` is `maxlan`.

`-223` Error was encountered while trying to close the checkpoint files after write them.

SOLUTION: This error probably can be ignored. Consult you system administrator.

This list represents all error code defined in TRLAN version 1.0. The function `trlan` should never return an error code not listed here. If you encountered one and you are sure that all the arguments to TRLAN functions are correct, you have uncovered a flaw in TRLAN, contact the authors with a description of your problem. A short example is always welcome.

5 Main function interfaces

The majority of the arithmetic computations to solve an eigenvalue problem are executed in the main function of TRLAN package and the user's own matrix-vector multiplication function. This section gives a more detail description of the interfaces of these two functions.

5.1 Interface of `trlan`

The main computation kernel of TRLAN package is named `trlan`. We have see a short description of its arguments in Chapter 3 [Example], page 4. To provide a different view of the interface, we show its `Fortran 90 interface` block.

```
Subroutine trlan(op, info, nrow, mev, eval, evec, lde, wrk, lwrk)
  Use trl_info
  Implicit None
  Type(TRL_INFO_T) :: info
  Integer, Intent(in) :: lde, mev, nrow
  Integer, Intent(in), Optional :: lwrk
  Double Precision, Intent(inout) :: eval(mev), evec(lde,mev)
  Double Precision, Target, Dimension(:), Optional :: wrk
  Interface
     Subroutine OP(nrow, ncol, xin, ldx, yout, ldy)
       Integer, Intent(in) :: nrow, ncol, ldx, ldy
       Double Precision, Dimension(ldx*ncol), Intent(in) :: xin
       Double Precision, Dimension(ldy*ncol), Intent(out) :: yout
     End Subroutine OP
  End Interface
End Subroutine trlan
```

Most of the arguments of this subroutine are explained before in Chapter 3 [Example], page 4. However for completeness, we will list all of them here.

`op` The operator routine. It applies the operator on `xin` and stores the resulting vectors in `yout`. In linear algebra terms, this is a matrix-vector multiplication routine. The vectors to be multiplied are stored in `xin` and the resulting vectors are stored in `yout`.

`info` A variable of `Fortran 90` derived type `TRL_INFO_T`. It carries the information to and from TRLAN. See Chapter 4 [TRL_INFO module], page 7, for more details.

`nrow` The number of rows on this processor if the problem is distributed using MPI, else the number of total rows in a Lanczos vector.

`mev` The number of elements in array `eval` and the number of columns in array `evec`. It denotes the maximum number of eigenpairs that can be stored in `eval` and `evec`.

NOTE: Since the array `evec` will be used internal to store
`mev` Lanczos vectors, even if you do not think TRLAN is able to compute `mev` eigenvectors at the end, you still declare `evec` as large as (`nrow, mev`).

eval

evec The arrays used to store the eigenvalue values (eval) and the eigenvectors (evec). On entry to trlan, if info%nec is greater than zero (0), the first info%nec elements of eval shall contain the eigenvalues already known and the first info%nec columns of evec shall contain the corresponding eigenvectors. These eigenpairs are assumed to have zero residual norms and will not be modified by TRLAN. On exit from trlan, the converged solutions are stored in the front of eval and evec, i.e. the first info%nec eigenvalues in eval contains the converged eigenvalues and the first info%nec columns of evec are the corresponding eigenvectors.

lde The leading dimension of array evec. It is expected to be no less than nrow, otherwise the eigenvectors can not be stored properly and trlan will abort with error code info%stat = -2.

wrk

lwrk These two are optional arguments. If both are present, wrk will be used as workspace inside and lwrk shall be the number of elements in array wrk. TRLAN will try to use this workspace if it is large enough for either misc (size maxlan * (maxlan + 10)) or base (size (maxlan + 1 - mev) * nrow). If wrk is present but not lwrk, the workspace size is assumed to be mev. It does not make sense to have only lwrk without argument wrk. If it is the case, lwrk is ignored.

If argument wrk is present and there is enough space to store the residual norms of the solutions, the first info%nec elements of wrk will contain the residual norms corresponding to the info%nec converged solutions.

5.2 Operator interface

TRLAN program require the user to provide his/her own matrix-vector multiplication routine. The matrix-vector multiplication routine needs to have the following interface.

```
Subroutine OP(nrow, ncol, xin, ldx, yout, ldy)
  Integer, Intent(in) :: nrow, ncol, ldx, ldy
  Double Precision, Dimension(ldx*ncol), Intent(in) :: xin
  Double Precision, Dimension(ldy*ncol), Intent(out) :: yout
End Subroutine OP
```

nrow The number of rows on this processor if the problem is distributed using MPI, otherwise the number of total rows in a Lanczos vector.

ncol The number of vectors (columns in xin and yout) to be multiplied.

xin The array to store the input vectors to be multiplied. The following two declarations are equivalent on most machines,

```
Double Precision, Dimension(1:ldx,1:ncol), Intent(in)::xin
Real*8 xin(1:ldx, 1:ncol)
```

The ith column of xin is xin((i-1)*ldx+1 : (i-1)*ldx+nrow) if xin is declared as one-dimensional array. If the user routine actually declare it as a two-dimensional array, the ith column should be xin(1:nrow, i). TRLAN calls OP using Fortran 77 style argument matching, only starting address of `xin` will be passed.

For those who are familiar with `C/C++`: `xin` is actually passed as `double *` that points to the first element of array. Elements in a column are ordered consecutively and the `i`th column starts at `(i-1)*ldx`.

`ldx` The leading dimension of the array `xin` when it is declared as two-dimensional array.

`yout` The array to store results of the multiplication. It can be equivalently declared as

```
Double Precision,Dimension(1:ldy,1:ncol),Intent(out)::yout
Real*8 yout(1:ldy, 1:ncol)
```

The usage notes on `xin` also apply to `yout`.

`ldy` The leading dimension of the array `yout` when it is declared as two-dimensional array.

This simple interface only has enough information to describe the input and output vectors. Here are some possible ways of passing the matrix information to this subroutine. In `Fortran 90`, we recommend using a module to encapsulate information related to the matrix. If `Fortran 77` is used, a `common block` may be used for the same purpose. Normally, if another language like `C` or `C++` is used, the matrix can be packaged in a `struct` or a `class`, and accessed through a global variable.

In case the user does not want to write his/her own matrix-vector multiplication routine. There are a number of packages out there that can be used. Useful software depots and information archives include

`ACM TOMS` *http://www.acm.org/toms/*

`ACTS Toolkit` *http://acts.nersc.gov/*

`National HPCC Software Exchange` *http://nhse.cs.utk.edu/*

`NETLIB` *http://www.netlib.org*

`Scientific Application on Linux` *http://SAL.KachinaTech.COM/*

Potentially useful packages include

`Aztec` *http://www.cs.sandia.gov/CRF/aztec1.html*

`BlockSolve` *http://www.mcs.anl.gov/sumaa3d/BlockSolve/*

`P_SPARSLIB` *http://www.cs.umn.edu/Research/arpa/p_sparslib/psp-abs.html*

`PETSc` *http://www.mcs.anl.gov/petsc/*

`SPARSKIT` *http://www.cs.umn.edu/Research/arpa/SPARSKIT/sparskit.html*

NOTE: All of the packages mentioned above have matrix-vector multiplications routines. However some of them are designed for solving linear systems or even larger granularity tasks, some effort may be required to directly using their matrix-vector multiplication routines.

6 Recommended parameter choices

Before calling `trlan`, the user needs to decide a few parameters. The most important parameters are arguments to function `trl_init_info`. The parameters like `nrow`, `lohi` and `ned` are determined by the problem to be solved, other parameters to control the execution of TRLAN might not be familiar to casual users. This part of the manual will give some recommendations on how to determine those parameters.

6.1 Selecting the maximum basis size (`maxlan`)

A few factors come into play when picking the maximum basis size `maxlan`, for example, the available computer memory size, the number of eigenvalues wanted, and the separation of wanted eigenvalues from the others. The first rule of thumb is that `maxlan` should at least as large as

```
ned + min(6, ned).
```

Generally, the larger it is, the better TRLAN will perform. The limitation on using a very large basis is that there might not be enough computer memory to store the basis in memory. Another concern regarding using a large basis is that the Gram-Schmidt orthogonalization process will be expensive. In addition, if the basis size is larger than 1000, then the time spent in finding the eigenvectors of the projection matrix may be a substantial portion of the overall execution time.

If the wanted eigenvalues are easier to compute compared to others, then it does not matter how large the basis size is, the restarted Lanczos method will find the solutions fairly quickly. If the wanted eigenvalues converge slower than the unwanted ones, such as the example in Chapter 3 [Example], page 4, then the above recommended minimum size is too small to be effective. In this case, the user should look at how many eigenvalues were locked and compare it with the number of eigenvalues converged. In difficult case, it is not unusually to see a large number of unwanted eigenpairs converge before the wanted one are finally computed. In the previous example, the minimum recommended basis size is 11. Since it is relatively small and we know the eigenvalue problem is relatively hard, we first tried `maxlan = 20`. After 2,000 matrix-vector multiplications, there are two wanted eigenvalues converged, and six eigenvalues were locked. After this first test, we use the following guidelines to choose the next basis size.

1. Add two to `maxlan` for each locked eigenvalue.
2. Increase the basis size by a factor of `ned / nec`.

The first rule suggests the new basis size of about 30 and the second suggest the next choice could be 50. The basis size used in the example is 40. We are able to find the five smallest eigenvalues with this choice. Further tests show that using basis size of 30 can compute the same 5 eigenvalues in 1056 matrix-vector multiplications, and using a basis size of 50 TRLAN only need 777 matrix-vector multiplications. However, in both cases, more time was used. This demonstrates the complexity of the choice. In this particular case, either 30, 40 or 50 is a reasonable choice.

6.2 Selecting the tolerance

The convergence test used in this program is `r < tol * || A ||`. Normally, if the matrix is stored, the accuracy of the matrix-vector multiplication routine is on the order of `epsilon * || A ||`. The unit round-off error (`epsilon`) of a 64-bit IEEE floating-point number is approximately `2.2E-16`. The default value of `tol` is about `1.49E-8`. Typically, if 5 digits of accuracy is desired for the eigenvectors, `tol` should be set to `1E-5`.

6.3 Selecting a restarting scheme

This is another parameter that can change the execution time dramatically. However, effective restarting schemes are still subject of active academic researches. On the example given before, schemes 1 and 2 uses about the same amount of matrix-vector multiplications which are more than the number of matrix-vector multiplications used with schemes 3 and 4. However, because schemes 3 and 4 perform more restarts and they save more basis vectors during restarting, their restarting procedures are more expensive. The actual execution time with schemes 3 and 4 are longer than those with schemes 1 and 2. Based on these observations, schemes 3 and 4 are better if the matrix-vector multiplication is very time-consuming, say, one matrix-vector multiplication takes more time than an average restart. If the matrix-vector multiplication is relatively inexpensive, then schemes 1 and 2 are preferred. Scheme 5 attempts to mimic the restarting strategy in ARPACK, in many cases, it has comparable performance as the scheme 1.

6.4 Selecting the maximum iterations

TRLAN is stopped usually after it has found all the wanted eigenvalues and the corresponding eigenvectors. The other normal stopping condition is to stop after `maxmv` number of matrix-vector multiplications. Typically, we would allow a fixed number of matrix-vector multiplications for each eigenvalue, for example, 100 per eigenvalue. When we are trying to find the correct value to use for `maxlan` and `restart` we may limit the number of matrix-vector multiplications used to reduce the time consumed. The default value in `trl_init_info` is very large especially for large problems. An more acceptable limit might be 1000 matrix-vector multiplications per eigenvalue. This should be sufficient for most problems. If more than 1000 matrix-vector multiplications are used to compute one eigenvalue, other means of computing eigenvalues should be tried. For example, the shift-and-invert Lanczos method is often able to compute the desired eigenvector in a few steps. The shift-and-invert scheme computes the extreme eigenvalues of $(A - \lambda I)^{-1}$ first, then derive the actual eigenvalues of A. To use this scheme, one need to either invert the matrix or at least being able to solve linear systems, $(A - \lambda I)u = v$. If neither is feasible, then the Davidson method might be an alternative to consider.

If TRLAN does not return with status 0, consult Section 4.2 [debug parameters], page 8, to setup a debugging session, and refer to Section 4.7 [error code], page 14, for error encountered and possible solutions.

7 Miscellaneous issues

7.1 Workspace requirement

Some of the issues related to workspace requirements have been mentioned through out this manual. This section provide a central location to collect all the information for ease of reference.

Inside of `trlan`, there are three large chucks of workspace, `evec`, `base` and `misc`. The user always provides the array `evec`, since it is necessary to carry input and output information for TRLAN. Its size is clearly defined in the calling sequence by `lde` and `mev`. The array `base` is used to store the basis vectors if the array `evec` can not store `maxlan+1` vectors. Given the maximum basis size `maxlan`, the size of `base` is `(maxlan + 1 - mev) * nrow)`. The array `misc` is used to store the projection matrix, the eigenvalues and eigenvectors of the projection matrix, workspace required by all lower level routines of TRLAN, library routines from `LAPACK` and `BLAS`. Its size should be no less than `maxlan * (maxlan + 10)`. If it is larger in size, some library routines might run faster. Thus if there is large amount of computer memory, the user can let TRLAN use it by pass in a large array `wrk`.

If the user provides a workspace `wrk` to `trlan`, then its size is checked to see either one of `misc` or `base` or both of them can fit inside the workspace. If at lease one of them can fit into `wrk`, it would be used. If `wrk` is large enough for both `base` and `misc`, the array `base` will use `(maxlan + 1 - mev) * nrow)` elements and the rest is given to `misc`. If `trlan` can not use `wrk`, it will allocate workspace of appropriate size internally.

If `wrk` is provided, its content is not used on input. However before returning, `trlan` will copy the residual norms of the converged Ritz pairs in the first `nec` elements of `wrk`.

7.2 Variations of TRLAN

We have isolated the communication needs of TRLAN in four subroutines, `trl_init_info`, `trl_sync_flag`, `trl_g_sum`, and `trl_g_dot`. The four subroutines are located in a file called '`trl_comm_mpi.f90`' for the MPI version and '`trl_comm_none.f90`' for sequential version. If the the matrix-vector multiplication routine and the main program are written for sequential machine, the user can simply compile with '`trl_comm_none.f90`' to get the sequential version of the program. This setup makes it easy to adopt TRLAN for different types of eigenvalue problems.

The function `trl_init_info` is used to initialize the `TRL_INFO_T` type variable to be used by `trlan`. The function `trl_sync_flag` is used to synchronize the status flags used inside TRLAN. In the current implementation, it computes the minimum value of `info%stat` on each processor and reset all `info%stat` to the minimum value. Given that the error flags are all less than zero (0), if any processor has detected an error, all of them will be set to indicate an error. `Trl_g_sum` computes the global sum of an input array and it returns the global sum in the same array. The subroutine `trl_g_dot` computes the dot-products among the Lanczos vectors.

If desired, one can change these four routines to suit different situations. For example, if the physical domain of the eigenvalue problem has certain symmetry, usually the discretization does not contain the whole domain but only a portion of it. Since not every element of a vector is stored, the dot-product routine needs to be modified. In this case, only `trl_g_dot` and `trl_g_sum` need to be modified in order for TRLAN for function properly.

7.3 Calling from other languages

TRLAN program is implemented in `Fortran 90`. Since `Fortran 90` is backward compatible with previous versions of `Fortran`. There should be no problem to use it in any other `Fortran` program. However, at the moment, the authors are not aware of a scheme to reliably access `Fortran 90` subroutine with optional arguments, a subroutine with fixed arguments is created to get around this problem. The fixed arguments subroutine has the following interface,

```
       subroutine trlan77(op, ipar, nrow, mev, eval, evec, lde,
     &                    wrk, lwrk)
       integer ipar(32), nrow, mev, lde, lwrk
       double precision eval(mev), evec(lde, mev), wrk(lwrk)
       external op
```

The `Fortran 90` derived type `TRL_INFO_T` variable is removed from this user interface since its primary access function `trl_init_info` contains optional arguments as well. Here is a list showing how the integer array `ipar` is mapped to the elements of `TRL_INFO_T`,

- `ipar(1) = stat`,
- `ipar(2) = lohi`,
- `ipar(3) = ned`,
- `ipar(4) = nec`,
- `ipar(5) = maxlan`,
- `ipar(6) = restart`,
- `ipar(7) = maxmv`,
- `ipar(8) = mpicom`,
- `ipar(9) = verbose`,
- `ipar(10) = log_io`,
- `ipar(11) = iguess`,
- `ipar(12) = cpflag`,
- `ipar(13) = cpio`,
- `ipar(14) = mvop`,
- `ipar(24) = locked`,
- `ipar(25) = matvec`,
- `ipar(26) = nloop`,
- `ipar(27) = north`,
- `ipar(28) = nrand`,
- `ipar(29) = total time in milliseconds`,
- `ipar(30) = MATVEC time in milliseconds`.
- `ipar(31) = re-orthogonalization time in milliseconds`.
- `ipar(32) = restarting time in milliseconds`.

Among the parameters, `ipar(2 : 14)` are input parameters, `ipar(1)`, `ipar(4)` and `ipar(24 : 32)` are output parameters.

There are two floating-point number elements in `TRL_INFO_T`, `tol` and `crat`. Before calling `trlan77`, the first element of `wrk` should be set to the residual tolerance `tol`. Inside `trlan77`, `wrk(1)` is transfered to `tol`. On return from `trlan77`, the first `ipar(4)` elements of `wrk` store the residual norms corresponding to the converged eigenvalues and eigenvectors. Element `ipar(4)+1` of `wrk` will store the last known value of `crat`. **Caution**: Fail to set `wrk(1)` to a valid floating-point number will cause TRLAN to produce floating-point exceptions!

The subroutine `trlan77` is relative simple to call from `C/C++`. The file '`simplec.c`' in the distribution of TRLAN version 1.0 has equivalent functionalities as '`simple.f90`' and '`simple.f77`'.

7.4 Debugging

Here are a few suggestions on what to watch out for when using TRLAN. Some of suggestions are simply good programming practices.

- Use `implicit none` in `Fortran` programs. This is very effective in catching typos. It is also a good practice to check the programs with automated tools such as `lint` and `ftnchek`.
- Make sure the arrays passed to TRLAN have correct dimensions and make sure the argument `lwrk` is actually the size of array `wrk`.
- Make sure all input variables to TRLAN are initialized correctly and arguments to `trlan` matches arguments to `trl_init_info`.
- When encountering problems, turn on debugging options in TRLAN by calling `trl_set_debug` prior to calling `trlan`. Set the ninth and tenth element of array `ipar` to appropriate values when `trlan77` is used. Consult Section 4.2 [debug parameters], page 8 to setup a debugging session. Refer to Section 4.7 [error code], page 14 for error encountered and possible solutions.
- If all above have been done and there is still a problem, contact the author at the address given in Section 7.5 [contacting authors], page 28.

7.5 Contacting the authors

The authors of TRLAN and this document can be contacted at the following email addresses: `kwu@lbl.gov` (Kesheng Wu), `hdsimon@lbl.gov` (Horst Simon). Kesheng Wu can also be reached at `kwu@ieee.org` and `kwu@computer.org`. The authors also maintain their own research pages on the web at *http://www.nersc.gov/~kewu* and *http://www.nersc.gov/research/SIMON*. The updated software package can also be found at both web addresses.

8 References

This is a list books and research papers on the Lanczos algorithm and restarting. See Section 5.2 [operator interface], page 21, for a list of software archives.

1. J. Baglama, D. Calvetti, and L. Reichel. Iterative methods for the computation of a few eigenvalues of a large symmetric matrix. *BIT*, 36:400–421, 1996.
2. F. Chatelin. *Eigenvalues of Matrices.* Wiley, 1993.
3. J. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Theory*, volume 3 of *Progress in Scientific Computing.* Birkhauser, Boston, 1985.
4. J. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Programs*, volume 4 of *Progress in Scientific Computing.* Birkhauser, Boston, 1985.
5. T. Ericsson and A. Ruhe. The spectral transformation Lanczos method for numerical solution of large sparse generalized symmetric eigenvalue problems. *Math. Comp.*, 35:1251–1268, 1980.
6. R. G. Grimes, J. G. Lewis, and H. D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM J. Matrix Anal. Appl.*, 15(1):228–272, 1994.
7. Beresford N. Parlett. *The symmetric eigenvalue problem.* SIAM, Philadelphia, PA, 1998.
8. A. Ruhe. Rational Krylov sequence methods for eigenvalue computation. *Lin. Alg. Appl.*, 58:391–405, 1984.
9. A. Ruhe. Rational Krylov algorithm for nonsymmetric eigenvalue problems II: matrix pairs. *Lin. Alg. Appl.*, 197/198:283–296, 1994.
10. A. Ruhe. The rational Krylov algorithm for nonsymmetric eigenvalue problems III: Complex shifts for real matrices. *BIT*, 34:165–176, 1994.
11. Yousef Saad. *Numerical Methods for Large Eigenvalue Problems.* Manchester University Press, 1993.
12. D. S. Sorensen. Implicit application of polynomial filters in a K-step Arnoldi method. *SIAM J. Matrix Anal. Appl.*, 13(1):357–385, 1992.
13. A. Stathopoulos, Y. Saad, and K. Wu. Dynamic thick restarting of the Davidson and the implicitly restarted Arnoldi methods. Technical Report UMSI 96/123, Minnesota Supercomputer Institute, University of Minnesota, 1996.
14. K. Wu and H. Simon. Thick-restart Lanczos method for symmetric eigenvalue problems. Technical Report 41412, Lawrence Berkeley National Laboratory, 1998.
15. K. Wu and H. Simon. Dynamic Retarting Schemes For Thick-Restart Lanczos Method. Technical Report xxxxx, Lawrence Berkeley National Laboratory, 1999.

Index