ForTrilinos Documentation

Release *unknown release*

Seth R. Johnson **Andrey Prokopenko**

Aug 17, 2023

Contents

CHAPTER 1

Getting started with ForTrilinos

1.1 Overview

ForTrilinos is an open-source software library providing object-oriented Fortran interfaces to Trilinos C++ packages.

The new implementation using SWIG code generation started in late 2016 and is currently under active development. As of version 2.0.0, the documented ForTrilinos module interfaces are stable and available for downstream consumption. See *Version compatibility* for details on Trilinos version compatibility.

1.2 ForTrilinos Development Team

ForTrilinos is developed and maintained by:

- Seth Johnson
- Andrey Prokopenko

Former developers include:

- Tim Fuller
- Karla Morris
- Damian Rouson

1.3 Questions, Bug Reporting, and Issue Tracking

Questions, bug reporting and issue tracking are provided through ForTrilinos GitHub repository. The recommended way to report any bug is by creating a new issue in the repository. You can also ask questions by creating a new issue with the question tag.

CHAPTER 2

Installation

ForTrilinos is built as an independent software library with the Trilinos software library as its only dependency. ForTrilinos can be installed through the Spack HPC package manager or independently from your local installation of Trilinos.

2.1 Version compatibility

ForTrilinos wrappers are tightly coupled to the Trilinos API, so upstream changes require new releases of ForTrilinos. Since the wrapper generation is dependent on SWIG-Fortran capabilities, changes there will also affect the ability to rebuild ForTrilinos wrappers. (Note that SWIG is always optional; the version here simply denotes the version used to generate the included wrappers.)

The version scheme is based on semantic versioning:

- Major version numbers with Trilinos and minor versions of SWIG-Fortran (since it's still not officially upstreamed) can result in major version number changes for ForTrilinos.
- New features in Trilinos, and new support by ForTrilinos, can result in minor version number changes. Features removed or deprecated by a minor version change in Trilinos may also result in a minor version change.
- Minor changes to the SWIG-Fortran implementation (which don't affect the interface in the .F90 files) result in a patch version.

Essentially, the versioning will be driven by what the Fortran-only users see in the committed version of the generated wrappers.

Table 1: Guaranteed version compatibility table for ForTrilinos.

ForTrilinos	Trilinos	SWIG
2.3.0	14.0	4.1.1+fortran
2.2.0	13.4	4.1.1+fortran
2.1.0	13.2	4.1.0-dev1+fortran
2.0.1	13.0:13.2	4.1.0-dev1+fortran
2.0.0	13.0:13.2	4.0.2+fortran
2.0.0-dev3	12.18.1	4.0.2+fortran
2.0.0-dev2	12.18.1	4.0.0+fortran+15e6ed59
2.0.0-dev1	12.17+8a82b322	4.0.0+fortran+15e6ed59
1.0	12.8.1	—

In *the version table above*, the `+fortran` suffix for SWIG indicates the SWIG-Fortran fork. `+sha` refers to a specific Git commit that comes after the given version.

Basically, the versioning will be driven by what the Fortran-only users see in the committed version of the generated wrappers.

The original implementation of the ForTrilinos was developed prior to 2012. That code is no longer developed and maintained, and is available using the `trilinos-release-12-8-1` tag in the ForTrilinos repository and the corresponding Trilinos 12.8.1 version.

2.2 E4S

As of this writing, ForTrilinos is distributed as part of the E4S Project and should be available as a pre-built binary on a variety of user and HPC systems.

2.3 Spack

To install ForTrilinos version `2.1.0` through an existing Spack installation (v0.19 or higher, or the `develop` branch):

```
$ spack install fortrilinos
```

2.4 Manual

To install manually, you can point to the target Trilinos installation with the `CMAKE_PREFIX_PATH` or `Trilinos_ROOT` environment variables, or with a `Trilinos_ROOT` CMake variable:

```
$ git clone https://github.com/trilinos/ForTrilinos && cd ForTrilinos
$ mkdir build && cd build
$ cmake -DTrilinos_ROOT=/opt/trilinos -DCMAKE_INSTALL_PREFIX=/opt/fortrilinos ..
$ make install
```

The build options are most easily viewed using the `ccmake` GUI. All ForTrilinos options are scoped with a `ForTrilinos_` prefix. Some options (e.g. `ForTrilinos_USE_MPI`) are derived from the Trilinos installation and cannot be changed.

2.5 Documentation

To build this documentation, (re)configure with `-D ForTrilinos_DOCS=ON` and run:

```
$ make doc
```

then open `$BUILD/doc/html/index.html`.

CHAPTER 3

Usage

To use ForTrilinos as part of your CMake app, simply ensure that CMake can find it (using the standard `CMAKE_PREFIX_PATH` or `ForTrilinos_ROOT` environment variables, or the `ForTrilinos_ROOT` CMake variable); and add

```
find_package(ForTrilinos)
```

An example application that uses ForTrilinos and MPI-provided Fortran bindings might look like:

```
cmake_minimum_required(VERSION 3.12)
project(ForTrilinosInstallTest VERSION 0.0.1 LANGUAGES Fortran)

find_package(ForTrilinos)

add_executable(downstream-app downstream-app.F90)
target_link_libraries(downstream-app ForTrilinos::ForTrilinos MPI::MPI_Fortran)
```

and the `downstream-app.F90` app will simply need

```
use forteuchos
use fortpetra
```

CHAPTER 4

Modules

ForTrilinos includes several Fortran modules (`forteuchos`, `forbelos`, `fortpetra`) that are thin layers built on top of Trilinos packages. The `fortrilinos_hl` package is a high-level set of wrappers that exposes linear, nonlinear, and eigenvalue solvers.

Different Trilinos packages are required for different levels of functionality:

Package	Module
Teuchos	`forteuchos`
Belos	`forbelos`
Tpetra	`fortpetra`
Anasazi	`fortrilinos_hl`
NOX	`fortrilinos_hl`
Stratimikos	`fortrilinos_hl`
Thyra	`fortrilinos_hl`
Amesos2	`fortrilinos_hl` (optional)
Ifpack2	`fortrilinos_hl` (optional)
MueLu	`fortrilinos_hl` (optional)

4.1 Fortpetra

Tpetra

- is a package within the Trilinos ecosystem;
- implements linear algebra objects, including sparse graphs, sparse matrices, and dense vectors;
- provides parallel data redistribution facilities for many other Trilinos' packages; and
- is "hybrid parallel," meaning that it uses at least two levels of parallelism.

Within Trilinos, Tpetra can be used mostly as a stand-alone package, with explicit dependencies on Teuchos and Kokkos. There are adapters allowing the use of Tpetra operators and multivectors in both the Belos linear solver package and the Anasazi eigensolver package.

4.1.1 Hybrid Parallel

Hybrid parallel refers to the fact that Tpetra uses at least two levels of parallelism:

- MPI (the Message Passing Interface) for distributed-memory parallelism; and
- any of various shared-memory parallel programming models within an MPI process.

Tpetra uses the Kokkos package's shared-memory parallel programming model for data structures and computational kernels. Kokkos makes it easy to port Tpetra to new computer architectures and to extend its use of parallel computational kernels and thread-scalable data structures. Kokkos supports several different parallel programming models, including

- OpenMP
- POSIX Threads (Pthreads)
- Nvidia's CUDA programming model for graphics processing units (GPUs)

4.1.2 Features

Tpetra has the following unique features:

- Native support for representing and solving very large graphs, matrices, and vectors. "Very large" means over two billion unknowns or other entities.
- Matrices and vectors may contain many different kinds of data, such as floating-point types of different precision, complex-valued types, automatic differentiation objects from the Sacado package, or stochastic PDE discretization types from the Stokhos package.
- Support for many different shared-memory parallel programming models

4.1.3 Tpetra Support in ForTrilinos

Tpetra is a large library and its full functionality has not been exposed in ForTrilinos, though coverage is actively being expanded. The Tpetra *Examples* provide a good overview of the Tpetra functionality currently supported.

4.2 Examples

Note: The Tpetra examples are not intended to be lessons on the theory and use of Tpetra linear algebra objects. They are intended only to demonstrate implementation of operations commonly performed on Tpetra objects using the ForTrilinos interface. See the Tpetra documentation and lessons for more background on Tpetra theory and usage.

4.2.1 Maps And Vectors

Overview

This example demonstrates the following:

- how to create a `TpetraMap` object that describes the distribution of other Tpetra object entries over processes; and

- how to create the simplest kind of Tpetra linear algebra object: a Vector, whose entries are distributed over the process(es) in a communicator

This is a ForTrilinos implementation of the code example associated with Tpetra Lesson 02: Map and Vector, which should be consulted for background and theory.

Code example

The following example initializes Tpetra (MPI) then creates two distributed Tpetra Maps and some Tpetra Vectors, and does a few computations with the vectors

---

Note: In the code example below, it is assumed the the following ForTrilinos modules are `used`

- `forteuchos`: provides interfaces to the Trilinos' Teuchos package
- `fortpetra`: provides interfaces to the Trilinos' Tpetra package
- **`fortpetra_types`: provides named constants of type default integer for** Tpetra data types that can be used as the `kind` type parameters

---

```
! Copyright 2017-2018, UT-Battelle, LLC
!
! SPDX-License-Identifier: BSD-3-Clause
! License-Filename: LICENSE
program main

#include "ForTrilinos_config.h"
#include "ForTrilinos.h"

use iso_fortran_env
use, intrinsic :: iso_c_binding
use forerror, only : get_fortrilinos_version
use forteuchos
use fortpetra

#if FORTRILINOS_USE_MPI
use mpi
#endif

implicit none

! -- Parameters
real(scalar_type), parameter :: one=1.d0, fortytwo=42.d0
integer(size_type), parameter :: num_vecs=1

! -- ForTrilinos objects
type(TeuchosComm) :: comm
type(TpetraMap) :: contig_map, contig_map2, contig_map3, cyclic_map
type(TpetraMultiVector) :: x, y, z

! -- Scalars
integer :: ierr
integer :: my_rank, num_procs, k
integer :: num_local_entries, num_elements_per_proc
integer(global_size_type) :: num_global_entries
logical :: zero_out
```

(continues on next page)

(continued from previous page)

```
real(scalar_type) :: alpha, beta, gamma
real(mag_type) :: the_norm
real(mag_type), allocatable :: norms(:)
integer(global_ordinal_type), allocatable :: element_list(:)

! ------------------------------------------------------------------------ !


! Initialize MPI subsystem, if applicable
#if FORTRILINOS_USE_MPI
call MPI_INIT(ierr)
if (ierr /= 0) then
  stop "MPI failed to init"
endif
comm = TeuchosComm(MPI_COMM_WORLD)
#else
comm = TeuchosComm()
#endif

my_rank = comm%getRank()
num_procs = comm%getSize()

if (my_rank == 0) then
  write(error_unit, '(A,A)') "ForTrilinos version ", get_fortrilinos_version()
endif

! Tpetra objects are templated on several parameters.  ForTrilinos
! provides concrete specializations of Tpetra objects and exposes them to
! Fortran users.  The module fortpetra provides named constants of type
! default integer for Tpetra data types that can be used as the kind type
! parameters.

! Tpetra has local and global Maps.  Local maps describe objects that are
! replicated over all participating MPI processes.  Global maps describe
! distributed objects.  You can do imports and exports between local and global
! maps; this is how you would turn locally replicated objects into distributed
! objects and vice versa.
!
! num_local_entries: The local (on the calling MPI process) number of
! entries (indices) in the first Map that we create.  ForTpetra
! expects a size_type for this value.
num_local_entries = 5

! num_global_entries: The total (global, i.e., over all MPI processes) number of
! entries (indices) in the Map.  ForTpetra expects global_size_t for this value.
! This type is at least 64 bits long on 64-bit machines.
!
! For this example, we scale the global number of entries in the Map with the
! number of MPI processes.  That way, you can run this example with any number
! of MPI processes and every process will still have at least one entry
num_global_entries = num_procs * num_local_entries

! Create some Maps.  All Map constructors must be called as a collective over
! the input communicator.  Not all Map constructors necessarily require
! communication, but some do, so it's best to treat them all as collectives.
!
! Create a Map that puts the same number of equations on each processor.  The
```

(continues on next page)

(continued from previous page)

```
! resulting Map is "contiguous and uniform."
contig_map = TpetraMap(num_global_entries, comm)

! contig_map is contiguous by construction.  Test this at run time.
if (.not. contig_map%isContiguous()) then
  write(error_unit, '(A)') 'Expected contiguous map!'
  stop 1
end if

! contig_map2: Create a Map which is the same as contig_map, but uses
! a different Map constructor.  This one asks for the number of entries on each
! MPI process.  The resulting Map is "contiguous" but not necessarily uniform,
! since the numbers of entries on different MPI processes may differ.  In this
! case, the number of entries on each MPI process is the same, but that doesn't
! always have to be the case.
contig_map2 = TpetraMap(num_global_entries, num_local_entries, comm);

! Since contig_map and contig_map2 have the same communicators, and the same
! number of entries on all MPI processes in their communicators, they are "the
! same."
if (.not. contig_map%isSameAs(contig_map2)) then
  write(error_unit, '(A)') 'Expected conti_map and contig_map2 to be the same!'
  stop 1
end if

! contig_map3: Use the same Map constructor as contig_map3, but don't specify
! the global number of entries.  This is helpful if you only know how many
! entries each MPI process has, but don't know the global number.  Instead of
! num_global_entries, we use -1
contig_map3 = TpetraMap(TPETRA_GLOBAL_INVALID, num_local_entries, comm)

! Even though we made contig_map3 without specifying the global number of
! entries, it should still be the same as contig_map2.
if (.not. contig_map2%isSameAs(contig_map3)) then
  write(error_unit, '(A)') 'Expected conti_map2 and contig_map3 to be the same!'
  stop 1
end if

! Create a Map which has the same number of global entries per process as
! contig_map, but distributes them differently, in round-robin (1-D cyclic)
! fashion instead of contiguously.

! We'll use the version of the Map constructor that takes, on each MPI process,
! a list of the global entries in the Map belonging to that process.  You can
! use this constructor to construct an overlapping (also called "not 1-to-1")
! Map, in which one or more entries are owned by multiple processes.  We don't
! do that here; we make a nonoverlapping (also called "1-to-1") Map.
num_elements_per_proc = 5
allocate(element_list(num_elements_per_proc))
do k = 1, num_elements_per_proc
  element_list(k) = int(my_rank + k * num_procs, kind=global_ordinal_type)
end do
cyclic_map = TpetraMap(num_global_entries, element_list, comm)
deallocate(element_list)

! If there's more than one MPI process in the communicator, then cyclic_map is
! definitely NOT contiguous.
```

(continues on next page)

(continued from previous page)

```
if (num_procs > 1 .and.  cyclic_map%isContiguous()) then
  write(error_unit, '(A)') 'cyclic map should NOT be contiguous!'
  stop 1
end if

! contig_map and cyclic_map should always be compatible.  However, if the
! communicator contains more than 1 process, then contig_map and cyclic_map are
! NOT the same.
if (.not. contig_map%isCompatible(cyclic_map)) then
  write(error_unit, '(A)') 'Expected contig_map to be compatible with cyclic_map!'
  stop 1
end if

if (num_procs > 1 .and.  contig_map%isSameAs(cyclic_map)) then
  write(error_unit, '(A)') 'contig_map should not be same as cyclic_map!'
  stop 1
end if

! We have maps now, so we can create vectors.

! Create a Vector with the contiguous Map.  This version of the constructor will
! fill in the vector with zeros.
x = TpetraMultiVector(contig_map, num_vecs)

! The two-argument copy constructor with second argument Teuchos::Copy performs
! a deep copy.  x and y have the same Map.  The one-argument copy constructor
! does a _shallow_ copy.
y = TpetraMultiVector(x, TeuchosCopy)

! Create a Vector with the 1-D cyclic Map.  Calling the constructor
! with false for the second argument leaves the data uninitialized,
! so that you can fill it later without paying the cost of
! initially filling it with zeros.
zero_out = .false.
z = TpetraMultiVector(cyclic_map, num_vecs, zero_out)

! Set the entries of z to (pseudo)random numbers.  Please don't consider this
! a good parallel pseudorandom number generator.
call z%randomize()

! Set the entries of x to all ones.
call x%putScalar(one)

alpha = 3.14159;
beta = 2.71828;
gamma = -10.0;

! x = beta*x + alpha*z
!
! This is a legal operation!  Even though the Maps of x and z are not the same,
! their Maps are compatible.  Whether it makes sense or not depends on your
! application.
call x%update(alpha, z, beta)

call y%putScalar(fortytwo)

! y = gamma*y + alpha*x + beta*z
```

(continues on next page)

(continued from previous page)

```
call y%update(alpha, x, beta, z, gamma)
call y%update(alpha, x, beta)

! Compute the 2-norm of y.
allocate(norms(num_vecs))
call y%norm2(norms)

the_norm = norms(1)

! Print the norm of y on Proc 0.
if (my_rank == 0) then
    write(*, '(A,f8.3)') 'Norm of y: ', the_norm
end if

! Release objects created and no longer in use
call z%release()
call y%release()
call x%release()
call cyclic_map%release()
call contig_map3%release()
call contig_map2%release()
call contig_map%release()
call comm%release()
deallocate(norms)

#if FORTRILINOS_USE_MPI
! Finalize MPI must be called after releasing all handles
call MPI_FINALIZE(ierr)
#endif

end program main
```

4.2.2 Power Method for Finding the Largest Eigenvalue

Overview

This example demonstrates the following:

- how to construct a `TpetraCrsMatrix` (a distributed sparse matrix);
- how to modify the entries of a previously constructed `TpetraCrsMatrix`; and
- how to use `TpetraCrsMatrix` and `TpetraMultiVector` to implement a simple iterative eigensolver (the power method)

This example is a ForTrilinos implementation of the code example associated with Tpetra Lesson 03: Power Method, which should be consulted for background and theory.

Code example

The following code example shows how to fill and compute with a Tpetra sparse matrix, using the procedure discussed in the text above.

Note: In the code example below, it is assumed the the following ForTrilinos modules are `used`

- `forteuchos`: provides interfaces to the Trilinos' Teuchos package
- `fortpetra`: provides interfaces to the Trilinos' Tpetra package
- **`fortpetra_types`: provides named constants of type default integer for** Tpetra data types that can be used as the `kind` type parameters

```
! Copyright 2017-2018, UT-Battelle, LLC
!
! SPDX-License-Identifier: BSD-3-Clause
! License-Filename: LICENSE
program main
! ------------------------------------------------------------------------ !
! Power method for estimating the eigenvalue of maximum magnitude of
! a matrix.
!
! We don't intend for you to write your own eigensolvers; the Anasazi
! package provides them.  You should instead see this class as a surrogate
! for a ForTrilinos interface to the Tpetra package.
! ------------------------------------------------------------------------ !
#include "ForTrilinos_config.h"

use iso_fortran_env
use, intrinsic :: iso_c_binding

#include "ForTrilinos.h"
use forteuchos
use fortpetra

#if FORTRILINOS_USE_MPI
use mpi
#endif

implicit none

! -- ForTrilinos objects
type(TeuchosComm) :: comm
type(TpetraMap) :: map
type(TpetraCrsMatrix) :: A

! -- Scalars
integer :: ierr
real(scalar_type) :: lambda
integer(global_size_type) :: num_gbl_indices
integer :: my_rank
integer(size_type) :: num_entries_in_row, max_entries_per_row, i
integer :: lcl_row, row_nnz, n
integer :: num_my_elements, iconv
integer(global_ordinal_type) gbl_row, id_of_first_row

! -- Arrays
integer(global_ordinal_type), allocatable :: cols(:)
real(scalar_type), allocatable :: vals(:)
! ------------------------------------------------------------------------ !


! Initialize MPI subsystem, if applicable
#if FORTRILINOS_USE_MPI
```

(continues on next page)

(continued from previous page)

```
call MPI_INIT(ierr)
if (ierr /= 0) then
  stop "MPI failed to init"
endif
comm = TeuchosComm(MPI_COMM_WORLD)
#else
comm = TeuchosComm()
#endif

my_rank = comm%getRank()

! The number of rows and columns in the matrix.
num_gbl_indices = 50

map = TpetraMap(num_gbl_indices, comm)
FORTRILINOS_CHECK_IERR()

! Check that the map was created with the appropriate number of elements
if (map%getGlobalNumElements() /= num_gbl_indices) then
  write(error_unit, '(A,I3,A)') 'Expected ', num_gbl_indices, ' global indices'
  stop 1
end if


if (my_rank == 0) &
  write(*, *) "Creating the sparse matrix"

! Create a Tpetra sparse matrix whose rows have distribution given by the Map.

! TpetraOperator implements a function from one Tpetra(Multi)Vector to another
! Tpetra(Multi)Vector.  TpetraCrsMatrix implements TpetraOperator; its apply()
! method computes a sparse matrix-(multi)vector multiply.  It's typical for
! numerical algorithms that use Tpetra objects to be templated on the type of
! the TpetraOperator specialization.
max_entries_per_row = 3
A = TpetraCrsMatrix(map, max_entries_per_row)

! Fill the sparse matrix, one row at a time.
allocate(vals(3))
allocate(cols(3))
num_my_elements = int(map%getLocalNumElements(), kind=kind(num_my_elements))
fill: do lcl_row = 1, num_my_elements
  gbl_row = map%getGlobalElement(lcl_row)
  if (gbl_row == 1) then
    ! A(1, 1:2) = [2, -1]
    row_nnz = 2
    cols(1:2) = [gbl_row, gbl_row+1]
    vals(1:2) = [2d0, -1d0]
  else if (gbl_row == num_gbl_indices) then
    ! A(N, N-1:N) = [-1, 2]
    row_nnz = 2
    cols(1:2) = [gbl_row-1, gbl_row]
    vals(1:2) = [-1d0, 2d0]
  else
    ! A(i, i-1:i+1) = [-1, 2, -1]
    row_nnz = 3
    cols(1:3) = [gbl_row-1, gbl_row, gbl_row+1]
```

(continues on next page)

(continued from previous page)

```
    vals(1:3) = [-1d0, 2d0, -1d0]
  end if
  call A%insertGlobalValues(gbl_row, cols(1:row_nnz), vals(1:row_nnz))
end do fill
deallocate(vals)
deallocate(cols)

! Tell the sparse matrix that we are done adding entries to it.
call A%fillComplete()

! Run the power method and report the result.
call PowerMethod(A, lambda, iconv, comm)
if (iconv > 0 .and. my_rank == 0) then
  write(*, *) "Estimated max eigenvalue: ", lambda
end if
if (abs(lambda-3.99)/3.99 >= .005) then
  write(error_unit, '(A)') 'Largest estimated eigenvalue is not correct!'
  stop 1
end if

! Now we're going to change values in the sparse matrix and run the power method
! again.  We'll increase the value of the (1,1) component of the matrix to make
! the matrix more diagonally dominant.  It should decrease the number of
! iterations required for the power method to converge. In Tpetra, if
! fillComplete() has been called, you have to call resumeFill() before you may
! change the matrix(either its values or its structure).

! Increase diagonal dominance
if (my_rank == 0) &
  write(*, *) "Increasing magnitude of A(1,1), solving again"

! Must call resumeFill() before changing the matrix, even its values.
call A%resumeFill()

id_of_first_row = 1
if (map%isNodeGlobalElement(id_of_first_row)) then
  ! Get a copy of the row with with global index 1.  Modify the diagonal entry
  ! of that row.  Submit the modified values to the matrix.
  num_entries_in_row = A%getNumEntriesInGlobalRow(id_of_first_row);
  n = int(num_entries_in_row, kind=kind(n))
  allocate(vals(n))
  allocate(cols(n))

  ! Fill vals and cols with the values resp.(global) column indices of the
  ! sparse matrix entries owned by the calling process.
  !
  ! Note that it's legal(though we don't exercise it in this example) for the
  ! row Map of the sparse matrix not to be one to one.  This means that more
  ! than one process might own entries in the first row.  In general, multiple
  ! processes might own the (1,1) entry, so that the global A(1,1) value is
  ! really the sum of all processes' values for that entry.  However, scaling
  ! the entry by a constant factor distributes across that sum, so it's OK to do
  ! so.
  call A%getGlobalRowCopy(id_of_first_row, cols, vals, num_entries_in_row)
  do i = 1, n
    if (cols(i) == id_of_first_row) then
      vals(i) = vals(i) * 10.
```

(continues on next page)

(continued from previous page)

```
    end if
  end do

  ! "Replace global values" means modify the values, but not the structure of
  ! the sparse matrix.  If the specified columns aren't already populated in
  ! this row on this process, then this method throws an exception.  If you want
  ! to modify the structure(by adding new entries), you'll need to call
  ! insertGlobalValues().
  i = A%replaceGlobalValues(id_of_first_row, cols, vals)

  deallocate(vals)
  deallocate(cols)

end if

! Call fillComplete() again to signal that we are done changing the
! matrix.
call A%fillComplete()

! Run the power method again.
call PowerMethod(A, lambda, iconv, comm)
if (iconv > 0 .and. my_rank == 0) then
  write(*, *) "Estimated max eigenvalue: ", lambda
end if
if (abs(lambda-20.05555)/20.05555 >= .0001) then
  write(error_unit, '(A)') 'Largest estimated eigenvalue is not correct!'
  stop 1
end if

call A%release()
call map%release()
call comm%release()

#if FORTRILINOS_USE_MPI
! Finalize MPI must be called after releasing all handles
call MPI_FINALIZE(ierr)
#endif


contains

  subroutine PowerMethod(A, lambda, iconv, comm, verbose_in)
    ! ------------------------------------------------------------------------ !
    ! Power method for estimating the eigenvalue of maximum magnitude of
    ! a matrix.  This function returns the eigenvalue estimate.
    ! ------------------------------------------------------------------------ !
    use fortpetra
    implicit None
    integer(int_type), intent(out) :: iconv
    real(scalar_type), intent(out) :: lambda
    logical, intent(in), optional :: verbose_in
    type(TpetraCrsMatrix), intent(in) :: A
    type(TeuchosComm), intent(in) :: comm
    ! ------------------------------------------------------------------------ !
    ! Arguments
    ! ---------
    ! A          (input) The sparse matrix
```

(continues on next page)

(continued from previous page)

```
    !
    ! lambda     (output) The estimated eigenvalue of maximum magnitude
    !
    ! iconv      (output) Flag indicated whether or not the procedure
    !    converged:
    !    iconv = 0  did not converge
    !          = 1  converged based on tol1 (more stringent)
    !          = 2  converged based on tol2 (less stringent)
    !
    ! verbose_in (input, optional) Print information diagnositics
    ! ------------------------------------------------------------------------ !
    ! -- Local Parameters
    real(scalar_type), parameter :: one=1., zero=0.
    integer(size_type), parameter :: num_vecs=1
    integer(int_type), parameter :: maxit1=500, maxit2=750
    real(scalar_type), parameter :: tol1=1.e-5, tol2=1.e-2

    ! -- Local Scalars
    real(scalar_type) :: normz, residual
    integer(int_type) :: report_frequency, it
    integer(size_type) :: my_rank
    logical :: verbose

    ! -- Local Arrays
    real(scalar_type) :: norms(num_vecs), dots(num_vecs)

    ! -- Local ForTrilinos Objects
    type(TpetraMultiVector) :: q, z, resid
    type(TpetraMap) :: domain_map, range_map
    ! ------------------------------------------------------------------------ !

    verbose = .false.
    if (present(verbose_in)) verbose = verbose_in
    my_rank = comm%getRank()

    ! Set output only arguments
    lambda = 0.
    iconv = 0

    ! Create three vectors for iterating the power method.  Since the power
    ! method computes z = A*q, q should be in the domain of A and z should be in
    ! the range. (Obviously the power method requires that the domain and the
    ! range are equal, but it's a good idea to get into the habit of thinking
    ! whether a particular vector "belongs" in the domain or range of the
    ! matrix.)  The residual vector "resid" is of course in the range of A.
    domain_map = A%getDomainMap()
    range_map = A%getRangeMap()
    q = TpetraMultiVector(domain_map, num_vecs)
    z = TpetraMultiVector(range_map, num_vecs)
    resid = TpetraMultiVector(range_map, num_vecs)

    ! Fill the iteration vector z with random numbers to start.  Don't have
    ! grand expectations about the quality of our pseudorandom number generator,
    ! but it is usually good enough for eigensolvers.
    call z%randomize()

    ! lambda: Current approximation of the eigenvalue of maximum magnitude.
```

(continues on next page)

(continued from previous page)

```
    ! normz: 2-norm of the current iteration vector z.
    ! residual: 2-norm of the current residual vector 'resid'.
    lambda = zero
    normz = zero
    residual = zero

    ! How often to report progress in the power method.  Reporting progress
    ! requires computing a residual, which can be expensive.  However, if you
    ! don't compute the residual often enough, you might keep iterating even
    ! after you've converged.
    report_frequency = 10

    ! Do the power method, until the method has converged or the
    ! maximum iteration count has been reached.
    iconv = 0
    iters: do it = 1, maxit2
      call z%norm2(norms)
      normz = norms(1)
      call q%scale(one / normz, z)  !  q := z / normz
      call A%apply(q, z)  !  z := A * q
      call q%dot(z, dots)
      lambda = dots(1)  !  Approx. max eigenvalue

      ! Compute and report the residual norm every report_frequency
      ! iterations, or if we've reached the maximum iteration count.
      if (mod(it-1, report_frequency) == 0 .or. it == maxit2) then
        call resid%update(one, z, -lambda, q, zero)  ! z := A*q - lambda*q
        call resid%norm2(norms)
        residual = norms(1)  ! 2-norm of the residual vector
        if (verbose .and. my_rank == 0) then
          write(*, '(A,I3,A)') "Iteration ", it, ":"
          write(*, '(A,F8.4)') "- lambda = ", lambda
          write(*, '(A,E8.2)') "- ||A*q - lambda*q||_2 = ", residual
        end if
      end if

      if (it <= maxit1) then
        if(residual < tol1) then
          iconv = 1
          exit iters
        end if
      else
        if (residual < tol2) then
          iconv = 2
          exit iters
        end if
      end if

    end do iters

    if (iconv == 0 .and. my_rank == 0) then
      write(*, *) "PowerMethod failed to converge after ", maxit2, " iterations"
    end if

    call q%release()
    call z%release()
    call resid%release()
```

(continues on next page)

(continued from previous page)

```
    return

  end subroutine PowerMethod

end program main
```