



**User Guide for the interface between ParaFEM and PETSc**

|  |  |
| --- | --- |
| Project Title | **eCSE06-ParaFEM** |
| Authors | **Mark Filipiak** |
| Document Version | **0.1 21/02/2017** |

**Contents**

[12 2](#_Toc475432425)

[12.5 PETSc 2](#_Toc475432426)

[12.5.1 Installing PETSc 2](#_Toc475432427)

[12.5.2 Building ParaFEM with PETSc 2](#_Toc475432428)

[12.5.3 Using the ParaFEM-PETSc interface 3](#_Toc475432429)

[12.5.4 Using PETSc directly in ParaFEM 9](#_Toc475432430)

[Program xx18 Three-dimensional analysis of an elastic solid. Compare Program 12.1 9](#_Toc475432431)

[Program xx17 Three-dimensional steady-state Navier-Stokes analysis. Compare Program 12.6 11](#_Toc475432432)

[Program xx15 Three-dimensional nonlinear analysis of an elastoplastic solid. Experimental 12](#_Toc475432433)

[12.7 Glossary of Variable Names 13](#_Toc475432434)

[Scalar integers: 13](#_Toc475432435)

[Scalar reals: 13](#_Toc475432436)

[Scalar characters: 13](#_Toc475432437)

[Scalar logicals: 13](#_Toc475432438)

[Dynamic integer arrays: 14](#_Toc475432439)

[Dynamic real arrays: 14](#_Toc475432440)

[Appendix G 14](#_Toc475432441)

[Subroutine descriptions 14](#_Toc475432442)

This User Guide is written as extra sections to be added to Chapter 12 of ‘Programming the Finite Element Method’ by I.M. Smith, D.V. Griffiths and L. Margetts, 5th edition, 2014 (referred to as PFEM below), which is the complete user guide to ParaFEM. It is expected that the reader will be already familiar with ParaFEM and has already read at least Chapters 1, 2, 3 and 12 of PFEM.

# 

(The Conclusions section is shifted to 12.6)

## PETSc

ParaFEM provides two parallel iterative solvers for the stiffness matrix equation: diagonally preconditioned conjugate gradient (see Program 12.1) and BiCGStab(*l*) (see Program 12.6). These have been made into subroutines pcg\_ver1 and bicgstabl\_p to give a general interface to these solvers, usable in any parallel driver program. These solvers use an unassembled stiffness matrix and perform the matrix-vector multiplication at each iteration using gather and scatter to convert vectors to and from assembled form.

To provide a wider range of solvers and preconditioners, an interface to the PETSc library (<https://www.mcs.anl.gov/petsc>) is available. PETSc provides the same solvers and preconditioners provided by ParaFEM (useful for comparison) and many more: iterative solvers MINRES (useful for indefinite symmetric problems), GMRES (general purpose iterative solver for all types of problem, but needs tuning of restart), and others; preconditioners Jacobi (i.e. diagonal), block Jacobi, ILU, AMG (not all of these are better than simple Jacobi preconditioning). PETSc can be built to use other libraries to extend the range of solvers and preconditioners (and these libraries can be provided with PETSc if they are not already installed), for example the parallel direct solvers MUMPS and SuperLU\_DIST.

### Installing PETSc

If you are lucky, PETSc is already installed on your computer. Otherwise you will have to install PETSc. This isn’t difficult since all the required libraries are usually already installed (e.g., BLAS and LAPACK, also used by ParaFEM) or will be downloaded automatically in the PETSc installation process. Detailed installation instructions are given on the PETSc website. There are some configuration options that need to be set to use PETSc in ParaFEM:

* The PETSc Fortran interface must be included.
* For very large problems use 64-bit indices (and ParaFEM INTEGER variables will need to be made 64-bit).
* For production runs switch off debugging and choose the highest optimisation levels for the optimisation flags in the configuration step.

### Building ParaFEM with PETSc

(Tested on a Cray XC30 with Linux and the Cray Fortran and C compilers.)

The version of ParaFEM with the PETSc interface is in branch parafem\_petsc. The stable revision is 2228. Checkout the branch with

svn checkout svn://svn.code.sf.net/p/parafem/code/branches/petsc/parafem

and then use make-parafem as normal but with the new option -parafem\_petsc to build the ParaFEM-PETSC interface and –xx to build the driver programs (only the experimental programs xx15, xx17 and xx18 use PETSc). The build/xc30.inc file gives the settings for building on a Cray XC30. These can be modified for other systems (see the other .inc files in the build directory) but the location of the PETSc Fortran modules and library will need to be set, for example

PETSC\_INCLUDE=-I/work/petsc/include  
PETSC\_LDFLAGS=-L/work/petsc/lib –lpetsc

For ARCHER (<http://www.archer.ac.uk>), a wrapper script (build.bash) has been written to assist in building the programs.

### Using the ParaFEM-PETSc interface

The PETSc library can be used directly in a ParaFEM program (see Section 12.5.4) but the structure of most ParaFEM programs makes it possible to have a simple interface to the linear solvers and preconditioners. This interface is provided by the parafem\_petsc module.

The driver programs for linear problems have a sequence like

Create stiffness matrix K and vectors x, f  
Set the entries in K

Solve Ku = f

For non-linear problems the sequence is

Create stiffness matrix K and vectors x, f

DO Newton-Raphson iteration

Set the entries in K

Solve Ku = f

END DO

and for time-stepping (or load-stepping) problems the sequence is

Create stiffness matrix K and vectors x, f

DO time/load step

DO Newton-Raphson iteration

Set the entries in K

Solve Ku = f

END DO

END DO

In all cases there only one matrix, one solution vector and one load vector. The matrix structure is fixed (the grid connectivity is unchanged) but the matrix entries may change in non-linear problems.

When using the ParaFEM solvers (pcg\_ver1 or bicgstabl\_p) the stiffness matrix is stored in unassembled form – each process has an array storkm\_pp(:,:,nels\_pp) (or storekm\_pp or storke\_pp) holding all the element matrices for the elements on that process. The load and solution vectors are converted between the global form and the per-element form (used for the matrix-vector product) with the ParaFEM subroutines gather and scatter.

When using the PETSc solvers the stiffness matrix is stored in assembled form as a PETSc Mat object – each process holds the rows of the global stiffness matrix corresponding to the equations on that process. The solution and load vectors are stored as PETSc Vec objects and PETSc handles the matrix-vector product internally. The ParaFEM-PETSc interface stores the matrix and vectors together with some auxiliary data in a global object and the driver program accesses all of the PETSc objects through the interface subroutines only.

An example sequence when using PETSc is:

Initialise PETSc (p\_initialize)  
Create PETSc matrix K and vectors x,f (p\_create)

DO time/load step

DO Newton-Raphson iteration  
 DO element

Create element stiffness matrix km  
 Add km to global matrix K (p\_add\_element)  
 END DO  
 Complete the setup of K (p\_assemble)

Solve Ku = f (p\_solve)

END DO

END DO

There is an extra ‘assemble’ step when creating the stiffness matrix. The p\_add\_element calls set up entries in the local process and but store entries for other processes locally – there is no communication between processes. The p\_assemble subroutine distributes these entries to the correct processes.

#### Choosing the solver to use

When using the ParaFEM solvers, there are two solvers to choose from and this is done by calling the corresponding subroutine (pcg\_ver1 or bicgstabl\_p) and the solver parameters are set in the input file <filename base>.dat.

When using the PETSc solvers, there is a large range of solvers and preconditioners to choose from. There is one subroutine (p\_solve) and the solver, preconditioner and solver parameters are set in the configuration file (<filename base>.petsc) which is read when PETSc is initialized with p\_initialize. The solver parameters set in the .petsc file are used instead of those in the .dat file:

* solver tolerance
* maximum number of solver iterations
* for BiCGStab(*l*), the value of *l*

The .petsc file is in the standard PETSc options file format. This means that all the PETSc documentation (<http://www.mcs.anl.gov/petsc/documentation>) can be used and makes all of the PETSc options available (only some will be useful when using the ParaFEM-PETSc interface, others will be useful if using PETSc directly).

The solver, solver tolerance, maximum number of iterations, and preconditioner are **required** in the .petsc file, so the simplest example is

# Comments start with a # and blank lines are ignored

-ksp\_type cg # use the conjugate gradient solver

-ksp\_rtol 1.0e-5 # the relative error tolerance of the   
 # preconditioned residual

-ksp\_max\_it 4000 # the maximum number of solver iterations

-pc\_type jacobi # use Jacobi (diagonal) preconditioning

This .petsc file sets up one solver (“ksp” stands for “Krylov solver”). Information about the solver and preconditioner can be printed by adding the options

-ksp\_view  
-pc\_view

The PETSc relative error tolerance is for the preconditioned residual, while the ParaFEM relative error tolerance is for the unpreconditioned (‘true’) residual. With –pc\_type none (no PETSc preconditioner) these will be the same. With –pc\_type Jacobi you may need to reduce the PETSc tolerance by a factor 2‒5 to achieve the same true residual. With other preconditioners some experimentation is needed to choose a preconditioned tolerance that will achieve a specified unpreconditioned tolerance. Calling p\_print\_info after p\_solve will print information about the true residual.

When solving a non-linear problem, the matrix will depend on the solution and the matrix type may change during the iterations of the non-linear solver, requiring a different type of linear solver. For example, in solid mechanics with elasto-plastic softening or buckling, the matrix will be symmetric positive definite during the elastic stages and become indefinite and possibly unsymmetric during the softening/buckling stages. It can be efficient to use a conjugate gradient solver for the elastic stages until softening is detected and then switch to use (for example) MINRES if the matrix becomes indefinite or BiCGStab(*l*) if the matrix becomes unsymmetric.

When using the ParaFEM solvers, the choice is between conjugate gradient and BiGCStab(*l*), and in the non-linear loop there would be some test for elastic or plastic stage and then a choice between the two solvers

IF (elastic) THEN  
 CALL pcg\_ver1(...)  
ELSE IF (plastic) THEN  
 CALL bicgstabl\_p(...)  
END IF

When using the PETSc solvers, switching between solvers is done by specifying the possible solvers in the .petsc file and choosing between them by calling p\_use\_solver before p\_solve. For example

-nsolvers 2

-prefix\_push solver\_1\_ # call p\_use\_solver(1,...)

-ksp\_type cg

-ksp\_rtol 1.0e-5

-ksp\_max\_it 2000

-pc\_type jacobi

-prefix\_pop

-prefix\_push solver\_2\_ # call p\_use\_solver(2,...)

-ksp\_type minres

-ksp\_rtol 1.0e-5

-ksp\_max\_it 2000

-pc\_type jacobi  
 -pc\_jacobi\_abs # MINRES needs a positive definite  
 # preconditioner  
-prefix\_pop

with the choice in the non-linear loop written as

IF (elastic) THEN  
 CALL p\_use\_solver(1,...)  
ELSE IF (plastic) THEN  
 CALL p\_use\_solver(2,...)  
END IF

CALL p\_solve(...)

Any number of solvers can be set up in the .petsc file: solver\_3\_ corresponds to p\_use\_solver(3,...), etc.

#### Choosing between ParaFEM and PETSc solvers

When testing the performance (and correctness) of different solvers, it is useful to be able to compare ParaFEM and PETSc solvers. The get\_solvers subroutine can be used to read in a second command line argument which specifies the solvers to use, either ‘parafem’ or ‘petsc’.

#### PETSc error and signal handling

If PETSc is used in a ParaFEM program (i.e., p\_initialize is called) PETSc handles **all** errors (e.g., running out of memory to store the stiffness matrix) and signals (e.g., from accessing elements beyond the end of an array) even if they occur outside PETSc subroutines. If errors occur, a large amount of information is printed by each process.

#### Advice on solvers and preconditioners to use

The type of solver to use depends on the matrix type, which depends on the type of problem to be solved:

|  |  |
| --- | --- |
| **Matrix type** | Symmetric positive-definite (SPD) |
| **Problem type** | Elastic solids, elasto-plastic solids, Laplacian flow, heat conduction |
| **Solvers** | Conjugate gradient: ParaFEM pcg\_ver1, PETSc –ksp\_type cg |
| Conjugate gradient is the standard solver to use for SPD matrices. MINRES is an alternative and has similar convergence properties. | |

|  |  |
| --- | --- |
| **Matrix type** | Symmetric indefinite |
| **Problem type** | Elasto-plastic solids after buckling |
| **Solvers** | MINRES: PETSc –ksp\_type minres |
| MINRES needs an SPD preconditioner: if using Jacobi preconditioning then the absolute values of the diagonal should be used (PETSc -pc\_jacobi\_abs). | |

|  |  |
| --- | --- |
| **Matrix type** | Non-symmetric |
| **Problem type** | Incompressible flow, Biot poro-elastic solid |
| **Solvers** | BiCGStab(*l*): ParaFEM bicgstabl\_p, PETSc –ksp\_type bcgsl  GMRES(r): PETSC –ksp\_type gmres |
| BiCGStab(*l*) is generally recommended but can sometimes give very slow convergence. Using a larger value for *l* may improve the convergence – a typical value for *l* is 4) or GMRES(r) (restarted GMRES) can be tried instead. With GMRES(r), small values for r may give slow convergence or stagnation (no convergence) but large values of r increase the time and memory requirements. There is no rule-of-thumb for choosing a good value for r, some experimentation is needed. The option -ksp\_monitor\_true\_residual in the .petsc file will print the preconditioned and true residual at each iteration – this can be used test the effect of different values of r. | |

|  |  |
| --- | --- |
| **Matrix type** | Non-symmetric |
| **Problem type** |  |
| **Solvers** | Sparse direct: PETSc –ksp\_type preonly with –pc\_type lu; choose the solver with -pc\_factor\_mat\_solver\_package superlu\_dist or -pc\_factor\_mat\_solver\_package mumps |
| Serial sparse direct solvers are available in ParaFEM (see earlier chapters) and PETSc. If PETSc has be installed with a direct solver package (SuperLU\_DIST or MUMPS) then PETSc will provide parallel sparse direct solvers as well. Direct solvers can be faster than iterative solvers for smaller problems (less than 2.5 M degrees of freedom). The memory required for the factorised matrix scales as N4/3and the operation count as N2 and direct solvers do not scale beyond about 100 processes, all of which mean that large problems are more efficiently solved by iterative solvers. | |

Preconditioners are usually applicable to any iterative solver, so these are listed by preconditioner type:

|  |  |
| --- | --- |
| **Preconditioner** | Jacobi (diagonal): ParaFEM in the driver programs that use pcg\_ver1, PETSc –pc\_type jacobi |
| For most problems, the Jacobi preconditioner is the simplest preconditioner to use and will often be sufficient to get adequate convergence. | |

|  |  |
| --- | --- |
| **Preconditioner** | None: PETSc in the driver programs that use bicgstabl\_p, PETSc –pc\_type none |
| Incompressible Navier-Stokes leads to a saddle-point system (a block matrix with one diagonal block all zero) and the Jacobi preconditioner is not suitable. For the driver programs 12.6 and xx18, using no preconditioning gives adequate convergence. | |

|  |  |
| --- | --- |
| **Preconditioners** | Various: block Jacobi, incomplete factorisations (ILU, ILU(t), ICC) |
| PETSc provides a range of other preconditioners which can be experimented with. These preconditioners reduce the number of iterations to convergence but the time for each iteration increases so that, overall, the solution time may not reduce.  (http://www.mcs.anl.gov/petsc/documentation/linearsolvertable.html) | |

|  |  |
| --- | --- |
| **Preconditioner** | Algebraic multigrid |
| For many problems, especially those leading to SPD matrices, multigrid provides the best preconditioner in terms of reduction of iterations. Each iteration takes more time but, overall, the solution time reduces. PETSc provides an algebraic multigrid preconditioner (-pc\_type gamg), suitable for problems on unstructured grids such as those used in ParaFEM. The multigrid method used is smoothed aggregation, which needs all the degrees of freedom at each node to be kept. ParaFEM treats restrained dofs by removing the equations for these from the problem, leading to a smaller matrix, which means that PETSc’s algebraic multigrid preconditioner is only suitable for scalar problems, for example heat flow (Programs 12.4 and 12.5). | |

#### Libraries

The subroutines for the PETSc solvers are in the parafem\_petsc library and get\_solvers is in the choose\_solvers library.

#### Global Variables

|  |  |
| --- | --- |
| p\_object | used only in the parafem\_petsc module but available if PETSc is used directly in a driver program. Contains the PETSc data structure for the stiffness matrix (a PETSc Mat), the solution and load vectors (Vec), the solver and preconditioner (Ksp) and several auxiliary variables (workspace, information about convergence). |
| choose\_solvers\_string\_length | Size of string returned by get\_solvers |
| parafem\_solvers | Command line argument for ParaFEM solvers (‘parafem’) |
| petsc\_solvers | Command line argument for PETSc solvers (‘petsc’) |

### Using PETSc directly in ParaFEM

It is recommended that the parafem\_petsc library is used to access the PETSc linear solvers but PETSc can be used directly. The subroutines in the parafem\_petsc library can be used to interface with ParaFEM and then the PETSc objects in p\_object accessed with PETSc subroutine calls. See the comments at the start of the xx18 driver program for information on using PETSc Fortran interfaces with the parafem\_petsc library.

PETSc provides non-linear solvers (e.g., Newton-Raphson) and also load-stepping (or time-stepping). These could be used to replace the methods used in the driver programs. Currently there is no interface in parafem\_petsc for these.

### Program xx18 Three-dimensional analysis of an elastic solid. Compare Program 12.1



This is based on Program 12.1 but with a choice between a ParaFEM solver (preconditioned conjugate gradient, in the subroutine pcg\_ver1) and PETSc solvers. After the input data is read, the second command line argument is read to choose between ParaFEM and PETSc solvers (get\_solvers).

The p12meshgen program is used as before to generate the input files from xx18.mg. An example PETSc configuration is given in xx18.petsc and an example job submission script for ARCHER is given in xx18.pbs.

There are IF/THEN/ELSE/ENDIF blocks throughout the program choosing between ParaFEM-specific and PETSc-specific code. ParaFEM uses the storkm\_pp array for the stiffness matrix; the corresponding data structures for PETSc are set up when p\_create is called. Assembly of the matrix is done by assembling each element matrix in turn and adding it to the global matrix. For ParaFEM, this is by copying the element matrix to the corresponding subarray of storkm\_pp; for PETSc is by calling the subroutine p\_add\_element (to add the element matrix to the internal data structure set up by p\_create) and finalising the assembly after all the elements have been added (p\_assemble).

The preconditioner for the ParaFEM solver is set up in the program; the preconditioner for the PETSc solver is specified in the PETSc configuration file and set up in the call to p\_solve. The code to call the PETSc solver calls the p\_use\_solver subroutine: in this case there is only one solver, so this could be omitted (solver 1 will be used in p\_solve).

Figure 1 compares the ParaFEM and PETSc conjugate gradient solvers, both using Jacobi (diagonal) preconditioning on a Cray XC30 (ARCHER), for the Program 12.1 test case with 1003 elements. For this case, PETSc is about 30% faster than ParaFEM but requires much more memory as the number of processes increases. (The peak memory occurs during the assembly of the stiffness matrix – off-process entries are stored temporarily as elements are added with p\_add\_element and then distributed with p\_assemble. The relative amount of temporary storage depends on how the mesh is partitioned among the processes but will always increase with increasing numbers of processes.) On distributed-memory computers (e.g., clusters), the total memory available will usually increase faster than the peak memory requirement.

Figure 1 Time and memory scaling of a conjugate gradient solver with Jacobi (diagonal) preconditioning for an elastic solid test case (xx18) with 1003 elements. Time is the sum of the matrix assembly, preconditioner setup and solver times. Memory is the peak memory use. Cray XC30, 8 Intel E5-2650 v2 processors (4 nodes) with 12 cores per processor. The slow-down as the number of processes increases beyond 8 is due to increasing memory contention, not the increase in communications/computation ratio. For 8 processes or less, there is one process per processor, and each processor has low-latency access to 32 GB of memory. As the number of processes increases, they are spread evenly over the processors, leading to 2, 3 and 6 processes per processor. There is a fixed memory bandwidth, so increasing the numbers of processes accessing the same memory increases the time per memory access.

### Program xx17 Three-dimensional steady-state Navier-Stokes analysis. Compare Program 12.6



This is based on Program 12.6 but with a choice between a ParaFEM solver (bicgstabl\_p) and PETSc solvers. The choice between ParaFEM and PETSc is treated in the same way as in Program xx18.

Note that a penalty method is not used (despite the description): the matrix and load vector are modified for the fixed freedoms. When using the ParaFEM solver this is done in the program; when using the PETSc solver this is done by the subroutine p\_zero\_rows.

Note also that Jacobi (diagonal) preconditioning leads to poorer convergence than no preconditioning: the matrix is non-symmetric and has zeroes on the diagonal. When using ParaFEM, the bicgstabl\_p subroutine does not apply the diagonal preconditioner; when using PETSc, the preconditioner type is set to ‘none’ in the example .petsc configuration files.

The number of iterations reported by PETSc is *l* times the number of ‘outer’ iterations. The number of iterations reported by ParaFEM is the number of ‘outer’ iterations. So, the iteration limit for PETSc set in the .petsc file should be *l* times the iteration limit for ParaFEM set in the .dat file.

For the xx17\_tiny example (cuboidal lid-driven cavity with 103 elements, same as p126\_demo) on 24 processes, PETSc BiCGStab(*l=4*) takes that same number of iterations, and slightly less time to converge than ParaFEM. For larger problems, PETSc BiCGStab is slower than ParaFEM BiCGStab: for the xx17\_medium example (cuboidal lid-driven cavity with 503 elements) on 960 processes, PETSc BiCGStab(*l=4*) is 60% slower than ParaFEM (40% more iterations and 15% longer per iteration). PETSC BiCGStab is also sensitive to the order of global reductions – repeated runs give different convergence. PETSc’s BiCGStab(*l*) is an enhanced algorithm (D.R. Fokkema, "Enhanced implementation of BiCGStab(*l*) for solving linear systems of equations", preprint from [www.citeseer.com](http://www.citeseer.com)) and this may explain the differences. As explained in the Program xx18 section, PETSc matrix assembly requires more memory than ParaFEM matrix assembly – in this case 2.5 times as much.

Numdiff (<http://www.nongnu.org/numdiff>) is a useful tool for comparing the results from the ParaFEM programs. It is useful in comparing the output of xx17 for the ParaFEM and PETSc solvers but it is not so useful for the xx15 cases, where vector or tensor quantities need to be compered. For the xx15 cases, a program to check the norms of differences is provided (normdiff).

### Program xx15 Three-dimensional nonlinear analysis of an elastoplastic solid. Experimental

 

The xx15 programs are experimental programs for finite-strain elasto-plasticity. They have similar structure, with a sequence of load steps until yield occurs:

Create stiffness matrix K and vectors x, f

DO load step

DO Newton-Raphson iteration

Set the entries in K

Solve Ku = f

END DO

END DO

The two programs use different constitutive models (UMATs) and the quadric linear hardening program tests for non-convergence of the UMAT calculation. The programs will be combined into one in the future, with a library of UMAT subroutines to choose from.

The programs use different conventions for the fixed freedoms and restraints from the rest of ParaFEM. For the Program xx15 restraints, 0 means free and 1 means restrained (to zero if there is no corresponding entry in the fixed freedoms file). So, there are no equations for the fixed freedoms and no need to zero any rows and columns in the global stiffness matrix as in Program xx17. The right-hand side is modified separately in xx15. For the rest of ParaFEM, 0 means restrained to zero and 1 means free (and the fixed freedoms can override that).

The choice of solver between ParaFEM and PETSc is handled in much the same way as for Program xx17. When using PETSc there is an additional array fixkm\_pp to hold the information about the fixed freedoms – when using ParaFEM that information is available in the storekm\_pp array which stores the stiffness matrix. The von Mises non-linear hardening program includes a mock example of switching to a second solver during Newton-Raphson iterations. A utility program normdiff is provided to compare results.

Small test examples are provided in the examples/dev/xx15 directory and the quadric linear hardening program has been used for simulations of trabecular bone under compression (Levrero-Florencio *et al.*, 2017). Figure 2 compares the time and memory use for ParaFEM and PETSc solvers for a simulation of trabecular bone using 7 million elements, performed on a Cray XC-30 system. The same solver and preconditioner is used, the differences are from the different implementation. PETSc is 10% slower than ParaFEM at large numbers of processes and, as in Program xx18, requires much more memory at the assembly stage than ParaFEM.

Figure 2 Time and memory scaling of a conjugate gradient solver with Jacobi (diagonal) preconditioning for a simulation of bone with 7M elements. Cray XC30, Intel E5-2650 v2 processors.

## Glossary of Variable Names

Additional names required to use PETSc in the driver programs. This does not list the large number of other new variables in the xx15 programs.

### Scalar integers:

no new variables

### Scalar reals:

peak\_memory\_use maximum amount of memory used so far in the program, in GB

### Scalar characters:

solvers name of solvers to use, either ‘parafem’ or ‘petsc’  
tab tab character, used for output formatting.

### Scalar logicals:

error flag for errors from the PETSc interface subroutines

### Dynamic integer arrays:

no new variables

### Dynamic real arrays:

km single element stiffness matrix  
ke single element stiffness matrix (Navier-Stokes program)  
fixkm\_pp distributed copy of the elements with fixed displacements

# Appendix G

(The External Subprograms appendix is shifted to Appendix H.)

This appendix describes the additional library subroutines used by the Chapter 12 programs illustrating the use of PETSc solvers and preconditioners in parallel finite element computations. All the subroutines can be downloaded from http://parafem.org.uk.

## Subroutine descriptions

The following descriptions indicate the library subroutines in alphabetic order, together with the meaning of their arguments. Arguments in **bold** are those returned by the subroutine. Arguments in *italic* or ***bold italic*** are optional.

| Name | Arguments | Description |
| --- | --- | --- |
| get\_solvers |  | (Function) Gets the name of the solvers to use, parafem or petsc, from the command line. |
| p\_add\_element | g, km | Adds the element matrix km to the global PETSc matrix |
| p\_assemble |  | Assembles the PETSc global matrix (distributes the matrix elements created by the previous calls of p\_add\_element). |
| p\_create | ntot\_max, g\_g\_pp, **error** | Creates the PETSc data structure to store the global matrix and vectors. |
| p\_destroy |  | Destroys the PETSc data structure (needed only if the matrix structure changes, e.g. if the mesh is changed). |
| p\_finalize |  | Tidies up after PETSc |
| p\_initialize |  | Initialize PETSc |
| p\_memory\_peak |  | (Function) Returns the maximum amount of memory used so far in the program. Linux only. |
| p\_memory\_use |  | (Function) Returns amount of memory in use in the program. Linux only. |
| p\_print\_info | output\_pe, unit | Prints the convergence information. |
| p\_release\_memory |  | (Function) Releases all freed memory back to the system. Linux only. |
| p\_shutdown |  | Destroys the PETSc data structures and finalizes PETSc. |
| p\_solve | r\_pp, x\_pp, ***reason***, *initial\_guess\_nonzero*, *reuse\_preconditioner* | Solve using PETSc, using the only solver, or the current solver chosen using p\_use\_solver |
| p\_use\_solver | solver, **error** | Choose one of the PETSc solvers specified in the .petsc configuration files |
| p\_version |  | (Function) Returns the PETSc version. |
| p\_zero\_matrix |  | Zeroes the PETSc global matrix. |
| p\_zero\_rows | fixed\_eqns, diag, fixed\_values, **r\_pp** | Modifies the PETSc global matrix and the load vector r\_pp by the fixed freedoms. The resulting matrix in not symmetric. |
| solvers\_valid | solvers | (Function) Checks if solvers is one of the supported solvers (ParaFEM or PETSc) |