Incompressible Flow & Iterative Solver Software (IFISS) Installation & Software Guide [1]

David J. Silvester[2] Howard C. Elman[3] Alison Ramage[4]

Version 3.5 | released 22 September 2016

[1]This project was supported in part by the U.S. National Science Foundation under grant DMS0208015, the U.S. Department of Energy under grant DOEG0204ER25619, and the UK Engineering and Physical Sciences Research Council under grant EP/C000528/1.

[2]School of Mathematics, University of Manchester, Manchester, UK.
[3]Department of Computer Science, University of Maryland, College Park, Maryland, USA.
[4]Department of Mathematics and Statistics, University of Strathclyde, Glasgow, UK.

Contents

1.1 Background

This document is an overview of the IFISS software library that has been developed in conjunction with the monograph [ESW2014]

Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics
Howard Elman, David Silvester and Andy Wathen
Oxford University Press, Second Edition, 2014.

The IFISS software library is "open-source" and is written in MATLAB.[1] It can also be installed and run using the freely available Octave package.[2]

IFISS can be downloaded from either of the following sites:

`http://www.manchester.ac.uk/ifiss`
`http://www.cs.umd.edu/~elman/ifiss.html`.

It can be distributed and/or modified under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or any later version. For precise details, see the file `readme.m`. The software is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU Lesser General Public License `http://www.gnu.org/licenses/lgpl.html` for a definitive statement. The IFISS library may be freely copied as long as the file `readme.m` is included. The current version of the software: IFISS 3.5 has been tested using Windows, Unix and Mac OS X. Previous releases were developed using MATLAB Versions 5.3 to 8.5. (If you are not sure which version of MATLAB you have running, just type `version` at the system prompt.) The current release was developed and tested using MATLAB 9.1 (R2016b) and is fully compatible with Octave 4.0.3.

1.1.1 Installation

IFISS is downloaded in the form of a gzipped tar file (Linux/MacOS) or a `zip` file (Windows). After unpacking the `tar` or `zip` file, installation is achieved by manually editing the scriptfile `gohome.m` in the top level directory. This scriptfile identifies the "home" directory of the package via a command of the form

`cd('<local directory>/ifiss3.5')`, or `cd('<local directory>\ifiss3.5')`,

when installing in a Linux/MacOS or Windows environment, respectively. After this has been done, IFISS is set to run without additional user intervention.

Once IFISS is installed, for all subsequent uses the MATLAB path must include the IFISS home directory. This requirement can be enforced each time MATLAB or Octave is initiated either by

[1] Copyright © The MathWorks, software is available from `http://www.mathworks.com/`.
[2] See `http://www.gnu.org/software/octave/`.

typing `setpath` in response to the prompt, or by incorporating the functionality of the `setpath` command into the user's MATLAB startup file. Having run `setpath` at the MATLAB or Octave prompt, simply type `helpme` to get started.

1.1.2 Overview

The IFISS package originally focussed on four specific partial differential equations (PDEs) that arise in modelling steady-state phenomena: these are the Poisson equation, the convection-diffusion equation, and the Stokes and Navier–Stokes equations. The first two of these equations are ubiquitous in scientific computing, and the latter two constitute the basis for computational modelling of the flow of an incompressible Newtonian fluid.

The latest release has a total of twenty eight built-in problems. A full description of these problems can be found in [ESW2014]. Seventeen of these involve the Poisson equation and steady-state versions of the convection-diffusion equation, the Stokes equations and the Navier–Stokes equations. A short summary of the associated test problems is given in the first Appendix. The remaining eleven problems are associated with the heat equation (which models the evolution of the temperature of a material), and time-dependent versions of the convection-diffusion equation, the Navier–Stokes equations and their Boussinesq flow combination. A short summary of the unsteady test problems is given in the second Appendix. Another new feature of IFISS is the inclusion of examples of optimisation problems subject to a PDE constraint. This topic is discussed in Chapter 5 of [ESW2014].

The software package has two important components.[3] The first component of IFISS concerns problem specification and finite element discretisation. For each of the equations listed above, IFISS offers a choice of two-dimensional domains on which the problem can be posed, along with boundary conditions and other aspects of the problem, and a choice of finite element discretisations on a quadrilateral element mesh. The package allows the study of accuracy of finite element solutions, different choices of elements, and a posteriori error analysis. In addition, special features associated with individual problems can be explored. These include the effects of boundary layers on solution quality for the convection-diffusion equation, and the effects of discrete inf-sup stability on accuracy for the Stokes and Navier–Stokes equations.

The second major feature of the package concerns iterative solution of the discrete algebraic systems that arise: either at every time step when solving an unsteady problem, or in the generation of a steady-state solution. The focus is on preconditioned Krylov subspace that are chosen to match the underlying problem. For example, the discrete Poisson equation, which has a symmetric positive definite coefficient matrix, can be treated by the conjugate gradient method (CG) whereas the discrete convection-diffusion and Navier–Stokes equations require a method such as the generalized minimum residual method (GMRES), which is designed for non-symmetric systems. The key for fast solution lies in the choice of effective preconditioning strategies. The package offers a range of options, including algebraic methods such as incomplete LU factorisations, as well as more sophisticated and state-of-the-art multigrid methods designed to take advantage of the structure of the discrete linearised Navier–Stokes equations.

[3] See the software review: "IFISS: A computational laboratory for investigating incompressible flow problems" `http://eprints.ma.man.ac.uk/1969/`.

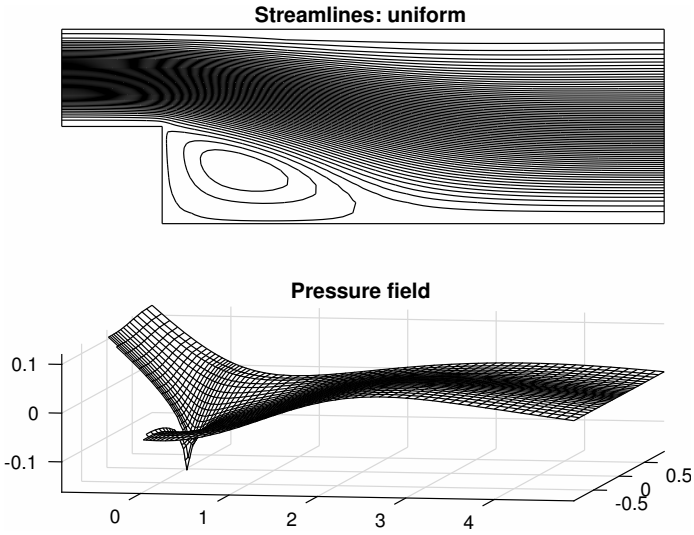
1.1.3 Sample Session

A sample IFISS session for test problem **NS2** (see the Appendix), which models flow over a step is reproduced below. The driver `navier_testproblem` asks the user to choose the problem, the grid resolution and the type of finite element discretisation to be used. The resulting nonlinear system is then solved using a hybrid Picard/Newton solver and an estimate of the error is computed as described in Chapter 8 of [ESW2014].

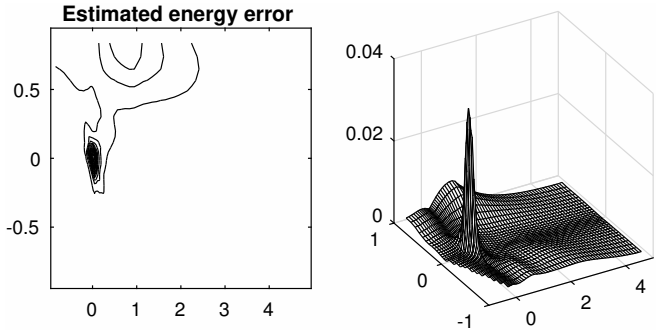
```
>> setpath
>> ifiss
This is IFISS version 3.5: released 20 September 2016
For help, type "helpme".
>> navier_testproblem

specification of reference Navier-Stokes problem.

choose specific example (default is cavity)
     1  Channel domain
     2  Flow over a backward facing step
     3  Lid driven cavity
     4  Flow over a plate
     5  Flow around a square obstruction
     6  Flow in a symmetric step channel
 : 2
horizontal dimensions [-1,L]: L? (default L=5) : 5

Grid generation for backward-facing step domain.
grid parameter: 3 for underlying 8x24 grid (default is 4) : 5
grid stretch factor (default is 1) : 1.2
   Grid generation for x-channel ...done.
   outlet subdivision parameter set to 5
   ... associated uniform grid value is 8
   Grid generation for x-channel ...done.
   Merger of two x-channel grids
   zip distance is 0.0000e+00 ... it should be close to zero!
  All done.

Q1-Q1/Q1-P0/Q2-Q1/Q2-P1: 1/2/3/4? (default Q1-P0) : 4
setting up Q2-P1 matrices...  done
system matrices saved in step_stokes_nobc.mat ...
Incompressible flow problem on step domain ...
viscosity parameter (default 1/210) : 1/50
Picard/Newton/hybrid linearization 1/2/3 (default hybrid) :
number of Picard iterations (default 3) : 2
number of Newton iterations (default 5) : 5
nonlinear tolerance (default 1.1*eps) :
stokes system ...
```

```
setting up Q2 convection matrix...  done.
uniform/nonuniform streamlines 1/2 (default uniform) :
number of contour lines (default 50) : 75


initial nonlinear residual is 3.781416e+00
Stokes solution residual is 8.524251e-01

Picard iteration number 1
setting up Q2 convection matrix...  done.
nonlinear residual is 9.933692e-03
   velocity change is 3.986598e+00

Picard iteration number 2
setting up Q2 convection matrix...  done.
nonlinear residual is 3.894324e-03
   velocity change is 1.938277e+00

Newton iteration number 1
setting up Q2 Newton Jacobian matrices...  done.
setting up Q2 convection matrix...  done.
nonlinear residual is 5.551608e-04
   velocity change is 1.453605e+00

Newton iteration number 2
setting up Q2 Newton Jacobian matrices...  done.
setting up Q2 convection matrix...  done.
nonlinear residual is 1.284289e-06
   velocity change is 7.097577e-02

Newton iteration number 3
setting up Q2 Newton Jacobian matrices...  done.
setting up Q2 convection matrix...  done.
nonlinear residual is 3.441705e-11
   velocity change is 3.025599e-04

Newton iteration number 4
setting up Q2 Newton Jacobian matrices...  done.
setting up Q2 convection matrix...  done.
nonlinear residual is 4.530095e-16
   velocity change is 4.859822e-09

finished, nonlinear convergence test satisfied

Natural outflow on right hand vertical boundary ..
FAST Q2-P1 NS a posteriori error estimation
checking edge numbering and computing edge lengths ... done
```

```
Q2-P1 local N-S error estimator ...
interior residual RHS assembly took 1.1257e-01 seconds
flux jump RHS assembly took 1.4966e-02 seconds
LDLT factorization took 4.4378e-03 seconds
triangular solves took 3.9028e-03 seconds
computing divergence of discrete velocity solution ...  done
estimated velocity divergence error:  1.052721e-03
estimated energy error is 8.0507e-02
plotting element data... done
```



(a) Solution to problem **NS2** with stabilised $\mathbf{Q}_2$–$\mathbf{P}_1$ approximation.



(b) Estimated error in the computed solution.

Figure 1.1: Sample output from `navier_testproblem`.

The computed solution and the estimated error are shown in Figure 1.1 (a) and (b).

Once a problem has been set up in this way, the performance of iterative solution methods and preconditioners can be explored using the driver `it_solve`. Here, the chosen iterative method is GMRES with ideal pressure convection-diffusion preconditioning. As shown below the method converges in 74 iterations, and the convergence curve is the blue line shown in Figure 1.2.

```
>> it_solve
inflow/outflow (step) problem ...
solving Jacobian system generated by solution from last Newton step

setting up Q2 Newton Jacobian matrices...  done.
GMRES/Bicgstab(l)/IDR(s)  1/2/3 (default GMRES) :
stopping tolerance? (default 1e-6) :
maximum number of iterations? (default 100) :
preconditioner:
   0  none
   1  unscaled least-squares commutator (BFBt)
   2  pressure convection-diffusion (Fp)
   3  least-squares commutator (LSC)
   4  modified pressure convection-diffusion (Fp*)
   5  boundary-adjusted least-squares commutator (LSC*)
default is modified pressure convection-diffusion : 2
ideal / AMG iterated  preconditioning? 1/2 (default ideal) :
setting up Q2-P1 pressure preconditioning matrices...
NonUniform grids are fine.
fixed pressure on inflow boundary
ideal pressure convection-diffusion preconditioning ...

GMRES iteration ...
convergence in  74 iterations

    k  log10(||r_k||/||r_0||)
    0           0.0000
    1          -0.0244
    2          -0.0255
    3          -0.0578
    4          -0.0878
    .
    .
    .
   73          -5.8587
   74          -6.0021
Bingo!

  2.5431e+00 seconds

use new (enter figno) or existing (0) figure, default is 0 : 1
colour (b,g,r,c,m,y,k): enter 1--7 (default 1) : 1
```

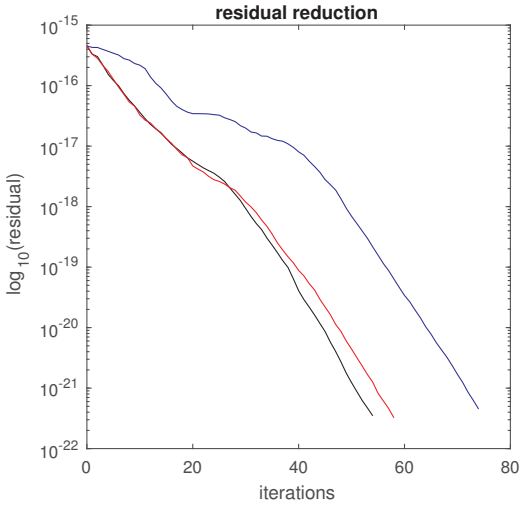


Figure 1.2: Sample output from `it_solve`.

To produce the second (black) curve in Figure 1.2, `it_solve` must be rerun using *modified* pressure convection-diffusion preconditioning. This converges in fewer iterations, reducing the CPU time.

```
>> it_solve
inflow/outflow (step) problem ...
solving Jacobian system generated by solution from last Newton step

setting up Q2 Newton Jacobian matrices...  done.
GMRES/Bicgstab(l)/IDR(s)  1/2/3 (default GMRES) :
stopping tolerance? (default 1e-6) :
maximum number of iterations? (default 100) :
preconditioner:
   0  none
   1  unscaled least-squares commutator (BFBt)
   2  pressure convection-diffusion (Fp)
   3  least-squares commutator (LSC)
   4  modified pressure convection-diffusion (Fp*)
   5  boundary-adjusted least-squares commutator (LSC*)
default is modified pressure convection-diffusion : 4
ideal / AMG iterated  preconditioning? 1/2 (default ideal) :
setting up modified Q2-P1 pressure preconditioning matrices...
NonUniform grids are fine.
Robin pressure on inflow boundary

modified pressure convection-diffusion preconditioning ...
GMRES iteration ...
convergence in  54 iterations
```

```
    k  log10(||r_k||/||r_0||)
    0           0.0000
    1          -0.1350
    2          -0.1771
    3          -0.3148
    4          -0.4667
    .
    .
    .
   53          -5.9844
   54          -6.1137
Bingo!

  1.8230e+00 seconds

use new (enter figno) or existing (0) figure, default is 0 :
figure number (default is current active figure) :
colour (b,g,r,c,m,y,k): enter 1--7 (default 1) : 7
```

To produce the third (red) curve in Figure 1.2, `it_solve` must be run using modified pressure convection-diffusion preconditioning, this time replacing the sparse direct solves in the preconditioning step with an AMG V-cycle.[4]

```
>> it_solve
inflow/outflow (step) problem ...

solving Jacobian system generated by solution from last Newton step

setting up Q2 Newton Jacobian matrices...  done.
GMRES/Bicgstab(l)/IDR(s)  1/2/3 (default GMRES) :
stopping tolerance? (default 1e-6) :
maximum number of iterations? (default 100) :
preconditioner:
   0  none
   1  unscaled least-squares commutator (BFBt)
   2  pressure convection-diffusion (Fp)
   3  least-squares commutator (LSC)
   4  modified pressure convection-diffusion (Fp*)
   5  boundary-adjusted least-squares commutator (LSC*)
default is modified pressure convection-diffusion : 4
ideal / AMG iterated  preconditioning? 1/2 (default ideal) : 2
setting up modified Q2-P1 pressure preconditioning matrices...
NonUniform grids are fine.
Robin pressure on inflow boundary
```

[4]The number of iterations may not be the same in Octave however ...

```
compute / load convection-diffusion AMG data? 1/2 (default 1) :
AMG grid coarsening ...  13 grid levels constructed.

AMG fine level smoothing strategy? PDJ/ILU 1/2  (default ILU) :
ILU smoothing on finest level..
AMG iterated PCD* preconditioning ...
AMG grid coarsening ...  8 grid levels constructed.
AMG setup of Ap done.
ILU smoothing on finest level..

GMRES iteration ...
convergence in  58 iterations

    k  log10(||r_k||/||r_0||)
    0           0.0000
    1          -0.1359
    2          -0.2053
    3          -0.3203
    4          -0.4195
    .
    .
    .
   57          -5.9924
   58          -6.1471
Bingo!

  3.6941e-01 seconds

use new (enter figno) or existing (0) figure, default is 0 :
figure number (default is current active figure) :
colour (b,g,r,c,m,y,k): enter 1--7 (default 1) : 3
```

Replacing the sparse solves by AMG we see that GMRES converges in 58 rather than 54 iterations. The CPU time is reduced by an additional factor of five however!

The same flow problem could also be solved by time stepping from a quiescent state to the final steady state (problem **T–NS2** in the Appendix). This is the approach in the session that is reproduced below. The driver `unsteady_navier_testproblem` asks the user to choose the problem, the spatial discretisation, the time of integration and an estimate of the required temporal accuracy. The resulting system of differential algebraic equations is solved using the implicit AB2–TR time-stepping algorithm that is described in Chapter 10 of [ESW2014].

```
>> save steadysol xns nnv % save the converged steady solution
>> unsteady_step_navier
Unsteady flow in a step domain ...
viscosity parameter (default 1/220) : 1/50
Discrete Saddle-Point DAE system ...
```

```
target time? (default 1e8) : 1e14
accuracy tolerance? (default 3e-5) : 1e-4
number of Picard steps? (default 2) : 0
averaging frequency? (default 10) :
plot solution evolution? 1/0 : 0
Solving DAE system using stabilized TR ...

AxBhandle =@defaultAxB

  initial nonlinear residual is 3.360822e-02
             boundary change is 2.783973e-08
setting up Q2 convection matrix...  done.

 StabTR with no nonlinear corrections
   step  timestep       time        divresidual     acceleration
     1   1.000e-09     2.000e-09    0.000e+00        1.314e+01
     2   1.000e-09     2.000e-09    2.533e-24        1.314e+01
     3   1.602e-05     1.602e-05    4.607e-24        1.314e+01
     4   3.914e-04     4.074e-04    4.334e-20        1.314e+01
     5   4.036e-03     4.443e-03    1.092e-18        1.308e+01
     6   4.593e-03     9.036e-03    1.122e-17        1.257e+01
     7   5.281e-03     1.432e-02    1.891e-17        1.201e+01
     8   6.888e-03     2.120e-02    2.887e-17        1.140e+01
     9   8.243e-03     2.945e-02    4.709e-17        1.065e+01
    10   9.533e-03     3.898e-02    5.735e-17        9.814e+00
    10   4.767e-03     3.421e-02  --- Averaging step
    11   1.004e-02     4.426e-02    2.685e-04        9.374e+00
    12   7.428e-03     5.169e-02    8.813e-17        8.679e+00
     .
     .

   180   1.290e+13     5.074e+13    2.982e-16        1.411e-17
   180   6.452e+12     4.429e+13  --- Averaging step
   181   1.453e+13     5.881e+13    2.588e-16        2.454e-19
   182   4.119e+13     1.000e+14    3.154e-16        1.717e-19
finished in  182 steps!
Integration took  1.711e+01 seconds

use new (enter figno) or existing (0) figure, default is 0 : 17
183 timesteps
step 183 : final time is 1.000e+14
minimum energy is 1.31376e-08 and maximum is 1.50735

Sanity check: is the step symmmetric? enter 0/1 (yes) : 0

computing divergence of discrete velocity solution ...  done
estimated velocity divergence error:  1.052722e-03
```

```
>> snaptime=[50,70,130];
>> step_unsteadyflowref(qmethod,mv,U,time,A,By,Bx,G,xy,xyp,x,y,bound,snaptime,symm);

   Plotting flow field snapshots ...
   step   time    mean_vorticity    min_phi  max_phi
    50    0.322    4.401e-05        0.00000  6.401e-01
    70    1.080    7.798e-05       -0.00395  6.667e-01
   130  196.045   -7.481e-04       -0.02829  6.667e-01
   All done

>> ufinal=U(:,end); load steadysol % compare velocity solutions
>> fprintf('difference in steady solutions is %6.2e\n',norm(ufinal-xns(1:nnv),inf))
difference in steady solutions is 2.30e-06
```

If the display solution switch is enabled then a dynamic flow animation will be generated that shows the computed vorticity and the magnitude of the velocity at each time step. As anticipated, the unsteady solution goes to a steady state. Three snapshots of the stationary streamlines are illustrated in Figure 1.3.

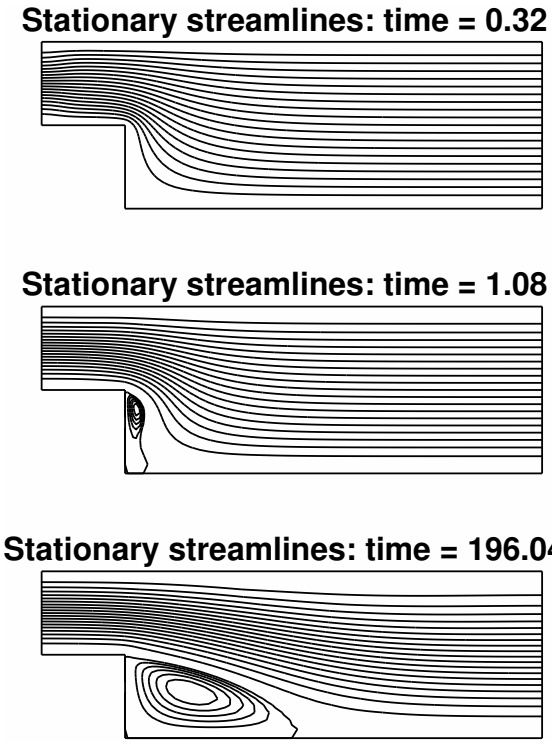


Figure 1.3: Sample output from `step_unsteadyflowref`.

Once a problem has been set up in this way, the performance of iterative solution methods and preconditioners can be explored using the driver `snapshot_solveflow`. Here, the chosen iterative method is GMRES with AMG modified PCD preconditioning. At the selected time step the preconditioned iteration converges in 18 iterations.

```
>> snapshot_solveflow

Iterative solution of a SNAPSHOT linear system
Solution data available for 1e+14 seconds
Approximate time for the SNAPSHOT? (default is the end) : 10

Time step number 105
Constructing system at time 9.64368 seconds
    current timestep is 0.485082 seconds
inflow/outflow (step) problem ...
stopping tolerance? (default 1e-8) :
maximum number of iterations? (default 100) :
preconditioner:
   0  none
   2  modified pressure convection-diffusion (Fp)
   3  least-squares commutator
   4  corrected pressure convection-diffusion (Fp*)
   5  boundary-adjusted least-squares commutator (LSC*)
default is Fp* : 3
ideal / AMG iterated  preconditioning? 1/2 (default ideal) :
ideal least-squares commutator preconditioning ...

GMRES iteration ...
convergence in  18 iterations

    k  log10(||r_k||/||r_0||)
    0           0.0000
    1          -0.0601
    2          -0.4017
    3          -1.0920
   .
   .
   .
   17          -7.6155
   18          -8.0593
Bingo!

  8.0041e-01 seconds
```

A good way to explore the capabilities of IFISS is to run the software demos in the `/ifissdemos/` directory: type `help ifissdemos` for a list. The demos are generated by running IFISS in batch mode. Details are given in the next section.

1.1.4 Running jobs in batchmode

IFISS contains a `batchmode` facility enabling data to be input from a pre-prepared file rather than directly from the terminal. The specific parameters that need to be input will of course vary from problem to problem, and the input file must be prepared accordingly. Sample input files for each of the model problems are provided (located in the appropriate `test_problems` subdirectory, see later) and can be easily modified by the user for a particular run. The names of these input files must have the form "`*_batch.m`" where "`*`" begins with one of "`P`", "`CD`", "`S`" or "`NS`" for the Poisson, convection-diffusion, Stokes or Navier–Stokes equations, respectively. For example, typing the command

```
batchmode('P2')
```

uses the file `P2_batch.m` to generate and solve the discrete Poisson equation on an L-shaped domain without interactive input. The results of the run are stored in the file `batchrun` in the `datafiles` subdirectory.

A similar `batchmode` facility is available for running the driver `it_solve` without interactive input after a discrete system has been generated in batchmode. Input files must have the names `itsolve*_batch.m`. A template, `itsolve_batch.m`, which applies multigrid preconditioned CG to the discrete Poisson equation, is available in the `solvers` subdirectory:

```
batchmode('itsolve')
```

This file would have to be modified by the user to contain the appropriate parameter values for other problems. The list of parameters required in each case can be generated by carrying out an initial run in interactive mode. For further details, type `help batchmode`.

2.1 Directory structure

As already noted, IFISS comprises functions which generate finite element approximations of the following PDE problems that arise in incompressible flow modelling:

- the Poisson equation: directory `/ifiss/diffusion/`
- the convection-diffusion equation: directory `/ifiss/convection/`
- the Stokes equations: directory `/ifiss/stokes_flow/`
- the Navier-Stokes equations: directory `/ifiss/navier_flow/`

The temporal discretization functions associated with unsteady versions of the above PDEs can be found in a separate directory:

- `/ifiss/timestepping/`

The Boussinesq flow functions are in the following directory:

- `/ifiss/boussinesq_flow/`

Each of these directories has a subdirectory `/test_problems/`. These contain the boundary and coefficient function files associated with the PDE reference problems described in [ESW2014]. The functions associated with the domain geometry and grid generation are independent of the PDE being solved. These functions are located in a separate directory:

- `/ifiss/grids/`

The computed solutions are visualized using the three-dimensional plotting functions that are built into MATLAB. The functions that generate this visual output are also located in a separate directory:

- `/ifiss/graphs/`

For each class of discrete problem we provide specialized fast iterative solvers. The associated functions are contained in two directories:

- `/ifiss/solvers/` ; `/ifiss/stokes_minres/`

The PDE optimization functionality is also contained in the separate directory:

- `/ifiss/pde_control/`

Type `helpme_poissoncontrol` to get further information. Finally, there are two directories that are used for storing intermediate data (for example, finite element matrices and multigrid data) and plot files:

- data (`.mat`) files : directory `/ifiss/datafiles/`
- plot (`.pdf`) files : directory `/ifiss/plotfiles/`

Help for the package is integrated into the MATLAB help facility. The command `help ifiss` gives a pointer to the IFISS general help command `helpme`. Typing `help` ⟨`directory name`⟩ lists the files in that directory that users may want to look at more closely. Using MATLAB version 7, the function names are "clickable" to give additional information.

The IFISS package consists of over 400 MATLAB functions and script files. Simply type `help <file-name>` for further information on any of these. For a complete list of functions and scripts in a specific directory type `help <directory-name>`.

2.1.1 IFISS figures

As is evident from the sample runs described above, several figures are "preallocated" within IFISS. That is, particular figures are always used for generating specific graphical output. A list of these figures is given below.

Figure	Description
10	grid plot (diffusion or convection-diffusion)
11	diffusion problem solution & error plot ($\boldsymbol{Q}_1$)
12	diffusion problem solution & error plot ($\boldsymbol{Q}_2$)
19	heat equation temperature evolution
20	convective wind
22	convection-diffusion solution & error plot ($\boldsymbol{Q}_1$)
29	unsteady convection-diffusion problem temperature evolution
30	grid plot (Stokes or Navier-Stokes flow problem)
33	Stokes flow solution plot
34	Stokes flow solution error plot ($\boldsymbol{Q}_1$–$\boldsymbol{P}_0$, $\boldsymbol{Q}_1$–$\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$–$\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$)
66	Navier-Stokes flow solution plot
67	Navier-Stokes flow solution error plot ($\boldsymbol{Q}_1$–$\boldsymbol{P}_0$, $\boldsymbol{Q}_2$–$\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$)
167	unsteady Navier-Stokes flow solution evolution

3.1 Generating new domains or model problems

For each PDE problem that is treated, IFISS contains a number of built-in model problems. It is also straightforward to adapt the code to include different domains, PDE features or boundary conditions. The way to achieve this is outlined below.

3.1.1 Introducing a new problem domain

The first task in setting up a different PDE domain is the grid generation, which should be done by a function `<newproblem>_domain.m` analogous to those in the `/grids/` directory (see Section 2.1 for details of the directory structure). This function should generate grid information (specifically, node point coordinates and element connectivity information), which will also serve to define the domain. The results should be saved in a data file called `/datafiles/<newproblem>_grid.mat`.

The grid information should be organized into a collection of MATLAB arrays specified as follows.

- A nodal coordinate matrix (`xy`) of size (# of nodes)×2, in which the first column contains the x-coordinates of the nodes and the second column contains the y-coordinates.
- A matrix (`mv`) of size (# of macroelements)×9 containing the so-called "macroelement mapping". For biquadratic $\mathbf{Q}_2$ approximation this is simply the connectivity array of the grid, i.e. entry `mv(nel,nv)` contains the global node number of node `nv` on element `nel`. The PDE problem drivers in IFISS tacitly assume that the same nine-node data structure is used in

the case of bilinear $\mathbf{Q}_1$ approximation. This means that the associated subdivision is formed from macroelements each consisting of four elements. This concept is explained in more detail below.

- A boundary vertex vector (`bound`) containing a list of nodes on the Dirichlet part of the boundary.
- A macroelement boundary edge matrix (`mbound`) of size (# of macroedges)×4. For each macroelement having an edge on the Dirichlet part of the boundary, the first column is a pointer to the macroelement number and the second is an integer 1, 2, 3 or 4, which uniquely defines the orientation of the edge (bottom, right, top or left).
- Two vectors `x` and `y`. These are used solely for plotting purposes (MATLAB assumes a rectangular matrix of values when calling `mesh` and `contour` plotting functions). The default choice is to match the data in `x` and `y` to the nodal coordinate data in `xy`. For a cartesian product grid this enables generation of mesh plots showing the underlying element structure. For general quadrilateral grids the data in `x` and `y` determine the interpolation points used in generating the solution plots.

After a grid is created by the grid generator, the data can be visually checked using the function `macrogridplot`. This facility is illustrated in the sample IFISS session below, which produces a nonrectangular domain and a nonrectangular grid.

```
>> quad_domain            % generates the datafile quad_grid.mat

Grid generation for quadrilateral domain.
grid parameter: 3 for underlying 8x8 grid (default is 16x16) : 3
aspect ratio (x:1) (default is 4:1) : 1
height contraction ratio (default is 4) : 2
outflow boundary: natural/prescribed 1/2 (default is natural) :
plotting grid points ...

>> load quad_grid
>> macrogridplot(xy,mv,bound,mbound);

Subdivision logistics ..
  81 nodes
  16 (2x2 macro)elements
  25 nodes on Dirichlet boundary
  12 macroelement edges on Dirichlet boundary

>> disp(mv)
     1    19    21     3    10    20    12     2    11
     3    21    23     5    12    22    14     4    13
     5    23    25     7    14    24    16     6    15
     ...
    59    77    79    61    68    78    70    60    69
    61    79    81    63    70    80    72    62    71
```

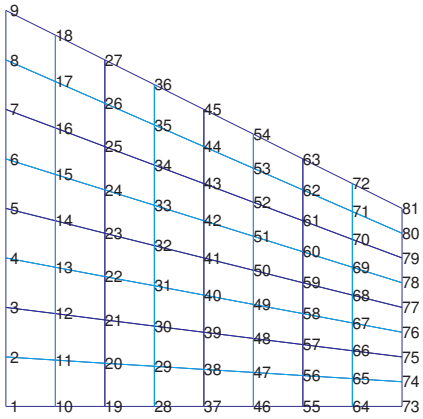
```
>> [ev,ebound]=q1grid(xy,mv,bound,mbound);
>> gridplot(xy,ev,bound,ebound);

Grid logistics ..
  81 nodes
  64 elements
  25 nodes on Dirichlet boundary
  24 element edges on Dirichlet boundary
```

The graphical output from `macrogridplot` is shown in Figure 3.4. The macroelement connectivity matrix `mv` can be correlated with the numbered nodes in the figure. The macroelement boundaries are distinguished from the internal element boundaries by their darker colour and represent the actual grid lines if biquadratic approximation is employed. In the example above, the call to `q1grid` is used to generate a bilinear approximation, for which the underlying grid is that represented by the union of the light and dark blue boundaries. This function generates the element connectivity matrix `ev` and the element boundary edge matrix `ebound`. This information can also be visualized by calling the function `gridplot` (graphical output not shown).

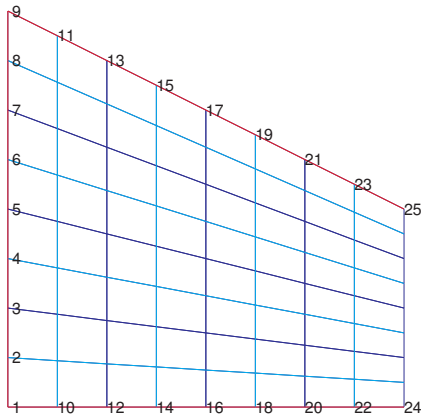
In addition to the grid generator, the other requirement is to write the PDE driver. Templates for this are the function `quad_diff` (solving Poisson's equation) and the function `quad_stokes` (which sets up matrices for subsequent solution using `solve_stokes` or `solve_navier`).

3.1.2 Changing PDE features or boundary conditions

Information on PDE attributes and boundary conditions are held in specific `m-files` in the directory associated with each PDE. For example, for Navier-Stokes flow problems these are the two files `/stokes_flow/specific_flow.m` and `/stokes_flow/stream_bc.m`. These specify the velocity boundary conditions and stream-function boundary conditions, respectively. To define a different Navier-Stokes flow problem on the same domain one simply needs to edit these two `m-files`. Further information can be obtained by using `helpme`.



(a) Indices of nodes of the macroelement grid.



(b) Indices of nodes on the Dirichlet boundary.

Figure 3.4: Sample output from `macrogridplot`.

Appendix A

Steady-state problems

In this appendix we give a brief description of the seventeen test problems currently implemented in IFISS. A fuller description can be found in [ESW2014]. Each problem can be generated by running the appropriate driver routine (as identified below) and is based on one of the following physical domains:

$\Omega_{\square}$: the square $(-1,1)\times(-1,1)$;

$\Omega_{\square\!\!\square}$: the L-shaped region generated by taking the complement in $(-1,L)\times(-1,1)$ of the quadrant $(-1,0]\times(-1,0]$;

$\Omega_{\square\!\!-\!\!\square}$: the rectangular region $(-1,5)\times(-1,1)$, with a slit along the line where $0\le x\le 5$ and $y=0$.

$\Omega_{\square\,\square}$: a disconnected rectangular region $(0,8)\times(-1,1)$ generated by deleting the square $(7/4,9/4)\times(-1/4,1/4)$.

The Poisson equation: `diff_testproblem`

P1 The domain is $\Omega_{\square}$, the source is the constant function $f(x,y)=1$, and zero Dirichlet conditions are applied on all boundaries. This represents a simple diffusion model for the temperature distribution $u(x,y)$ in a square plate, with uniform heating of the plate whose edges are kept at an ice-cold temperature.

P2 The source function and boundary conditions are the same as above but here the domain is $\Omega_{\square\!\!\square}$. As a result, the underlying solution to the Poisson problem has a singularity at the origin.

P3 The domain is $\Omega_{\square}$ and the source function f is identically zero. The boundary conditions are chosen so that the problem has the exact analytic solution

$$u(x,y)=\frac{2(1+y)}{(3+x)^2+(1+y)^2}.$$

P4 This is a second analytic test problem, which is associated with the singular solution of problem **P2** given by

$$u(r,\theta) = r^{2/3}\sin\left(\frac{2\theta+\pi}{3}\right),$$

where r represents the radial distance from the origin, and θ is the angle with the vertical axis.

The convection-diffusion equation: `cd_testproblem`

All of these reference problems are posed on the square domain $\Omega_\square$ with convective velocity of order unity, that is, $\|\vec{w}\|_\infty = O(1)$.

CD1 For constant convective velocity vector $\vec{w} = (0,1)$, the function

$$u(x,y) = x\left(\frac{1-e^{\frac{y-1}{\epsilon}}}{1-e^{-\frac{2}{\epsilon}}}\right)$$

satisfies the convection-diffusion equation exactly. For this problem, Dirichlet conditions on the boundary are determined by this solution, and satisfy

$$u(x,-1) = x, \quad u(x,1) = 0, \quad u(-1,y) \approx -1, \quad u(1,y) \approx 1,$$

where the latter two approximations hold except near $y = 1$. The dramatic change in the value of u near $y = 1$ constitutes an *exponential boundary layer*. This problem also has an option for applying a natural (Neumann) boundary condition on the outflow (top) boundary.

CD2 Here $\vec{w} = (0, 1 + (x+1)^2/4)$, so the wind is again vertical but increases in strength from left to right across the domain. The function u is set to unity on the inflow boundary and decreases to zero quadratically on the right wall and cubically on the left wall. On the outflow (top) boundary, either a Dirichlet or Neumann condition can be applied (both homogeneous). The fixed values on the side boundaries generate *characteristic boundary layers*.

CD3 For this problem, $\vec{w} = \left(-\sin\frac{\pi}{6}, \cos\frac{\pi}{6}\right)$, that is, the wind is still constant but is now at an angle of 30° to the left of vertical. The Dirichlet boundary conditions are zero on the left and top boundaries and unity on the right boundary, with a jump discontinuity (from 0 to 1) on the bottom boundary at the point $(0,-1)$. The resulting discontinuity in the solution is smeared by the presence of diffusion, producing an *internal layer*. There is also an exponential boundary layer near the top boundary $y = 1$.

CD4 This is a simple model for the temperature distribution in a cavity with a 'hot' external wall. The wind $\vec{w} = (2y(1-x^2),\ -2x(1-y^2))$ determines a recirculating flow. The Dirichlet boundary conditions imposed have value one on the right-hand (hot) wall and zero everywhere else. There are therefore discontinuities at the two corners of the hot wall, $x = 1$, $y = \pm 1$, which lead to boundary layers near these corners.

The Stokes equations: `stokes_testproblem`

S1 This problem represents steady horizontal flow in a channel driven by a pressure difference between the two ends, or *Poiseuille flow*. Here a solution is computed numerically on $\Omega_{\square}$ using the velocity $\vec{u} = (1 - y^2,\ 0)$ to define a Dirichlet condition on the inflow boundary $x = -1$. The no-flow Dirichlet condition $\vec{u} = \vec{0}$ is applied on the characteristic boundaries $y = -1$ and $y = 1$. At the outflow boundary ($x = 1, -1 < y < 1$), there is a choice of applying a Neumann or a Dirichlet condition.

S2 This example represents slow flow in a rectangular duct with a sudden expansion, or *flow over a step*. The domain is $\Omega_{\sqsubset}$ with $L = 5$. A Poiseuille flow profile is imposed on the inflow boundary ($x = -1; 0 \le y \le 1$), and a no-flow (zero velocity) condition is imposed on the top and bottom walls. A Neumann condition is applied at the outflow boundary which automatically sets the mean outflow pressure to zero.

S3 This is a classical test problem used in fluid dynamics, known as *driven-cavity flow*. It is a model of the flow in a square cavity (the domain is $\Omega_{\square}$) with the lid moving from left to right. A Dirichlet no-flow condition is applied on the side and bottom boundaries. Different choices of the nonzero horizontal velocity on the lid give rise to different computational models:

$$\begin{array}{ll} \{y = 1; -1 \le x \le 1 | u_x = 1\}, & \text{a } \textit{leaky} \text{ cavity;} \\ \{y = 1; -1 < x < 1 | u_x = 1\}, & \text{a } \textit{watertight} \text{ cavity;} \\ \{y = 1; -1 \le x \le 1 | u_x = 1 - x^4\}, & \text{a } \textit{regularised} \text{ cavity.} \end{array}$$

S4 This is a simple model of *colliding flow*. It is an analytic test problem on $\Omega_{\square}$ associated with the solution of the Stokes equations given by

$$\vec{u} = (20xy^3,\ 5x^4 - 5y^4), \quad p = 60x^2y - 20y^3 + \text{constant}.$$

The interpolant of $\vec{u}$ is used to specify Dirichlet conditions everywhere on the boundary.

Navier-Stokes equations: `navier_testproblem`

The first three of these test problems are fast-flowing analogues of the first three Stokes flow problems.

NS1 The Poiseuille channel flow solution $\vec{u} = (1 - y^2,\ 0)$, $p = -2\nu x$ is also an analytic solution of the Navier-Stokes equations, since the convection term $\vec{u} \cdot \nabla \vec{u}$ is identically zero. The only difference is that here the pressure gradient is proportional to the viscosity parameter. As for the analogous Stokes problem **S1**, at the outflow boundary ($x = 1, -1 < y < 1$) there is a choice of Neumann or Dirichlet boundary conditions.

NS2 This example represents flow over a step of length L (the domain is $\Omega_{\sqsubset}$ with L chosen by the user). For high Reynolds number flow, longer steps are required in order to allow the flow to fully develop (unlike in problem **S2**, where $L = 5$ is sufficient). The boundary conditions are identical to those in problem **S2**.

NS3 This problem again models flow in a cavity $\Omega_{\square}$. The boundary conditions are the same as in problem **S3**, with the choice of a leaky, watertight or regularised lid boundary condition.

NS4 This is a classical problem in fluid dynamics which is referred to as *Blasius flow*: the objective is to compute the steady flow over a flat plate moving at a constant speed through a fluid that is at rest. The flow domain is $\Omega_{\boxminus}$. The 'parallel flow' Dirichlet condition $\vec{u} = (1, 0)$ is imposed at the inflow boundary ($x = -1; -1 \leq y \leq 1$) and also on the top and bottom of the channel ($-1 \leq x \leq 5; y = \pm 1$), representing walls moving from left to right with speed unity. A no-flow condition is imposed on the internal boundary ($0 \leq x \leq 5; y = 0$), and a Neumann condition is applied at the outflow boundary ($x = 5; -1 < y < 1$).

NS5 This is another classical problem. The domain is $\Omega_{\boxdot}$ and is associated with modelling flow in a rectangular channel with a square cylindrical obstruction. A Poiseuille profile is imposed on the inflow boundary ($x = 0; -1 \leq y \leq 1$), and a no-flow (zero velocity) condition is imposed on the obstruction and on the top and bottom walls. A Neumann condition is applied at the outflow boundary which automatically sets the mean outflow pressure to zero.

Appendix B

Time-dependent problems

In this appendix we describe the eleven unsteady test problems that are currently implemented in IFISS. Each problem can be generated by running the appropriate driver routine (as identified below) and is based on one of the following four physical domains:

$\Omega_{\square}$: the square $(-1,1)\times(-1,1)$;

$\Omega_{\boxtimes}$: the rectangle $(0,L)\times(0,H)$;

$\Omega_{\sqsupset}$: the L-shaped region generated by taking the complement in $(-1,L)\times(-1,1)$ of the quadrant $(-1,0]\times(-1,0]$;

$\Omega_{\boxdot}$: a disconnected rectangular region $(0,8)\times(-1,1)$ generated by deleting the square $(7/4,9/4)\times(-1/4,1/4)$.

In all cases a slow "start-up" from zero to a steady-state Dirichlet temperature/horizontal velocity u_∞ is achieved via the time-dependent boundary condition

$$u_*(t) = u_\infty(1-e^{-10t}).$$

The heat equation: `heat_testproblem`

H1 The heat equation is solved on the rectangular domain $\Omega_{\boxtimes}$ with $L=1$ and with the vertical edges heated/cooled to values of $\pm 1/2$. Either a zero Dirichlet or a zero Neumann condition can be specified on the horizontal edges. In the latter case the solution tends to a steady state with a linear temperature variation in the x-direction.

H2 The heat equation is solved on the L-shaped domain $\Omega_{\sqsupset}$ with $L=1$. A zero Dirichlet condition is imposed at all points on the boundary except for the vertical edge $x=-1$ and the point value $(1,0)$ which are forced to a constant temperature of unity.

H3 The heat equation is solved on the backward-facing step domain $\Omega_{\sqsupset}$. A Dirichlet condition is imposed on the top and bottom walls to give a vertical temperature differential (the bottom edge is hot and the top is cold). A zero Neumann condition is applied on the left and right vertical boundaries.

The convection-diffusion equation: `unsteady_cd_testproblem`

These reference problems are posed on the square domain $\Omega_{\square}$ and the convective wind is independent of time.

T–CD2 This is the unsteady analogue of **CD2**. The convective field $\vec{w} = (0, 1 + (x+1)^2/4)$, so the wind is vertical but increases in strength from left to right across the domain. The temperature u is forced to unity on the inflow boundary but decreases to zero quadratically on the right wall and cubically on the left wall. On the outflow (top) boundary, either a Dirichlet or Neumann condition can be applied (both homogeneous). The wind pushes a hot 'wave' through the domain which exits in 2–3 time units. The steady-state solution can be compared with that given by solving **CD2**.

T–CD4 This is the unsteady analogue of **CD4**. This is a model for the development of the temperature distribution in a cavity with a 'hot' external wall. The wind $\vec{w} = (2y(1-x^2),\ -2x(1-y^2))$ determines a recirculating flow. The Dirichlet boundary conditions imposed have value one on the right-hand (hot) wall and zero everywhere else. There are therefore discontinuities at the two corners of the hot wall, $x = 1$, $y = \pm 1$, which lead to boundary layers near these corners. The steady-state solution can be compared with that given by solving **CD4**.

Navier-Stokes equations: `unsteady_navier_testproblem`

T–NS2 This is the unsteady analogue of **NS2**. This example models the development of flow over a step of length L (the domain is $\Omega_{\sqsupset}$ with L chosen by the user). The boundary conditions ultimately tend to those in **NS2**. If the viscosity parameter is sufficiently large ($\nu \geq 1/600$) and L is sufficiently long $L \sim 40$ the flow solution tends to a steady state. In this case the steady-state solution can be compared with that given by solving **NS2**.
For high Reynolds number flow, the steady state is not stable and the flow is forever unsteady.

T–NS3 This is the unsteady analogue of **NS3**. This problem models "spin-up" flow in a cavity $\Omega_{\square}$. The boundary conditions are the same as in **S3**, with the choice of a leaky, watertight or regularised lid boundary condition. If the viscosity parameter is sufficiently large ($\nu \geq 1/1500$) the flow solution ultimately tends to a steady state.

T–NS5 This is the unsteady analogue of **NS5**. The critical value of the viscosity parameter (associated a bifurcation from a symmetric steady flow to unsymmetric vortex shedding) is relatively large in this case. For example, a periodic shedding solution can be computed for $\nu = 1/400$ using $\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$ approximation.

Boussinesq flow equations: `unsteady_bouss_testproblem`

B–NS0 This problem models buoyancy driven flow in a rectangular cavity with a vertical temperature profile. Steady Rayleigh–Bénard convection rolls can be computed if the cavity is long and thin (e.g., has aspect ratio 8:1) and the Rayleigh and Prandtl numbers are set appropriately. An specific example is given in the batchfile `B-NS42_batch.m`.

B–NS1 This problem models buoyancy driven flow in a rectangular cavity with a horizontal temperature profile (the temperature boundary conditions problem are as in **H1**). Time periodic solutions can be computed both for a horizontal orientation (aspect ratio 4:1) and for a vertical orientation (aspect ratio 1:8) if the Rayleigh and Prandtl numbers are set appropriately. Examples are given in the batchfiles `B-NS41_batch.m` and `B-NS43_batch.m` respectively.

B–NS2 This is the thermally driven analogue of **T-NS2**. This example models the development of flow over a step of length L (the domain is $\Omega_{\sqsubset}$ with L chosen by the user). The temperature boundary conditions are as in **H3**. An example which gives slow evolution to a steady state is provided in the batchfile `B-NS2_batch.m`.