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

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

Version 3.1 | released 25 January 2011

[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. d.silvester@manchester.ac.uk.

[3]Department of Computer Science, University of Maryland, College Park, Maryland, USA. elman@cs.umd.edu.

[4]Department of Mathematics and Statistics, University of Strathclyde, Glasgow, UK. a.ramage@strath.ac.uk.

Contents

1.1 Background

This is an overview of the IFISS software library that is associated with the book [ESW2005]

Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics
Howard Elman, David Silvester and Andy Wathen
Oxford University Press, 2005.[1]

The IFISS software library is "open-source" and is written in MATLAB.[2] It can also be installed and run using the freely available Octave package, see `http://www.gnu.org/software/octave/`.

IFISS can be downloaded from either of the following sites:

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

 The current version of the software: IFISS 3.1 has been tested using Windows, Unix and Mac OS X. Previous releases were developed using MATLAB Versions 5.3 to 7.9. (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 7.11 (R2010b) and is fully compatible with Octave 3.2.4.

We are happy to receive feedback, especially if it is positive. If you find any bugs please send an email to `a.ramage@strath.ac.uk`.

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>/ifiss3.1')`, or `cd('<local directory>\ifiss3.1')`,

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

[1] For further details see `http://ukcatalogue.oup.com/product/9780198528685.do` or `http://www.oup.com/us/catalog/general/subject/Mathematics/AppliedMathematics/?view=usa&ci=019852868X`

[2]

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 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 `ifiss, helpme` to get started.

1.2 Steady-state problems

The IFISS package focuses on four specific partial differential equation problems (PDEs): these are the Poisson equation, the convection-diffusion equation, and the Stokes and Navier-Stokes equations. Investigation of these is facilitated by a suite of drivers that set up and solve four reference test problems for each PDE. These test problems have been selected to illustrate characteristics of typical solutions and to allow users to experiment with various aspects of the modelling and solution processes. A full description of the reference problems can be found in [ESW2005]. For convenience, a short summary of this information is given in the first Appendix.

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 7 of [ESW2005].

```
>> setpath
>> ifiss
This is IFISS version 3.1: finally released on 25 January 2011
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 over an obstacle
 : 2
horizontal dimensions [-1,L]: L? (default L=5) :

Grid generation for a step shaped domain.
grid parameter: 3 for underlying 8x4*(L+1) grid (default is 4) : 5
Q1-Q1/Q1-P0/Q2-Q1/Q2-P1: 1/2/3/4? (default Q1-P0) :
setting up Q1-P0 matrices...  done
system matrices saved in step_stokes_nobc.mat ...
Incompressible flow problem on step domain ...
viscosity parameter (default 1/50) :
```

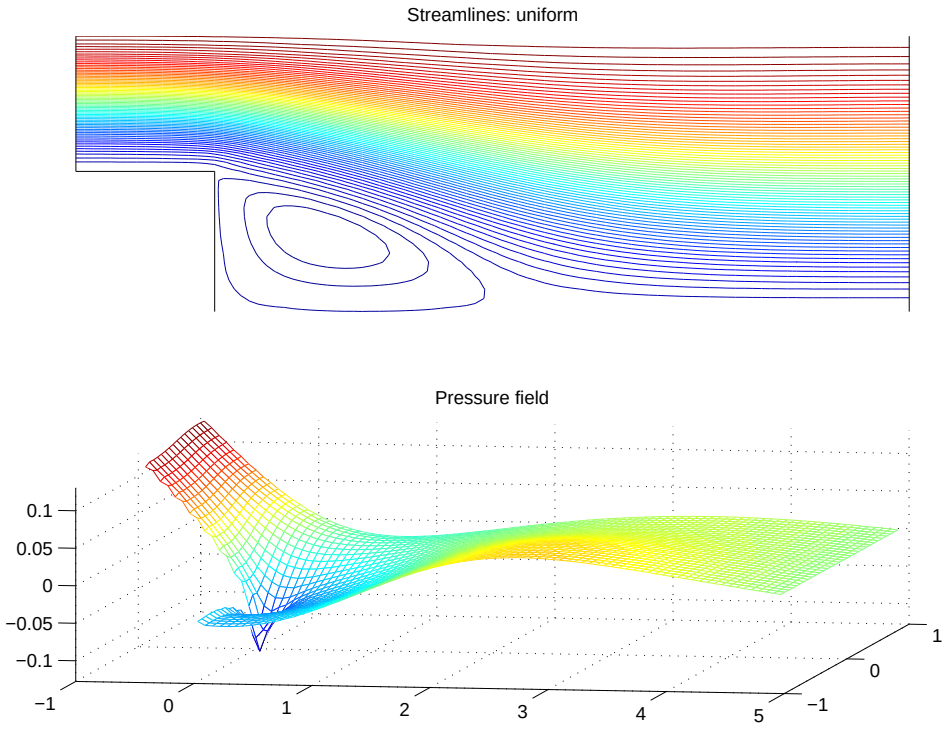
```
Picard/Newton/hybrid linearization 1/2/3 (default hybrid) :
number of Picard iterations (default 2) :
number of Newton iterations (default 4) :
nonlinear tolerance (default 1.e-8) :
stokes system ...
Stokes stabilization parameter (default is 1/4) :
setting up Q1 convection matrix...  done.
uniform/exponential streamlines 1/2 (default uniform) :
number of contour lines (default 50) : 75
computing Q1-P0 element stress flux jumps...  done
computing local error estimator... done.
estimated velocity error (in energy):  (3.158849e-01,1.655539e-01)
computing divergence of discrete velocity solution ...  done
estimated velocity divergence error:  5.554454e-03
plotting element data... done

initial nonlinear residual is 4.108490e+00
Stokes solution residual is 8.538847e-01

Picard iteration number 1
setting up Q1 convection matrix...  done.
nonlinear residual is 1.093384e-02
   velocity change is 4.345363e+00

Picard iteration number 2
setting up Q1 convection matrix...  done.
nonlinear residual is 4.003099e-03
   velocity change is 2.073700e+00

Newton iteration number 1
setting up Q1 Newton Jacobian matrices...  done.
setting up Q1 convection matrix...  done.
nonlinear residual is 4.397459e-04
   velocity change is 1.528600e+00

Newton iteration number 2
setting up Q1 Newton Jacobian matrices...  done.
setting up Q1 convection matrix...  done.
nonlinear residual is 9.297262e-07
   velocity change is 7.807388e-02

Newton iteration number 3
setting up Q1 Newton Jacobian matrices...  done.
setting up Q1 convection matrix...  done.
nonlinear residual is 1.855600e-11
   velocity change is 2.985901e-04
```

```
finished, nonlinear convergence test satisfied

computing Q1-P0 element stress flux jumps...  done
computing oseen local error estimator... done.
estimated velocity error (in energy):  (3.169732e-01,1.032707e-01)
computing divergence of discrete velocity solution ...  done
estimated velocity divergence error:  4.506274e-03
estimated overall error is 3.334024e-01
plotting 2x2 element data... done
```

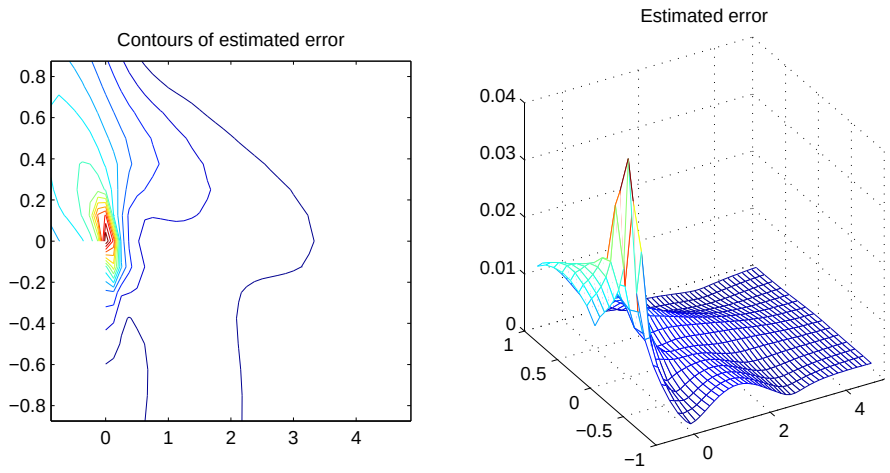
The computed solution and the estimated error are shown in Figure 1.1 (a) and (b).

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 62 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 Q1 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
   4  modified pressure convection-diffusion (Fp*)
default is modified pressure convection-diffusion : 2
ideal / AMG iterated  preconditioning? 1/2 (default ideal) :
setting up Q0 pressure preconditioning matrices...
NonUniform grids are fine.
fixed pressure on inflow boundary
ideal pressure convection-diffusion preconditioning ...

GMRES iteration ...
convergence in  62 iterations

    k  log10(||r_k||/||r_0||)
    0           0.0000
    1          -0.0115
    2          -0.0156
    3          -0.0171
```



(a) Solution to problem **NS2** with stabilised $\mathbf{Q}_1$–$\mathbf{P}_0$ approximation.



(b) Estimated error in the computed solution.

Figure 1.1: Sample output from `navier_testproblem`.

```
    .
    .
    .
   61          -5.8082
   62          -6.0196
Bingo!

  6.1107e+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) :
```

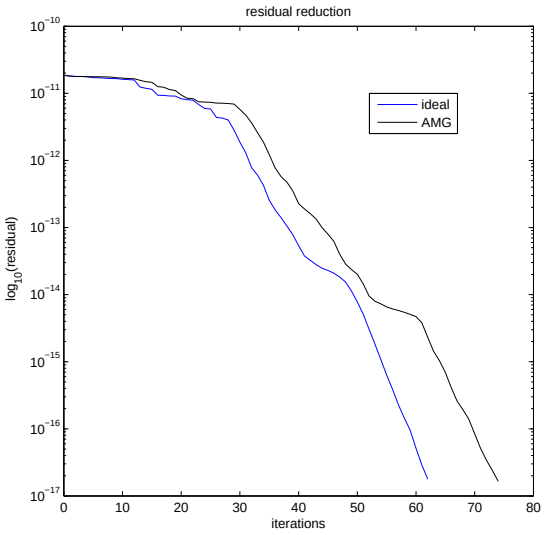


Figure 1.2: Sample output from `it_solve`.

To produce the second (black) curve in Figure 1.2, `it_solve` must be rerun using pressure convection-diffusion preconditioning, this time replacing the sparse direct solves in the preconditioning step with an AMG V-cycle.

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

solving Jacobian system generated by solution from last Newton step

setting up Q1 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
   4  modified pressure convection-diffusion (Fp*)
default is modified pressure convection-diffusion : 2
ideal / AMG iterated  preconditioning? 1/2 (default ideal) : 2
setting up Q0 pressure preconditioning matrices...
NonUniform grids are fine.
fixed pressure on inflow boundary
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 ...  9 grid levels constructed.

Pressure Poisson setup done.
ILU smoothing ..

GMRES iteration ...
convergence in  74 iterations

    k  log10(||r_k||/||r_0||)
    0           0.0000
    1          -0.0098
    .
    .
    .
   73          -5.8880
   74          -6.0523
Bingo!

  1.2567e+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
>> legend('ideal','AMG')
```

In this case GMRES converges in 74 iterations—but the CPU time for solution is reduced by a factor of five!

The IFISS package includes an implementation of the modified pressure convection-diffusion preconditioner that was proposed by Elman & Tuminaro[3]. This strategy is described in the online

[3]Available online from `http://hdl.handle.net/1903/8940`

publication ETNA 35:257–280, 2009 [ET2008], and is particularly effective in the case of non-enclosed flow.

A new feature of the IFISS 3.1 package is that it includes an implementation of the `est_minres` optimal Stokes solver developed by Silvester & Simoncini.[4] A run that illustrates this feature is reproduced below.

```
>> stokes_testproblem

specification of reference Stokes problem.

choose specific example (default is cavity)
     1  Channel domain
     2  Flow over a backward facing step
     3  Lid driven cavity
     4  Colliding flow
 : 3
cavity type leaky/tight/regularised 1/2/3 (regularised) :

Grid generation for cavity domain.
grid parameter: 3 for underlying 8x8 grid (default is 16x16) : 6
uniform/stretched grid (1/2) (default is uniform) : 2
computed stretch ratio is     1.0977
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 square_stokes_nobc.mat ...
imposing boundary conditions and solving system ...
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 3.630023e-18.

This should not cause difficulty for enclosed flow problems.

Stokes system solved in 6.643e-01 seconds
uniform/exponential streamlines 1/2 (default uniform) : 2

Enclosed flow ..
FAST Stokes Q2-P1 a posteriori error estimation
checking edge numbering and computing mesh lengths ...
Q2-P1 local error estimator ...
interior residual RHS assembly took 5.3161e-01 seconds
flux jump RHS assembly took 8.1748e-02 seconds
LDLT factorization took 1.5545e-02 seconds
triangular solves took 2.1097e-02 seconds
computing divergence of discrete velocity solution ...  done
estimated velocity divergence error:  1.386888e-04
estimated energy error is 2.8616e-02
```

[4] Available online from `http://eprints.ma.man.ac.uk/1450/` .

```
>> itsolve_stokes
Inexact AMG block preconditioning ..
number of V-Cycles? (default 1) :
AMG grid coarsening ...  12 grid levels constructed.
AMG with point damped Gauss-Seidel smoothing ..
Enclosed flow ..
checking edge numbering and computing mesh lengths ...
Q2-P1 local error estimator ...
interior residual RHS assembly took 4.7206e-01 seconds
flux jump RHS assembly took 4.8778e-02 seconds
LDLT factorization took 1.6855e-02 seconds
triangular solves took 1.3280e-02 seconds
Call to EST_MINRES with built in error control ...

   k    Estimated-Error   Algebraic-Bound   Residual-Error
   1    6.7839e+00                          1.2057e+00
   2    3.5953e+00                          1.1510e+00
   3    2.6205e+00                          6.7249e-01
   4    5.2730e-01                          5.1638e-01
   5    7.9704e-01                          4.2154e-01
   6    6.1398e-01        1.1261e+00        2.1421e-01
   7    6.2805e-01        1.1260e+00        2.1419e-01
   8    3.1579e-01        7.0126e-01        1.2523e-01
   9    1.3678e-01        6.5200e-01        1.1587e-01
  10    2.3234e-01        4.9444e-01        8.6172e-02
  11    1.3283e-01        3.1041e-01        5.2587e-02
  12    1.5694e-01        3.0879e-01        5.2307e-02
  13    5.1152e-02        1.6155e-01        2.6595e-02
  14    3.4364e-02        1.3696e-01        2.2454e-02
  15    6.7931e-02        9.9896e-02        1.6223e-02
  16    3.5868e-02        6.4090e-02        1.0263e-02
  17    4.7005e-02        5.9944e-02        9.5826e-03
  18    2.9131e-02        2.9867e-02        4.6880e-03
  19    2.8869e-02        2.8548e-02        4.4777e-03  Bingo!

convergence in  19 iterations
  5.3300e+00 seconds

 Eigenvalue convergence
    k     infsup     lambda
    3        NaN        NaN
    4     0.4312     0.9543
    5     0.3630     0.9371
    6     0.3308     0.9183
    7     0.2690     0.8634
    8     0.2690     0.8633
```

```
    9      0.2525      0.8134
   10      0.2513      0.8035
   11      0.2465      0.7644
   12      0.2396      0.7034
   13      0.2396      0.7027
   14      0.2328      0.6448
   15      0.2319      0.6341
   16      0.2297      0.6217
   17      0.2265      0.6136
   18      0.2261      0.6128
   19      0.2220      0.6084

 Final estimated error is 2.8869e-02
 Optimality in  19 iterations

>> [np,nu]=size(Bst);
>> xdiff=norm(xest(1:nu)-xst(1:nu),inf);
>> fprintf('velocity solution difference is %7.3e\n',xdiff)
velocity solution difference is 9.306e-04
```

1.3 Unsteady problems

Version 3.0 of the IFISS package extended the functionality to PDE models that include time derivatives: the heat equation, the time-dependent convection-diffusion equation, and the Navier-Stokes equations modelling unsteady fluid flow. As in the steady state case, a selection of test problems have been selected which illustrate the characteristics of typical solutions and which allow users to experiment with fast solvers for the linear equation systems that are solved at every time step. A description of the built-in problems is given in the second Appendix.

A sample IFISS session for test problem **T**–**NS3** (see the Appendix), which models flow in a lid driven cavity, 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 system of Differential Algebraic equations is solved using the self-adaptive AB2-TR algorithm that is described in the MIMS Eprint[5] 2008.6 [KGGS2008].

```
>> unsteady_navier_testproblem

specification of reference unsteady Navier-Stokes problem.

choose specific example (default is step)
     2  Flow over a backward facing step
     3  Lid driven cavity
     5  Flow around a square obstruction
 : 3
```

[5]Available online from `http://eprints.ma.man.ac.uk/1110/`

```
cavity type leaky/tight/regularised 1/2/3 (regularised) :

Grid generation for cavity domain.
grid parameter: 3 for underlying 8x8 grid (default is 16x16) : 6
uniform/stretched grid (1/2) (default is uniform) : 2
computed stretch ratio is     1.0977
Q1-Q1/Q1-P0/Q2-Q1/Q2-P1: 1/2/3/4? (default Q1-P0) : 3
setting up Q2-Q1 matrices...  done
system matrices saved in square_stokes_nobc.mat ...
Unsteady flow in a square domain ...
viscosity parameter (default 1/200) : 1/500
Discrete Saddle-Point DAE system ...
target time? (default 100) :
accuracy tolerance? (default 1e-4) :
averaging frequency? (default 10) :
Solving DAE system using stabilized TR ...

AxBhandle =
    @defaultAxB

  initial nonlinear residual is 2.569596e-03
             boundary change is 5.188603e-08
setting up Q2 convection matrix...  done.

   step  timestep        time
     2   1.000e-09      2.000e-09
     3   5.010e-05      5.010e-05
     4   1.765e-03      1.815e-03
     5   3.046e-03      4.861e-03
     6   3.674e-03      8.535e-03
     7   5.203e-03      1.374e-02
     8   7.364e-03      2.110e-02
     9   9.355e-03      3.046e-02
    10   1.167e-02      4.213e-02 --- Averaging
    11   1.450e-02      5.079e-02
     .
     .
     .
   117   4.361e+00      7.957e+01
   118   4.626e+00      8.419e+01
   119   4.844e+00      8.904e+01
   120   4.947e+00      9.399e+01 --- Averaging
   121   5.212e+00      9.672e+01
   122   3.276e+00      1.000e+02
finished in  123 steps!
Integration took  8.711e+01 seconds
```

```
plotting timestep sequence ...
specify plot symbol without quotes, default "bo" :
use new (enter figno) or existing (0) figure, default is 0 : 9
123 timesteps

running flow field animation ...
   step   mean_vorticity
     1      1.600e-08
     2      1.600e-08
     3      3.200e-08
    .
    .
    .
   121      1.600e+00
   122      1.600e+00
   123      1.600e+00

All done
step 123 : time is 1.000e+02
minimum w is -20.9748 and maximum w is 24.7171
... to generate a movie run square_flowmovie
    see help square_flowmovie ...
square_flowmovie               - generates flow movie on square shaped domain
computing divergence of discrete velocity solution ...  done
estimated velocity divergence error:  1.657197e-03
```

The associated flow animation shows the computed vorticity and the total velocity at each time step. The solution tends to a steady state: this is illustrated in in Figure 1.3.

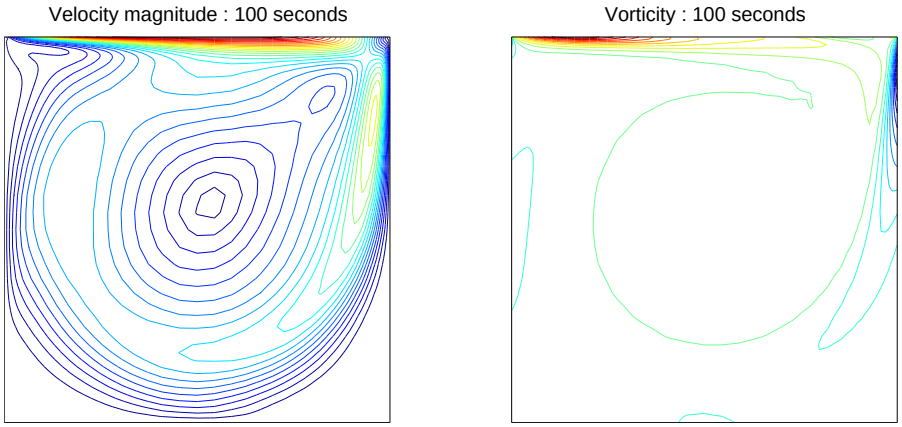


Figure 1.3: Sample output from `unsteady_navier_testproblem`.

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_solve`. Here, the chosen iterative method is GMRES with AMG least-squares commutator preconditioning. At the selected time step the preconditioned iteration converges in 20 iterations.

```
>> snapshot_solve
Iterative solution of a SNAPSHOT linear system
Solution data available for 100 seconds
Approximate time for the SNAPSHOT? (default is the end) : 10

Time step number 71
Constructing system at time 9.91164 seconds
    current timestep is 0.370012 seconds
enclosed flow (cavity) problem ...
stopping tolerance? (default 1e-8) :
maximum number of iterations? (default 100) :
preconditioner:
   0  none
   3  least-squares commutator
   9  modified pressure convection-diffusion (Fp*)
default is Fp* : 3
ideal / AMG iterated  preconditioning? 1/2 (default ideal) : 2
AMG grid coarsening ...  15 grid levels constructed.
AMG fine level smoothing strategy? PDJ/ILU 1/2  (default ILU) :
ILU smoothing on finest level..
AMG iterated LSC preconditioning ...
fixing singularity in pressure matrix... done
AMG grid coarsening ...  9 grid levels constructed.
BinvGB setup done.
ILU smoothing on finest level..

GMRES iteration ...
convergence in  20 iterations

    k  log10(||r_k||/||r_0||)
    0           0.0000
    1          -0.0091
    2          -0.2777
    3          -1.0799
    .
    .
    .
   19          -7.9153
   20          -8.4136
Bingo!

  7.0681e-01 seconds
```

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:

- `/timestepping/`

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 [ESW2005]. 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/`

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/`

The finite element approximations, preconditioners and iterative solution methods are described in detail in [ESW2005]. Indeed, all the numerical results described in the book have been computed (and can be reproduced) using IFISS. Additionally, the computational exercises at the end of each chapter are designed to be carried out using the IFISS software.

2.1.1 Help facility and IFISS function glossary

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 350 MATLAB functions and script files, of which the high level ones are listed below. 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>`.

IFISS3.1	
activemode	turns off batch processing for IFISS
batchmode	enables batch processing for IFISS testproblem
gohome	positions command prompt at top level directory
helpme	IFISS interactive help facility
ifiss	returns IFISS version number
ifisslogo	generates IFISS logo
recall	query the current discretized problem
setpath	sets IFISS search path

DIFFUSION	
diffpost	driver for a posteriori error postprocessing
diffpost_q1	local Poisson error estimator for Q_1 solution
diffpost_res	computes Q_1 element residual estimator
diffpost_q2_with_q4	local Poisson error estimator for Q_2 solution
element_lusolve	vectorized local backward-forward solves
ell_diff	solve Poisson problem in L-shaped domain
femq1_diff	vectorized bilinear coefficient matrix generator
femq2_diff	vectorized biquadratic coefficient matrix generator
gausspoints_oned	constructs one-dimensional Gauss Point Rule
gausspoints_twod	constructs tensor product Gauss Point Rule
helpme_diff	diffusion problem interactive help
nonzerobc	imposes Dirichlet boundary condition
q1fluxjmps	computes flux jumps for rectangular Q_1 grid
q1res_diff	computes interior residuals for rectangular Q_1 grid
quad_diff	solve Poisson problem in quadrilateral domain
specific_bc	(current) problem boundary condition
specific_rhs	(current) problem forcing function
square_diff	solve Poisson problem in unit square domain

GRIDS	
box_domain	rectangular cavity $\boldsymbol{Q}_2$ grid generator
cavity_domain	square cavity $\boldsymbol{Q}_2$ grid generator
ell_domain	L-shape domain $\boldsymbol{Q}_2$ grid generator
channel_domain	standard square shaped domain $\boldsymbol{Q}_2$ grid generator
edgegen	edge information for flux jump computation
eexgen	$\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$ reorientation for flux jump computation
forwardstep_domain	forward step domain $\boldsymbol{Q}_2$ grid generator
gridplot	quadrilateral grid verification
longstep_domain	extended L-shaped domain $\boldsymbol{Q}_2$ grid generator
macrogridplot	quadrilateral macroelement grid verification
obstacle_domain	obstacle domain $\boldsymbol{Q}_2$ grid generator
perturb_grid	perturbed bilinear element grid generator
plate_domain	slit shaped domain $\boldsymbol{Q}_2$ grid generator
q1grid	bilinear ($\boldsymbol{Q}_1$) element grid generator
q1p0grid	$\boldsymbol{Q}_1$–$\boldsymbol{P}_0$ element grid generator
q1q1grid	$\boldsymbol{Q}_1$–$\boldsymbol{Q}_1$ element grid generator
q2grid	biquadratic ($\boldsymbol{Q}_2$) element grid generator
q2p1grid	$\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$ element grid generator
q2q1gridx	$\boldsymbol{Q}_2$–$\boldsymbol{Q}_1$ element grid generator
quad_domain	quadrilateral domain $\boldsymbol{Q}_2$ grid generator
ref_domain	reference square domain $\boldsymbol{Q}_2$ grid generator
shish_grid	Shishkin grid generator on unit domain
square_domain	square domain $\boldsymbol{Q}_2$ grid generator
step_domain	standard L-shaped domain $\boldsymbol{Q}_2$ grid generator
subint	geometrically stretched subdivision generator

CONVECTION	
cdpost_bc	postprocesses local Poisson error estimator
cdpost_p	computes local Poisson error estimator for $\boldsymbol{Q}_1$ solution
femq1_cd	vectorized bilinear coefficient matrix generator
femq1_cd_supg	vectorized $\boldsymbol{Q}_1$ streamline diffusion matrix generator
helpme_cd	convection-diffusion problem interactive help
ref_cd	set up problem in reference square domain
solve_cd	solve convection-diffusion problem in square domain
specific_wind	(current) problem convective wind
square_cd	set up problem in unit square domain

STOKES_FLOW	
altinfsup	computes inf-sup eigenvalue distribution of potential flow
box_stokes	set up flow problem in rectangular domain
channel_stokes	setup inflow/outflow problem in square domain
flowbc	imposes inflow boundary condition
helpme_stokes	Stokes flow problem interactive help
infsup	computes inf-sup eigenvalue distribution
longstep_stokes	setup flow problem in extended step domain
obstacle_stokes	setup inflow/outflow problem in obstacle domain
plate_stokes	setup flow problem in slit domain
q1div	computes norm of divergence of $\boldsymbol{Q}_1$ flow solution
quad_stokes	setup Stokes problem in quadrilateral domain
solve_step_stokes	solve Stokes problem in step domain
solve_stokes	solve Stokes problem in square domain
specific_flow	(current) problem imposed boundary condition
square_stokes	setup flow problem in unit square domain
step_stokes	setup flow problem in standard step domain
stokes_q1p0	$\boldsymbol{Q}_1$–$\boldsymbol{P}_0$ matrix generator
stokes_q1q1	$\boldsymbol{Q}_1$–$\boldsymbol{Q}_1$ matrix generator
stokes_q2p1	$\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$ matrix generator
stokes_q2q1	$\boldsymbol{Q}_2$–$\boldsymbol{Q}_1$ matrix generator
stokespost	estimates Stokes error distribution
stokespost_q2p1	vectorized $\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$ local Poisson error estimator
stokespost_q1p0_p	computes Poisson error estimator for $\boldsymbol{Q}_1$–$\boldsymbol{P}_0$
stressjmps_q1p0	stress jumps for rectangular $\boldsymbol{Q}_1$–$\boldsymbol{P}_0$ grid
stressjmps_q2p1	vectorised stress jumps for $\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$ grid
vorticity_q1	computes $\boldsymbol{Q}_1$ total vorticity

STOKES_MINRES	
algpost_q2p1	computes $\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$ error estimate inside iteration
est_errorq2p1	computes energy error estimate for $\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$ solution
est_minres	MINRES with discretization error estimation
helpme_stopping	optimal iterative solvers interactive help
hydro	removes hydrostatic pressure mode from Stokes solution vector
itsolve_stokes	specialized iterative solution of Stokes problem
minres_esterror	MINRES with energy norm estimation
param_est	determines adaptive stopping criteria for EST_MINRES

NAVIER_FLOW	
Cpre_q1p0	stabilization for least squares commutator for $\boldsymbol{Q}_1$-$\boldsymbol{P}_0$
fpsetup_q0	$\boldsymbol{Q}_0$ pressure convection-diffusion matrix
fpsetup_q1	$\boldsymbol{Q}_1$ pressure convection-diffusion matrix
fpsetup_q2p1	$\boldsymbol{P}_{-1}$ pressure convection-diffusion matrix
fpzsetup_q0	modified $\boldsymbol{Q}_0$ pressure convection-diffusion matrix
fpzsetup_q1	modified $\boldsymbol{Q}_1$ pressure convection-diffusion matrix
fpzsetup_q2p1	modified $\boldsymbol{P}_{-1}$ pressure convection-diffusion matrix
helpme_navier	Navier-Stokes flow problem interactive help
navier_q1	$\boldsymbol{Q}_1$ convection matrix
navier_q2	$\boldsymbol{Q}_2$ convection matrix
navierpost_q1p0_bc	postprocesses Poisson error estimator
navierpost_q1p0_p	computes Poisson error estimator for $\boldsymbol{Q}_1$–$\boldsymbol{P}_0$
newton_q1	$\boldsymbol{Q}_1$ convection derivative matrices
newton_q2	$\boldsymbol{Q}_2$ convection derivative matrices
solve_navier	solve Navier-Stokes problem in square domain
solve_plate_navier	solve Navier-Stokes problem in slit domain
solve_obstacle_navier	solve Navier-Stokes problem in obstacle domain
solve_step_navier	solve Navier-Stokes problem in step domain
navierpost	estimates $\boldsymbol{Q}_2$–$\boldsymbol{P}_{-1}$ or $\boldsymbol{Q}_2$–$\boldsymbol{Q}_1$ NS error distribution

TIMESTEPPING	
box_heat	solves heat equation in rectangular cavity domain
colamdAxB	refined saddlepoint system solver
defaultAxB	standard MATLAB saddlepoint system solver
dtflowbc	imposes updated flow BC at the current timestep
dtheatbc	imposes updated Dirichlet BC at the current timestep
ell_heat	solves heat equation in L-shaped domain
helpme_heat	heat equation interactive help
helpme_timestepping	unsteady problem solution interactive help
helpme_unsteady_cd	unsteady convection-diffusion problem interactive help
helpme_unsteady_navier	unsteady flow problem interactive help
stabtr	standard integrator based on stabilized TR
stabtrNS	Navier-Stokes integrator based on stabilized TR
unsteady_cd	unsteady convection-diffusion in square-shaped domain
unsteady_navier	unsteady Navier-Stokes flow in square-shaped domain
unsteady_obstacle_navier	unsteady Navier-Stokes flow in obstructed domain
unsteady_step_navier	unsteady Navier-Stokes flow in step domain
unsteadyflowbc	imposes inflow boundary condition

GRAPHS	
box_heatplot	evolves temperature data on rectangular domain
dttplot	plots timestep data
ell_heatplot	evolves temperature data on L-shaped domain
eplot	plots element data on square-shaped domain
eplotl	plots element data on L-shaped domain
errplot	plots solution and error on square-shaped domain
errplotl	plots solution and error on L-shaped domain
flowplot09	plots flow data
flowvolume	plots/explores flow solution on vertical cross-section
logoplot	plots IFISS logo
mplot	plots 2x2 macroelement data on square-shaped domain
mplotl	plots 2x2 macroelement data on L-shaped domain
obstacle_flowmovie	generates movie of flow around square obstacle
obstacle_unsteadyflowplot	evolves flow data on obstructed channel domain
outflow	plots/explores tangential flow solution on outflow
solplot	plots nodal data on square-shaped domain
solplotl	plots nodal data on L-shaped domain
solsurf	plots solution surface on square-shaped domain
solsurfl	plots solution surface on L-shaped domain
square_flowmovie	generates flow movie on square-shaped domain
square_heatplot	evolves temperature data on square-shaped domain
square_unsteadyflowplot	evolves flow data on square-shaped domain
square_vorticityplot	plots vorticity data on square-shaped domain
step_flowmovie	generates flow movie on extended step-shaped domain
step_unsteadyflowplot	evolves flow data on extended step-shaped domain
step_vorticityplot	plots vorticity data on step domain
wwwplotl	plots solution and error estimate on L-shaped domain
wwwplotobs	plots streamlines and error estimate on general domain
xrefplot	comparison plot of iterative solver residuals
xsection	plots/explores solution on horizontal cross-section

SOLVERS	
a_cdt	matrix-vector product for convection-diffusion operator
a_nst	matrix-vector product for linearized Navier-Stokes operator
amg_grids_setup	performs algebraic multigrid setup phase
amg_smoother	performs AMG smoothing
amg_smoother_params	generates structure for AMG smoother parameters
amg_v_cycle	performs one AMG V-cycle
bicgstab_ell_r	BICGSTAB(ℓ) iteration with right preconditioning
bsolve	solves Galerkin system using backslash
cg_test	CG convergence demo
gmres_r	GMRES iteration with right preconditioning
helpme_it	iterative solvers interactive help
helpme_mg	geometric multigrid interactive help
idrs_r	Induced Dimension Reduction iteration
it_solve	driver for iterative solution of predefined problem
m_amgzt	AMG preconditioner for scalar operator
m_amgzz	AMG preconditioner with multiple v-cycles
m_bfbt	ideal least squares commutator preconditioner (unscaled)
m_diagt	action of diagonal preconditioning operator
m_fp	ideal pressure convection-diffusion preconditioner
m_fp_amgz	AMG iterated PCD preconditioner
m_fp_mg	GMG iterated PCD preconditioner
m_ilut	incomplete LU preconditioner
m_masscheb	mass matrix Chebyshev preconditioning operator
m_massdiag	mass matrix diagonal preconditioning operator
m_mg	GMG preconditioner for scalar problems
m_st_amgz	AMG block preconditioner for Stokes equations
m_st_block	block preconditioner for Stokes equations
m_st_mg	block MG preconditioner for Stokes equations
m_sxbfbt	ideal stabilized least squares commutator preconditioner
m_xbfbt	ideal least squares commutator preconditioner
m_xbfbt_amgz	AMG iterated LSC preconditioner
m_xbfbt_mg	GMG iterated LSC preconditioner
m_xfp	modified ideal PCD preconditioner
m_xfp_amgz	AMG iterated modified PCD preconditioner
mg_apsetup_q1	$\boldsymbol{Q}_1$ pressure diffusion matrix for GMG
mg_cd	GMG preconditioner for convection-diffusion problem
mg_diff	GMG preconditioner for diffusion problem
mg_iter	performs one GMG iteration
mg_ns	GMG preconditioner for Navier-Stokes equations
mg_post	postsmoothing for GMG
mg_pre	presmoothing for GMG
mg_prolong	GMG prolongation operator for square domain
mg_smooth	smoothers for GMG on square domain
mg_solve	driver for GMG solution of predefined problem
resplot	plot residuals computed by iterative solvers
snapshot_solve	driver for iterative solution of predefined unsteady problem

2.1.2 IFISS figures

As is evident from the sample runs described in the previous sections, 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

2.1.3 Running jobs in batchmode

IFISS also provides a `batchmode` facility via which data may 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); these 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.

2.2 Generating new domains or model problems

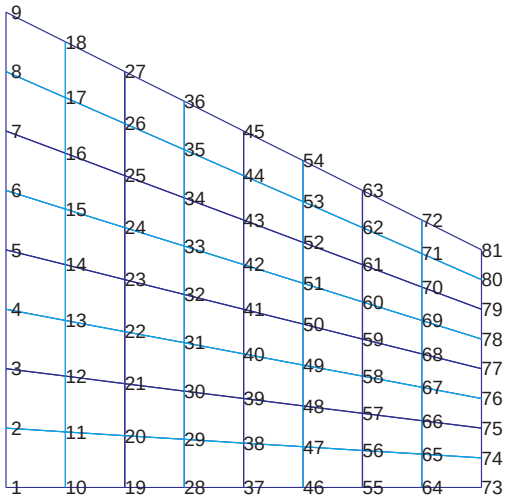
For each of the four PDEs treated, IFISS contains four built-in model problems. These are outlined in the Appendix. It is also straightforward to adapt the code to include different domains, PDE features or boundary conditions.

2.2.1 Introducing a new problem domain

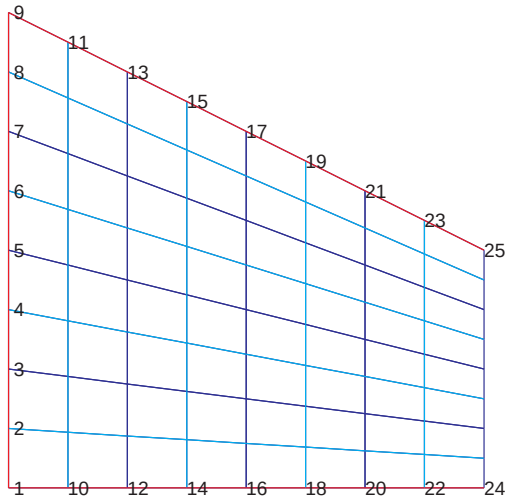
The first task in setting up a different PDE domain is the grid generation, which should be done by a function `<newproblem>_domain.m` analogous to those in the `/grids/` directory (see Section 2.1 for details of the directory structure). This function should generate grid information (specifically, 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.



(a) Indices of nodes of the macroelement grid.



(b) Indices of nodes on the Dirichlet boundary.

Figure 2.4: Sample output from `macrogridplot`.

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 2.4. 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`).

2.2.2 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 `/diffusion/specific_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`.

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 [ESW2005]. 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 $(-1,1)\times(-1,1)$;

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

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

$\Omega_{\square}$: 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 $\Omega_{\square}$. 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 $\Omega_{\sqsupset}$ 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 $\Omega_{\sqsupset}$ 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 $\Omega_{\square}$. 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 $\Omega_{\square}$ 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 seven unsteady test problems 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 $(0,L)\times(0,H)$;

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

$\Omega_{\square}$: 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 is $\Omega_{\boxtimes}$, 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 $\Omega_{\square}$. 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.

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.