

Incompressible Flow Iterative Solution Software (IFISS) Installation & Software Guide *

David J. Silvester[†] Howard C. Elman[‡] Alison Ramage[§]

Version 2.1, released 28 February 2006

Contents

1	Background	2
2	Installation	2
3	Model problems	2
4	Directory structure	7
5	Help facility and IFISS function glossary	7
6	IFISS figures	11
7	Running jobs in batchmode	11
A	Appendix: Description of model problems	12
A.1	The Poisson equation: <code>diff_testproblem</code>	12
A.2	The convection-diffusion equation: <code>cd_testproblem</code>	12
A.3	The Stokes equations: <code>stokes_testproblem</code>	13
A.4	Navier-Stokes equations: <code>navier_testproblem</code>	14

*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 U. K. Engineering and Physical Sciences Research Council under grant EP/C000528/1.

[†]School of Mathematics, University of Manchester, Manchester, UK. na.silvester@na-net.ornl.gov.

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

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

1 Background

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

Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics
H.C. Elman, D.J. Silvester and A.J. Wathen
Oxford University Press, 2005.¹

The IFISS software library is “open-source” and is written in `matlab`. It is freely available and can be downloaded from either of the 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. Older releases of IFISS were tested under Matlab Versions 5.3 to 6.5 and could be run under Windows, Unix and Mac architectures. The current version, IFISS 2.1, was developed using the syntax of `matlab` 6.5 (Release 13), but it has also been extensively tested using `matlab` 7.0 (Release 14). (If you are not sure which version of `matlab` you have running, just type `version` at the system prompt.)

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.

2 Installation

IFISS is downloaded as the `tar` file `ifiss.tar`, which contains all the individual files bundled together. After unpacking the `tar` file, two steps are required to set the package up:

1. The file `gohome.m` identifies the ‘home’ directory of the package via a command of the form

```
cd('<local directory information>/ifiss')
```

(where the name `ifiss` has the appropriate version number appended). This must be edited to point to the user’s local directory.

2. After `matlab` is invoked from the IFISS home directory, one of the commands `install_unix` or `install_pc` must be executed.² These commands initialise all Unix/Linux-dependent and PC-dependent files, respectively. After this has been done, IFISS is set to run in a Unix/Linux or Windows environment 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` is initiated either by typing `setpath` in response to the `matlab` prompt, or by incorporating the functionality of the `setpath` command into the user’s `matlab` startup file. Having run `setpath` at the `matlab` prompt, simply type `helpme` to get started.

3 Model problems

The IFISS package focuses on four specific PDEs. Easy investigation of approximation properties and behaviour of solvers for each of these is facilitated via a set of built-in drivers that set up and solve four reference test problems for each PDE. The test problems have been selected to illustrate the interesting features of examples of each type of equation, 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 the book. For

¹For further details see <http://www.oup.co.uk/isbn/0-19-852868-X> or <http://www.oup.com/us/catalog/general/subject/Mathematics/AppliedMathematics/?view=usa&ci=019852868X>

²In the downloaded package, the default files are set for running in a Unix or Linux environment.

convenience, a short summary of this information is given in the Appendix to this document.

As an example, we reproduce here a sample IFISS session for test problem **NS2** (see the Appendix), which models flow over a step. The driver `navier_testproblem` asks the user to choose the problem, grid and 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 the book.

```
>> navier_testproblem

specification of reference Navier-Stokes problem.

choose specific example (default is cavity)
  1 Poiseuille channel flow
  2 Flow over a backward facing step
  3 Lid driven cavity
  4 Flow over a plate
: 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.d-5) :
stokes system ...
Stokes stabilization parameter (default is 1/4) :
setting up Q1 convection matrix... done.
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

finished, nonlinear convergence test satisfied

computing Q1-P0 element stress flux jumps... done
computing Useen local error estimator... done.
estimated velocity error (in energy): (3.169739e-001,1.032718e-001)
computing divergence of discrete velocity solution ... done
estimated velocity divergence error: 4.506309e-003
estimated overall error is 3.334034e-001
plotting 2x2 element data... done

```

The computed solution and the estimated error are shown in Figures 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 the default choice, namely GMRES with ideal pressure convection-diffusion preconditioning. As shown below the method converges in 66 iterations, and the convergence curve is the blue line shown in Figure 1(c).

```

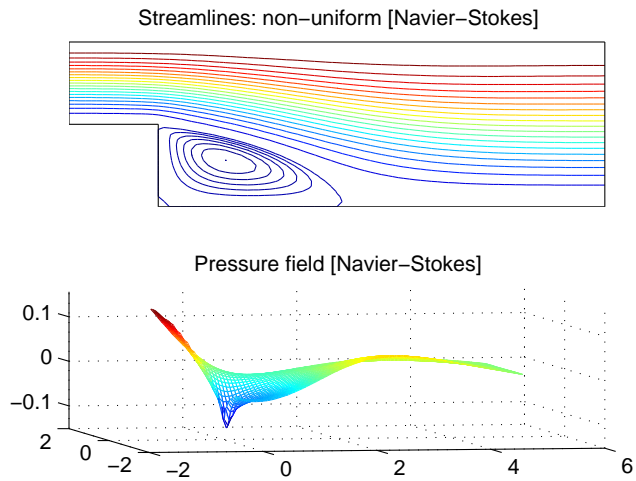
>> it_solve
inflow/outflow (step) problem ...
solving Jacobian system generated by solution from last Newton step
setting up Q1 Newton Jacobian matrices... done.

GMRES/Bicgstab(2) 1/2 (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
default is pressure convection-diffusion :
ideal / GMG iterated / AMG iterated preconditioning? 1/2/3 (default
ideal) :
setting up Q0 pressure preconditioning matrices...
uniform grid code: hx is 0.0625 and hy is 0.0625
fixed pressure on inflow boundary
ideal pressure convection-diffusion preconditioning ...
GMRES iteration ...
convergence in 66 iterations

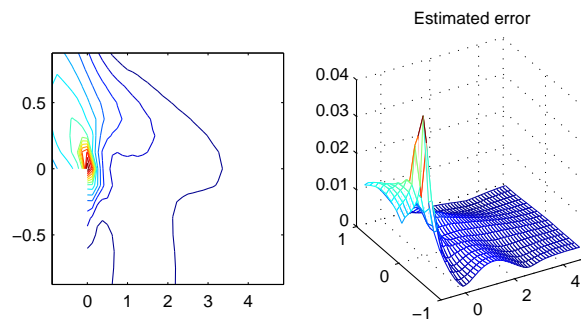
  k  log10(||r_k||/||r_0||)
  0      0.0000
  1     -0.0176
  2     -0.0222
    .
    .
    .
 64     -5.7428
 65     -5.9643
 66     -6.2042

```

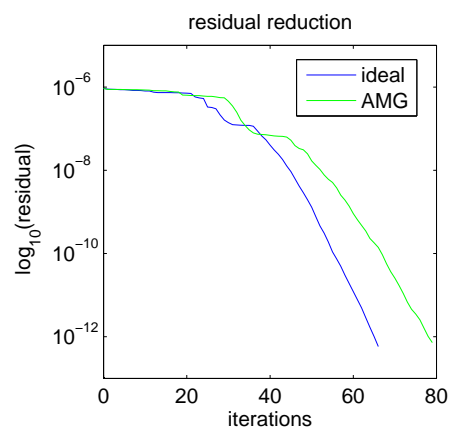
Bingo!



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



(b) Estimated error in the computed solution.



(c) Iteration counts for GMRES with pressure convection-diffusion preconditioning.

Figure 1: Sample output from `navier_testproblem` and `it_solve`.

2.3291e+01 seconds

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

To produce the second (green) curve in Figure 1(c), `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.³ In this case GMRES converges in 79 iterations (but the CPU time for solution is much faster!).

```
>> 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(2) 1/2 (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
default is pressure convection-diffusion :
ideal / GMG iterated / AMG iterated preconditioning? 1/2/3 (default
ideal) : 3
setting up Q0 pressure preconditioning matrices...
uniform grid code: hx is 0.0625 and hy is 0.0625
fixed pressure on inflow boundary
AMG iterated pressure convection-diffusion preconditioning ...
GMRES iteration ...
convergence in 79 iterations
```

k	log10(r_k / r_0)
0	0.0000
1	-0.0236
2	-0.0248
	.
	.
	.
77	-5.7663
78	-5.9591
79	-6.1085

Bingo!

5.5836e+00 seconds

use new (1) or existing (0) figure, default is 0 : 0
figure number (default is current active figure) : 1
colour (b,g,r,c,m,y,k): enter 1--7 (default 1) : 2

An important feature of IFISS is that it is straightforward user to adapt the code to use different domains, PDE features or boundary conditions. The first of these can be done by creating a file `<new problem>.domain` analogous to those in the `\ifiss\grids\` subdirectory, to specify a list of x and y coordinates for the desired quadrilateral mesh (see the next section for details on directory structure). The information on PDE attributes and boundary conditions is held in two specific `m`-files in the directory associated with each PDE. For example, for Navier-Stokes problems like the one above, these are files `specific_flow.m` and `specific_bc.m` in the directory `\ifiss\stokes_flow\`, which specify the velocity boundary conditions and

³The IFISS code provides an interface to the algebraic multigrid solver of the commercial finite element code FEMLAB, which is available from <http://www.comsol.com>.

stream-function boundary conditions, respectively. All that is needed to define a new problem is to edit these two `m`-files without changing any other IFISS files.

4 Directory structure

IFISS comprises functions which generate finite element approximations of the following Partial Differential Equations (PDEs) 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\`

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 the book. The functions associated with the domain geometry and grid generation are independent of the PDE being solved. These functions are thus located in a separate directory:

- `\ifiss\grids\`

The computed solutions are visualized using the 3-D plotting functions that are built into `matlab`. The functions that call these built in functions to 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 the following directory:

- `\ifiss\solvers\`

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 (`.ps`) files : directory `\ifiss\plotfiles\`

The finite element approximations, preconditioners and iterative solution methods are described in detail in the book. 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.

5 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. In `matlab` version 7, the function names are ‘clickable’ to give additional information.

The IFISS package consists of over 290 `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>`.

IFISS2.1	
gohome	positions command prompt at top level directory
helpme	IFISS interactive help facility
ifiss	returns IFISS version number
install_pc	sets up IFISS on non-UNIX computer
install_unix	sets up IFISS on UNIX computer
readme	distribution information file
setpath	sets IFISS search path
activemode	turns off batch processing for IFISS
batchmode	enables batch processing for IFISS testproblem

GRIDS	
cavity_domain	square cavity Q_2 grid generator
ell_domain	L-shape domain Q_2 grid generator
longstep_domain	extended L-shaped domain Q_2 grid generator
pipe_domain	standard square shaped domain Q_2 grid generator
plate_domain	slit shaped domain Q_2 grid generator
q1grid	bilinear (Q_1) element grid generator
q1p0grid	Q_1-P_0 element grid generator
q1q1grid	Q_1-Q_1 element grid generator
q2grid	biquadratic (Q_2) element grid generator
q2p1grid	Q_2-P_{-1} element grid generator
q2q1grid	Q_2-Q_1 element grid generator
quad_domain	quadrilateral domain Q_2 grid generator
ref_domain	reference square domain Q_2 grid generator
shish_grid	Shishkin grid generator on unit domain
square_domain	square domain Q_2 grid generator
step_domain	standard L-shaped domain Q_2 grid generator
subint	geometrically stretched subdivision generator

DIFFUSION	
diffpost_bc	postprocesses local Poisson error estimator
diffpost_p	computes local Poisson error estimator for Q_1 solution
ell_diff	solve Poisson problem in L-shaped domain
femq1_diff	vectorized bilinear coefficient matrix generator
femq2_diff	vectorized biquadratic coefficient matrix generator
helpme_diff	diffusion problem interactive help
nonzerobc	imposes Dirichlet boundary condition
q1fluxjumps	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

CONVECTION	
cdpost_bc	postprocesses local Poisson error estimator
cdpost_p	computes local Poisson error estimator for Q_1 solution
femq1_cd	vectorized bilinear coefficient matrix generator
femq1_cd_supg	vectorized 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	
helpme_stokes	Stokes flow problem interactive help
infsup	computes inf-sup eigenvalue distribution
longstep_stokes	setup flow problem in extended step domain
pipe_stokes	setup inflow/outflow problem in square domain
plate_stokes	setup flow problem in slit domain
q1div	computes norm of divergence of Q_1 flow solution
q2div	computes norm of divergence of Q_2 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	vectorized Q_1-P_0 matrix generator
stokes_q1q1	vectorized Q_1-Q_1 matrix generator
stokes_q2p1	vectorized Q_2-P_{-1} matrix generator
stokes_q2q1	vectorized Q_2-Q_1 matrix generator
stokespost_q1p0_bc	postprocesses Poisson error estimator
stokespost_q1p0_p	computes Poisson error estimator for Q_1-P_0
stressjumps_q1p0	stress jumps for rectangular Q_1-P_0 grid

NAVIER_FLOW	
fpsetup_q0	Q_0 pressure convection-diffusion matrix
fpsetup_q1	Q_1 pressure convection-diffusion matrix
fpsetup_q2p1	P_{-1} pressure convection-diffusion matrix
helpme_navier	Navier-Stokes flow problem interactive help
navier_q1	Q_1 convection matrix
navier_q2	Q_2 convection matrix
navierpost_q1p0_bc	postprocesses Poisson error estimator
navierpost_q1p0_p	computes Poisson error estimator for Q_1-P_0
newton_q1	Q_1 convection derivative matrices
newton_q2	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_step_navier	solve Navier-Stokes problem in step domain
Cpre_q1p0	generate stabilizations for least sqrs commutator for Q_1-P_0

SOLVERS	
a_cdt	matrix-vector product for convection-diffusion operator
a_nst	matrix-vector product for linearized Navier-Stokes operator
bicgstab_ell_r	BICGSTAB(ℓ) iteration with right preconditioning
cg_test	CG convergence demo
gmres_r	GMRES iteration with right preconditioning
helpme_it	iterative solvers interactive help
helpme_mg	geometric multigrid interactive help
it_solve	driver for iterative solution of predefined problem
m_amgt	AMG preconditioner for convection-diffusion operator
m_bfbt	ideal least squares commutator preconditioner (unscaled)
m_diagt	action of diagonal preconditioning operator
m_fp	ideal pressure convection-diffusion preconditioner
m_fp_amg	AMG iterated pressure convection-diffusion preconditioner
m_fp_mg	GMG iterated pressure convection-diffusion preconditioner
m_ilut	incomplete LU preconditioner
m_mg	GMG preconditioner for scalar problems
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_amg	AMG iterated least squares commutator preconditioner
m_xbfbt_mg	GMG iterated least squares commutator preconditioner
mg_apsetup_q1	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

GRAPHS	
bookplot	saves IFISS figure as postscript file
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
flowplot	plots flow data on standard square-shaped domain
flowplotl	plots flow data on standard step-shaped domain
flowplotp	plots flow data on slit-shaped domain
flowplotz	plots flow data on extended step-shaped domain
flowvolume	plots/explores flow solution on vertical cross-section
htmlplot	saves IFISS figure as html file
mplot	plots 2x2 macroelement data on square-shaped domain
mplotl	plots 2x2 macroelement data on L-shaped domain
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
xsection	plots/explores solution on horizontal cross-section

6 IFISS figures

As is evident from the sample run 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 (Q_1)
12	diffusion problem solution (Q_2)
20	convective wind
22	convection-diffusion solution & error plot (Q_1)
30	grid plot (Stokes or Navier-Stokes flow problem)
33	Stokes flow solution plot
34	Stokes flow solution error plot (Q_1-P_0 , Q_1-Q_1)
66	Navier-Stokes flow solution plot
67	Navier-Stokes flow solution error plot (Q_1-P_0)

7 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 `itsolve` 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. This file is used via the command

batchmode('itsolve')

It 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.

A Appendix: Description of model problems

In this appendix we give a brief outline of the sixteen test problems currently implemented in IFISS (four for each featured PDE). A fuller description of these reference problems and the underlying PDEs can be found in the book. Each problem can be generated by running the appropriate driver routine (as identified below) and is based on one of the following three physical domains:

- Ω_{\square} : the square $(-1, 1) \times (-1, 1)$;
- $\Omega_{\mathcal{P}}$: the L-shaped region generated by taking the complement in $(-1, L) \times (-1, 1)$ of the quadrant $(-1, 0] \times (-1, 0]$;
- Ω_{\square} : the rectangular region $(-1, 5) \times (-1, 1)$, with a slit along the line where $0 \leq x \leq 5$ and $y = 0$.

A.1 The Poisson equation: `diff_testproblem`

- **P1**: The domain is Ω_{\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 L-shaped. In this case, the underlying solution to the Poisson problem has a singularity at the origin.
- **P3**: The domain is Ω_{\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 **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.

A.2 The convection-diffusion equation: `cd_testproblem`

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

- **CD1**: For constant convective velocity vector $\mathbf{w} = (0, 1)$, 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 $\mathbf{w} = (0, 1 + (x + 1)^2/4)$, 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, $\mathbf{w} = (-\sin \frac{\pi}{6}, \cos \frac{\pi}{6})$, 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 $\mathbf{w} = (2y(1 - x^2), -2x(1 - y^2))$ 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.

A.3 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 Ω_{\square} using the velocity $\mathbf{u} = (1 - y^2, 0)$ to define a Dirichlet condition on the inflow boundary $x = -1$. The no-flow Dirichlet condition $\mathbf{u} = \vec{0}$ 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 $\Omega_{\mathbb{P}}$ with $L = 5$. A Poiseuille flow profile is imposed on the inflow boundary ($x = -1; 0 \leq y \leq 1$), 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 Ω_{\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:

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

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

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

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

A.4 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 $\mathbf{u} = (1 - y^2, 0)$, $p = -2\nu x$ is also an analytic solution of the Navier-Stokes equations, since the convection term $\mathbf{u} \cdot \nabla \mathbf{u}$ 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 $\Omega_{\mathbb{P}}$ 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 **S2**, where $L = 5$ is sufficient). The boundary conditions are identical to **S2**.
- **NS3:** This problem again models flow in a cavity $\Omega_{\mathbb{Q}}$. The boundary conditions are the same as in **S3**, with the choice of a leaky, watertight or regularised lid boundary condition.
- **NS4:** This model is a two-dimensional version of boundary layer flow over a flat plate, or *Blasius flow*. The problem is equivalent to that of computing the steady flow over a flat plate moving at a constant speed through a fluid that is at rest. The ‘parallel flow’ Dirichlet condition $\mathbf{u} = (1, 0)$ 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 ($x = 5; -1 < y < 1$).