RUF 1.0 User Manual

Philip T. Keenan

August 1994

TR94-30
RUF 1.0 User Manual

Philip T. Keenan

May 26, 1994

1This research was supported in part by the Department of Energy, the State of Texas Governor’s Energy Office, and project grants from the National Science Foundation. The author was also supported in part by an NSF Postdoctoral Fellowship.

2Department of Computational and Applied Mathematics, Rice University, P.O. Box 1892, Houston, TX 77251-1892.
Contents

1 Introduction ........................................... 1
   1.1 Disclaimer ......................................... 2
   1.2 Supported Platforms ............................... 2

2 The Single Phase Flow Equation ....................... 3

3 Input Commands .......................................... 4
   3.1 Mesh Description Commands ....................... 5
   3.2 Solve and Display Commands ....................... 6
   3.3 Keyword Options ................................... 8

4 Some Mathematical Details ............................. 13

5 Running the Programs .................................. 15
   5.1 Command Line Arguments ........................... 15
   5.2 Sample Input Files ................................ 16

1 Introduction

This is the user manual for version 1.0 of the Rice Unstructured Flow code (RUF).

RUF models the flow of fluid through a porous medium. It can be applied to the
study of both petroleum reservoirs and groundwater aquifers. RUF 1.0 models single
phase flow; future versions are planned which will handle multiphase, multicompo-
nent flow and transport. Reservoirs and aquifers are modeled geometrically as a grid
or mesh consisting of polygonal or polyhedral elements. Unlike some models which
require rectangular or brick elements arranged in a two or three dimensional lattice,
RUF allows using unstructured meshes in which elements of a variety of shapes may
be combined without restriction. In particular, it supports tetrahedral elements in
three dimensions, as well as hexahedra and bricks, and triangular elements in two
dimensions, as well as quadrilaterals and rectangles.

RUF has a number of other features. It handles general boundary conditions.
It allows general permeability tensors, not just diagonal ones. It can both refine
and smooth the meshes it builds. It can set up test problems with known analytic
solutions from a wide range of polynomial and non-polynomial reference solutions.
It can output text or graphics files describing the mesh, the solution, the gradient
and the flux; and when using a known reference solution it can also compute and
plot errors and their norms. In two dimensions it allows wells to be represented
as point sources (delta-functions); in two and three dimensions one can use the
general boundary conditions to model a variety of wells. RUF also includes a flux
postprocessor for triangular elements, which increases the accuracy of the computed fluxes on unstructured meshes, as described in [4] and [2].

RUF uses a very simple mesh geometry input language. While simple meshes can be constructed by hand, RUF is intended to be used in conjunction with a commercial mesh generation package. Any package will do, as long as its output mesh description can be translated into RUF's input format.

1.1 Disclaimer

RUF is a research tool, not a commercial product. It is made available to other researchers subject to the following restrictions and disclaimers.

Copyright (C) Philip Thomas Keenan and the Subsurface Modeling Group at Rice University, 1994. All rights reserved. No portion of this document or program may be reproduced, transmitted or otherwise copied without prior written permission of the copyright holder. The author makes no representations or warranties about the correctness of any program code or documentation in this or any other document or program file, nor about the correctness of the executable program or its suitability for any purpose, and is in no way liable for any damages resulting from its use or misuse. Any program source code made available is for non-commercial use only, and is subject to the same restrictions on copying and modification. Modified versions must cite the original author and include a disclaimer stating the modified nature of the code.

1.2 Supported Platforms

RUF is written entirely in C++ and builds on the Keenan C++ Foundation Class Library, which is subject to the same terms and conditions as RUF. In the experience of the author, C++ currently provides a powerful and efficient mechanism for writing highly complex software for scientific computation. The resulting code is very portable and has been run on a variety of machines including Sun Sparc workstations, IBM RS6000 workstations, the Intel Hyercube and the Intel Paragon. The last two machines mentioned are distributed memory parallel supercomputers; the present version of RUF runs on a single processor as a sequential program. An initial parallel version (pRUF 1.0) is currently under development and is presently targeted for release in Summer 1994.

RUF does not require a native C++ compiler for the target machine: after using the standard cfront C++ to C translator, the resulting C code can be compiled on any machine with an ANSI C compiler.

On many modern machines C/C++ and FORTRAN achieve the same level of efficiency in numerical computation [3]. However, in the event that you wish to
run on a machine for which the manufacturer did not put the same effort into the C compiler as the FORTRAN one, the Keenan C++ Foundation Class Library allows you to represent vectors and matrices in FORTRAN format, so that you can link with any previously developed FORTRAN linear algebra routines you wish. Indeed, FORTRAN and C/C++ can be mixed throughout the code, subject to the restriction that FORTRAN only understands very simple data structures and will therefore have a hard time with the trees and other pointer based data structures found in much of RUF.

RUF has been written in a modular style which should be easy to extend and modify. However, this manual addresses only the user interface and does not attempt to describe the source code itself. The user interface is flexible enough to describe a wide variety of application scenarios.

2 The Single Phase Flow Equation

RUF 1.0 consists of 6 executable programs named elliptic_mNd_X. Here $N$ is 2 or 3 for running two or three dimensional scenarios, while $X$ is the name of the particular numerical method used by that variant. In particular, $X$ is saddlepoint for the MFEM, hybrid for the MHFEM, stencil for the CCSM, and enhanced for the ECCSM. Only hybrid and enhanced were implemented in 3-D.

The programs are otherwise identical and build on a substantial C++ library of tools for partial differential equations, general geometry, linear algebra and user interfaces. The various numerical methods are defined in detail in [1], which also presents numerical examples and explains which methods are preferred.

All the RUF 1.0 executables solve the scalar linear elliptic partial differential equation

$$-\nabla \cdot (K(x) \nabla p(x)) = f(x), \quad x \in \Omega,$$

In two dimensions, $\Omega$ is a two dimensional polygonal region defined by triangles, rectangles and quadrilateral; in three dimensions it is a polyhedral region defined by tetrahedra or by hexahedra and bricks. $\Omega$ need not be convex; it also need not be simply connected; for instance wells may be represented as actual holes drilled in the domain.

$K$ is a symmetric positive definite tensor function of position. The scalar $p$ represents a potential and is called pressure; $u = -K \nabla p$ is called velocity. If $\hat{n}$ is a unit normal to an edge then $u \cdot \hat{n}$ is called the normal flux across the edge in the direction of the normal. On each external boundary edge, one of the following two boundary conditions must be supplied: either a scalar boundary condition

$$p(x) = p_0(x),$$

3
or a flux boundary condition

\[ u(\mathbf{x}) \cdot \mathbf{n} = g_0(\mathbf{x}) \].

Note that pressure and flux boundary conditions may be mixed throughout the boundary of \( \Omega \). The boundary functions \( p_0 \) and \( g_0 \), the coefficient tensor \( K \), and the source/sink function \( f \) must all be specified. The 2-D programs also allow "wells" to be specified as delta-function sources and sinks within elements.

The programs use the lowest order Raviart-Thomas spaces to construct a mixed finite element approximation to the PDE. The programs differ in the manner in which they solve the resulting saddle point problem. The saddlepoint variant rewrites it as a positive definite linear operator and applies conjugate gradient. However, evaluation of the operator itself requires a nested application of conjugate gradients, resulting in very slow run times for large problems. The hybrid variant introduces Lagrange multipliers on all mesh edges, yielding a sparse positive definite system, but with substantially more unknowns. The stencil variant uses special quadrature rules to reduce the problem to a sparse positive definite system in the scalar variables, analogous to the finite difference method on rectangles. Finally, the enhanced stencil method combines features of the stencil and hybrid methods to handle certain problems that can arise when using regular, rather than smooth, mesh refinement.

Future versions of RUF are planned which would incorporate a number of extensions including:

- handling the nonlinear case;
- incorporating a domain decomposition preconditioner into the planned parallel version;
- using multilevel mesh techniques for adaptive and local mesh refinement;

as well as extensions to general multiphase, multicomponent flow and transport.

3 Input Commands

The RUF user interface is based on a command interpreter which was built automatically using Phil Keenan's C++ code generator tool cmdGen, version 3.0.

The user interface takes a "free format" input file which can contain any of a large number of commands. Some of the commands are standard for all command interpreter based programs; others are specific to RUF. All command interpreters include commands for online help, include file capability, and comments. A space-delimited \# symbol comments out the rest of the line on which it occurs. In all other
contexts, white space (spaces, tabs, line breaks, and so on) serves only to delimit commands and their arguments. For a complete and up-to-date list of commands, run the program and type

    help cmds

Commands specific to RUF are described in the following subsections.

3.1 Mesh Description Commands

    bndy <bndyConds bc>
    Select the type of boundary condition to impose on subsequent faces.

    prop <double prop>
    Specify a property value for subsequent faces and elements.

    v <char* name> <double x> <double y>
    Define a vertex by its coordinates.

    V <char* name> <double x> <double y>
    Define a fixed vertex by its coordinates. Fixed vertices and vertices on
    the boundary of the region are not moved by the mesh smoother.

    e <char* name> <char* id1> <char* id2>
    Define an edge by its vertices.

    tri <char* name> <char* id1> <char* id2> <char* id3>
    Define a triangle by its edges.

    rect <char* name> <char* id1> <char* id2> <char* id3> <char* id4>
    Define a rectangle by its edges.

    quad <char* name> <char* id1> <char* id2> <char* id3> <char* id4>
    Define a quadrilateral by its edges.

    tetra <char* name> <char* id1> <char* id2> <char* id3> <char* id4>
    Define a tetrahedron by its faces (3-D only).

    brick <char* name> <char* id1> <char* id2> <char* id3> <char* id4> <char* id5> <char* id6>
    Define a brick by its faces (3-D only).

    hexahedron <char* name> <char* id1> <char* id2> <char* id3> <char* id4> <char* id5> <char* id6>
    Define a hexahedron by its faces (3-D only).
setBndy <char* faceName> <bndyConds bc> <double value>
    Modify the boundary condition for a given face.

setProp <char* elementName> <double propValue>
    Modify the property value for a given element.

subdivide <int N> <int R>
    Globally subdivide N times, each time using R smoother cycles. Use R = 0
    for strictly hierarchical mesh refinement.

3.2 Solve and Display Commands

solve
    Solve the partial differential equation on the current mesh.

colorRange auto
    The colors in plots are automatically selected to fit the full observed range
    of values.

colorRange range min max
    Values in the range [min, max] are shown in color in plots; values outside
    this range are shown as black.

colorTransform <transforms t>
    Select a transformation to apply to variables before coloring.

plot <colorVariables var>
    Append a plot of the specified variable to the plot file.

write <colorVariables var>
    Append transformed values to the log file.

norms <colorVariables var>
    Append norms of the indicated variable to the log file.

label <labels obj>
    Label the specified objects in the current plot.

vectorScale <double factor>
    Change the scale factor used in drawing vectors.

plotCommands <'' text ''>
    Append low level plot commands to the plot file.

redirect <outputFiles file> <char* newFileName>
    Redirect any output file. New output is appended to old when the specified
    file already exists.
refSoln <refSolnKinds kind> <array maxIndices>

Specify a reference solution. This sets up the boundary conditions and forcing term to match the specified analytic solution, for testing purposes and for running convergence studies. Very general analytic solutions may be specified as discussed under the refSolnKinds keyword.

refSolnParams <array params>

Give an array of parameters for the reference solution. Arrays can be specified as curly-bracket delimited lists of numbers. Very general analytic solutions may be specified as discussed under the refSolnKinds keyword.

tensor <array params>

Specify the components of the coefficient tensor as an array. In 2-D, for instance, the command

\[
\text{tensor } \{ k_{11} \ k_{12} \ k_{22} \}
\]

specifies

\[
K = \begin{pmatrix}
k_{11} & k_{12} \\
k_{12} & k_{22}
\end{pmatrix}.
\]

In 3-D, the tensor has 6 independent components. Tensors which vary with position can be used if the program is linked with an appropriate user function. Future versions of the program may provide additional user interface support to simplify this task in layered and random media.

method <methods method>

Select among variant solution procedures.

preCond <preconditioners method>

Select a preconditioner.

iterations <double C> <double p> <double rtol>

Iteration parameters: allow up to \( C \times N^p \) iterations for \( N \) equations, while seeking to reduce the relative error by \( rtol \).

dump

Print the internal solution arrays.

postProc <postprocessors method>

Select method for postprocessing fluxes.

well <double x> <double y> <double flowRate>

Add a well at the specified position, 2-D only. The flow rate \( q \) is positive for injection wells and negative for production wells. The well acts like \( q \) times a delta function in the right hand side of \( \nabla \cdot u = f \).
info
    Print information about this program.

dimension
    Prints number of space dimensions.

showState
    Print internal state variables in input format.

verbosity <int level>
    Zero turns off status messages, positive integers turn them on, when de-
    bugging is enabled.

3.3 Keyword Options

Many of the commands above take keyword arguments. For a complete and up-to-
date list of allowed values for a particular keyword, use the enum command. The
command

    enum enum

will list all of the keyword argument types known to the program; the enum command
can also provide complete details on each.

3.3.1 Boundary Conditions

An argument of type bndyConds can take any of the following values.

    scalar
        the scalar solution variable (Dirichlet condition)

    flux
        the normal flux in the solution across a face (Neumann condition)

3.3.2 Color Variables

Originally named for use in the plot command, color variables also work with the
write and norm commands. An argument of type colorVariables can take any of
the following values.

    mesh
        This draws the mesh.

    edgeLines
        This draws mesh edges in wireframe style.

    mesh-measure
        This colors each element based on its measure.
mesh-map
This displays the determinant of each element's map to the reference element.

mesh-regularity
For each element, the shortest edge length divided by the longest.

property
This displays the property code for each element.

boundary-conditions
The boundary conditions on each face.

sources-sinks
This displays the right hand side of the scalar PDE.

tensor
The determinant of the coefficient tensor.

scalar
The computed scalar solution.

gradient
The gradient of the computed solution.

flux
The flux: the coef-tensor times the gradient.

divergence
The divergence of the computed solution.

reference-solution
The reference scalar solution.

absolute-error
The absolute value of the error in the scalar solution.

relative-error
The relative error (computed-reference)/reference.

reference-gradient
The gradient of the reference solution.

gradient-error
The vector error in the gradient.

gradient-absolute-error
The absolute error in the gradient.
gradient-relative-error
    The relative error in the gradient.

reference-flux
    The flux from the reference solution.

flux-error
    The vector error in the flux.

flux-absolute-error
    The absolute error in the flux.

flux-relative-error
    The relative error in the flux.

reference-divergence
    The divergence of the reference solution.

divergence-absolute-error
    The absolute error in the divergence.

divergence-relative-error
    The relative error in the divergence.

3.3.3 Mesh Labels

An argument of type labels can take any of the following values.
vertices
edges
faces
    in 2-D, this is the same as edges
boundary
    boundary faces only
elements

3.3.4 Solution Methods

An argument of type methods can take any of the following values.
default
    the default method for this program

Certain of the programs may recognize additional solution method options; run the enum command to check.
3.3.5 Output Files

An argument of type `outputFiles` can take any of the following values.

- **plot**
  - the plot file
- **log**
  - the log file

3.3.6 Postprocessors

An argument of type `postprocessors` can take any of the following values.

- **none**
  - no postprocessing is done
- **flux**
  - apply Keenan's flux postprocessor directly to flux vectors
- **grad**
  - apply Keenan's flux postprocessor indirectly via gradient vectors
- **12**
  - apply the full linear least square postprocessor

3.3.7 Preconditioners

An argument of type `preconditioners` can take any of the following values.

- **none**
  - no preconditioning is done
- **inherit**
  - use parent mesh's solution as initial guess. This option is currently ignored by certain of the programs.

3.3.8 Reference Solution Kinds

An argument of type `refSolnKinds` can take any of the following values.

- **property**
  - piecewise constant coefficients based on element and face properties.
- **user**
  - user defined functions

The **user** case only works if user defined coefficient functions have been written and linked into the code. The **property** case makes the forcing term \( f \) a piecewise constant function defined by the element property values, and makes the boundary
functions $p_0$ and $g_0$ piecewise constants defined by the property values of boundary faces.

The remaining cases, listed below, specify an analytic solution of the form (written here for $n$ space dimensions)

$$p(x_1, \ldots, x_n) = \sum_{i_1=0}^{k_1} \cdots \sum_{i_n=0}^{k_n} c_{i_1 \ldots i_n} f_{i_1 \ldots i_n}(x_1, \ldots, x_n).$$

Here $f$ is a primitive function selected by one of the following named options:

- **poly**
  
  polynomials:
  
  $$f_{i_1 \ldots i_n}(x_1, \ldots, x_n) = \prod_{k=1}^{n} x_k^{i_k}.$$  

- **trigSum**
  
  trigonometrics:
  
  $$f_{i_1 \ldots i_n}(x_1, \ldots, x_n) = \cos \left( \sum_{k=1}^{n} i_k x_k \right).$$  

- **trigProd**
  
  trigonometrics:
  
  $$f_{i_1 \ldots i_n}(x_1, \ldots, x_n) = \prod_{k=1}^{n} \cos (i_k x_k).$$  

- **bump**
  
  bump functions:
  
  $$f_{i_1 \ldots i_n}(x_1, \ldots, x_n) = \left( 1 + \sum_{k=1}^{n} (x_k - i_k)^2 \right)^{-1}.$$  

- **exp**
  
  exponential functions:
  
  $$f_{i_1 \ldots i_n}(x_1, \ldots, x_n) = \exp \left( \sum_{k=1}^{n} i_k x_k \right).$$

The second argument to the **refSoln** command is the array of maximum indices \( \{k_1 \ldots k_n\} \). The coefficients $c_{i_1 \ldots i_n}$ are supplied by the **refSolnParams** command, as a linear array, ordered lexicographically, as in the nested loop sequence.
for i1 = 0 to k1
  for i2 = 0 to k2
    ...
    for in = 0 to kn
read c(i1,...,in)

3.3.9 Transformations
An argument of type transforms can take any of the following values.
  identity
    x
  absolute-value
    |x|
  log10-of-absolute-value
    \log_{10}|x|

4 Some Mathematical Details
The programs solve the elliptic partial differential equation

$$-\nabla \cdot (K(x) \nabla p(x)) = f(x), \quad x \in \Omega,$$

with boundary conditions

$$p(x) = p_0(x), \quad x \in \partial \Omega_D,$$

and

$$-(K(x) \nabla p(x)) \cdot \hat{n}(x) = g_0(x), \quad x \in \partial \Omega_N,$$

where $\partial \Omega_N = \partial \Omega - \partial \Omega_D$.

Following the standard mixed finite element formulation let

$$u = -K \nabla p,$$

whence

$$\nabla \cdot u = f.$$

Let $(\cdot, \cdot)$ denote the $L^2$ inner product on $\Omega$, and $< \cdot, \cdot >$ the $L^2$ inner product on $\partial \Omega$. 

13
Multiplying by suitable test functions, integrating and applying the divergence theorem yields

\[(K^{-1}u, v) = -(\nabla p, v) = (p, \nabla \cdot v) = \langle p, v \cdot \hat{n} \rangle,\]

and

\[(\nabla \cdot u, w) = (f, w).\]

For \(u \in H(div)\) and \(p \in L^2\), the above equations are equivalent to the original partial differential equation, provided they hold for all \(v \in H_0(div)\) and \(w \in L^2\). Here \(H_0(div) = \{v \in H(div) : v \cdot \hat{n} = 0 \text{ on } \partial\Omega_N\}\). Thus the term \(\langle p, v \cdot \hat{n} \rangle\) becomes \(\langle p_0, v \cdot \hat{n} \rangle_{\partial\Omega_D}\).

Now let \(\{w_j : j \in I_w\}\) be a basis for a finite dimensional subspace of \(L^2\), and \(\{v_j : j \in I_v\}\) be a basis for a suitable corresponding finite dimensional subspace of \(H(div)\). In these programs the lowest order Raviart-Thomas spaces are used, corresponding to a decomposition of \(\Omega\) into mesh elements, so the \(w_j\) are piecewise constants and the \(v_j\) are discontinuous piecewise linear functions with continuous normal components across elements.

Let \(I_a\) be the set of indices \(j\) for which \(v_j \cdot \hat{n} = 0\) on \(\partial\Omega_N\), and let \(I_b = I_v - I_a\).

We then seek approximate solutions

\[U = \sum_{j \in I_v} U_j v_j,\]

and

\[P = \sum_{j \in I_w} P_j w_j.\]

The unknown coefficients \(U_j\) and \(P_j\) must satisfy

\[\sum_{j \in I_v} U_j (K^{-1}v_j, v_i) - \sum_{j \in I_w} P_j (w_j, \nabla \cdot v_i) = \langle p_0, v_i \cdot \hat{n} \rangle, \text{ for all } i \in I_a,\]

\[\sum_{j \in I_v} U_j (\nabla \cdot v_j, w_i) = (f, w_i), \text{ for all } i \in I_w,\]

and

\[U_j = g_0(x_j) \text{ for all } j \in I_b,\]

where \(x_j\) is the midpoint of the external edge on which \(v_j \cdot \hat{n} = 1\).

Let \(U = (U_j)_{j \in I_v}^T\) be the vector of unknown flux coefficients, and \(P = (P_j)_{j \in I_w}^T\) be the vector of unknown scalar coefficients. Let us use block notation for the \(I_v\)
index range, writing for example

\[ U = \begin{pmatrix} U_a \\ U_b \end{pmatrix}. \]

Then in block matrix form we have

\[ M_{aa} U_a - B_a P = R_a - M_{ab} G, \]
\[ B_a^T U_a = R_w - B_b^T G, \]

where

\[ M_{ij} = (K^{-1} v_j, v_i), \]
\[ B_{ij} = (w_j, \nabla \cdot v_i), \]
\[ R_{ai} = -< p_0, v_i \cdot \hat{n} >, \]

and

\[ R_{wi} = (f, w_i). \]

This is the symmetric indefinite sparse square linear system which must be solved in the saddlepoint (MFEM) variation. The other variants are similar and are described in greater detail in [1].

5 Running the Programs

5.1 Command Line Arguments

Executing a command like

`elliptic_m2d_hybrid -usage`

will bring up a complete list of the command line options and C-shell environment variables used by the program. In particular, the `-echo` option displays input commands as they are processed, which may help with debugging input files. The standard command line is

`elliptic_mNd_X inputFileName plotFileName logFileName`

Using `-` in place of a file name makes the program read from the keyboard or send output to the screen, which also happens if the output files are omitted. Two dimensional versions of RUF write graphics files suited for display with Phil Keenan’s `plot` program, which runs under X-11/Motif on workstations; three dimensional
versions write graphics files designed for Wavefront's commercial Data Visualizer program, which runs on Silicon Graphics machines. Both kinds of graphics files are simply text files in a straightforward format, so user conversion for other display programs should be possible. Moreover, the write command provides an easy way to write out the solution in numeric form, which (at least when using rectangular grids) can then be imaged using commercial products such as Matlab.

5.2 Sample Input Files

Suppose the file twistM contains the following lines:

```plaintext
# a pair of triangles stretched along the normal direction
v a -1 0
v b  0 1
v c  0 -1
v d  2 1.5

# boundary edges

e ab  a b
e ac  a c

bndy flux

e db  d b
e dc  d c

# internal edges

e bc  b c

tri t  ab ac bc
tri tt db dc bc
```

It defines a domain made from two triangles, as shown in Figure 1.

Next, suppose the file demo contains the following lines:

```plaintext
# a sample driver file

include twistM # this reads in the above mesh description
```
Figure 1: Sample coarse mesh
plot mesh
plotCommands { new }

refSoln poly { 3 3 }
refSolnParams 16 {
  1  -3  1.7 -4.1
  2  2.4 3.1  0
     -1.1 2.1  0  0
     1.2  0  0  0 }

tensor { 1 0.5 3 }

vectorScale 0.01

plot edgeLines
plot boundary-conditions
plotCommands { new }

iterations 1.1 1 1e-12
subdivide 3 0
solve

plot scalar
plot edgeLines
plotCommands { new }

plot reference-solution
plot edgeLines
plotCommands { new }

plot absolute-error
plot edgeLines
plotCommands { new }

norms absolute-error
norms flux-absolute-error

This subdivides the mesh and solves the PDE using a cubic polynomial as a reference solution, with a non-diagonal tensor for $K$. It produces several informative plots as well as discrete norms for the error in both pressure and velocity. If run via a command like
elliptic_m2d_enhanced demo demo.plot demo.log

the plots will be in the file demo.plot, while the norms and other convergence information will be in demo.log. For instance, figure 2 is a grayscale rendition of the color plot produced showing the pressure solution.

The output log shows, for instance, that the error in the pressure on this mesh, which is still a relatively coarse mesh, is already fairly small: a maximum error of 0.456 (as shown by the $l^\infty$ norm), while pressure itself is on the order of 20. Subdividing the mesh further yields more accurate answers, at the cost of additional computing time. For complete details on the convergence rates for each numerical method in RUF, see [1].

As a final example of the code's flexibility, Figure 3 shows the pressure and velocity fields surrounding a two dimensional horizontal well. The injection well is perforated only along the horizontal segment; the other well is a production well.

References


Figure 2: Pressure Solution
Pressure and Flux

Figure 3: A Partially Perforated Horizontal Well Example

P. T. Keenan - May 1994