Turbine Simulations with Eilmer3.

Mechanical Engineering Report 2011/xx
Peter Jacobs, Paul Petrie-Repar, Carlos de Miranda-Ventura,
Emilie Sauret, Jason Czapla, Peter Blyton,
Haiko Rijkers*and Paul van der Laan†,
Queensland Geothermal Energy Centre of Excellence,
The University of Queensland, Brisbane, Australia.

October 2011

Abstract

In order to perform simulations of turbomachinery flow paths. the in-house compressible-flow simulation code, *Eilmer3*, has been adapted. Specific adaptations include the addition of terms for the rotating frame of reference, programmable boundary conditions for periodic boundaries and a mixing plane interface between the rotating and non-rotating blocks.

Simulations of various turbomachines using Eilmer3 then compared to reference calculations and experimental data to demonstrate it effectiveness for turbomachinery calculations.

*University of Twente, Netherlands
†University of Twente, Netherlands

Contents

1 Introduction

Our in-house code, *Eilmer3*, has been developed over the past few years for the simulation of hypersonic flows. Being built specifically to model the flow in shock and expansion tubes, it solves the compressible Navier–Stokes equations via a cell-centred time-dependent finite-volume formulation. The governing equations are expressed in integral form over arbitrary hexahedral cells, with the time rate of change of conserved quantities in each cell specified as a summation of the mass, momentum and energy flux through the cell interfaces. The code has come from a hypersonics research background is also specialised for simulating flows in 2D axisymmetric geometries. The thermochemistry module can handle gases in chemical equilibrium or nonequilibrium. When simulating gases with finite-rate chemistry and radiation energy exchange, these physical processes are treated with an operator-split approach. The computational core of *Eilmer3* is written in a combination of C and C++, with the option for user-defined functions such as boundary conditions provided as Lua scripts. Preprocessing (*i.e.* grid generation) and postprocessing is handled by a collection of Python programs.

For a more detailed description of the numerical methods coded into `Eilmer3`, see the companion report [1] which covers the gas-dynamic formulation and the basic thermochemistry components.

2 The code

2.1 Gas-Dynamic Formulation

The code is formulated around the integral form of the Navier-Stokes equations, which can be expressed as

$$\frac{\partial}{\partial t}\int_V U dV = -\oint_S \left(\overline{F}_i - \overline{F}_v\right)\cdot\hat{n}\; dA + \int_V Q dV \;, \tag{1}$$

where S is the bounding surface and $\hat{n}$ is the outward-facing unit normal of the control surface. Two-dimensional and three-dimensional formulations are implemented somewhat separately in *Eilmer3*, however, there is much of the formulation and code that is the same for both cases. Considering a single-species gas, the vector of conserved quantities can be written as

$$U = \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho u_z \\ \rho E \end{bmatrix} \;. \tag{2}$$

Here, the conserved quantities are respectively density, x, y, z-momentum per volume and total energy per volume. For a rotating frame of reference, we choose the rotation to occur about the z-axis and follow the description in Ref. [2]. The relative velocity, $\vec{u}$, in the rotating frame is related to the intertial-frame velocity, $\vec{v}$, as $\vec{v} = \vec{u} + \omega_z(x\hat{j} - y\hat{i})$, where ω_z is the angular speed of the frame. The total energy is constructed as $E = e + \frac{1}{2}|\vec{u}|^2 - \frac{1}{2}\omega_z^2 r^2$, where e is the specific internal energy and $r^2 = x^2 + y^2$, r being the radial distance from the z-axis.

The inviscid component of the flux vector is

$$\overline{F}_i = \begin{bmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_y u_x \\ \rho u_z u_x \\ \rho E u_x + p u_x \end{bmatrix}\hat{i} + \begin{bmatrix} \rho u_y \\ \rho u_x u_y \\ \rho u_y^2 + p \\ \rho u_z u_y \\ \rho E u_y + p u_y \end{bmatrix}\hat{j} + \begin{bmatrix} \rho u_z \\ \rho u_z u_x \\ \rho u_z u_y \\ \rho u_z^2 + p \\ \rho E u_z + p u_z \end{bmatrix}\hat{k} \;. \tag{3}$$

The viscous component is

$$\begin{aligned} \overline{F}_v \;=\; & \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{zx} \\ \tau_{xx}u_x + \tau_{yx}u_y + \tau_{zx}u_z + q_x \end{bmatrix}\hat{i} + \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{zy} \\ \tau_{xy}u_x + \tau_{yy}u_y + \tau_{zy}u_z + q_y \end{bmatrix}\hat{j} + \\ & \begin{bmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ \tau_{xz}u_x + \tau_{yz}u_y + \tau_{zz}u_z + q_z \end{bmatrix}\hat{k} \;, \end{aligned} \tag{4}$$

and the viscous stresses are evaluated on the usual gradients of velocity.

The finite-volume cells are hexahedral with 6 (possibly-nonplanar) quadrilateral surfaces interfacing the neighbouring cells. Flux values are estimated at midpoints of the cell interfaces and the integral conservation equation (1) is approximated as the algebraic expression

$$\frac{dU}{dt} = -\frac{1}{V} \sum_{cell-surface} \left(\overline{F}_i - \overline{F}_v\right) \cdot \hat{n}\, dA + Q \ , \tag{5}$$

where U and Q now represent cell-average values of the conserved-quantity vector and the source vector, respectively. The centrifugal and Coriolis forces are included in the source vector as

$$Q = \begin{bmatrix} 0 \\ \rho(+2\omega_z u_y + \omega_z^2 x) \\ \rho(-2\omega_z u_x + \omega_z^2 y) \\ 0 \\ 0 \end{bmatrix} \ . \tag{6}$$

2.2 Grid and Boundary Conditions

The flow domain is discretised as a set of structured, body-fitted grids. Data arrays for each block are dimensioned such that there is a buffer region, two cells deep, around the *active cells*, which completely defines the flow domain covered by the block. The buffer region contains *ghost cells* which are used to hold a copy of the flow information from adjacent blocks or to implement the boundary conditions.

For a boundary common to two blocks, the ghost cells in the buffer region of each block overlap the active cells of the adjacent block. The only interaction that occurs between blocks is the exchange of boundary data, prior to the reconstruction phase of each time step. For the *shared memory* version of the code, the exchange of cell-average data along the block boundaries takes place as a direct copy from the active-cell of one block to the ghost-cell of the other block. Thus, the cells along the common boundary of each block must match in both number and position. Some logic is used within the exchange routines to set the appropriate indexing direction for each boundary. The information on the connections between block boundaries is stored in a (global) connectivity array. For each boundary on each block, this array stores the identity of the adjacent block and the name of the connecting boundary on the adjacent block.

The inviscid-component of applied boundary conditions is implemented by also filling in the ghost-cell data and then applying the normal reconstruction and flux calculation without further discrimination of the boundary cells. This approach covers solid/slip walls, inflow and outflow boundaries.

For viscous simulations, boundaries may also be assigned as fixed temperature, no-slip or catalytic (chemical equilibrium at wall gas state) boundary conditions. Such viscous

boundary conditions also use data specified at the cell interfaces that lie along the boundary surface. These data are used in the derivative calculations that subsequently feed into the viscous fluxes.

The primary data held by the code are cell-average data, associated with cell centres. To get the fluxes at cell interfaces, a variable-by-variable reconstruction is made of the flow field. This is done in a one-dimensional fashion, working along one-index direction at a time. *Left* and *Right* values of a flow variable at a cell interface are evaluated as the corresponding cell average value plus a limited higher-order interpolated increment. This is essentially a piecewise-parabolic reconstruction of the flowfield data. For flows without strong compressions or shocks, *Eilmer3* then uses a modified AUSMDV calculator [3], as described in Ref.[1]. Prior to applying the flux calculator at each interface, the velocity field is rotated into a coordinate system local to the cell interface, and after, that flux transformed back into the global coordinate system.

2.3 Mixing-Plane Interface

The stator-rotor interaction was modelled making use of a *mixing plane* (MP) model [4]. Other authors ([5],[6], [7]) used this approach to model the interaction between the bladed segments of radial turbines. Contrasting with the *multi-reference frame* (MRF) model [8], this latter model (MP) is applicable when the flow at the interface between stationary and moving zones is not uniform and therefore represents a more suitable choice when modelling cases where a strong interaction between the stationary and rotating zones exists (i.e. current case). When following this approach, each zone is regarded as a steady state problem, taking the flow data from both the stator and rotor interface cells and averaging it. This removes any unsteadiness that could occur in the flow field between passages. This also requires less computational effort when compared to the sliding mesh model since its calculations are unsteady and also as a consequence of a higher number of blade passages needed when using this latter model to simulate cases with different number of blades between each blade row. Presently, we have implemented the *MP* interface to model the stator-rotor interaction.

Within `Eilmer3` here are also boundary conditions that allow the user to specify the ghost-cell data and boundary interface data via user-written functions. These functions may also be used to bypass the internal flux calculators and specify the boundary fluxes directly. As described in Ref. [9] and shown in Appendix A for the Kofsky radial-inflow turbine example, these user-defined boundary conditions are used to implement:

- the mixing-plane interface between the rotating and non-rotating blocks;
- the circumferentially-periodic boundary connecting segments of the volute, the gap between stator and rotor, and the segments of the diffuser;

- the subsonic inflow and outflow conditions.

The Wilcox-2006 $k-\omega$ turbulence model is available in the code but has not been exercised for the turbomachinery calculations thus far.

3 Kofsky Radial-Inflow Turbine

For radial-inflow turbines, the availability of fully defined test cases in terms of geometry as well as operating conditions and experimental results was found to be scarce. Therefore, this work will allow to set a standard test case for our code development and testing. In this context, a small 10 kW shaft output turbine tested by *NASA* [10] was considered one of the most suitable sample cases since it was found to provide sufficient detail with respect to the overall test case (geometry and operating conditions). Experimental results regarding the turbine performance for two values of Reynolds number, considering both argon and air as the working fluids as well as different inlet conditions were also presented. The design point values mentioned in Ref. [10] allowed the definition of the input conditions to be used when considering using argon as the working fluid (see Table 1).

Total Mass Flow Rate [kg/s]	0.277
Total Inlet Pressure [kPa]	91
Total Inlet Temperature [K]	1083.3
Static Outlet Pressure [kPa]	56.4
Inlet Flow Angle [deg]	55.6
Rotational Speed [RPM]	38500

Table 1: Flow Conditions used as Boundary Conditions for both *CFX* and *Fluent* with argon as the working fluid.

3.1 Geometric model

A simplified three-dimensional geometry of the *NASA* turbine was also considered. For this case, the procedure reported by Krijgsman [11] was followed, considering both the stator and rotor and assuming an infinitesimal blade thickness as well as no clearances for both the stator vanes and rotor blades.

In Krijgsman's work, several steady state simulations were carried out using *Fluent* considering different approaches such as inviscid/viscous rotor analysis as well as different stator-rotor interaction models. An off-design performance evaluation was carried out by comparing the results against experimental data reported in Ref. [10] when considering several pressure-ratios, maintaining the total inlet pressure and changing the mass flow rate. Krijgsman's results showed good agreement with the *NASA* values [10] but the assumed stator inlet flow angles were found to deviate from the reported inlet conditions. Therefore, the input conditions presented in Table 1 were used for the current calculations and further tests (not shown in this paper) were performed to explore the influence of the input flow angles. Relative to Krijgsman's results, the new tests showed a difference of 5.45% and 6.50% for both the total-total efficiency and total-static efficiency respectively as well as a 1.2% difference in terms of power output.

Approximate three-dimensional geometries were also defined making use of commercial radial-inflow turbine design software packages *RITAL* and *AxCent* from *Concepts NREC* (`http://www.conceptsnrec.com`). These geometries were re-created by optimising the one-dimensional preliminary design obtained with *RITAL* in order to obtain a similar design as well as similar velocity triangles and using these results to create suitable three-dimensional geometries from *AxCent*. Two fully-defined three-dimensional geometries were obtained from *AxCent*. Both geometries take into account a pre-set blade thickness distribution and the main difference between these cases is the absence and presence of (blade-shroud) clearances respectively. The one-dimensional analysis results compared with the *NASA* reported data are presented in Table 2.

	NASA reported	*RITAL* computed	Error [%]
Inlet Stator			
α [deg]	55.60	55.40	0.36
C [m/s]	107.35	109.45	1.95
Outlet Stator			
α [deg]	72.00	72.35	0.49
C [m/s]	288.05	280.80	2.50
Inlet Rotor			
α [deg]	71.80	72.30	0.70
β [deg]	-23.60	-21.20	10.17
C [m/s]	283.30	287.00	1.31
W [m/s]	96.50	94.10	2.49
Outlet Rotor			
α [deg]	10.20	12.20	19.61
β [deg]	-50.80	-50.80	set
C [m/s]	148.80	116.80	21.51
W [m/s]	231.70	180.60	22.05
η_{t-t}	0.880	0.877	0.34
η_{t-s}	0.824	0.817	0.85
Total Power [kW]	22.39	22.24	0.67

Table 2: Comparison of *RITAL* computational results and reported data from Ref. [10].

The results are in fair agreement with the reported values, despite the differences in velocities at the outlet of the rotor. This is evidence that further improvement in the three-dimensional rotor description is needed. Moreover, the meanline (one-dimensional) design has been conducted without considering the presence of splitters which actually exist in the original turbine and which might explain some of the discrepancies between our preliminary calculations and the reported *NASA* results. However, at the present stage and since this work will focus on the testing and development of computational tools comparing results across different codes, these results were considered sufficient to recreate a description of three-dimensional geometry of the original turbine (Figure 1).

In addition, radial-inflow turbines are expected to be less sensitive to details of geometry and flow conditions [12]. The following sections show our attempt to re-create the NASA turbine and flow conditions without the inclusion of splitters. Further work will have to be performed in terms of geometric description as well as comparing both experimental and computational results when splitters are included.

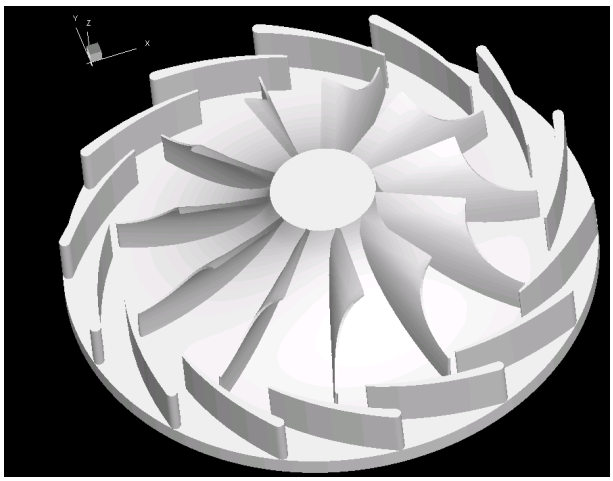

Figure 1: Three-dimensional Geometry obtained with *AxCent* from *RITAL* one-dimensional preliminary design.

Assuming periodicity, only one blade passage for each bladed segment was modelled. Meshes used for the calculations were generated with the *AxCent* program using geometric parameters imported from *RITAL* and then exported as *Fluent* mesh files. Boundary conditions were based on the reported values for the design-point input conditions, specifically, the reported design point values for the mass flow rate, total temperature at the stator inlet and rotor outlet as well as static backpressure at the rotor outlet. Density was calculated assuming an ideal gas behaviour and the transport properties for argon were calculated by the use of polynomial and Sutherland law approaches for thermal conductivity and viscosity respectively. As a first approach, the specific heat C_p as well as the ratio of specific heats γ were considered constants and equal to 520.3 J/(kgK) and 5/3 respectively.

Variables such as static pressure, mass flow rate and velocities were also monitored during the calculations and the presented results were obtained when their variation at the inlet and outlet of both the stator and rotor was considered residual.

3.2 Results

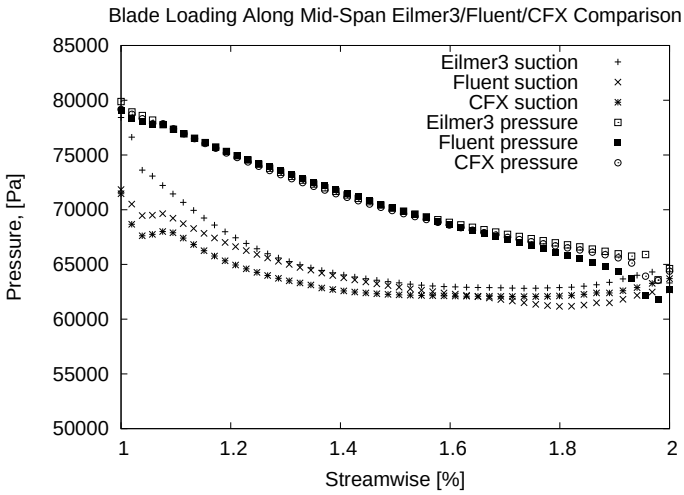

Figure 2: Comparison Between *Eilmer3*, *CFX* and *Fluent* Considering Inviscid Flow.

References

[1] P. A. Jacobs, R. J. Gollan, A. J. Denman, B. T. O'Flaherty, D. F. Potter, P. J. Petrie-Repar, and I. A. Johnston. Eilmer's theory book: Basic models for gas dynamics and thermochemistry. Mechanical Engineering Report 2010/09, The University of Queensland, Brisbane, Australia, 2010.

[2] C. Hirsch. *Numerical Computation of Internal and External Flows, 2nd Ed.* Butterworth-Heinemann, 2007.

[3] Y. Wada and M. S. Liou. A flux splitting scheme with high-resolution and robustness for discontinuities. AIAA Paper 94-0083, January 1994.

[4] J. D. Denton and U. K. Singh. Time marching methods for turbomachinery flow calculations. In *VKI Lecture Series 1979-7*. von Karman Institute for Fluid Dynamics, 1979.

[5] L. Barr, S. Spence, and D. Thornhill. A numerical and experimental performance comparison of an 86 mm radial and back swept turbine. In ASME, editor, *Proceedings of ASME Turbo Expo 2009: Power for Land, Sea and Air GT2009 June 8-12, 2009, Orlando, Florida, USA*. ASME, June 2009.

[6] X. Qiu, M. R. Anderson, and N. Baines. Meanline modeling of radial inflow turbine with variable area nozzle. In ASME, editor, *Proceedings of ASME Turbo Expo 2009: Power for Land, Sea and Air, GT2009, June 8-12, 2009, Orlando, Florida USA*. ASME, 2009.

[7] Y. Li and Q. Zheng. Numerical simulation of a multistage radial inflow turbine. In *Proceedings of ASME Turbo Expo 2006: Power for Land, Sea and Air, GT2006, May 8-11, 2006, Barcelona, Spain*. ASME, May 2006.

[8] J.Y. Luo, R.I. Issa, and A.D. Gosman. Prediction of impeller induced flows in mixing vessels using multiple frames of reference. In *Institution of Chemical Engineers, Symposium Series, No. 136*, pages 549–556, 1994.

[9] P. van der Laan. Advanced boundary conditions in turbomachinery using Eilmer3. QGECE Internal Report, The University of Queensland, 2009.

[10] M. G. Kofskey and D. E. Holeski. Cold performance evaluation of a 6.02-inch radial inflow turbine designed for a 10-kilowatt shaft output brayton cycle space power generation system. Technical report, NASA Technical Report, TN D-2987, 1966.

[11] D. Krijgsman. Radial turbine analysis using Fluent. QGECE Internal Report, The University of Queensland, 2008.

[12] H. E. Rohlik and M. G. Kofskey. Recent radial turbine research at the NASA Lewis Research Center. Technical Memorandum X-67903, NASA, 1972.

A UDF Boundary Conditions for the NASA Rotor

```
-- udf-periodic-bc.lua
-- Lua script for the user-defined periodic BC
--
-- This particular example sets up peroidic boundary conditions
-- for the turbine-blade simulation.
-- When called, this boundary conditions looks up the flow data
-- in a cell that would overlay the ghost cell,
-- shifted by 1 period in the y-direction.
-- We will assume that the boundary blocks are approximately aligned
-- with the x,y-axes so that we simply add or subtract the y_period value.
--
-- PJ, 07-Mar-2008
--     03-Sep-2008 port to Elmer3
--     03-Nov-2008 sc10_3D version
-- Max.F
--     14-Jul-2009 modification for simulation of Blade Radial Turbine Nasa 6.02-inch
-- Paul van der Laan
--     Okt-2009 testcase.py (mixing plane model)

-- We will remember where we found the appropriate cells.
g1_src_blk = {}; g1_src_i = {}; g1_src_j = {}; g1_src_k = {}; g1_src_theta = {};
g2_src_blk = {}; g2_src_i = {}; g2_src_j = {}; g2_src_k = {}; g2_src_theta = {};

theta_period = math.rad(360.0/11.0)

-- Here we have assumed that the midspan plane has mapped unstretched
-- to the midspan cylinder.

function ghost_cell_position(xc, yc, zc, xw, yw, zw)
   -- Returns the ghost cell centre position as a reflection of the original cell
   -- in the boundary surface.
   -- c represents the cell-centre
   -- w represents the wall-interface position
   -- We will work all movements in the radial (x,y)-plane.
   theta_w = math.atan2(yw,xw)
   rc = math.sqrt(xc*xc + yc*yc)
   theta_c = math.atan2(yc,xc)
   dtheta = theta_c - theta_w
   theta_ghost = theta_c - 2*dtheta
   return rc*math.cos(theta_ghost), rc*math.sin(theta_ghost), zc
end

function source_cell_position(xc, yc, zc, theta_change)
   -- Computes the cell position in the (x,y)-plane,
   -- theta_change radians around the axis.
   rc = math.sqrt(xc*xc + yc*yc)
   theta = math.atan2(yc,xc)
   new_theta = theta + theta_change
   return rc*math.cos(new_theta), rc*math.sin(new_theta), zc
end

function rotate_velocity_vector(cell, theta_old, theta_new)
   -- Rotates the (x,y)-plane velocity vector.
   -- First, transforms into radial and tangential components, then back.
   vx = cell.u; vy = cell.v

   vt = -vx * math.sin(theta_old) + vy * math.cos(theta_old)
   vr = vx * math.cos(theta_old) + vy * math.sin(theta_old)
   cell.u = -vt * math.sin(theta_new) + vr * math.cos(theta_new)
   cell.v = vt * math.cos(theta_new) + vr * math.sin(theta_new)
   return cell.u, cell.v
end

function ghost_cell(args)
   -- Function that returns the flow state for a ghost cell
   -- for use in the inviscid flux calculations.
   i = args.i; j = args.j; k = args.k
   x = args.x; y = args.y; z = args.z
```

```
   indx = j*nnk + k
   if g1_src_blk[indx] == nil then  --When a new block is searching for it's ghost cell, g1_src_blk[i
      if args.which_boundary == NORTH then
         c = sample_flow(block_id, i, j, k)
         xg, yg, zg = ghost_cell_position(c.x, c.y, c.z, x, y, z)
         xs, ys, zs = source_cell_position(xg, yg, zg, theta_period)
         g1_src_blk[indx], g1_src_i[indx], g1_src_j[indx], g1_src_k[indx]
            = locate_cell(xs, ys, zs)
         g1_src_theta[indx] = math.atan2(ys,xs)
         -- Locate cell corresponding to second ghost cell similarly.
         j = j - 1
         c = sample_flow(block_id, i, j, k)
         xg, yg, zg= ghost_cell_position(c.x, c.y, c.z, x, y, z)
         xs, ys, zs = source_cell_position(xg, yg, zg, theta_period)
         g2_src_blk[indx], g2_src_i[indx], g2_src_j[indx], g2_src_k[indx]
            = locate_cell(xs, ys, zs)
         g2_src_theta[indx] = math.atan2(ys,xs)
      elseif args.which_boundary == EAST then
         print("EAST boundary should not be periodic!")
      elseif args.which_boundary == SOUTH then
      --elseif args.which_boundary == BOTTOM then
         -- Search for the cell corresponding to the ghost-cell,
         -- offset by one period.
         c = sample_flow(block_id, i, j, k)
         xg, yg, zg = ghost_cell_position(c.x, c.y, c.z, x, y, z)
         xs, ys, zs = source_cell_position(xg, yg, zg, -theta_period)
         g1_src_blk[indx], g1_src_i[indx], g1_src_j[indx], g1_src_k[indx]
            = locate_cell(xs, ys, zs)
         g1_src_theta[indx] = math.atan2(ys,xs)
         -- Locate cell corresponding to second ghost cell similarly.
         j = j + 1
         c = sample_flow(block_id, i, j, k)
         xg, yg, zg = ghost_cell_position(c.x, c.y, c.z, x, y, z)
         xs, ys, zs = source_cell_position(xg, yg, zg, -theta_period)
         g2_src_blk[indx], g2_src_i[indx], g2_src_j[indx], g2_src_k[indx]
            = locate_cell(xs, ys, zs)
         g2_src_theta[indx] = math.atan2(ys,xs)
      elseif args.which_boundary == WEST then
         print("WEST boundary should not be periodic!")
      end
   end
   -- On subsequent calls, the array entries should be non-nil so
   -- we can immediately look up the flow data.
   cell1 = sample_flow(g1_src_blk[indx], i, g1_src_j[indx], g1_src_k[indx])
   cell2 = sample_flow(g2_src_blk[indx], i, g2_src_j[indx], g2_src_k[indx])
   if args.which_boundary == NORTH then
      theta_old = g1_src_theta[indx]
      theta_new = theta_old - theta_period
      cell1.u, cell1.v = rotate_velocity_vector(cell1, theta_old, theta_new)
      theta_old = g2_src_theta[indx]
      theta_new = theta_old - theta_period
      cell2.u, cell2.v = rotate_velocity_vector(cell2, theta_old, theta_new)
   elseif args.which_boundary == SOUTH then
   --elseif args.which_boundary == BOTTOM then
      theta_old = g1_src_theta[indx]
      theta_new = theta_old + theta_period
      cell1.u, cell1.v = rotate_velocity_vector(cell1, theta_old, theta_new)
      theta_old = g2_src_theta[indx]
      theta_new = theta_old + theta_period
      cell2.u, cell2.v = rotate_velocity_vector(cell2, theta_old, theta_new)
   end
   return cell1, cell2
end


function interface(args)
   -- Function that returns the conditions at the boundary
   -- when viscous terms are active.
   return sample_flow(block_id, args.i, args.j, args.k)
end
```

Index