ORNL/TM-2016/215

Dakota Uncertainty Quantification Methods Applied to the CFD code Nek5000

Marc-Olivier Delchini
Emilian L. Popov
William David Pointer

04/29/2016

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2016/215

Reactor and Nuclear Systems Division

Dakota Uncertainty Quantification Methods Applied to the CFD code Nek5000

Marc-Olivier Delchini, Emilian L. Popov, William David Pointer

Date Published: 06/2016

Prepared by
OAK RIDGE NATIONAL LABORATORY
P.O. Box 2008
Oak Ridge, Tennessee 37831-6285
managed by
UT-Battelle, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

ACRONYMS

NEAMS	Nuclear Energy Advanced Modeling and Simulation
CFD	computational fluid dynamics
UQ	uncertainty quantification
LES	large eddy simulation
URANS	unsteady Reynolds average Navier-Stokes
SEM	spectral element method

ACKNOWLEDGEMENT

This work was supported by the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program under the auspices of the US Department of Energy, Office of Nuclear Energy.

ABSTRACT

This report presents the state of advancement of a Nuclear Energy Advanced Modeling and Simulation (NEAMS) project to characterize the uncertainty of the computational fluid dynamics (CFD) code Nek5000 using the Dakota package [2] for flows encountered in the nuclear engineering industry. Nek5000 is a high order spectral element CFD code developed at Argonne National Laboratory for high resolution spectral-filtered large eddy simulations (LESs) and unsteady Reynolds averaged Navier-Stokes (URANS) simulations. The Dakota package developed at Sandia National Laboratory can be integrated with many scientific and engineering codes to facilitate efficient, effective uncertainty quantification (UQ) and sensitivity analyses. The objective of this work is to perform a UQ and a sensitivity analysis of the numerical and physical models implemented in Nek5000 and used to solve for flows in two geometries of interest: a 3-D pipe and a 7-pin bundle. The results presented in this report demonstrate loose integration of Dakota with Nek5000 and thus should not be used to assess the accuracy of the numerical methods and physical models implemented in Nek5000. All results were obtained by running the default version of Nek5000 which does not include any turbulent models. Thus, any experimental data with the current results is not expected to be validated.

1. INTRODUCTION

Predictive modeling and simulation of the performance of a nuclear reactor and its fuel is a challenging task because of the large number of coupled physical phenomena that must be addressed. In addition to this intrinsic complexity, model uncertainty must be accounted for in any analysis if the model will be used to facilitate design or operational decisions which may impact safety and performance. Rigorous, structured uncertainty analyses are performed by first characterizing the model's input uncertainties and, then propagating the uncertainty through the model in order to estimate the output's uncertainty.

1.1 OBJECTIVES

This project is part of the ongoing effort to assess modeling uncertainty in Nek5000 simulations of flow configurations relevant to the advanced reactor applications of the NEAMS program. Two geometries are under investigation in these preliminary assessments: a 3-D pipe and a 3-D 7-pin-bundle. Initial efforts have focused on gaining an understanding of Nek5000 modeling options and integration of Nek5000 with Dakota. This report focuses on an initial demonstration of the use of Dakota to assess parametric uncertainties in a simple pipe flow problem. This problem is used to optimize performance of the uncertainty quantification strategy and to estimate computational requirements for assessments considering more complex geometries.

The objective of this project is threefolds. In the first step, the uncertainty of the numerical model, the flow condition, and the material properties used in Nek5000 will be characterized for flow in a 3-D pipe by comparing the numerical results against experimental data [4, 5]. Once Nek5000 is calibrated using results from the first study, the flow in a 7-pin-bundle geometry will be simulated. Then, both an uncertainty quantification and a sensitivity analysis will be performed. Particular attention will be focused on UQ methods used (polynomial chaos or sampling methods for instance) in order to minimize the computational cost. In this report, only results of the first step are presented. Comparison with the commercial CFD code Star-CCM+ will be also investigated.

1.2 NEK5000 CODE AND DAKOTA

The computational fluid dynamic (CFD) code Nek5000 developed at Argonne National Laboratory is used to simulate flow behavior in the 3-D pipe. The incompressible Navier-Stokes equations are solved on a computational domain discretized by a spectral element method (SEM). Several temporal discretization methods are available for transient simulations. A second-order temporal discretization is used for the results presented in this report. The Nek5000 code is designed to run on massively supercomputers and has been widely used and tested for CFD simulations around the world.

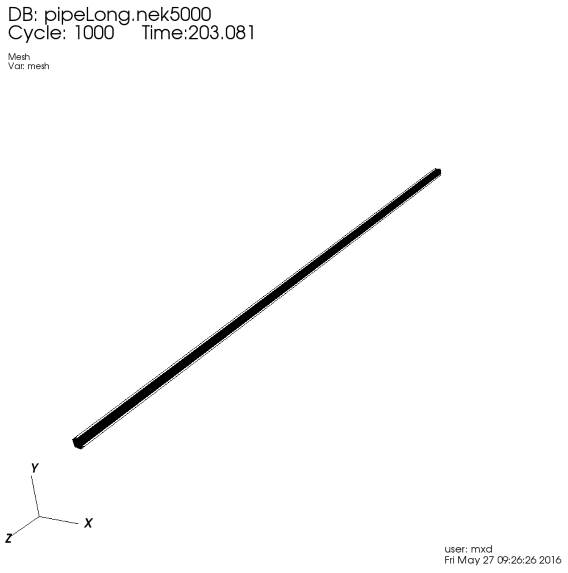
The UQ analysis is performed using the Dakota package [2] that includes methods for parameter studies, optimization, sensitivity analysis, and uncertainty quantification (UQ). Dakota is designed to interface with external packages/codes such as Nek5000 through a script-driven interface called *fork*. Such interface provides maximum flexibility to the users for pre- and post-processing tasks preceding and following each function evaluation. For this study, a given set of input parameters of known uncertainty is supplied to Nek5000 through the Dakota interface, and then the code which corresponds to a function evaluation is run. Outputs of interest, or response functions in UQ terms, are returned to Dakota to determine the outputs' uncertainty. More details regarding the input and output parameters of interest are provided in Section 2.

2. UNCERTAINTY QUANTIFICATION AND SENSITIVITY ANALYSES

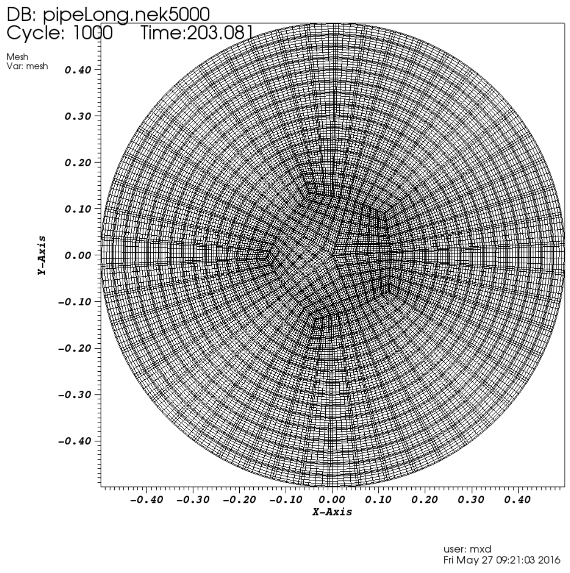
In this section, details regarding the geometry, as well as the input and output parameters, are presented.

2.1 THE MODEL OF INTEREST

Flow of Reynolds number $Re = 31 \cdot 10^3$ in a 3-D pipe with a nondimensional length of 70 in the z-direction is investigated using the code Nek5000; the time-dependent incompressible Navier-Stokes equations are solved on an unstructured mesh of $200,000$ hexagonal elements using fourth-order SEM (see Figs. 1a and 1b). Note that the mesh density is high enough to resolve the eddies.



(a) View of the 3-D pipe.



(b) Mesh density in a slice of the 3-D pipe.

For each time step, an estimate of the velocity profile is obtained through a Helmholtz solver, and then corrected by solving pressure Laplace equations. Inlet and outflow boundary conditions are used to set the inlet velocity profile and the outlet pressure, respectively.

We ran the large eddy simulation (LES) version of Nek5000 and monitor the L_2 error norm between the numerical and experimental data describing the fully developed radial velocity profile. The L_2 norm is computed using the following four-step process (all length are in nondimensional unit):

1. The time-averaged, z-direction velocity is extracted from the Nek5000 output file from a slice at (x,y,z)=(0,0,70) using Visit [1]. This information is collected from 0 to 5 s and stored in a text file along with node coordinates.
2. For each node of the slice S, the distance to the outside wall is computed using the following relation:

$$dist_{\text{to wall}} = \frac{D}{2} - \sqrt{(x^2 + y^2)}\,, \tag{1}$$

where $D = 1$ is the diameter of the pipe in nondimensional unit.

3. Once the variable $dist_{\text{to wall}}$ is obtained, the nodes are gathered in pools using the following rule: a node belongs to a pool if the relative difference between the value $dist_{\text{to wall}}(node)$ and the pool radius r_{pool} is less than a user-supplied tolerance: tol. When a node meets the previous criterion, the corresponding velocity magnitude $V(node)$ is added to the pool velocity V_{pool}. A script representing the above would be:

```
$for$ loop over nodes of slice $S$
$ratio = \frac{|r_{pool} - dist_{\text{to wall}}(node)|}{r_{pool}}$
$if$ ratio $\geq tol$
  node does not belong to pool $r_{pool}$
$else$
  node belongs to pool $r_{pool}$
  $V_{pool} + = V(node)$
$N_{nodes} + = 1$
$endif$
$endfor$
$V_{pool} * = 1./N_{nodes}$
```

   Note that the number of nodes, N_{nodes} added to each pool must be tracked in order to compute the pool's average velocity. The pool radius r_{pool} can be set equal to the coordinates of the experimental data. It is, however, preferred to have a sufficient number of pools such as all nodes from numerical data fall inside a pool. The number of nodes stored in each pool depends on the value of the tolerance tol.

4. The experimental and numerical values of the velocity magnitude are now available at the same coordinates and are used to compute the L_2 norm of the error, as follows:

$$L_2^{error} = \sqrt{\sum_{i}^{N_{pool}} ||V_{pool}(i) - V_{exp}(i)||^2}, \tag{2}$$

   where N_{pool} is the number of pool.

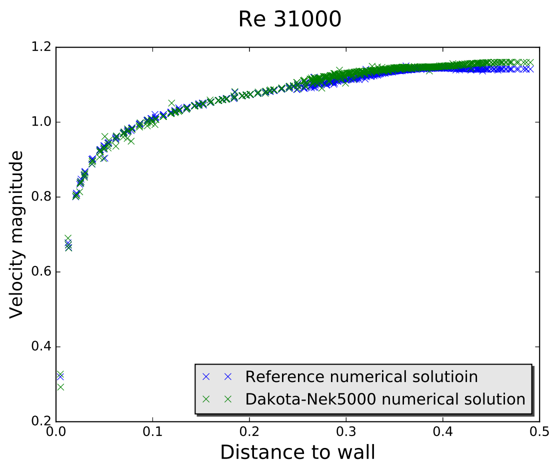
5. The L_2^{error} norm value is returned to Dakota.

The method described above to compute the L_2^{error} error norm between the experimental data and the numerical results is not used in this report but will be the focus of future investigations. The process described above is driven by a Python script (APPENDIX A.) and thus does not require any user intervention besides launching the Dakota input file. Once Dakota has collected all L_2^{error} norms from the Nek5000 runs, a uncertainty quantification or statistics assessment is performed.

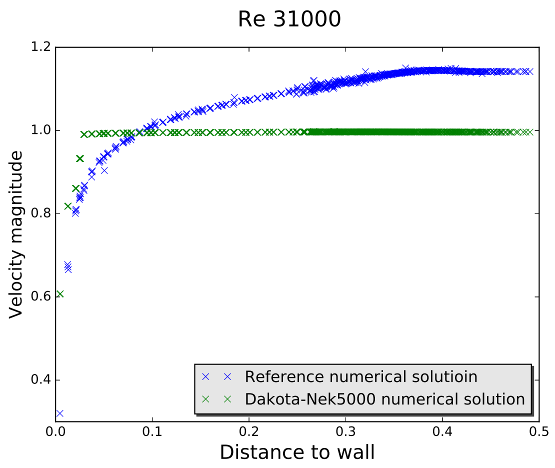
2.2 RESULTS

The results presented in this report should not be used to assess the accuracy of the physical models and the numerical methods implemented in Nek5000, but to demonstrate the interest of integrating Dakota with Nek5000 when simulating engineering flows.

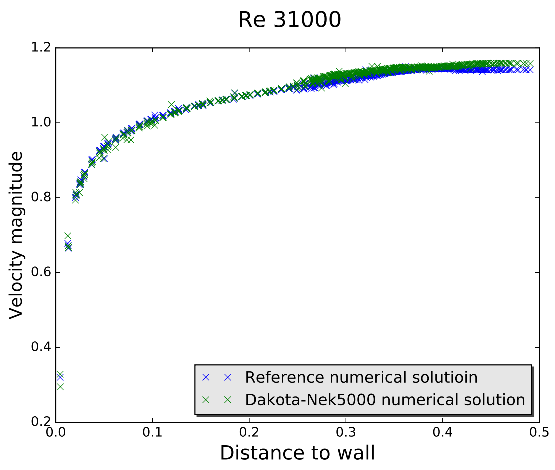
This report presents an initial assessment of the sensitivity of the Nek5000 numerical results for a flow in a 3-D pipe to the spectral filtering input parameters P101 and P103. When turned on, these two parameters control the filtering of modes for LES-type simulations [3]. The parameters $P101$ and $P103$ are varied in the ranges $[0, 6]$ and $[0.05, 0.2]$, respectively, using the *discrete_uncertain_set* method from the Dakota package. The parameters were treated as discrete uncertain variables (real or integer) whose values come from a set of admissible elements. The contributing simulation was allowed to complete $5,000$ time steps on 96 processors in about 5 hours. The time average velocity profile obtained with Nek5000 is plotted against the experimental data in Fig. 2 for different values of the input parameter $P101$. A solution obtained with default input parameters ($P101 = 0$ and $P103 = 0$) in Nek5000 is used as a reference solution to highlight the sensitivity of the numerical simulations when varying the input parameters $P101$ and $P103$.



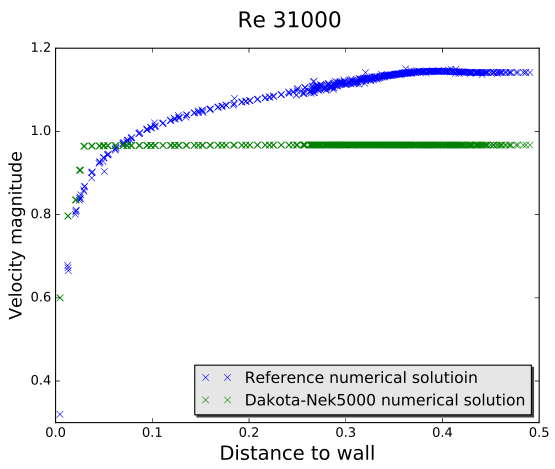
(a) $P101 = 0$ and $P103 = 5 \cdot 10^{-3}$.



(b) $P101 = 2$ and $P103 = 5 \cdot 10^{-3}$.



(c) $P101 = 4$ and $P103 = 5 \cdot 10^{-3}$.



(d) $P101 = 6$ and $P103 = 5 \cdot 10^{-3}$.

Fig. 2. Velocity magnitude profile in a slice located at $(x, y, z) = (0, 0, 70)$ for different values of the input parameters $P101$ and $P103$.

The numerical results presented in Fig. 2 show a strong dependence upon the parameter $P101$, i.e. the number of modes filtered when solving for the turbulent flow in the 3-D pipe. As the number of modes

filtered is increased, the time average velocity profile obtained from Nek5000 diverges from the reference profile. Although it is not shown here, the same behavior is observed when the input parameter $P103$ increases.
The average velocity magnitude profile on section S is shown in Fig. 3. Turbulence flow is well developed and is consistent with the 1-D velocity profiles presented in Fig. 2: the velocity is zero at the wall, and it increases towards the center of the pipe.

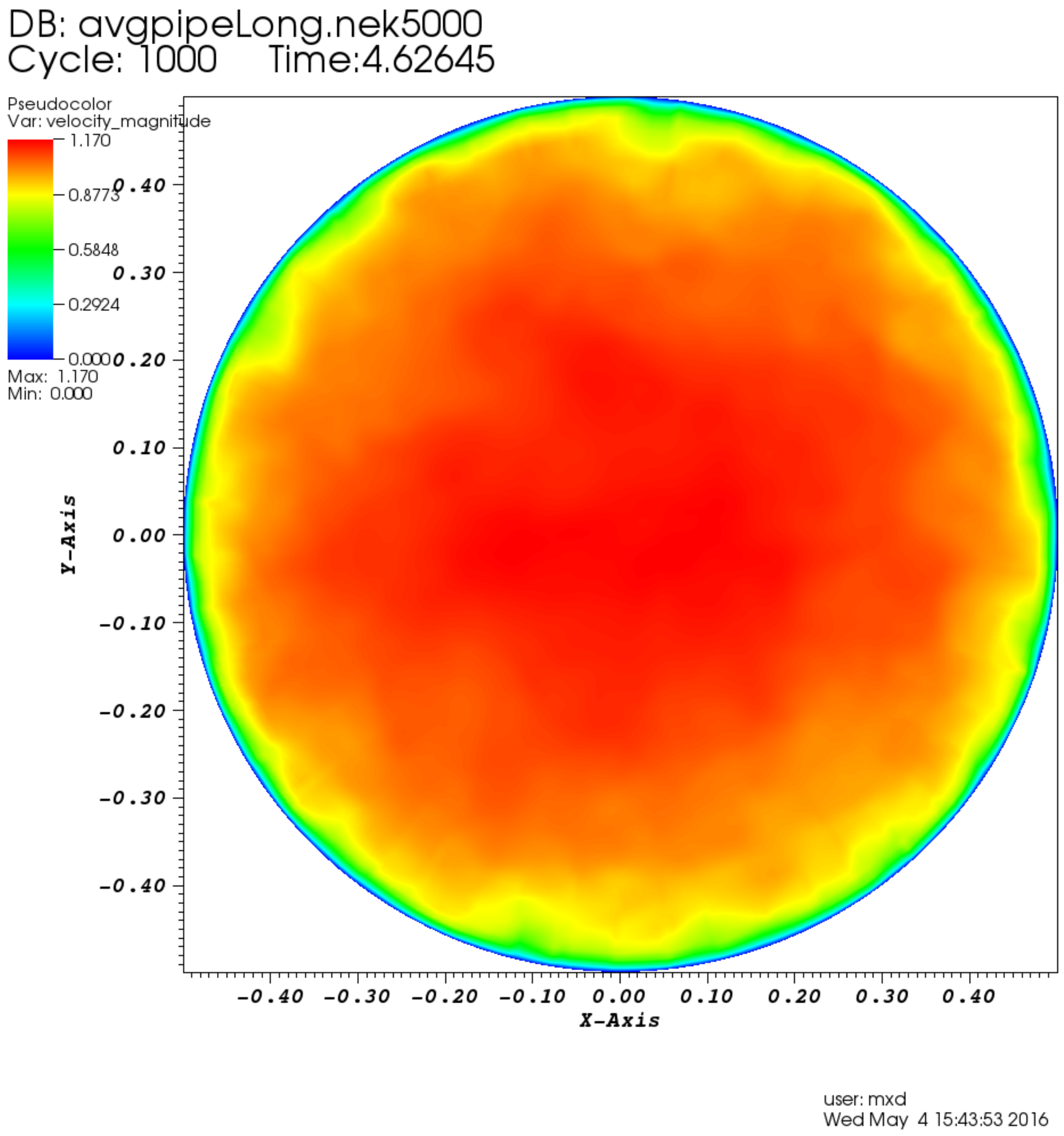


Fig. 3. Time average velocity magnitude profile on slice S.

3. CONCLUDING REMARKS

3.1 OUR ANALYSIS

This report demonstrates the sensitivity of Nek5000 predictions of axial velocity in a 3D pipe to input parameters P101 and P103 and outlines an approach for performing a UQ study using Nek5000 and Dakota that will be used in future work. It is estimated that an average of 100 hours computing time on 200 processors will be needed for further UQ analysis.

3.2 PATH FORWARD

At this stage, the combined use of Dakota and Nek5000 shows promising results. In the future, the influence of other input parameters on the numerical solution, the computing time, and the number of solver iterations, for instance, will be investigated. A calibration study will be also performed for the flow in a 3-D pipe using the experimental data as reference. In the long term, the same study for flow in a 7-pin bundle will be performed.

4. REFERENCES

References

[1] "Computer Codes," Lawrence Livermore National Laboratory, https://wci.llnl.gov/simulation/computer-codes/visit/, accessed June 7, 2016.

[2] B. M. Adams, L. E. Bauman, W. J. Bohnhoff, K. R. Dalbey, M. S. Ebeida, J. P. Eddy, M. S. Eldred, P. D. Hough, K. T. Hu, J. D. Jakeman, J. A. Stephens, L. P. Swiler, D. M. Vigil, and T. M. Wildey. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual*. Technical report, Sandia National Laboratory SAND2014-4633, (2014).

[3] P. Fisher and J. Mullen. Filter-based stabilization of spectral element method. *Numerical Analysis*, 332(Serie I):265–270, 2001.

[4] M. V. Zagarola and A. J. Smits. Mean-flow scaling of turbulent pipe flow. *J. Fluid Mech.*, 373(1998):33–79.

[5] M. V. Zagarola and A. J. Smits. "Scaling of the mean velocity profile for turbulent pipe flow". *Physical review Letter*, 78, 2(1997):349–242.

APPENDIX A. PYTHON SCRIPT

```
# This python script intends to read a data file (format xmd) and post-
   process it. The first part of the code is dedicated to reading and
   storing the data in arrays that can be easily manipulated in a
   second step. The second step is user dependent.

# import python utilities
import sys
import os
import math
import numpy as np
import itertools as it
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
from operator import add
from operator import mul

nek5000_inputfile=sys.argv[1] # Nek5000 input file name
exp_inputfile=sys.argv[2]
dakota_output_file=sys.argv[3]
num=sys.argv[4]
rel_tol=0.02 # relative tolerance in percentage
D=1. # assuming circular pipe of dimater 1

#-----------------------------------------------#
###### read and store data from input file ######
#-----------------------------------------------#
#print {AC_1}
print '-----------------------------------------'
print 'Reading␣and␣storing␣data␣from␣input␣file'
print '-----------------------------------------'
# declare variables
#nek5000_inputfile="avgpipeLong-cycle-2000-point-0-0-30.okc" # Nek5000
   input file name
nek5000_nb_head=9 # number of line to skip before reading Nek5000 data
exp_nb_head=2 # number of line to skip before reading experimental data
head=[] # list storing the head file
num_lines = sum(1 for line in open(nek5000_inputfile, 'r')) # number of
    lines in the file

# read data file
data=open(nek5000_inputfile, 'r')
for i in xrange(0,nek5000_nb_head,1):
  line=data.readline()
  head.append(line.split())

# set and print information from head file
nb_var=int(head[0][0]) # number of variables (including mesh)
print 'Number␣of␣variables:', nb_var
nb_points=int(head[0][1])
nb_phys_var=nb_var-3 # number of physical variables (excluding mesh)
print 'Number␣of␣variables␣excluding␣the␣mesh:', nb_phys_var
var_names=head[1:nb_var+1] # variables names from the input file
print 'Variable␣names:',var_names
max_var=head[nb_var+1:2*nb_var+1] # maximum values for all variables
for i in xrange(0,nb_var,1):
  print 'Minimum␣and␣maximum␣values␣for', var_names[i],':',max_var[i
    ][0:2]

# declare variables to store data
x = [] # x
y = [] # y
z = [] # z
var = [] # variables

# loop over the remaining of the file, make them float and store them
for i in xrange(nek5000_nb_head,num_lines,1):
  line=data.readline()
  row = line.split()
  x.append(float(row[0]))
  y.append(float(row[1]))
  z.append(float(row[2]))
  var.append(map(float,row[3:nb_var+1]))


print '-----------------------------------------------'
print 'Done␣reading␣and␣storing␣data␣from␣input␣file'
print '-----------------------------------------------'

#-----------------------------------------------------------------#
############### post-processing experimental data ###############
#-----------------------------------------------------------------#

print '-----------------------------------'
print 'Post-processing␣experimental␣data'
print '-----------------------------------'
# Read experimental data
exp_dist_to_wall = []
exp_vel = []
```

```
num_lines=sum(1 for line in open(exp_inputfile, 'r')) # number of lines
    in the file
exp_data=open(exp_inputfile, 'r')
for line in xrange(0, exp_nb_head, 1):
  exp_data.readline()
for line in xrange(exp_nb_head,num_lines,1):
  line=exp_data.readline()
  row = line.split()
  exp_vel.append(float(row[0]))
  exp_dist_to_wall.append(float(row[1]))

# Check experimental data consistency
exp_nb_points=len(exp_dist_to_wall)
if exp_nb_points != len(exp_vel):
  print 'ERROR:␣the␣number␣of␣experimental␣nodes␣and␣the␣number␣of␣
     nodal␣velocity␣do␣not␣match.'
  sys.exit()

# Convert list to array
idx=np.argsort(exp_dist_to_wall)
exp_dist_to_wall_temp=np.array(exp_dist_to_wall)[idx]
exp_dist_to_wall=exp_dist_to_wall_temp
exp_vel_temp=np.array(exp_vel)[idx]
exp_vel=exp_vel_temp
nb_exp_points=len(exp_dist_to_wall)

exp_delta_radius_min=max(exp_dist_to_wall)
for point in xrange(0,exp_nb_points-1):
  exp_delta_radius=abs(exp_dist_to_wall[point]-exp_dist_to_wall[point
     +1])
  exp_delta_radius_min=min(exp_delta_radius_min,exp_delta_radius)

print 'Number␣of␣experimental␣points:', nb_exp_points

print '-------------------------------------'
print 'Done␣post-processing␣experimental␣data'
print '-------------------------------------'

#-----------------------------------------------------------#
############### post-processing Nek5000 data ###############
#-----------------------------------------------------------#
print '---------------------------'
print 'Post-processing␣Nek5000␣data'
print '---------------------------'
# set variables
```

```
nb_points=len(x)
print 'Number␣of␣points/nodes␣to␣post-process:',nb_points
print 'Relative␣tolerance␣to␣use:', rel_tol

# declare variables as lists
dist_to_wall = []
flag = [0] * nb_points # set all flags to zero
vel_pool = []
radius_pool = []
nb_nodes_per_pool = []

# Compute the distance to wall for each node of the mesh
for nodes in xrange(0,nb_points,1):
  dist_to_wall.append(max(0.5*D-math.sqrt(x[nodes]**2+y[nodes]**2), 0.)
     )

# Sort the value from min to max in 'dist_to_wall' array
idx_nek=np.argsort(dist_to_wall)
dist_to_wall_temp=np.array(dist_to_wall)[idx_nek]
dist_to_wall=dist_to_wall_temp
var_temp=np.array(var)[idx_nek]
var=var_temp

# Sort the value from min to max in 'dist_to_wall' array
#print var
#idx_nek=np.argsort(var)
#var_temp=np.array(var)[idx_nek]
#var=var_temp
#print var
#dist_to_wall_temp=np.array(dist_to_wall)[idx_nek]
#dist_to_wall=dist_to_wall_temp

if min(dist_to_wall)>min(exp_dist_to_wall):
  print 'WARNING:␣the␣mesh␣domain␣does␣not␣cover␣the␣entire␣
     experimental␣data␣domain'

# find points with the same distance to the wall within the tolerance
# 'rel_tol' and group them in a pool to compute a mean value. Each node
# is flagged so that it only gets accounted for once.
#for nodes in xrange(0,10,1):
vel_pool = []
dist_to_wall_pool = []
nb_nodes_per_pool = []
node=0
while (node<nb_points):
```

```
            vel_pool.append(var[node])
            dist_to_wall_pool.append(dist_to_wall[node])
            nb_nodes_per_pool.append(1)
            if node==nb_points-1:
                    break
            #j=node+1 if node<nb_points-1 else nb_points-1
            j=node+1
            while (j<nb_points and abs(dist_to_wall[node]-dist_to_wall[j])
               < rel_tol*dist_to_wall[node]):
            #while (j<nb_points and abs(var[node]-var[j]) < rel_tol*var[
               node]):
                    vel_pool[-1]+=var[j]
                    dist_to_wall_pool[-1]+=dist_to_wall[j]
                    nb_nodes_per_pool[-1]+=1
                    j+=1
            node=j

# Output information on screen: number of pools
print 'Final␣number␣of␣pools␣after␣eliminating␣pools␣with␣zero␣point:',
    len(vel_pool)

# Convert list to array for easy post-processing
vel_pool=np.asarray(vel_pool)
vel_pool=vel_pool.flatten()
nb_nodes_per_pool=np.asarray(nb_nodes_per_pool)
nb_nodes_per_pool=nb_nodes_per_pool.flatten()
dist_to_wall_pool=np.asarray(dist_to_wall_pool)
dist_to_wall_pool=dist_to_wall_pool.flatten()

# Perform operation on velocity and radius for each pool
vel_pool=np.divide(vel_pool,nb_nodes_per_pool)
dist_to_wall_pool=np.divide(dist_to_wall_pool,nb_nodes_per_pool)

print '--------------------------------'
print 'Done␣post-processing␣Nek5000␣data'
print '--------------------------------'

#--------------------------------------------------------------#
############### Interpolating and computing error ###############
#--------------------------------------------------------------#

print '
   ---------------------------------------------------------------------------
   '
print 'Interpolating␣and␣computing␣L2␣norm␣error␣between␣experimental␣
```

```
    and␣Nek5000␣data'
print '
    ----------------------------------------------------------------------------
    '

# Get Nek5000 data at exoerimental nodes by interpolating
print 'Interpolating␣data␣at', len(exp_dist_to_wall), 'points'
nek5000_vel_pool_at_exp_nodes=np.interp(exp_dist_to_wall,
    dist_to_wall_pool, vel_pool)
nek5000_vel_pool_at_exp_nodes=np.array(nek5000_vel_pool_at_exp_nodes)

# Compute L2 error norm
sri=math.sqrt(np.linalg.norm(exp_vel-nek5000_vel_pool_at_exp_nodes))
print 'Compute␣L2␣error␣norm:', sri

# Saving plot
plt.plot(exp_dist_to_wall, exp_vel, '*', label='experimental␣data')
plt.plot(dist_to_wall_pool, vel_pool, 'x', label='Nek5000␣pool␣data')
plt.plot(exp_dist_to_wall, nek5000_vel_pool_at_exp_nodes, '+', label='
    Nek5000␣interpolated␣data')
legend = plt.legend(loc='lower␣right', shadow=True)
frame = legend.get_frame()
frame.set_facecolor('0.90')
for label in legend.get_texts():
  label.set_fontsize('large')
for label in legend.get_lines():
  label.set_linewidth(1.5)
plt.suptitle('Re␣31000', fontsize=20)
plt.xlabel('Distance␣to␣wall', fontsize=18)
plt.ylabel('Velocity␣magnitude', fontsize=16)
fig_name='Re-31k-'+num+'.eps'
plt.savefig(fig_name, format='eps', dpi=1000)
print 'Saving␣plot␣using␣Matplotlib:', fig_name

output_data_txt_file="velocity-data-"+num+".txt"
with open(output_data_txt_file,"w") as fin:
  header="##␣Nek5000-dist-to-wall"+"\t"+"Nek5000-velocity"+"\t"+"
      intepolated-velocity"+"\t"+"exp-dist-to-wall"+"\t"+"exp-velocity␣
      ##"+"\n"
  fin.write(header)
  for e in it.izip_longest(dist_to_wall_pool, vel_pool,
      nek5000_vel_pool_at_exp_nodes, exp_dist_to_wall, exp_vel,fillvalue
      =''):
    print >>fin,"{0:^8}␣\t␣{1:^8}␣\t␣{2:^8}␣\t␣{3:^8}␣\t␣{4:^8}".format
        (*e)
```

```
print 'Saving␣Nek5000␣data ,␣experimental␣data␣and␣interpolated␣data␣in'
    , output_data_txt_file

print '
    ----------------------------------------------------------------------------------------------------
    '
print 'Done␣interpolating␣and␣computing␣L2␣norm␣error␣between␣
    experimental␣and␣Nek5000␣data'
print '
    ----------------------------------------------------------------------------------------------------
    '

with open(dakota_output_file ,"a+") as f:
        f.write(str(sri))

data.close()
exp_data.close()
sys.exit()
```