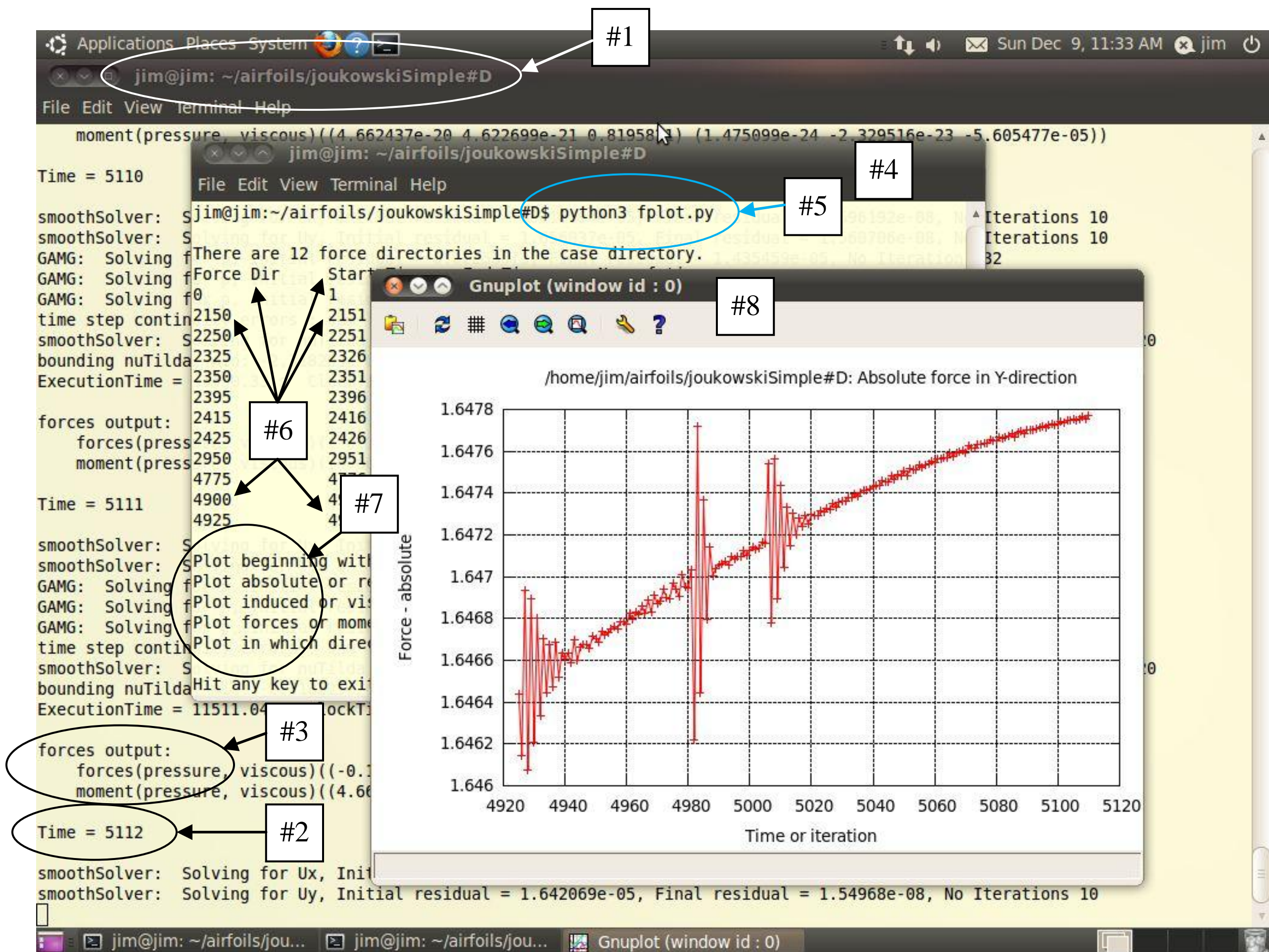
Plotting OpenFoam forces and moments in real time

The following screenshot shows three separate windows on the console. If you are an OpenFoam user, you will recognize the underlying window as the output from an OpenFoam job. The case directory is "/joukowskiSimple#D", which is located inside directory "/airfoils" in my home directory "jim@jim". This information is marked by the arrow #1. OpenFoam is in the midst of iteration #5112 (see arrow #2). The forces and moments are being logged to a file (see arrow #3).

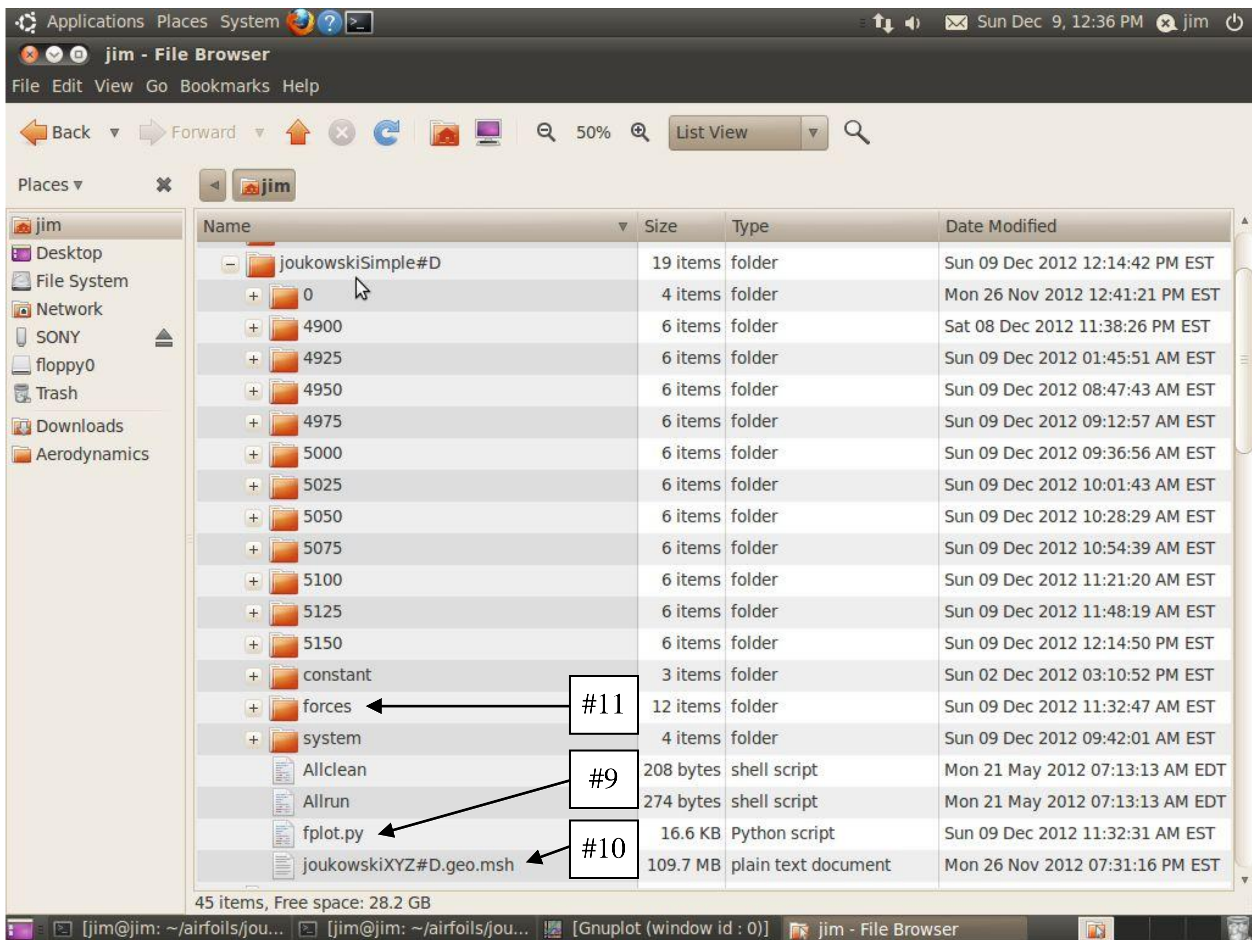
A second terminal has been opened (see textbox #4), and the directory in it changed to the case directory. In this directory, the command line "`python3 fplot.py`" begins execution of a python file named "fplot.py" (see arrow #5). This program – "fplot.py" – begins by summarizing the various force data files which exist, and the starting and ending time/iterations which they contain (see arrow #6). Then, the program asks the user to specify the force or moment and the direction to be plotted (see arrow #7). When that is done, the program produces the third window containing the plot (see textbox #8). The graph selected here shows the absolute values (as opposed to the relative values) of the induced force in the Y-direction. The object being simulated is a one millimeter thickness of an airfoil with a chord of about 1.4 meters, so the values of the force are reported in Newtons (e.g., 1.647 Newtons). Although the values of the force are known from the very first iteration of the job, the user selected to plot only the last two hundred or so. Note that the last iteration plotted is #5111, which was the last completed iteration at the time the user invoked "fplot.py" by typing "`python3 fplot.py`" in response to the prompt.



The graph shows three perturbations, which result from the user changing control parameters in the /system/fvSolution file on the fly. It is very handy to be able to see such consequences in real time. It is indeed unfortunate that OpenFoam is available only under Linux, since the plotting must therefore be done in Linux, too. The great disadvantage of Linux is that nothing can be done without delving into the operating system. A further disadvantage is that the development tools available for Linux are, at best, archaic. I hope that my instructions are complete enough to help you navigate through the shoals.

A quick overview of the "forces" subdirectory

The plotting procedure is contained in a single program, named "fplot.py", which resides in the case directory. The following screenshot shows the case directory "joukowskiSimple#D". Arrow #9 shows that the program "fplot.py" is simply one more file inside the case directory. The only other file inside this case directory, which is not a standard OpenFoam file, is the one identified by arrow #10. It is named "joukowskiXYZ#D.geo.msh" and has nothing to do with this plotting procedure. It contains the description of the mesh which I produced for this application using GMesh. Alert readers will notice that OpenFoam has continued to run since the screenshot above was taken – it is now executing an iteration after #5150.



OpenFoam is logging forces and moments into the subdirectory named "/forces" (see arrow #11). Just so that we have the same information, OpenFoam was instructed to log forces and moments by making two changes in the "/system" directory, adding a command to the "/system/controlDict" file and adding a file to the "/system" directory. The following screenshot shows the four files in the "/system" directory.

The first change is to add a command to the "/system/controlDict" file, which will cause it to execute the added file as a function.

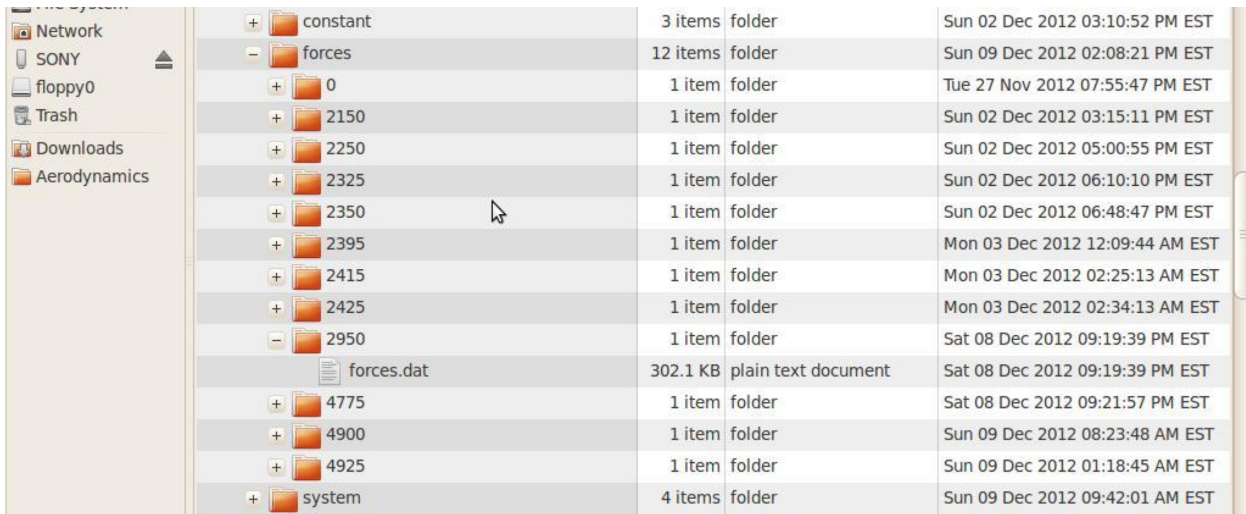
```
functions
{
     #include "ForceAndMoment"
}
```

This command causes OpenFoam to execute the file "ForceAndMoment" at the end of every iteration. The added file "ForceAndMoment" (see arrow #12) has the following contents.

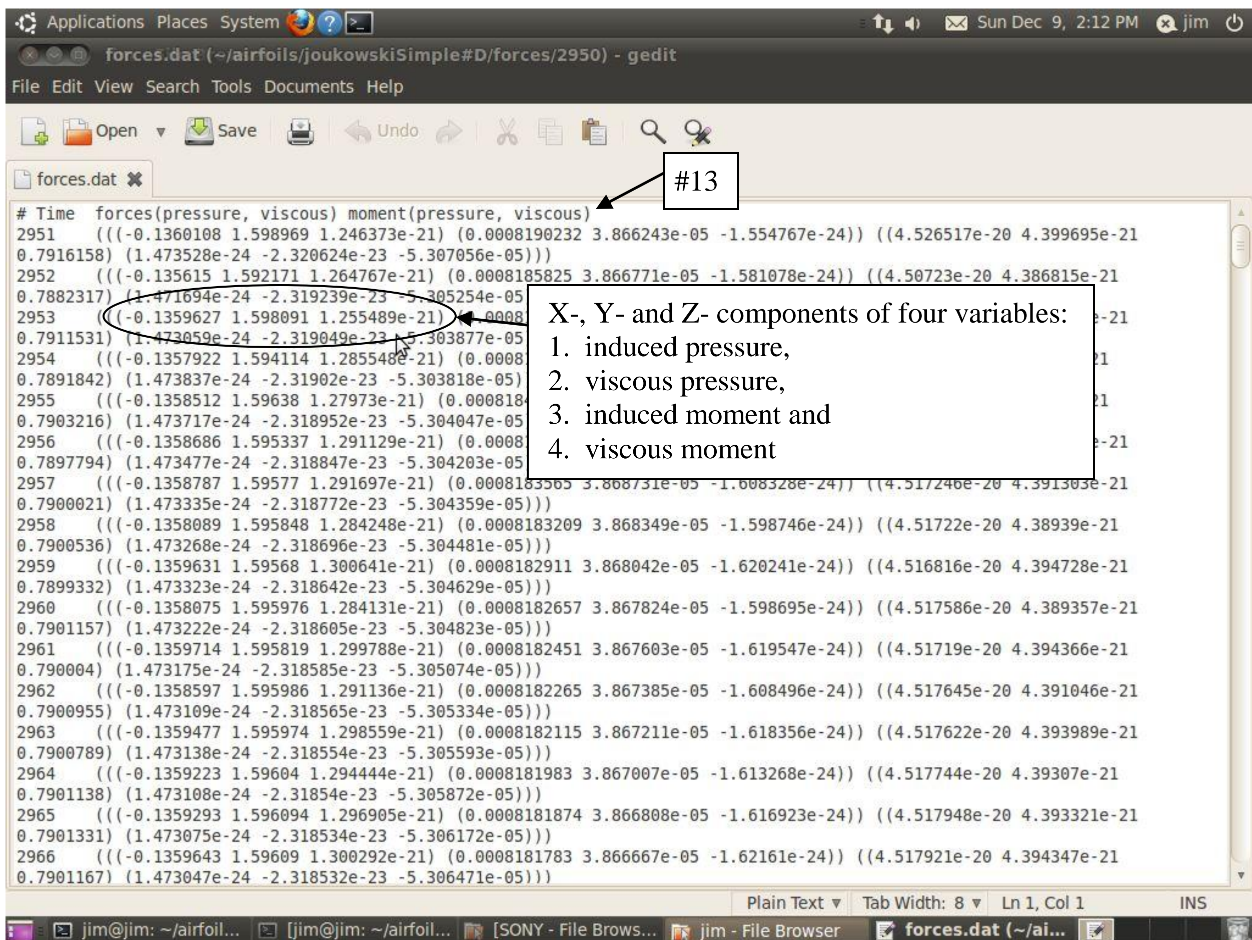
```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
forces
{
    type                    forces;
    functionObjectLibs      ( "libforces.so" );
    outputControl           timeStep;
    outputInterval          1;
    patches                 ( "airfoil.*" );
    pName                   p;
    UName                   U;
    rhoName                 rhoInf;
    log                     true;
    CofR                    ( 0 0 0.0005 );
    rhoInf                  1.225;
}
```

In my case, the forces are those which act on the patch named "airfoil". Note that the moment is calculated around the point which is one-half of the Z-thickness and that the air density is stated at sea level. Of course, you will have to change the contents of this file so that it calculates the forces and moments of interest in your application.

In any event, when OpenFoam logs the forces and moments, they are recorded in time-stamped files contained inside the "/forces" directory. The typical content of a "/forces" directory is shown in the following screenshot.



Every time you restart execution of an OpenFoam job, a new directory is created in the "/forces" directory, with the name being the starting time or iteration number. For example, the directory "/forces/2950" was created at the start of iteration #2951. Each time-stamped directory contains a single text file, named "forces.dat". The following screenshot shows the starting lines in the file "/forces/2950/forces.dat".



The first line is a text header (see arrow #13) describing the contents of the 13 columns of data. Each subsequent line contains the time or iteration number, followed by 12 force and moment values. The OpenFoam job will continue to add lines to this "/forces/2950/forces.dat" file until execution is halted. If, and when, execution is restarted, OpenFoam will create a new directory with the then-time stamp and begin to enter force and time data into that new file.

It follows that the "/forces" directory will contain an unknown number of time-stamped subdirectories. If the user has not deleted or changed any of the subdirectories or their "forces.dat" files, then the collection of all the "forces.dat" files will contain an iteration-by-iteration list of the force and moment data from the commencement of the job.

However, the user can delete or edit the time-stamped folders in the "/forces" directory. He can also change the contents of the "forces.dat" files themselves. For example, the user can manually edit the "forces.dat" files to merge the lists. We are going to have the plotting program do the merging for us.

A quick overview of the "fplot.py" procedure

The "fplot.py" program works through five main steps:
1. Organize the force and moment data and store it in vectors.
2. Get plotting instructions from the user.
3. Create a temporary data file with the data to be plotted, called "plotfile.dat".
4. Create a temporary gnuplot script file, called "gnuscript.txt".
5. Execute the gnuplot script file and then delete the two temporary files.

Step #1 – Organize the force and moment data

The program reviews the names of the time-stamped folders in the "/forces" directory in order to establish their sequence. It relies on the names. It assumes, for example, that the file named "forces/2350/forces.dat" contains data from times/iterations prior to those in the file named "forces/3725/forces.dat". The program does not care if files are missing or mislabeled but, if they are, the data will be plotted out of sequence.

The program reads the data from all the "forces.dat" files and stores it in 13 vectors (called "lists" in Python). The vectors have obvious names: Time[], FXinduced[], FYinduced[], …, MZviscous[].

The program reads the data and closes the "forces.dat" files as quickly as possible. Since the OpenFoam job will probably be executing when "fplot.py" is run, potential conflict over the use of the current "forces.dat" file is minimized by closing it as soon as the data has been read.

Step #2 – Get instructions from the user

The program will display a summary of the "forces.dat" files like that shown in the following screenshot. As described before, the user invokes the program by: (i) changing the current working directory to the case directory (see arrow #14) and then (ii) typing the command line "`python3 fplot.py`" (see arrow #15). In the case illustrated, there are 12 time-stamped subdirectories in the "/forces" directory. It can be seen that the user has deleted a number of files created earlier in the job. There is a gap, for example, between iteration #109 and iteration #2150 (see arrow #16). There is also a bit of back-tracking (see arrow #17). Folder "4775" accumulated force and moment data up until iteration #4912" but the following folder, "4900", begins at iteration #4901. [How did this happen? The "/system/controlDict" is set to record the complete state every 25 iterations. A complete state was therefore written at iteration #4900. Shortly thereafter, the user (me) decided to change a parameter in the "/system/fvSolution" file.

But, the parameter change did not work out, which is to say, that the solution began to diverge. I halted execution just after iteration #4912, and re-started from the last complete state, being iteration #4900.]

```
jim@jim: ~/airfoils/joukowskiSimple#D
File Edit View Terminal Help
jim@jim:~$ cd airfoils
jim@jim:~/airfoils$ cd joukowskiSimple#D
jim@jim:~/airfoils/joukowskiSimple#D$ python3 fplot.py

There are 12 force directories in the case directory.
Force Dir       Start Time      End Time        No. of times
0               1               109             109
2150            2151            2257            107
2250            2251            2325            75
2325            2326            2350            25
2350            2351            2398            48
2395            2396            2416            21
2415            2416            2425            10
2425            2426            2950            525
2950            2951            4795            1845
4775            4776            4912            137
4900            4901            4925            25
4925            4926            5326            401

Plot beginning with time/iteration number [1-5326]:
```

#14 #15 #16 #17

The plot will begin at the time/iteration typed by the user in response to the prompt shown and will continue up until the end of the data. Then, the program will ask the following four questions.

```
Plot beginning with time/iteration number [1-5348]: 4775
Plot absolute or relative values [a/r]: a
Plot induced or viscous results. [i/v]: i
Plot forces or moments. [f/m]: f
Plot in which direction? [x/y/z]: y
```

"Absolute" does not refer to the mathematical function. It refers instead to the unadjusted force and moment values calculated by OpenFoam. The alternate – "relative" – values are the percentage differences between the absolute values and the average of the absolute values over the range of times/iterations being plotted. The other three questions determine which of the 12 variables will be plotted.

The user's response can be either in small case or in upper case. In the event of an invalid response, the program will assume a default value.

Step #3 – Create a temporary data file

From this point onwards, program "fplot.py" proceeds without any further input from the user. In this Step #3, the program creates a temporary text file. The temporary text file has the name "plotfile.dat" and

is created inside the “/forces” directory. This text file consists of two columns of numbers. The first column contains the times/iterations in the range selected and the second column contains the value to be plotted on the vertical axis. There are no column or row headers.

Step #4 – Create a temporary Gnuplot script file

The program creates a second temporary file as well, with the name “gnuscript.txt”. This is also a text file and is also created in the “/forces” directory. This file is a so-called “script”. It contains a list of commands which will be executed by Gnuplot. In many ways, “gnuscript.txt” resembles a Windows batch file like autoexec.bat.

Gnuplot is a general purpose plotting program designed for use with Linux.

Step #5 – Execute the Gnuplot script file

“fplot.py” then executes Gnuplot as a subprocess. The file “gnuscript.txt” contains the instructions for the Gnuplot interpreter and the file “plotfile.dat” contains the values it will plot. The titles and labels which Gnuplot uses to format the plot were coded into the script file when it was built during Step #4.

Step #6 – Delete the two temporary files

The window containing the plot will persist until it is closed when the user clicks on the “X” button at the upper left corner. To clean up its mess, the program will delete the two temporary files “plotfile.dat” and “gnuscript.txt”. If at any time you discover files with these names in your “/forces” directory, they can be deleted without concern. In fact, the program deletes any old copies it finds before creating new ones.

A listing of the program “fplot.py”

In this section, I give a listing of the program “fplot.py”. It is a text file, meaning that it can be edited with any text editor, including “gedit” on Linux systems and “Notepad” on Windows systems. Although it is simply a text file, it has the syntax of a Python program, hence the extension “.py”. Python is a programming language which resembles BASICA from the mid-1980s. It is an interpreter, which means that it works its way through the lines in real time, executing them one-by-one.

[Aside for a brief rant: I do not worship Windows by any means, but I know a good tool when I find one. Microsoft’s Visual Basic is a good tool. Python is not nearly as good. Based on my recent internet searching, it seems that there was at one time an attempt by the Linux folks to build an interface which would run Windows executable programs (those with the extension “.exe”) on Linux machines. I think it was called “Wine”. It was a great idea, but constant upgrades by Microsoft left Wine far behind. It looks like another attempt was made, somewhat later, which would run Visual Basic programs on Linux machines. This effort, called “Mono”, seems to be well along its way to history, too. There seems to have been an heroic effort by a French man to produce from scratch a Linux version of Visual Basic, called “GamBas”. A commendable objective, I must say, but too much for one man. None of my efforts to download and install GasBas were successful. There must be a “C” or “C++” implementation for Linux somewhere – after all, OpenFoam is written in “C” – but I could not find a complete and functional package within the time I cared to devote to the task. That pretty much left Python, which can do the job I want it to do, but not nearly as well as one might hope. But, enough ranting, here is the listing.]

is a comment in Python.

```
# This is a Python program designed to plot forces and moments from the
# forces.dat files produced by OpenFoam.
#
# Assumptions:
# 1. That this file is named "fplot.py";
# 2. That this file is located in the case directory;
# 3. That /system/controlDict writes forces into a directory named /forces;
# 4. That an OpenFoam job is running in a terminal window.
#
# Invocation:
# 1. Open a second terminal window;
# 2. Change directory to the case directory;
# 3. Type the command "python3 fplot.py".

# File names.
InputFileName = "./forces.dat"  # A file containing force and moment data
PlotFileName = "./plotfile.dat" # The file with data selected for plotting
GnuFileName = "./gnuscript.txt" # The file with gnuplot script

# Define variables.
ForceTimeDirNames = []     # List of time-dirs containing force data files
NumForceTimeDirNames = 0   # Number of time-dirs containing force data files
StartTimesInDir = []       # The first time/iteration in a force data file
EndTimesInDir = []         # The last time/iteration in a force data file
NumTimesInDir = []         # Number of times/iterations in a force data file

Time = []                  # Vectors to hold original force and moment data
FXinduced = []
FYinduced = []
FZinduced = []
FXviscous = []
FYviscous = []
FZviscous = []
MXinduced = []
MYinduced = []
MZinduced = []
MXviscous = []
MYviscous = []
MZviscous = []

PlotTime = []              # Vectors to hold the X-Y values to be plotted
PlotValue = []

StartPlottingAtTime = 0    # Time/iteration at which plotting starts
PlotAbsOrRel = True        # True for absolute; False for relative
PlotIndOrVis = True        # True for induced; False for viscous
PlotForOrMom = True        # True for forces; False for moments
PlotX = True               # True to plot X-direction values
PlotY = False              # True to plot Y-direction values
PlotZ = False              # True to plot Z-direction values

# Step #1 - Get the name of the case directory.
import os
CaseDir = os.getcwd()
```

This syntax defines vectors with a zero-based index.

cwd is the current working directory.

```
# Step #2 - Change to the directory containing OpenFoam's force data files.
os.chdir('forces')
ForceTopDir = os.getcwd()
```

Indents are very important in Python.
Multiples of three blanks are used here.

```
# Step #3 - Make a list of the time-directories containing force data files.
for name in os.listdir("."):
   if os.path.isdir(name):
      ForceTimeDirNames.append(name)
NumForceTimeDirNames = len(ForceTimeDirNames)
if (NumForceTimeDirNames == 0):
   print("Error: No force data files were found.")
   import sys
   sys.exit()
else:
   print("\n" + \
      "There are " + str(NumForceTimeDirNames).strip() + \
      " force directories in the case directory.")
```

A “range” does not include the last number.

```
# Step #4 - Put the list of time-directories into chronological order.
#           Use a bubble-type sorting procedure.
if (NumForceTimeDirNames >= 2):
   for Index1 in range(1, NumForceTimeDirNames):
      for Index2 in range(Index1 + 1, NumForceTimeDirNames + 1):
         if (int(ForceTimeDirNames[Index1 - 1]) > \
            int(ForceTimeDirNames[Index2 - 1])):
            TempName = ForceTimeDirNames[Index1 - 1]
            ForceTimeDirNames[Index1 - 1] = ForceTimeDirNames[Index2 - 1]
            ForceTimeDirNames[Index2 - 1] = TempName

# Step #5 - Open each force data file and parse the data into vectors.
for Index in range(NumForceTimeDirNames):
   #
   # Step #5A - Change to the appropriate force directory.
   os.chdir(ForceTimeDirNames[Index])
   #
   # Step #5B - Open the force data file.
   try:
      FileNumber = open(InputFileName, "r")
   except:
      print("Error: Could not open file " + \
         str(ForceTimeDirNames[Index].strip()) + \
         InputFileName + ".")
      import sys
      sys.exit()
   #
   # Step #5C - Parse out the data in each line.
   LineNumber = 0
   import re
   for OriginalLine in FileNumber:
      # Do not process blank lines.
      if (len(OriginalLine) > 1):
         # Do not process the first line, with a header including "Time".
         if (OriginalLine.find("Time") < 0):
            LineNumber = LineNumber + 1
            # Replace all occurrences of "(" with a blank space.
            ModifiedLine = OriginalLine.replace("(", " ")
```

```
# Replace all occurrences of ")" with a blank space.
ModifiedLine = ModifiedLine.replace(")", " ")
# Replace each white space with a single blank space.
re.sub("\s\s+" , " ", ModifiedLine)
# Read and store the iteration/time number.
ModifiedLine.lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
Time.append(TempString)
# If this is the first line, make a note of the start time.
if (LineNumber == 1):
   StartTimesInDir.append(TempString)
# Read and store FXinduced.
ModifiedLine = parts[1].lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
FXinduced.append(TempString)
# Read and store FYinduced.
ModifiedLine = parts[1].lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
FYinduced.append(TempString)
# Read and store FZinduced.
ModifiedLine = parts[1].lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
FZinduced.append(TempString)
# Read and store FXviscous.
ModifiedLine = parts[1].lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
FXviscous.append(TempString)
# Read and store FYviscous.
ModifiedLine = parts[1].lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
FYviscous.append(TempString)
# Read and store FZviscous.
ModifiedLine = parts[1].lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
FZviscous.append(TempString)
# Read and store MXinduced.
ModifiedLine = parts[1].lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
MXinduced.append(TempString)
# Read and store MYinduced.
ModifiedLine = parts[1].lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
MYinduced.append(TempString)
# Read and store MZinduced.
ModifiedLine = parts[1].lstrip()
parts = ModifiedLine.split(" ",1)
TempString = parts[0].strip()
MZinduced.append(TempString)
```

“Split” breaks a string into two parts.

```
            # Read and store MXviscous.
            ModifiedLine = parts[1].lstrip()
            parts = ModifiedLine.split(" ",1)
            TempString = parts[0].strip()
            MXviscous.append(TempString)
            # Read and store MYviscous.
            ModifiedLine = parts[1].lstrip()
            parts = ModifiedLine.split(" ",1)
            TempString = parts[0].strip()
            MYviscous.append(TempString)
            # Read and store MZviscous.
            ModifiedLine = parts[1].lstrip()
            parts = ModifiedLine.split(" ",1)
            TempString = parts[0].strip()
            MZviscous.append(TempString)
   #
   # Step #5D - Make a note of the end time.
   LastEntry = len(Time)
   EndTimesInDir.append(Time[LastEntry - 1])
   #
   # Step #5E - Close the input force data file.
   FileNumber.close()
   #
   # Step #5F - Return to the parent directory.
   os.chdir("..")

# Step #6 - Describe the data files to the user.
print("Force Dir     Start Time    End Time      No. of times")
for Index1 in range(NumForceTimeDirNames):
   OutputString = ForceTimeDirNames[Index1]
   for Index2 in range(len(OutputString), 14):
      OutputString = OutputString + " "
   OutputString = OutputString + StartTimesInDir[Index1]
   for Index2 in range(len(OutputString), 28):
      OutputString = OutputString + " "
   OutputString = OutputString + EndTimesInDir[Index1]
   for Index2 in range(len(OutputString), 42):
      OutputString = OutputString + " "
   NumTimes = 1 + int(EndTimesInDir[Index1]) - int(StartTimesInDir[Index1])
   OutputString = OutputString + str(NumTimes).strip()
   print(OutputString)

# Step 7 - Prompt the user to select the number of times/iterations.
EarliestStartTime = int(StartTimesInDir[0])
LatestStartTime = int(EndTimesInDir[NumForceTimeDirNames - 1])
OutputString = "\n"\
   + "Plot beginning with time/iteration number ["\
   + str(EarliestStartTime).strip() + "-"\
   + str(LatestStartTime).strip() +"]: "
Response = input(OutputString)
StartPlottingAtTime = int(Response)
if (StartPlottingAtTime < EarliestStartTime):
   StartPlottingAtTime = EarliestStartTime
   print("Error: Start time is out of range. "\
      + "Using " + str(EarliestStartTime).strip + ".")
if (StartPlottingAtTime > LatestStartTime):
   StartPlottingAtTime = LatestStartTime
```

Add entries to a vector by "appending" them.

Use an ending backslash for line continuation.

```
   print("Error: Start time is out of range. "\
      + "Using " + str(LatestStartTime).strip() + ".")

# Step #8 - Prompt the user to select absolute or relative values.
OutputString = "Plot absolute or relative values [a/r]: "
Response = input(OutputString)
if ((Response == "a") or (Response == "A")):
   PlotAbsOrRel = True
else:
   if ((Response == "r") or (Response == "R")):
      PlotAbsOrRel = False
   else:
      print("Invalid response. Will plot absolute values.")
      PlotAbsOrRel = True

# Step #9 - Prompt the user to select induced or viscous results.
OutputString = "Plot induced or viscous results [i/v]: "
Response = input(OutputString)
if ((Response == "i") or (Response == "I")):
   PlotIndOrVis = True
else:
   if ((Response == "v") or (Response == "V")):
      PlotIndOrVis = False
   else:
      print("Invalid response. Will plot induced values.")
      PlotIndOrVis = True

# Step #10 - Prompt the user to select forces or moments.
OutputString = "Plot forces or moments [f/m]: "
Response = input(OutputString)
if ((Response == "f") or (Response == "F")):
   PlotForOrMom = True
else:
   if ((Response == "m") or (Response == "M")):
      PlotForOrMom = False
   else:
      print("Invalid response. Will plot forces.")
      PlotForOrMom = True

# Step #11 - Prompt the user to select the desired direction.
OutputString = "Plot in which direction? [x/y/z]: "
Response = input(OutputString)
if ((Response == "x") or (Response == "X")):
   PlotX = True
   PlotY = False
   PlotZ = False
else:
   if ((Response == "y") or (Response == "Y")):
      PlotX = False
      PlotY = True
      PlotZ = False
   else:
      if ((Response == "z") or (Response == "Z")):
         PlotX = False
         PlotY = False
         PlotZ = True
      else:
```

```
         print("Invalid response. Will plot X-values.")
         PlotX = True
         PlotY = False
         PlotZ = False

# Step #12 - Transfer the selected data items into new vectors.
del PlotTime[:]
del PlotValue[:]
for Index in range(len(Time)):
   if (int(Time[Index]) >= StartPlottingAtTime):
      PlotTime.append(Time[Index])
      # Block to plot forces.
      if (PlotForOrMom == True):
         if ((PlotIndOrVis == True) and (PlotX == True)):
            PlotValue.append(FXinduced[Index])
         if ((PlotIndOrVis == True) and (PlotY == True)):
            PlotValue.append(FYinduced[Index])
         if ((PlotIndOrVis == True) and (PlotZ == True)):
            PlotValue.append(FZinduced[Index])
         if ((PlotIndOrVis == False) and (PlotX == True)):
            PlotValue.append(FXviscous[Index])
         if ((PlotIndOrVis == False) and (PlotY == True)):
            PlotValue.append(FYviscous[Index])
         if ((PlotIndOrVis == False) and (PlotZ == True)):
            PlotValue.append(FZviscous[Index])
      # Block to plot moments.
      else:
         if ((PlotIndOrVis == True) and (PlotX == True)):
            PlotValue.append(MXinduced[Index])
         if ((PlotIndOrVis == True) and (PlotY == True)):
            PlotValue.append(MYinduced[Index])
         if ((PlotIndOrVis == True) and (PlotZ == True)):
            PlotValue.append(MZinduced[Index])
         if ((PlotIndOrVis == False) and (PlotX == True)):
            PlotValue.append(MXviscous[Index])
         if ((PlotIndOrVis == False) and (PlotY == True)):
            PlotValue.append(MYviscous[Index])
         if ((PlotIndOrVis == False) and (PlotZ == True)):
            PlotValue.append(MZviscous[Index])
NumValuesToPlot = len(PlotTime)
```

"Len" is defined for lists as well as strings.

```
# Step #12 - Convert to relative values, if selected.
if (PlotAbsOrRel == False):
   #
   # Step #12A - Calculate the total of the values.
   TotalValue = 0
   for Index in range(NumValuesToPlot):
      TotalValue = TotalValue + float(PlotValue[Index])
   #
   # Step #12B - Calculate the average value.
   AverageValue = TotalValue / NumValuesToPlot
   #
   # Step #12C - Calculate the relative value percentages.
   for Index in range(NumValuesToPlot):
      PlotValue[Index] = str((float(PlotValue[Index]) - AverageValue)\
         * 100 / AverageValue)
```

This is Python's Val() function.

```
# Step #13 - Write the selected data into the temporary plot file.
#
# Step #13A - Delete any existing temporary plot file.
try:
   os.remove(PlotFileName)
except:
   if (False):
      print("") # Do nothing if there is no old plot file.
#
# Step #13B - Open a new temporary file to hold the data to be plotted.
try:
   FileNumber = open(PlotFileName, "w")
except:
   print("Error: Could not open file " + \
      str(ForceTimeDirNames[Index].strip()) + \
      PlotFileName + ".")
   import sys
   sys.exit()
#
# Step #13C - Write the data.
for Index in range(NumValuesToPlot):
   FileNumber.write(PlotTime[Index] + " " + PlotValue[Index] + "\n")
#
# Step #13D - Close the temporary plot file.
FileNumber.close()

# Step #14 - Prepare a script file for gnuplot.
#
# Step #14A - Delete any existing gnuplot file.
try:
   os.remove(GnuFileName)
except:
   if (False):
      print("") # Do nothing if there is no old gnuplot file.
#
# Step #14B - Open a temporary file for the script to control gnuplot.
FileNumber = open(GnuFileName, "w")
#
# Step #14C - Organize and write the gnuplot script.
FileNumber.write("# GnuPlot script file\n")
FileNumber.write("set autoscale # scale axes automatically\n")
FileNumber.write("unset log     # remove any loraithmic scaling\n")
FileNumber.write("unset label   # remove any previous labels\n")
FileNumber.write("set xtic      # set xtics automatically\n")
FileNumber.write("set ytic auto # set ytics automatically\n")
FileNumber.write("unset key\n")
FileNumber.write("unset mouse\n")
FileNumber.write("set grid\n")
OutputString = CaseDir + ": "
if (PlotAbsOrRel == True):
   OutputString = OutputString + "Absolute "
else:
   OutputString = OutputString + "Relative "
if (PlotIndOrVis == True):
   OutputString = OutputString + "induced "
else:
   OutputString = OutputString + "viscous "
```

Python will not accept an empty block of code. Use a dummy.

These are all GnuPlot commands.

```
if (PlotForOrMom == True):
   OutputString = OutputString + "force "
else:
   OutputString = OutputString + "moment "
if (PlotX == True):
   OutputString = OutputString + "in X-direction"
else:
   if (PlotY == True):
      OutputString = OutputString + "in Y-direction"
   else:
      OutputString = OutputString + "in Z-direction"
FileNumber.write("set title '" + OutputString + "'\n")
FileNumber.write("set xlabel 'Time or iteration'\n")
if (PlotForOrMom == True):
   OutputString = "Force - "
else:
   OutputString = "Moment - "
if (PlotAbsOrRel == True):
   OutputString = OutputString + "absolute"
else:
   OutputString = OutputString + "relative"
FileNumber.write("set ylabel '" + OutputString + "'\n")
OutputString = "plot '" + PlotFileName + "' with linespoints"
FileNumber.write(OutputString)
#
# Step #14D - Close the gnuplot script file.
FileNumber.close()

# Step #15 - Execute the gnuplot script.
import os
import subprocess
GnuPlotProcess = subprocess.Popen("gnuplot -persist " + GnuFileName,\
   shell = True)
os.waitpid(GnuPlotProcess.pid, 0)

# Step #16 - Wait for the user to press any key.
response = input("\nHit any key to exit.")

# Step #17 - Delete the two temporary files.
import os
os.remove(PlotFileName)
os.remove(GnuFileName)

# Step #18 - Exit.
import sys
sys.exit()
```

This is where the program executes the Gnuplot script.

The plot will not stay visible unless “persisted”.

In order to create this program, use a text editor to type in the lines, or cut-and-paste. The file must be saved with the extension ”.py” or, alternatively, the extension changed to “.py” before use. Lastly, the file must be saved into or copied into the case directory of interest.

Finicky operating system stuff #1

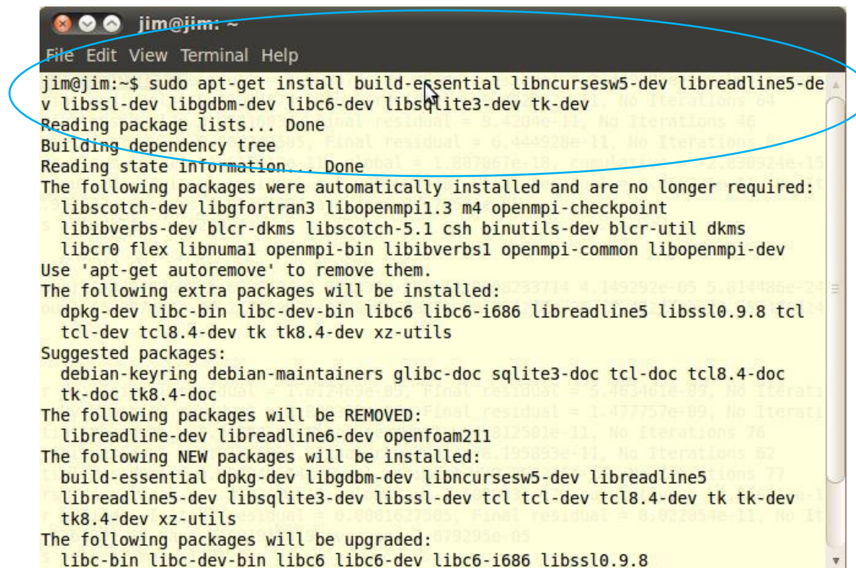
In order for Python to run, it must be installed on your Linux computer. Your Linux installation may or may not have arrived with a default version of Python. Regrettably, the Python folks are as busy as bees and revise their baby constantly. However, because Python is not very advanced, the revisions are significant and usually fatal to programs written for earlier versions. What works in one version of Python will not work in another. Be advised that the version of Python I used to develop this application is Python-3.3.0.

The Linux folks keep pretty busy, too, and there are umpteen versions of their baby as well. I use Ubuntu, which I believe is version 10.04 of Linux. However, I do not expect that your version of Linux will affect your ability to use this application – the version of Python is more important.

I used the following procedure to download and install Python-3.3.0, and then to convince Linux to find and execute it. Start up a new terminal window – the prompt I get is "jim@jim:~$", but your prompt will be different.

Step #1 – Download a bunch of stuff

Type in the line shown in the following screenshot, which begins with "`sudo apt-get install build-essential` " and ends with " `tk-dev`". You may be asked to enter the administrator's password. This command will download and install eight library-type things. I have no idea what these things do, why they are needed or even whether they are needed at all. The list of instructions from which I took guidance seemed to say that these files are needed to compile Python. But, I thought Python was an interpreter …



Step #2 – Download Python-3.3.0

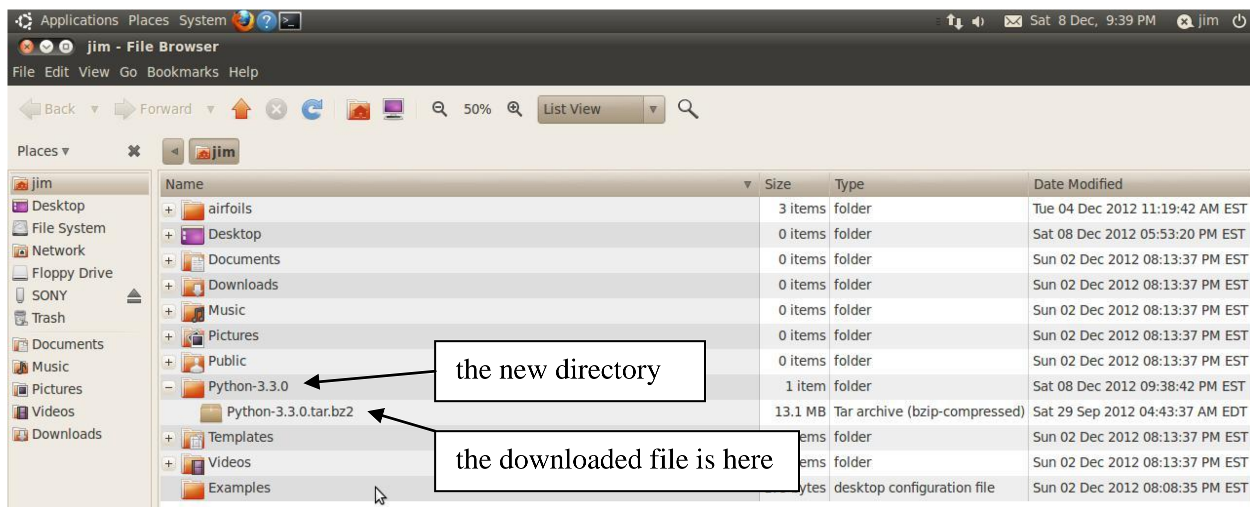
When the processing in Step #1 is finished, download Python-3.3.0 from the python.org website using the command "`wget http://www.python.org/ftp/python/3.3.0/Python-3.3.0.tar.bz2`", as shown in the following screenshot.

```
jim@jim: ~
File  Edit  View  Terminal  Help
Ignoring install-info called from maintainer script
The package libreadline5-dev should be rebuilt with new debhelper to get trigger
 support
Ignoring install-info called from maintainer script
The package libreadline5-dev should be rebuilt with new debhelper to get trigger
 support

Setting up libsqlite3-dev (3.6.22-1) ...
Setting up libssl-dev (0.9.8k-7ubuntu8.13) ...
Setting up tcl (8.4.16-2) ...
update-alternatives: using /usr/bin/tclsh-default to provide /usr/bin/tclsh (tcl
sh) in auto mode.

Setting up tcl8.4-dev (8.4.19-4) ...
Setting up tcl-dev (8.4.16-2) ...
Setting up tk (8.4.16-2) ...
update-alternatives: using /usr/bin/wish-default to provide /usr/bin/wish (wish)
 in auto mode.

Setting up tk8.4-dev (8.4.19-4) ...
Setting up tk-dev (8.4.16-2) ...
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place
jim@jim:~$ wget http://www.python.org/ftp/python/3.3.0/Python-3.3.0.tar.bz2
```

A "tar" file is Linux's version of a zipped file. The "`wget`" command downloads the file but does nothing with it. You can use your file browser to confirm that the "`wget`" command downloaded the file "`Python-3.3.0.tar.bz2`" into your HOME directory, in my case, the "jim" directory.

Step #3 – Make a directory for the Python-3.3.0 files

I next made a new directory named "Python-3.3.0" inside my HOME directory. I did this by typing "`mkdir Python-3.3.0`" in response to the "jim@jim:~$" prompt. Using the file browser, I moved the downloaded file into the new directory. This is a screenshot of my HOME directory after Step #3.



Step #4 – Unzip the downloaded file

The next step is to unzip the downloaded file. Since I had been playing around using the file browser, I opened a new terminal window. Before giving me the "jim@jim:~$" prompt, Linux gave me a disturbing message, which is shown in the following screenshot.

```
jim@jim: ~/Python-3.3.0
File Edit View Terminal Help
bash: /opt/openfoam211/etc/bashrc: No such file or directory
jim@jim:~$ cd Python-3.3.0
jim@jim:~/Python-3.3.0$ tar -xjf Python-3.3.0.tar.bz2
```

It seems that the operating system can no longer find my OpenFoam installation. Users of OpenFoam will recognize the phrase "`/opt/openfoam211/etc/bashrc`" as part of the line you added to the "~/.bashrc" file when you installed OpenFoam. It seems that the "`wget`" command did something to affect my OpenFoam installation. This is typical of the tomfoolery that plagues Linux. Fortunately, the remedy, which I describe below, did not take much time.

Because the downloaded file has now been moved into a new directory, it is necessary to change into that directory, using the command "`cd Python-3.3.0`". The downloaded file is extracted using the command "`tar -xjf Python-3.3.0.tar.bz2`", as shown in the screenshot above.

A review of your HOME directory after the extraction is completed will show that the extraction process extracted the file into a new directory, called "Python-3.3.0", which is a subdirectory of the directory "Python-3.3.0" which we created manually.

Step #4 – "Configure the build"

This is what my list of instructions calls this next step, which is the first step towards installing Python-3.3.0 from the files which have now been downloaded and extracted. The following screenshot shows the procedure, starting with a new terminal window.

```
jim@jim: ~/Python-3.3.0/Pytho   Browse all local and remote disks and
                                folders accessible from this computer
File Edit View Terminal Help
bash: /opt/openfoam211/etc/bashrc: No such file or directory
jim@jim:~$ cd Python-3.3.0
jim@jim:~/Python-3.3.0$ cd Python-3.3.0
jim@jim:~/Python-3.3.0/Python-3.3.0$ ./configure --prefix=/opt/python3
```

Because I have not yet fixed the OpenFoam problem, bash is still complaining. It is necessary to change directories twice, to get down into "HOME/Python-3.0.0/Python-3.3.0". Then, typing the command "`./configure --prefix=/opt/python3`" will begin the configuration.

Steps #5 and #6 – Make and install python3

After the configuration is completed, something needs to be made, which is accomplished in Step #5 by typing "`make`" in response to the prompt. Step #6 must be the installation itself, because it is executed by typing "`sudo make install`" in response to the prompt. Both of these operations take a fair amount of time and are accompanied by lots of activity on the screen. These two screenshots show the two commands just before the Return key is pressed.

```
jim@jim: ~/Python-3.3.0/Python-3.3.0
File Edit View Terminal Help
checking for resizeterm... yes
configure: checking for device files
checking for /dev/ptmx... yes
checking for /dev/ptc... no
checking for %lld and %llu printf() format support... yes
checking for %zd printf() format support... yes
checking for socklen_t... yes
checking for broken mbstowcs... no
checking for --with-computed-gotos... no value specified
checking whether gcc -pthread supports computed gotos... yes
checking for build directories... done
checking for -O2... yes
checking for glibc _FORTIFY_SOURCE/memmove bug... no
checking for gcc ipa-pure-const bug... no
configure: creating ./config.status
config.status: creating Makefile.pre
config.status: creating Modules/Setup.config
config.status: creating Misc/python.pc
config.status: creating Modules/ld_so_aix
config.status: creating pyconfig.h
creating Modules/Setup
creating Modules/Setup.local
creating Makefile
jim@jim:~/Python-3.3.0/Python-3.3.0$ make
```

Step #5

```
renaming build/scripts-3.3/pydoc3 to build/scripts-3.3/pydoc3.3
renaming build/scripts-3.3/idle3 to build/scripts-3.3/idle3.3
renaming build/scripts-3.3/2to3 to build/scripts-3.3/2to3-3.3
renaming build/scripts-3.3/pyvenv to build/scripts-3.3/pyvenv-3.3
/usr/bin/install -c -m 644 ./Tools/gdb/libpython.py python-gdb.py
gcc -pthread -c -Wno-unused-result -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-protot
ypes    -I. -I./Include    -DPy_BUILD_CORE -o Modules/_testembed.o ./Modules/_te
stembed.c
gcc -pthread   -Xlinker -export-dynamic -o Modules/_testembed Modules/_testembed
.o libpython3.3m.a -lpthread -ldl  -lutil   -lm
jim@jim:~/Python-3.3.0/Python-3.3.0$ sudo make install
```

Step #6

.

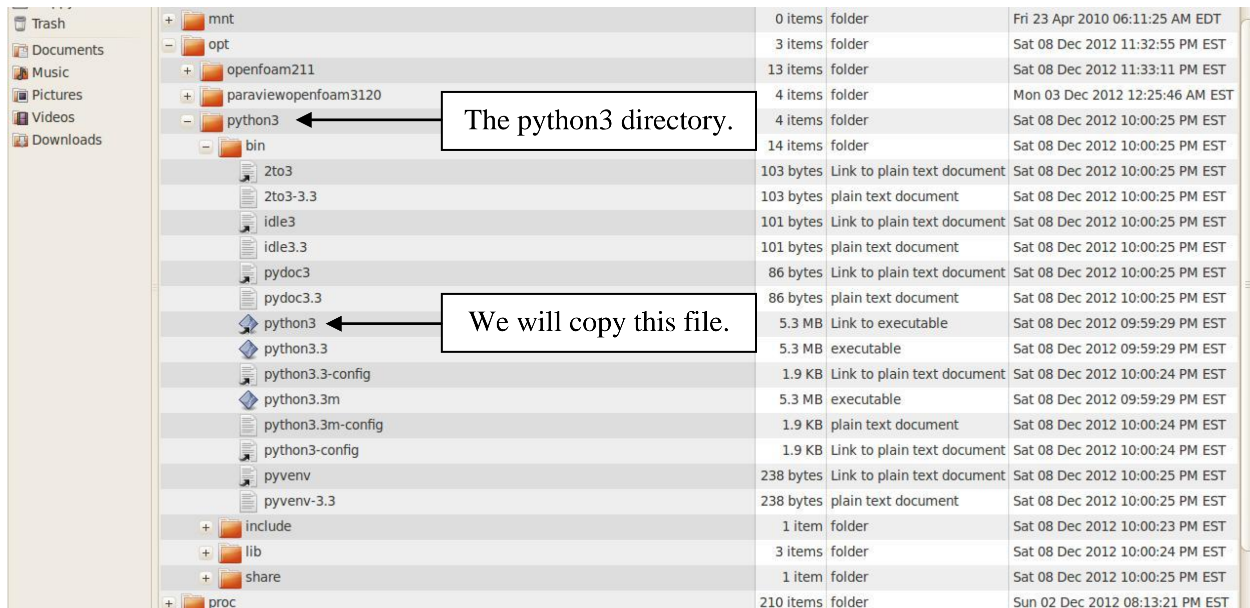
Step #7 – Adding python3 to the PATH

Although Python-3.3.0 is now installed, under the name "python3", it would be a miracle if your Linux installation could find it. In other words, it will say "`File not found`" if you type "`python3`" after the terminal prompt. According to the internet, there are several million ways to fix this small difficulty. None of them are guaranteed to work and, indeed, none of them worked for me.

Somewhere in your Linux installation, a PATH variable is set, which lists the directories the operating system searches through to find the name of any program you try to execute. The file which sets this PATH variable depends on which version of Linux you have. [One of the favorite tricks of the Linux folks is to change the location of this file. On my installation, it is in the file "/etc/environment".]

Wherever the file with the PATH variable is located, it is a good bet that one of the directory names it contains is "/usr/bin". I think it was anticipated that this directory would contain all the binary files installed by the user. We are not going to change the PATH setting. Instead, we are going to ensure that the directory "/usr/bin" contains a suitable reference to "python3".

You can use your file browser to confirm that the process we used to install Python-3.3.0 put the program files in the directory "/opt". You should see it there and, if you are an OpenFoam user, will see it as a sister directory to the OpenFoam and ParaView installations. The following screenshot shows the three directories as they appear on my system.



The file of interest now is the one "/opt/python3/bin/python3". It is described as a "Link to executable", which could be similar to a Windows shortcut file. However, its size – 5.3 MB – is awfully big for a shortcut file. Nevertheless, making a copy of this file in the "/usr/bin" directory does solve the problem.

Since I cannot make such changes using my file browser, I had to open a new terminal window to do so. In response to the terminal's prompt, type "`sudo cp /opt/python3/bin/python3 /usr/bin`". You may be asked to provide the administrator's password. After this, the operating system will recognize the command "`python3`".

Finicky operating system stuff #2

The program "fplot.py" invokes the program "gnuplot", which must also be installed on your Linux computer for this application to run. It is possible that your Linux computer already has Gnuplot installed. If so, it should work. It looks to me like the Gnuplot versions which are available on the internet are pretty stable. The Gnuplot folks may also be as busy as bees but, thankfully, their updates do not seem to deprecate everything from previous versions. Thank you, Gnuplot folks.

My Linux computer did not have Gnuplot, so I downloaded it using the following command, typed in response to the terminal prompt "`sudo apt-get install gnuplot`".

This command worked. It apparently downloaded and installed all the necessary things. Again, thank you, Gnuplot folks.

Restoring the reference to openfoam211

When I installed Python-3.3.0, something disturbed the operating system's ability to find "openfoam211". I solved this problem by typing the following command in response to a terminal prompt "`sudo apt-get install openfoam211`". OpenFoam users will recall that this is the command you used during your original installation of OpenFoam. Doing it again is much quicker. Apparently, the necessary files still exist somewhere on the computer, and the "`apt`" program does not download them again. It does a few things, including restoring the necessary links. The "source" line you originally added to the "~/.bashrc" file is still there and does not need to be repeated.

Conclusion

To summarize usage of "fplot.py":

1. Make sure there is a copy of "fplot.py" in the OpenFoam case directory of interest;
2. Open a new terminal window;
3. Change to the case directory and
4. Type "`python3 fplot.py`" in response to the terminal prompt.
5. This can be done while the job in the case directory is executing.

The program "fplot.py" as listed above only plots forces and moments. I do not keep track of force and moment coefficients during execution but, if you do, you could revise the program to parse the lines in the coefficients file which the /system/controlDict can produce. If you choose, you can also plot residuals. The /system/controlDict can be instructed to produce a running "log" file to contain the output which would otherwise be sent to the screen. "fplot.py" could be revised to parse the lines in the "log" files in order to extract the residuals of interest. Of course, the parsing algorithm will depend on the format of the data dumped into the "log" files.

Jim Hawley
December 2012

An e-mail setting out errors and omissions would be appreciated.