Wednesday Gasifier Training Problem
Part 2: Advanced Post-Processing

February 2018

CPFD Software LLC
10899 Montgomery Blvd. NE, Suite A
Albuquerque, NM 87111
+1.505.275.3849
www.cpfd-software.com

Training Objectives

- This post-processing training introduces several advanced techniques that are useful in monitoring and analyzing Barracuda simulations. We will use the Wednesday gasifier as the basis for demonstrating these techniques.
- The terminal (aka command-line, or shell) is introduced. Many advanced operations can be done using the terminal.
- Scripts are introduced, using Python to define plots that can be used to easily monitor ongoing simulations.
- MAKE_ANIMATIONS.sh scripts are introduced, using BATCHMOVIE.sh to automate the creation of animations.
- Combining multiple views of a simulation into a single animation
- Adding a logo to an animation

Data File Header Sections

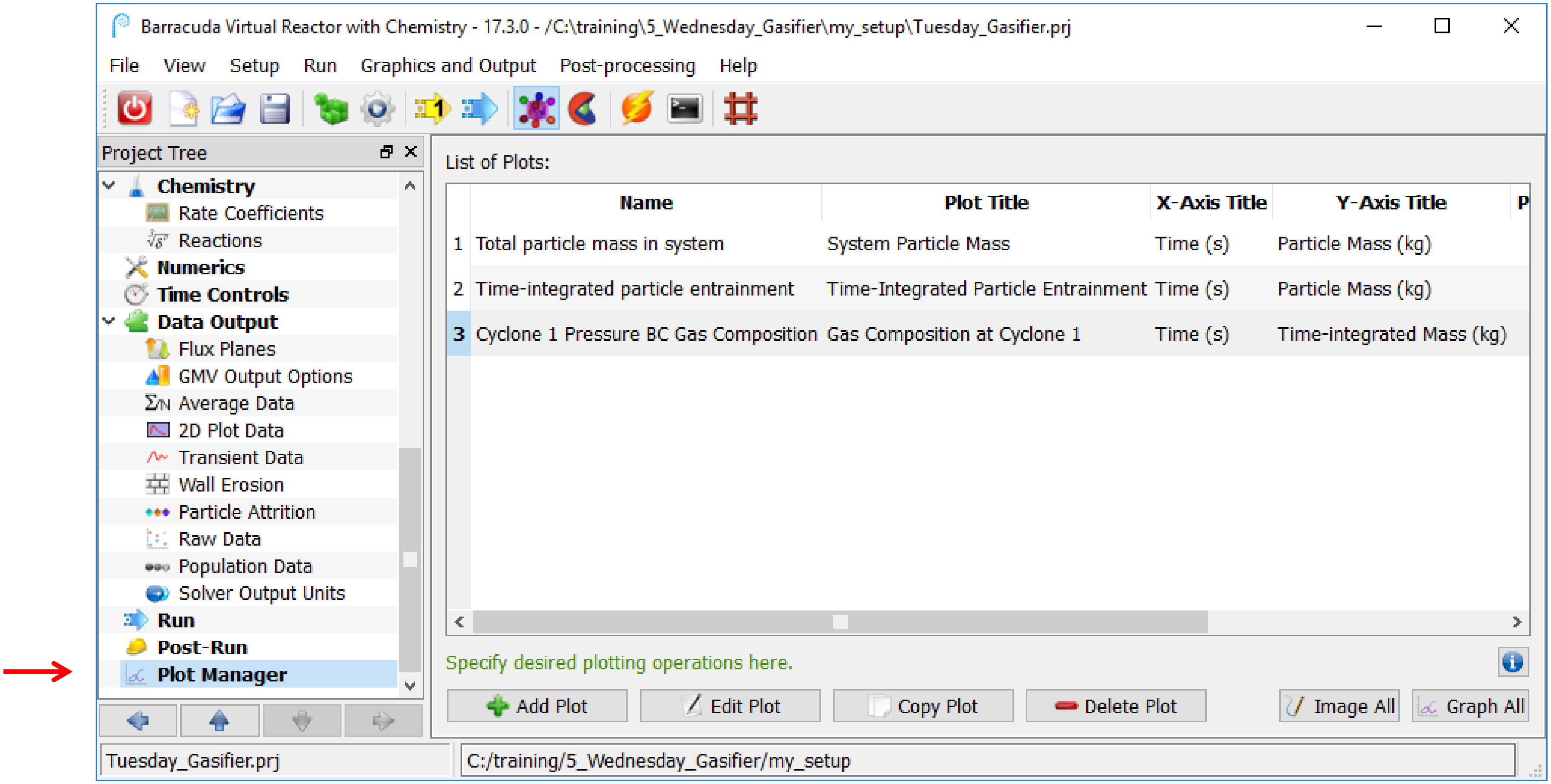
- In the discussions that follow, we will be plotting data from Barracuda output files. It is important to understand the header sections of these files in order to find the correct columns for plotting.
- When plotting with **Plot Manager**, use the **Preview** button to see the header sections.
- When writing custom plotting scripts, use the **Post-Run → View Log Files** to open text-based output files and inspect their header section information. Notice that by default only files ending in *.log and *.otp are listed. Other types of files can be viewed by using the **Files of type** dropdown.
- All text-based Barracuda output files have headers in Standard File Format (SFF), similar to the format used for *.sff input files at boundary conditions.

```
#@   1  "Time"                 "s“
#@   2  "dt"                   "s“
#@   3  "Volume iterations"  "“
#@   4  "Volume error"         "“
#@   5  "u iterations"         "“
#@   6  "u error"              "“
#@   7  "v iterations"         "“
#@   8  "v error"              "“
#@   9  "w iterations"         "“
... and so on ...
```

- Base on a review of the header information for an output file, you will be able to determine which column contains the data you wish to plot.

Plot Manager Review

- Whenever possible, use **Plot Manager**. For many simulation monitoring and analysis tasks, **Plot Manager** is the easiest way to quickly visualize data from Barracuda VR's text-based output files.
- Plots defined in **Plot Manager** are automatically stored within the Barracuda VR project file, making them easily accessible for future use.


Barracuda Virtual Reactor with Chemistry - 17.3.0 - /C:\training\5_Wednesday_Gasifier\my_setup\Tuesday_Gasifier.prj

List of Plots:

	Name	Plot Title	X-Axis Title	Y-Axis Title	P
1	Total particle mass in system	System Particle Mass	Time (s)	Particle Mass (kg)	
2	Time-integrated particle entrainment	Time-Integrated Particle Entrainment	Time (s)	Particle Mass (kg)	
3	Cyclone 1 Pressure BC Gas Composition	Gas Composition at Cyclone 1	Time (s)	Time-integrated Mass (kg)	

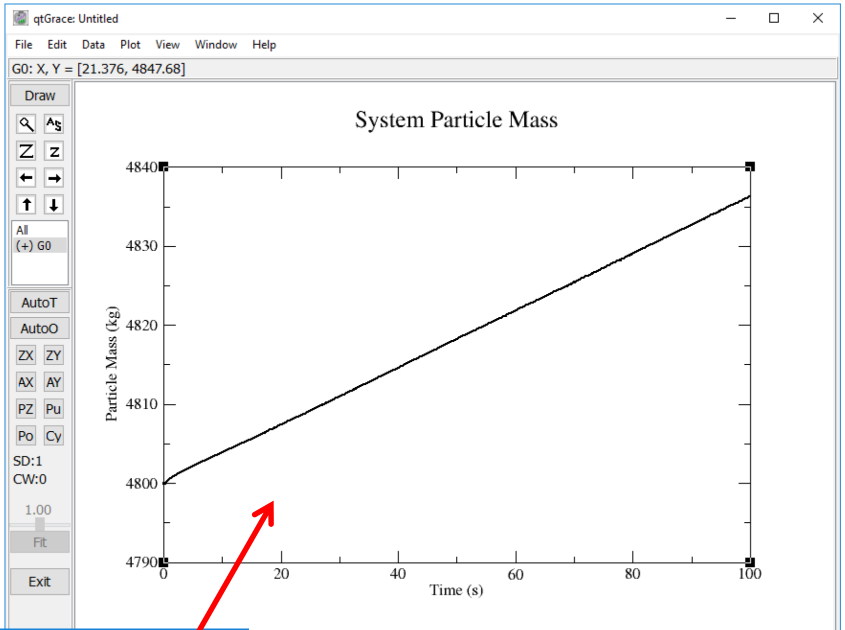
Specify desired plotting operations here.

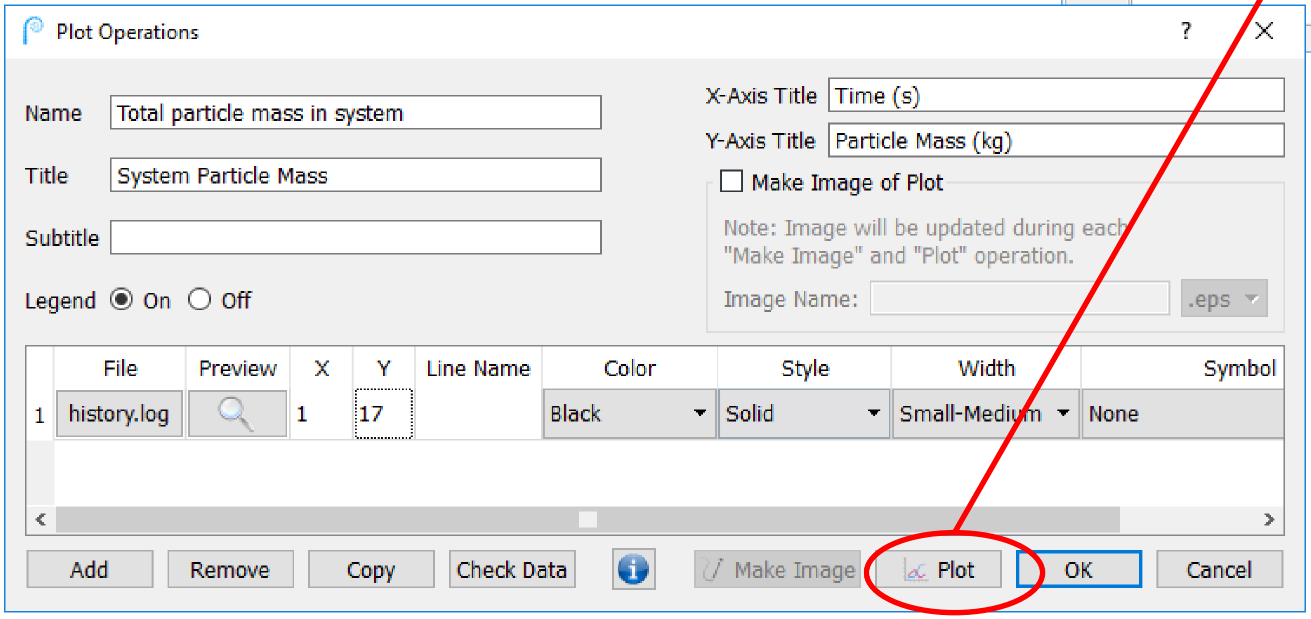
Tuesday_Gasifier.prj

C:/training/5_Wednesday_Gasifier/my_setup

Total Particle Mass in System

- A common parameter that you should monitor during a Barracuda VR simulation is the total particle mass in the system.
    - Most systems have a known mass of particles at steady state, and your simulation should match that known value.
- The total particle mass is recorded in **history.log** with a column name of **"particle mass".**
- Create a plot of the total system particle mass using **Plot Manager**.





Name: Total particle mass in system

Title: System Particle Mass

Subtitle:

Legend ● On ○ Off

X-Axis Title: Time (s)

Y-Axis Title: Particle Mass (kg)

☐ Make Image of Plot

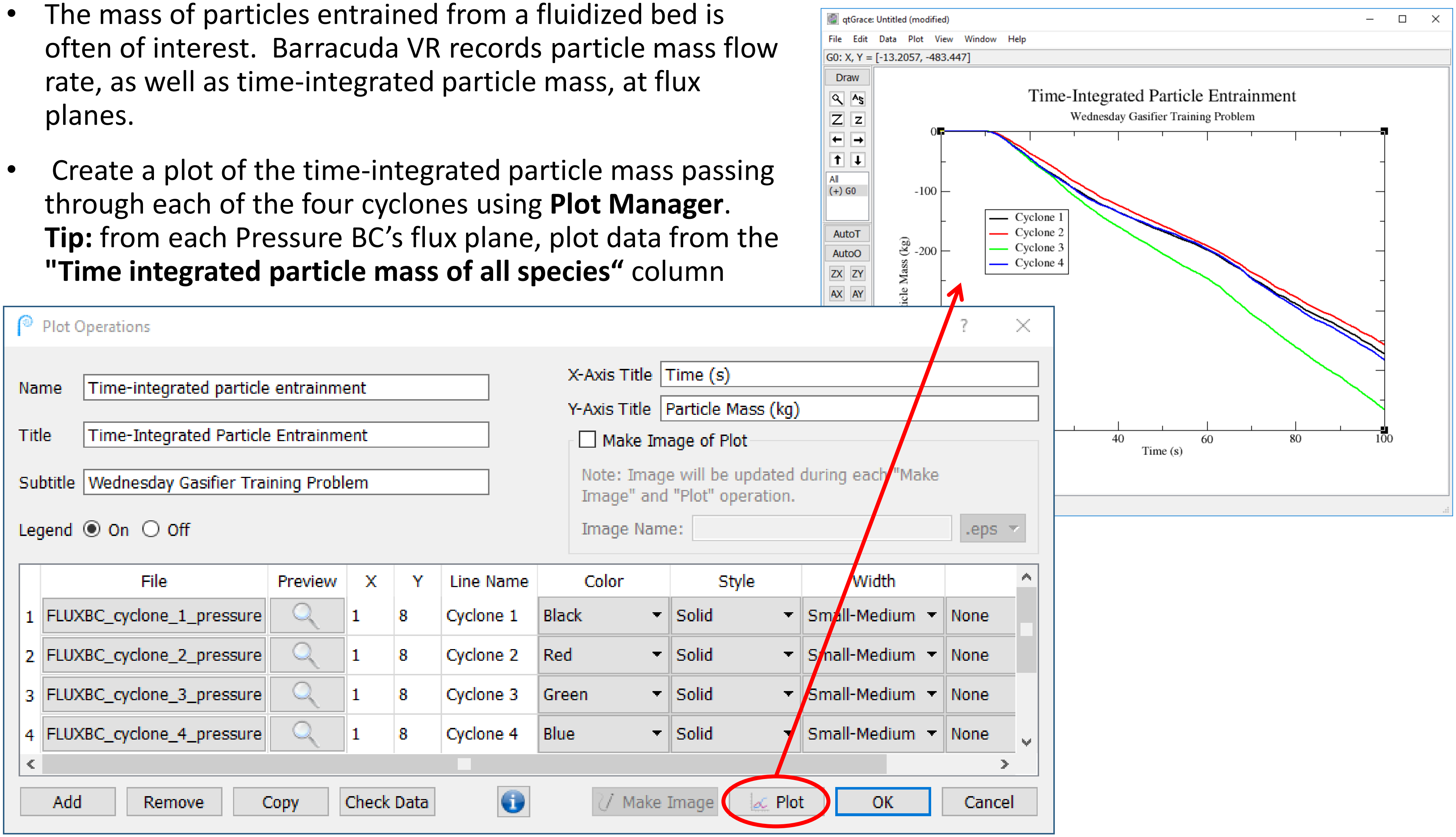
Note: Image will be updated during each "Make Image" and "Plot" operation.

Image Name: .eps

	File	Preview	X	Y	Line Name	Color	Style	Width	Symbol
1	history.log		1	17		Black	Solid	Small-Medium	None

Time-Integrated Particle Entrainment

- The mass of particles entrained from a fluidized bed is often of interest. Barracuda VR records particle mass flow rate, as well as time-integrated particle mass, at flux planes.
- Create a plot of the time-integrated particle mass passing through each of the four cyclones using **Plot Manager**. **Tip:** from each Pressure BC's flux plane, plot data from the **"Time integrated particle mass of all species"** column


qtGrace: Untitled (modified)
File Edit Data Plot View Window Help
G0: X, Y = [-13.2057, -483.447]
Draw
All
(+) G0
AutoT
AutoO
ZX ZY
AX AY
Time-Integrated Particle Entrainment
Wednesday Gasifier Training Problem
0
-100
-200
icle Mass (kg)
40
60
80
100
Time (s)
Cyclone 1
Cyclone 2
Cyclone 3
Cyclone 4
Plot Operations
?

Name: Time-integrated particle entrainment

Title: Time-Integrated Particle Entrainment

Subtitle: Wednesday Gasifier Training Problem

Legend ● On ○ Off

X-Axis Title: Time (s)

Y-Axis Title: Particle Mass (kg)

☐ Make Image of Plot

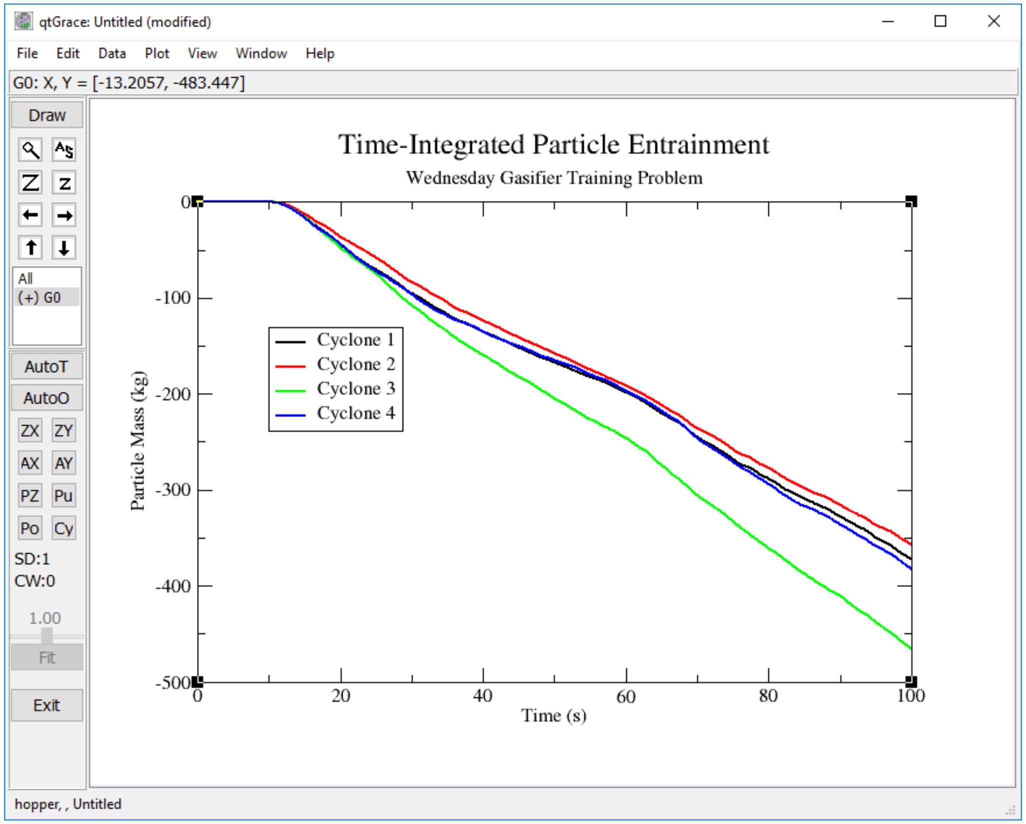
Note: Image will be updated during each "Make Image" and "Plot" operation.

Image Name: .eps

	File	Preview	X	Y	Line Name	Color	Style	Width	
1	FLUXBC_cyclone_1_pressure		1	8	Cyclone 1	Black	Solid	Small-Medium	None
2	FLUXBC_cyclone_2_pressure		1	8	Cyclone 2	Red	Solid	Small-Medium	None
3	FLUXBC_cyclone_3_pressure		1	8	Cyclone 3	Green	Solid	Small-Medium	None
4	FLUXBC_cyclone_4_pressure		1	8	Cyclone 4	Blue	Solid	Small-Medium	None

Interpreting Time-Integrated Data

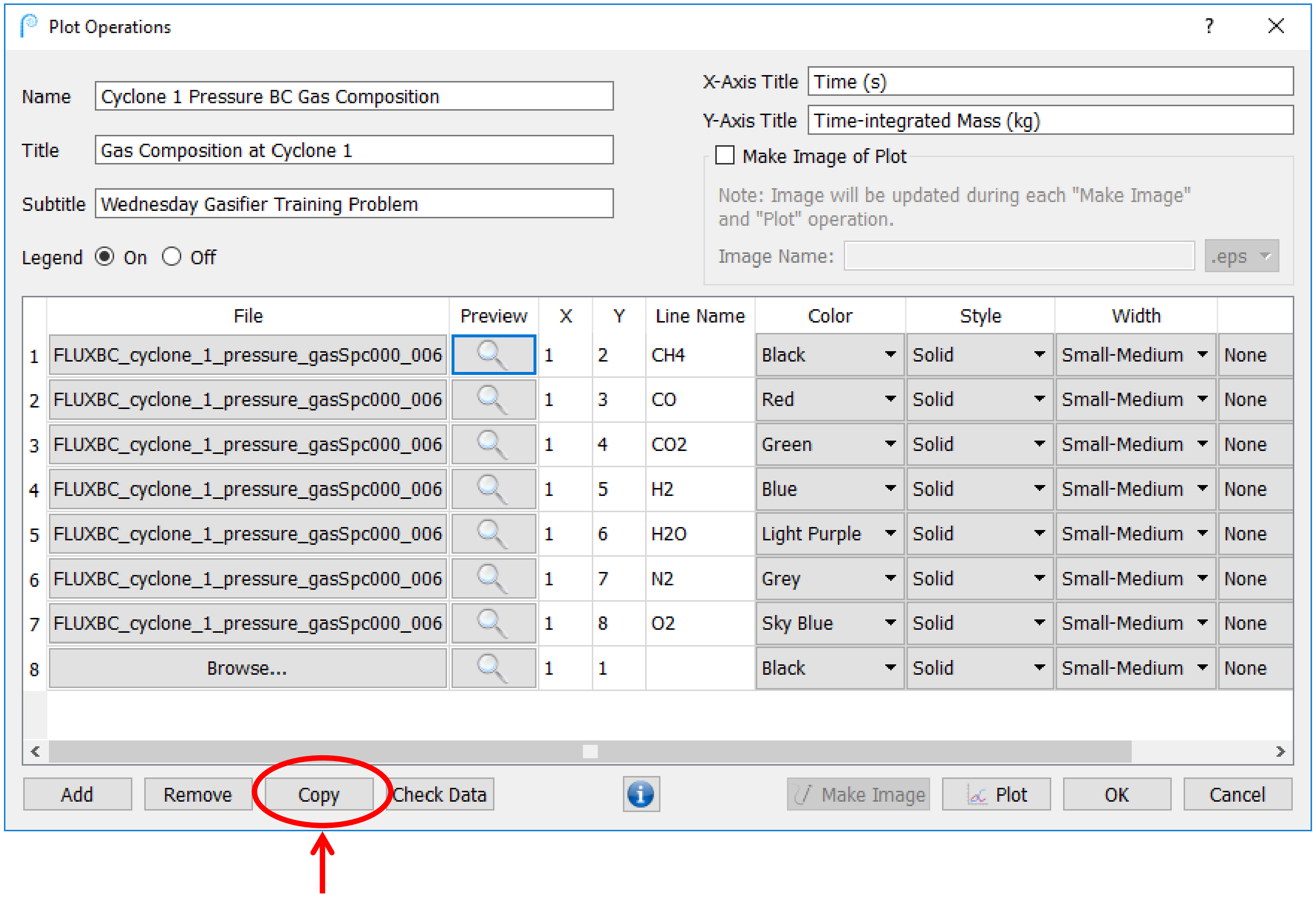
- It is important to have a clear understanding of time-integrated mass data, since many Barracuda files utilize this concept.
- In the plot at right, we are seeing curves that represent the total particle mass that has passed through each of the four flux planes since the beginning of the simulation.
- The time-integrated mass is negative due to the sign convention in Barracuda VR.
    - For boundary condition flux planes (e.g., Pressure BCs and Flow BCs), flow into the system is considered positive, while flow out of the system is negative.
    - For internal flux planes, the sign convention is based on axis directions. A z-direction flux plane reports positive values for flow in the positive z-direction, and negative values for negative z-direction flows.
- By calculating the slope of each time-integrated line, you can determine the entrainment rate in units of (kg/s).



G0: X, Y = [-13.2057, -483.447]

Outlet Gas Composition

- For chemically reacting systems, the gas composition at system outlets is often of interest.
- In Barracuda VR, a flux plane will record gas composition information according to the option selected in the drop-down box labeled **Gas species flux plane behavior**.
    - The gas composition flux plane file will have the same name as the normal flux plane file, with a suffix of **_gasSpc000_006** (the numbers at the end are determined by the number of gas species used in the project).
- For the Wednesday Gasifier example, we chose **Mass Time Cumulative** as the gas composition format at each of the four cyclone Pressure BCs.
- Using **Plot Manager**, create a plot of the gas composition at the Pressure BC of Cyclone 1.



Name: Cyclone 1 Pressure BC Gas Composition

Title: Gas Composition at Cyclone 1

Subtitle: Wednesday Gasifier Training Problem

Legend ● On ○ Off

X-Axis Title: Time (s)

Y-Axis Title: Time-integrated Mass (kg)

☐ Make Image of Plot

Note: Image will be updated during each "Make Image" and "Plot" operation.

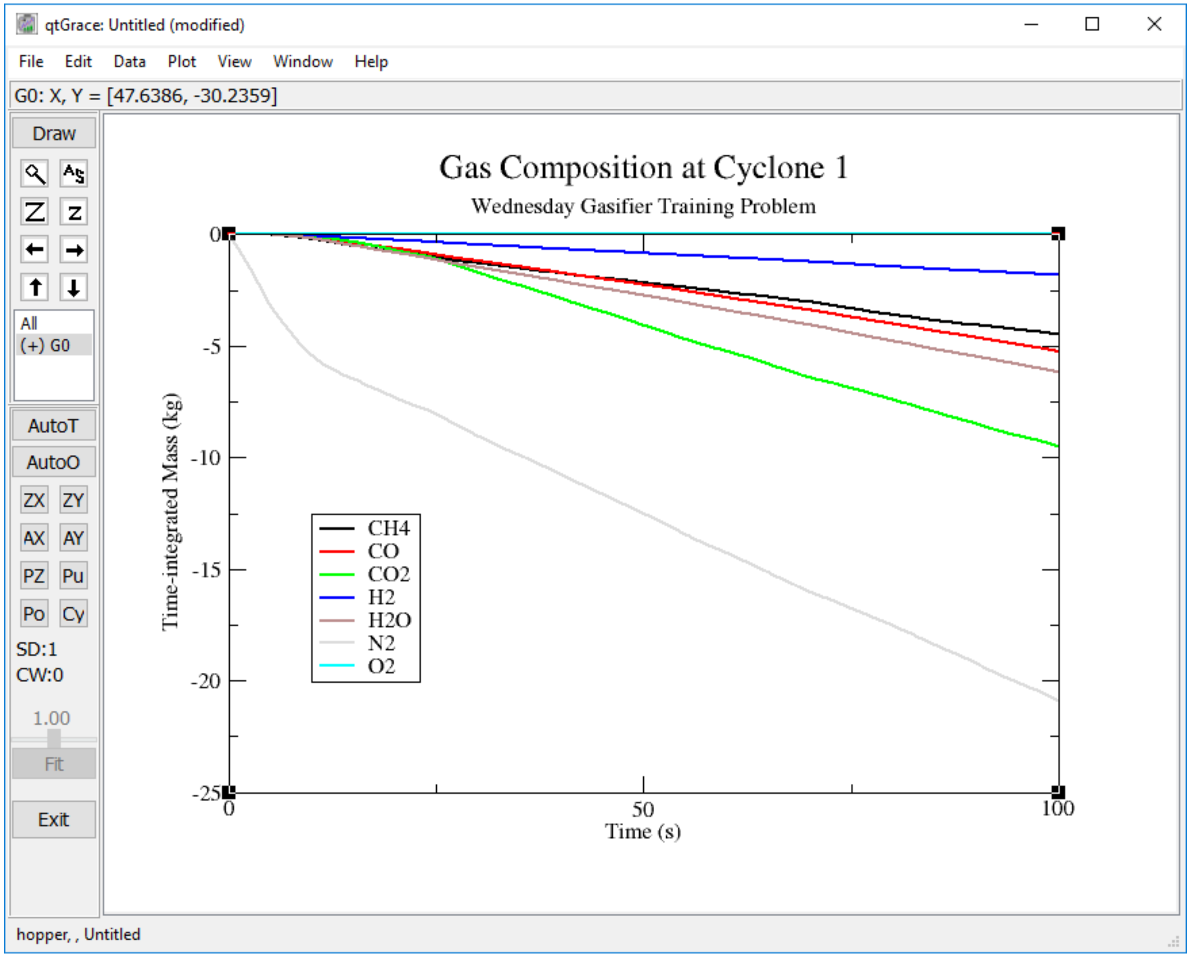
Image Name: .eps

	File	Preview	X	Y	Line Name	Color	Style	Width	
1	FLUXBC_cyclone_1_pressure_gasSpc000_006	🔍	1	2	CH4	Black	Solid	Small-Medium	None
2	FLUXBC_cyclone_1_pressure_gasSpc000_006	🔍	1	3	CO	Red	Solid	Small-Medium	None
3	FLUXBC_cyclone_1_pressure_gasSpc000_006	🔍	1	4	CO2	Green	Solid	Small-Medium	None
4	FLUXBC_cyclone_1_pressure_gasSpc000_006	🔍	1	5	H2	Blue	Solid	Small-Medium	None
5	FLUXBC_cyclone_1_pressure_gasSpc000_006	🔍	1	6	H2O	Light Purple	Solid	Small-Medium	None
6	FLUXBC_cyclone_1_pressure_gasSpc000_006	🔍	1	7	N2	Grey	Solid	Small-Medium	None
7	FLUXBC_cyclone_1_pressure_gasSpc000_006	🔍	1	8	O2	Sky Blue	Solid	Small-Medium	None
8	Browse...	🔍	1	1		Black	Solid	Small-Medium	None

A quick way to create rows 2 through 7 is to define row 1, and then use the “Copy” button. You can then change the “Y” and “Color” items for rows 2 through 7.

Interpretation of Gas Composition Plot

- This plot is showing time-integrated data, not mass flow rate. Be careful to note the difference in meaning.
  - As noted earlier, if you want to get the flow rate of each gas species, calculate the slope of each time-integrated line.
- The sign convention at Boundary Condition flux planes is:
  - In-flow = positive
  - Out-flow = negative
- The gas composition plot can be used to help judge when the system reaches pseudo-steady state. Once the time-integrated lines achieve constant slope, the gas composition at the outlet is no longer changing.



G0: X, Y = [47.6386, -30.2359]

hopper, , Untitled

Plot Total Particle Mass in System with a Script

- Previously in this presentation, you made a plot of the total particle mass using **Plot Manager**. Here, we will plot the same data using a script.
  - As noted previously, in general you should prefer using **Plot Manager** whenever possible. This example is meant to be instructive so that you can see how the same plot can be made using a script.
- We will be using a program called Jupyter notebook. It is an interactive computational environment which uses the Python scripting language to create plots.
  - If Jupyter notebook is not already installed on your system, see this Support Site post for instructions on downloading and installing the Anaconda Python distribution: http://cpfd-software.com/customer-support/knowledge-base/installing-the-anaconda-python-distribution
- When plotting data with Jupyter notebook (or any scripting method), it is convenient to start with a template, so you don't have to type repetitive parts of the plotting commands. Template scripts are included in the training directory.

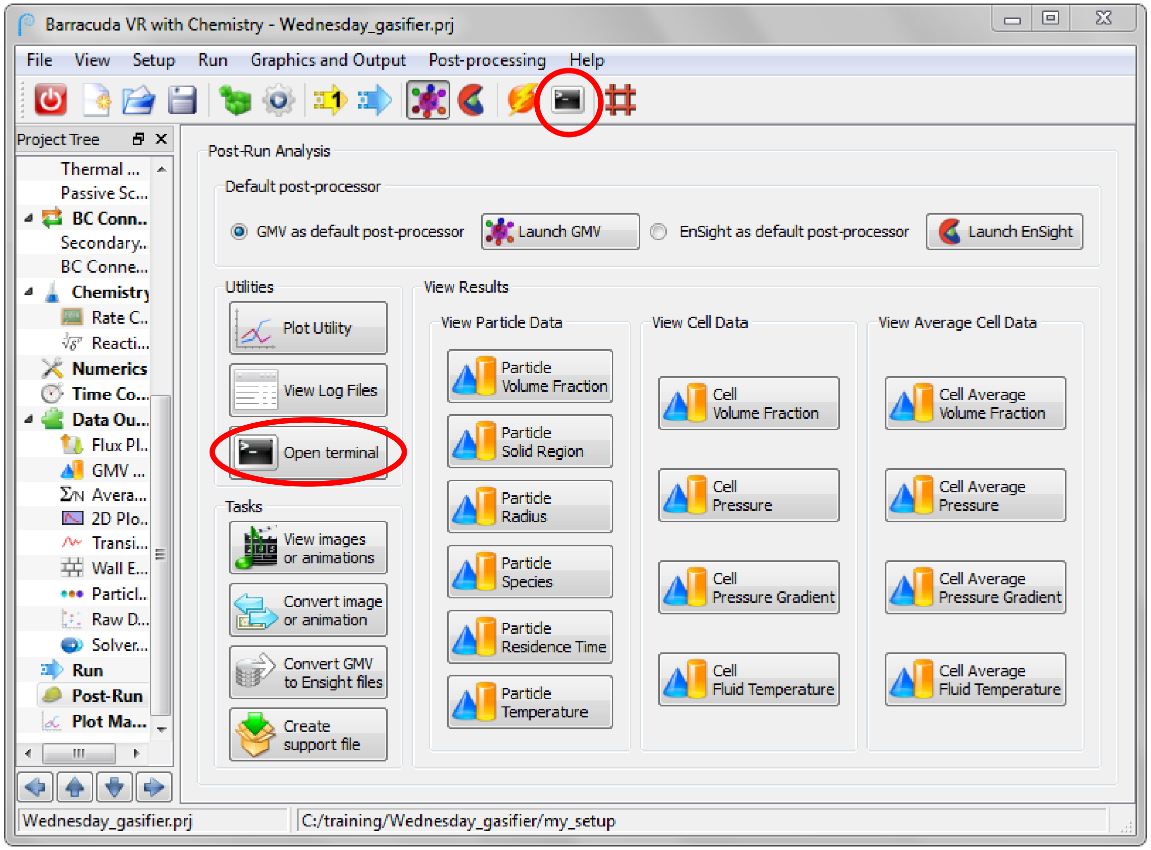
  In **Linux**:
  `/home/training/barracuda_training/scripts`

  In **Windows**:
  `C:\training\scripts`

- Using a file browser or command-line terminal, copy the template script `jupyter_notebook_template.ipynb` into your project directory.

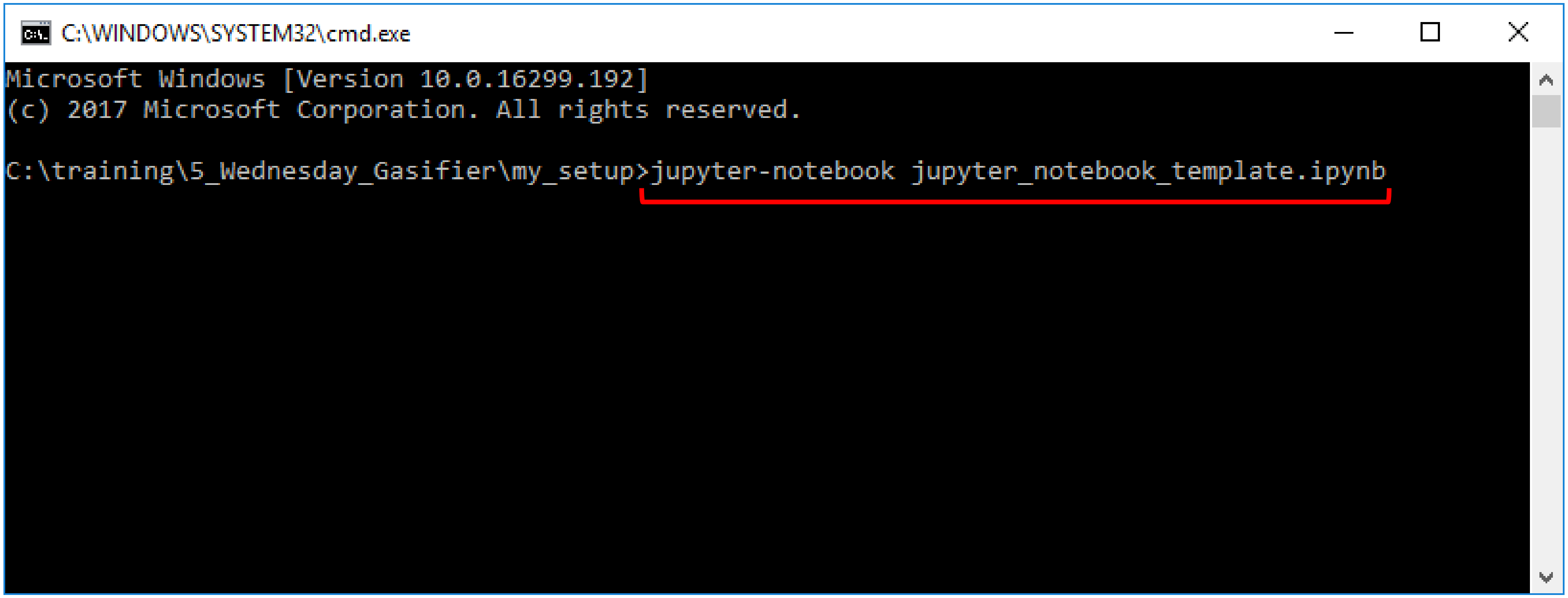
Opening a Command-Line Terminal

- The Barracuda VR GUI has several built-in ways to open a command-line terminal.
    - A shortcut button is always available on the top shortcut button bar.
    - The **Open Terminal** button is available in the **Post-Run** section of the GUI.
    - There is a menu item: **Post-processing → Open Terminal**
- When any of these is used, a new terminal is opened in the currently opened project's working directory.



Back to Making that Plot…

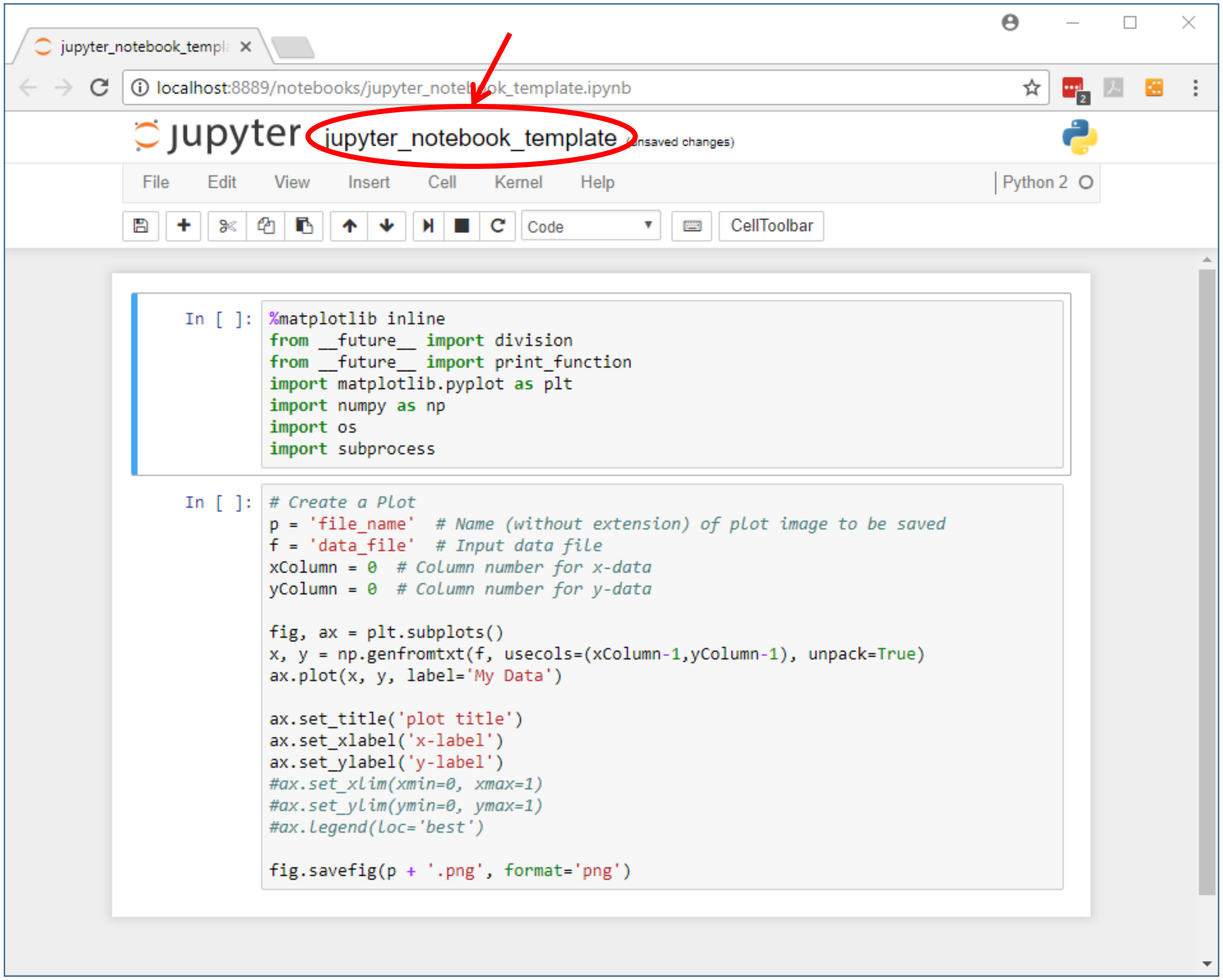
- Using a file browser or command-line terminal, copy the template script `jupyter_notebook_template.ipynb` into your Wednesday Gasifier `my_setup` directory.
- Open a terminal through the Barracuda GUI.
- Type the command seen below to open the Jupyter notebook template.
- This will open the template in your web browser.



```
Microsoft Windows [Version 10.0.16299.192]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\training\5_Wednesday_Gasifier\my_setup>jupyter-notebook jupyter_notebook_template.ipynb
```

Jupyter Notebook – System Mass plot

- Rename the notebook with a name of your choice. Click on the file name to enter a new notebook name.
- A Jupyter notebook is set up with a top cell of information that is necessary to run and create plots.
- The second cell contains the template coding to create a plot. Cells can be added in order to create additional plots.
- In order to “run” the notebook, you can click in each cell (starting with the first cell and moving down) and **Ctrl-Enter**. Or to run the whole notebook at one time, use the top menu bar **Cell→Run All**.
    - Cells should generally be run in order from top to bottom. This will ensure that all functions and variables are properly defined.


jupyter_notebook_templ
localhost:8889/notebooks/jupyter_notebook_template.ipynb
jupyter
jupyter_notebook_template (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Help
Python 2
Code
CellToolbar

In [ ]:

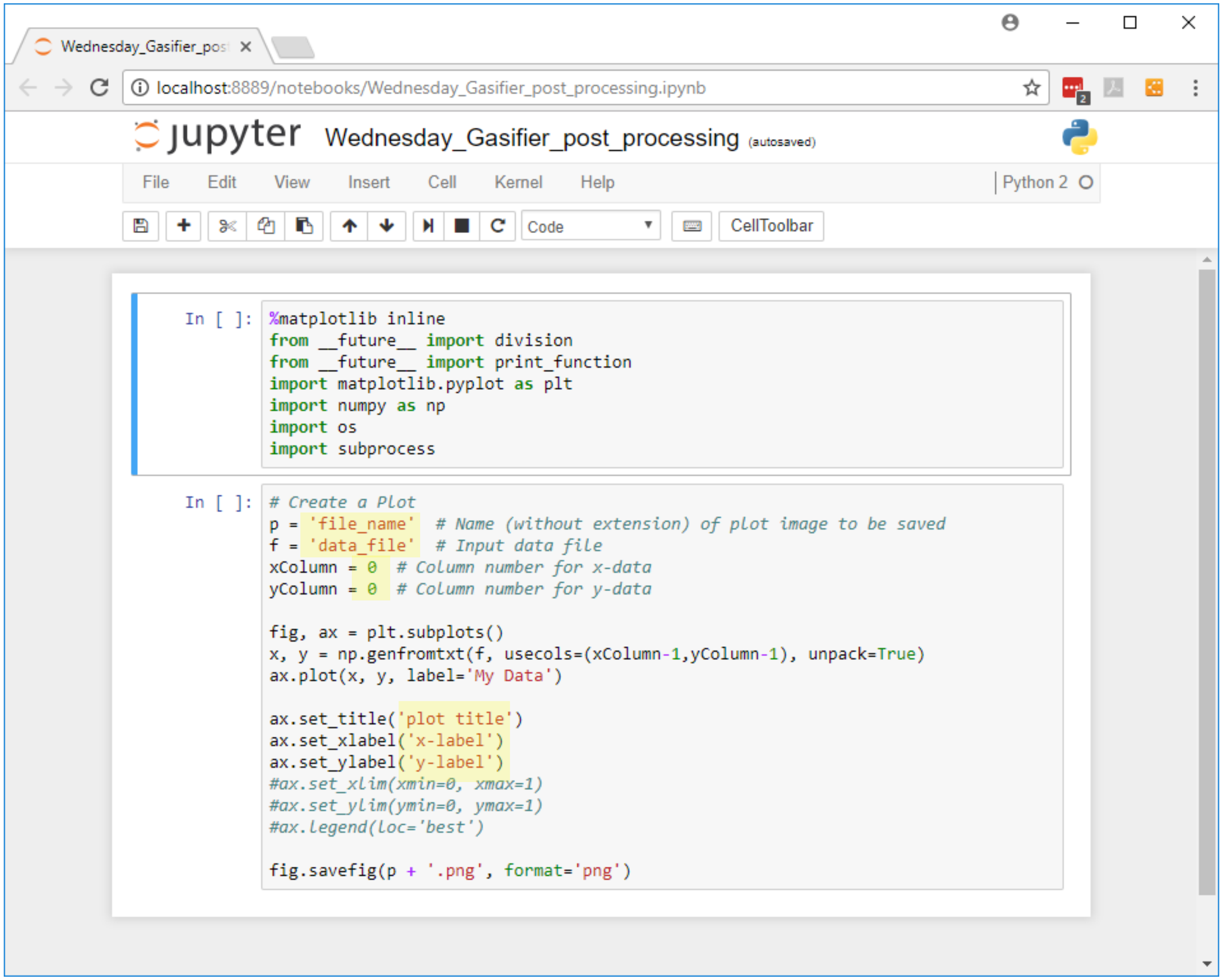
```
%matplotlib inline
from __future__ import division
from __future__ import print_function
import matplotlib.pyplot as plt
import numpy as np
import os
import subprocess
```

In [ ]:

```
# Create a Plot
p = 'file_name'  # Name (without extension) of plot image to be saved
f = 'data_file'  # Input data file
xColumn = 0  # Column number for x-data
yColumn = 0  # Column number for y-data

fig, ax = plt.subplots()
x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
ax.plot(x, y, label='My Data')

ax.set_title('plot title')
ax.set_xlabel('x-label')
ax.set_ylabel('y-label')
#ax.set_xlim(xmin=0, xmax=1)
#ax.set_ylim(ymin=0, ymax=1)
#ax.legend(loc='best')

fig.savefig(p + '.png', format='png')
```

Jupyter Notebook – System Mass plot

- Text hightlighted in yellow needs to be replaced with relevant data for this system mass plot.
- Start with giving the plot a file name. Use underscores (instead of spaces) in all file names.
- Specify the name of the input data file. Tab completion will work in the notebook.
- Type in the correct numbers for the X column and Y column data.
- Specify plot title and X and Y axis labels.
- Once all data is input, **Ctrl-Enter** in each cell or **Cell→Run All** to run the code.


Wednesday_Gasifier_pos
localhost:8889/notebooks/Wednesday_Gasifier_post_processing.ipynb
jupyter
Wednesday_Gasifier_post_processing (autosaved)
File Edit View Insert Cell Kernel Help
Python 2
Code
CellToolbar

```
In [ ]: %matplotlib inline
        from __future__ import division
        from __future__ import print_function
        import matplotlib.pyplot as plt
        import numpy as np
        import os
        import subprocess
```

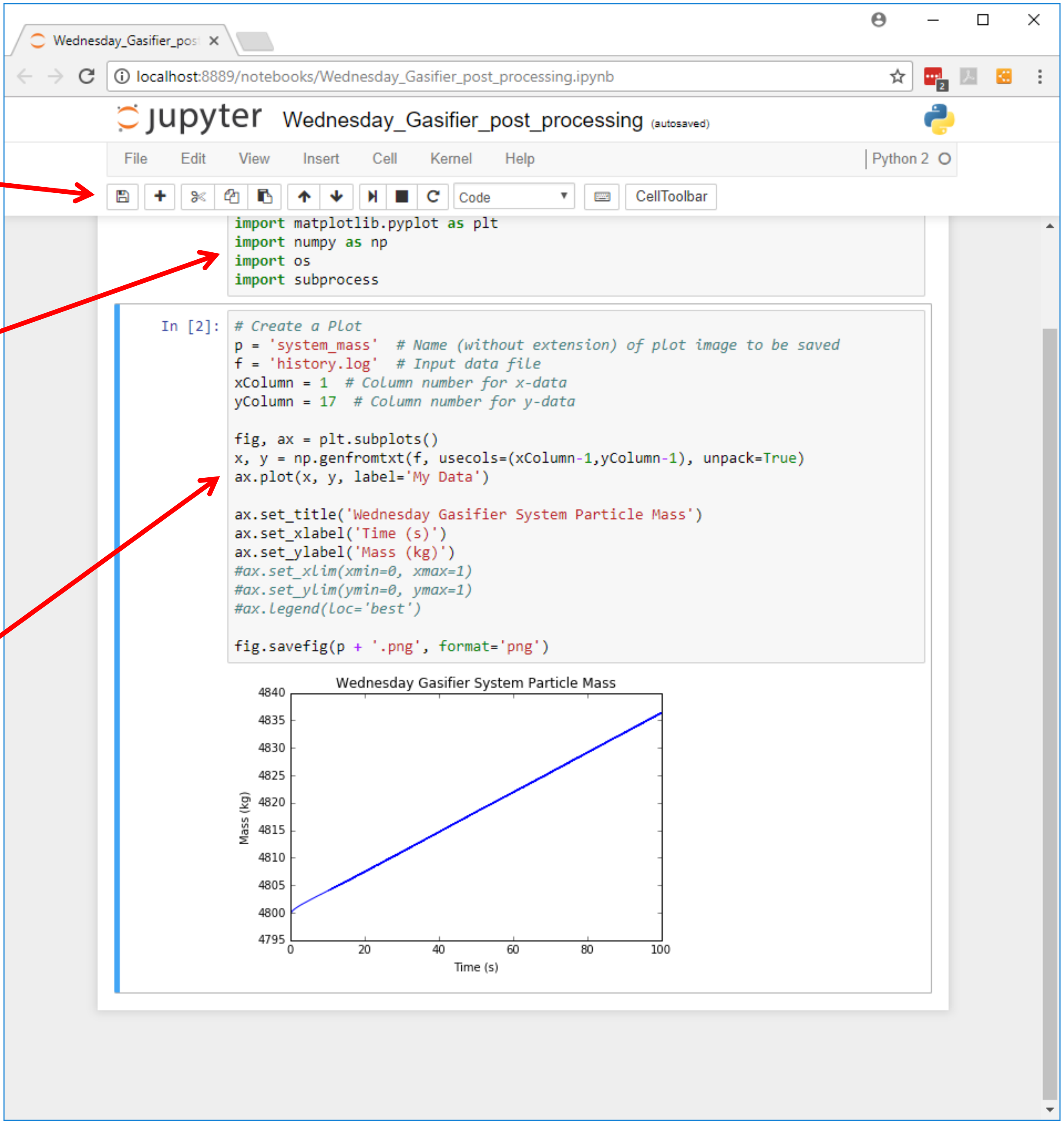
```
In [ ]: # Create a Plot
        p = 'file_name'  # Name (without extension) of plot image to be saved
        f = 'data_file'  # Input data file
        xColumn = 0  # Column number for x-data
        yColumn = 0  # Column number for y-data

        fig, ax = plt.subplots()
        x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y, label='My Data')

        ax.set_title('plot title')
        ax.set_xlabel('x-label')
        ax.set_ylabel('y-label')
        #ax.set_xlim(xmin=0, xmax=1)
        #ax.set_ylim(ymin=0, ymax=1)
        #ax.legend(loc='best')

        fig.savefig(p + '.png', format='png')
```

Jupyter Notebook – System Mass plot

Remember to save!

The first cell only needs to be run when the notebook has first been opened or when any new modules are added.

After the cell has been run, changes can be made to the cell and run again.



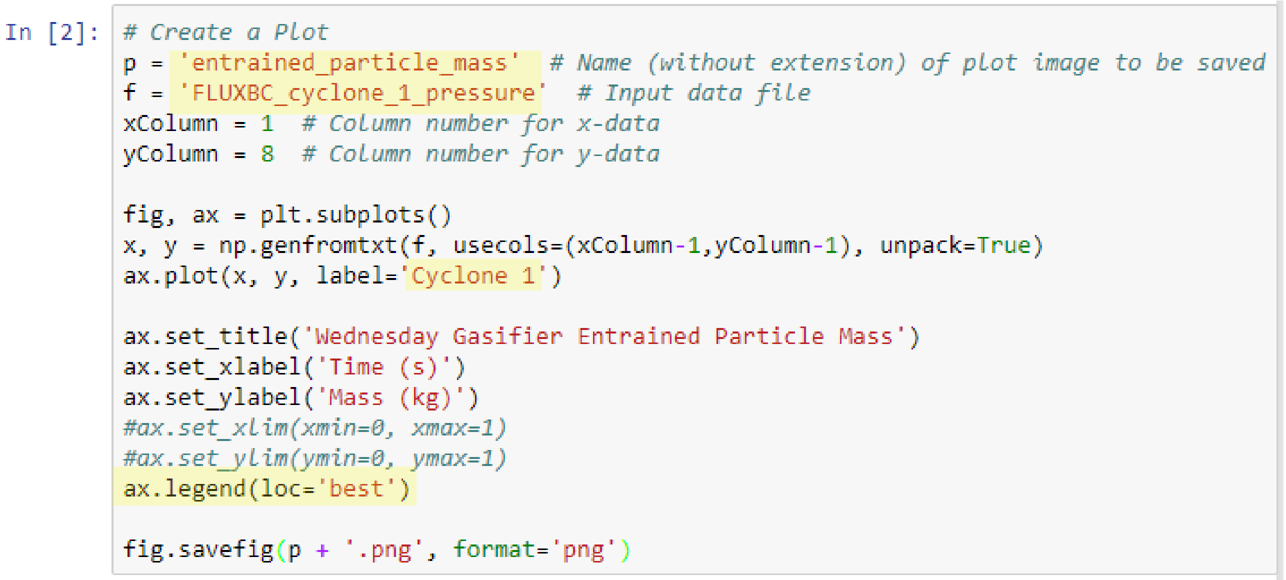
```
import matplotlib.pyplot as plt
import numpy as np
import os
import subprocess
```

In [2]:

```
# Create a Plot
p = 'system_mass'  # Name (without extension) of plot image to be saved
f = 'history.log'  # Input data file
xColumn = 1  # Column number for x-data
yColumn = 17  # Column number for y-data

fig, ax = plt.subplots()
x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
ax.plot(x, y, label='My Data')

ax.set_title('Wednesday Gasifier System Particle Mass')
ax.set_xlabel('Time (s)')
ax.set_ylabel('Mass (kg)')
#ax.set_xlim(xmin=0, xmax=1)
#ax.set_ylim(ymin=0, ymax=1)
#ax.legend(loc='best')

fig.savefig(p + '.png', format='png')
```

Jupyter Notebook – Time-integrated Particle Entrainment with Summation

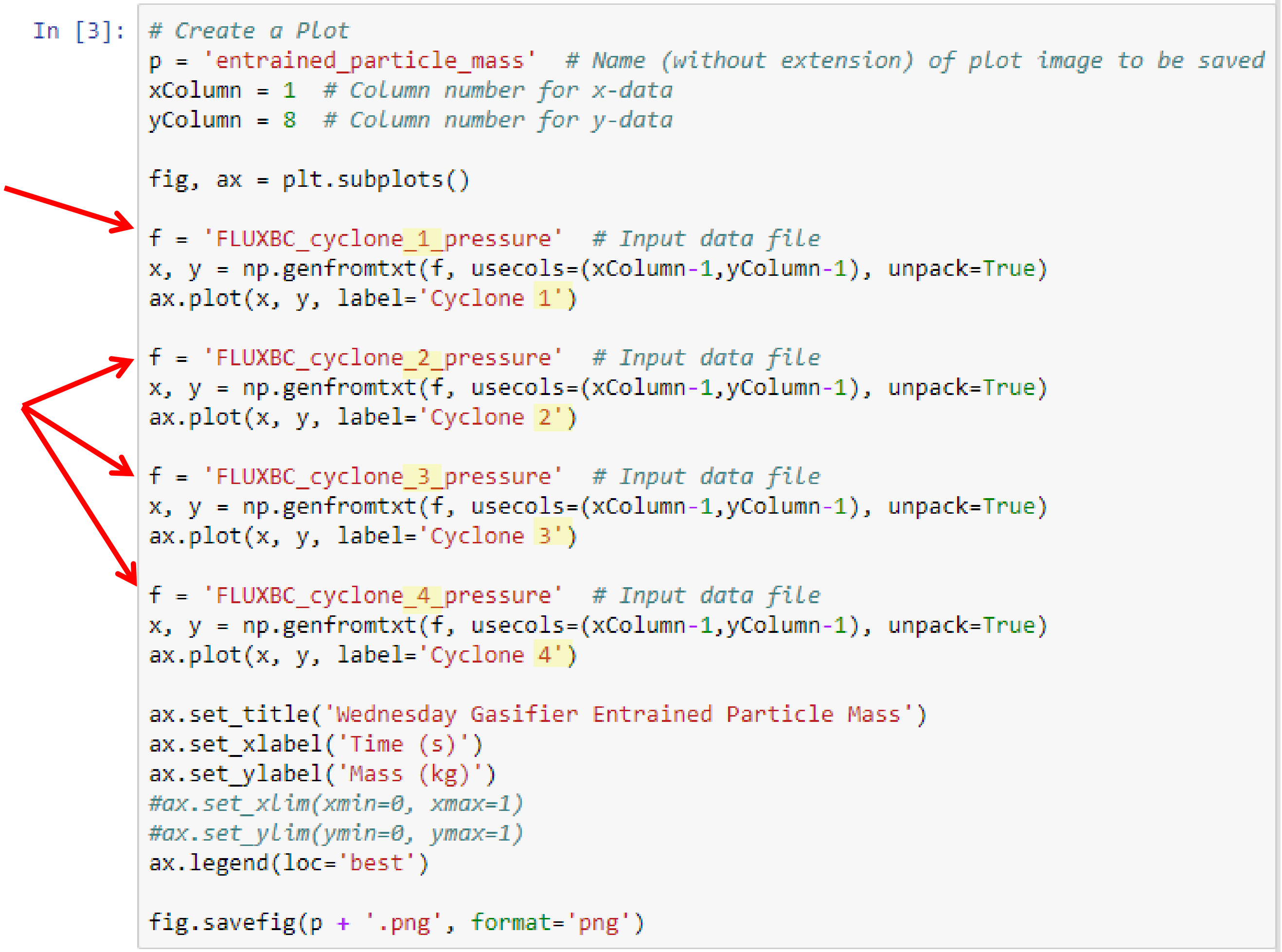
- Earlier in this presentation, we plotted the time-integrated particle mass leaving through the four cyclone Pressure BC flux planes. We can make the same plot from the notebook, and furthermore we can perform a summation of the four cyclones to include in the plot.
- Copy the system mass cell and paste it in a new cell below that cell. Click in the system mass cell, then:
  - **Edit→Copy Cells**
  - **Edit→Paste Cells Below**
- Replace name for plot, data file input, X and Y column, and plot title.
- Add a label and uncomment the legend line.



```
In [2]: # Create a Plot
        p = 'entrained_particle_mass'  # Name (without extension) of plot image to be saved
        f = 'FLUXBC_cyclone_1_pressure'  # Input data file
        xColumn = 1  # Column number for x-data
        yColumn = 8  # Column number for y-data

        fig, ax = plt.subplots()
        x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y, label='Cyclone 1')

        ax.set_title('Wednesday Gasifier Entrained Particle Mass')
        ax.set_xlabel('Time (s)')
        ax.set_ylabel('Mass (kg)')
        #ax.set_xlim(xmin=0, xmax=1)
        #ax.set_ylim(ymin=0, ymax=1)
        ax.legend(loc='best')

        fig.savefig(p + '.png', format='png')
```

Jupyter Notebook – Time-integrated Particle Entrainment with Summation

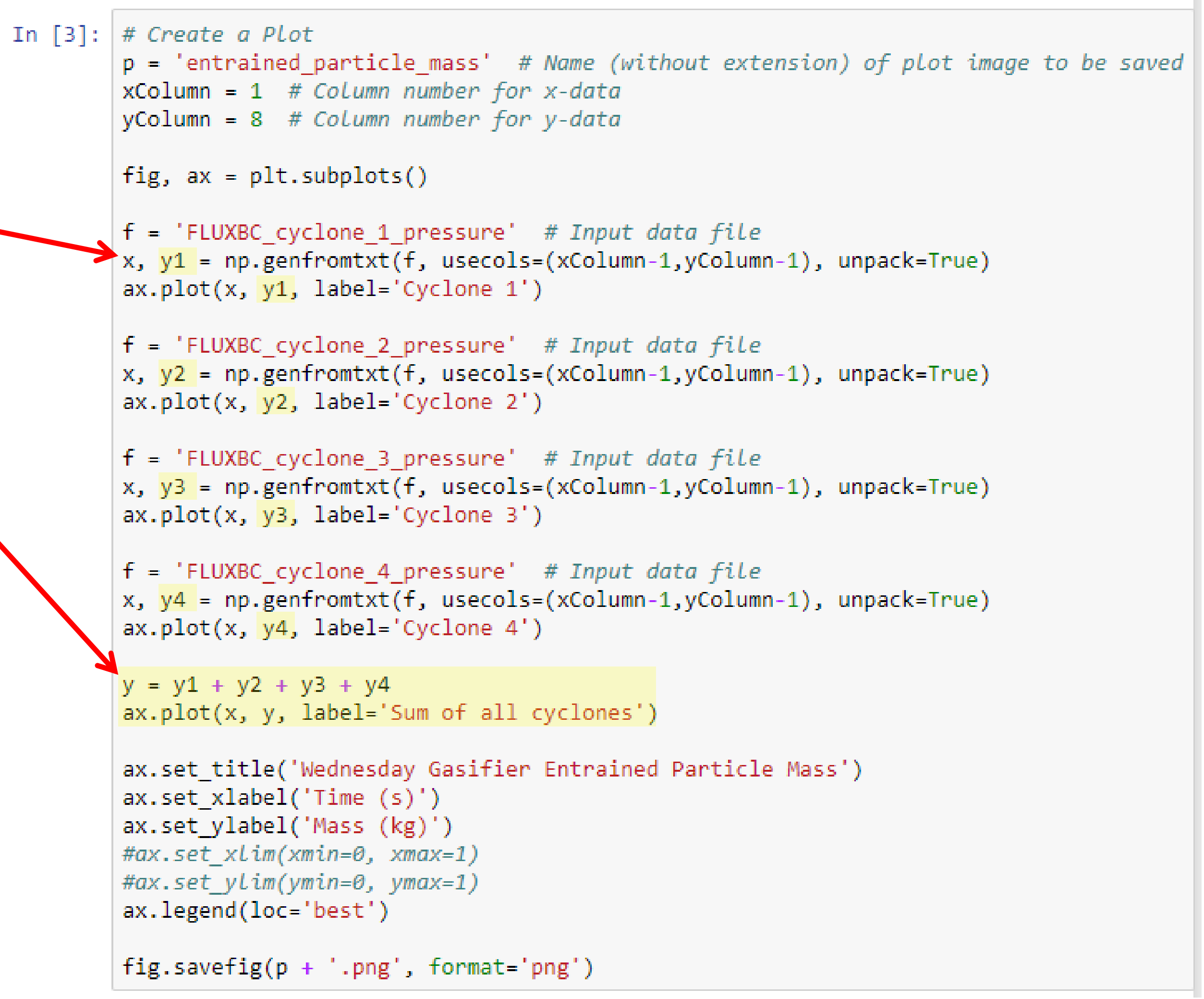
- In order to input the data for all four cyclones, we will need to move the lines around in the script.
- First, cut the input data file line and paste it as shown.
- Next, copy and paste the 3-line block of code three times. Then change the data file name and label to include all four cyclones.
- Next, we will add a line for the summation of all four cyclone's entrainment.



```
In [3]: # Create a Plot
        p = 'entrained_particle_mass'  # Name (without extension) of plot image to be saved
        xColumn = 1  # Column number for x-data
        yColumn = 8  # Column number for y-data

        fig, ax = plt.subplots()

        f = 'FLUXBC_cyclone_1_pressure'  # Input data file
        x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y, label='Cyclone 1')

        f = 'FLUXBC_cyclone_2_pressure'  # Input data file
        x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y, label='Cyclone 2')

        f = 'FLUXBC_cyclone_3_pressure'  # Input data file
        x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y, label='Cyclone 3')

        f = 'FLUXBC_cyclone_4_pressure'  # Input data file
        x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y, label='Cyclone 4')

        ax.set_title('Wednesday Gasifier Entrained Particle Mass')
        ax.set_xlabel('Time (s)')
        ax.set_ylabel('Mass (kg)')
        #ax.set_xlim(xmin=0, xmax=1)
        #ax.set_ylim(ymin=0, ymax=1)
        ax.legend(loc='best')

        fig.savefig(p + '.png', format='png')
```

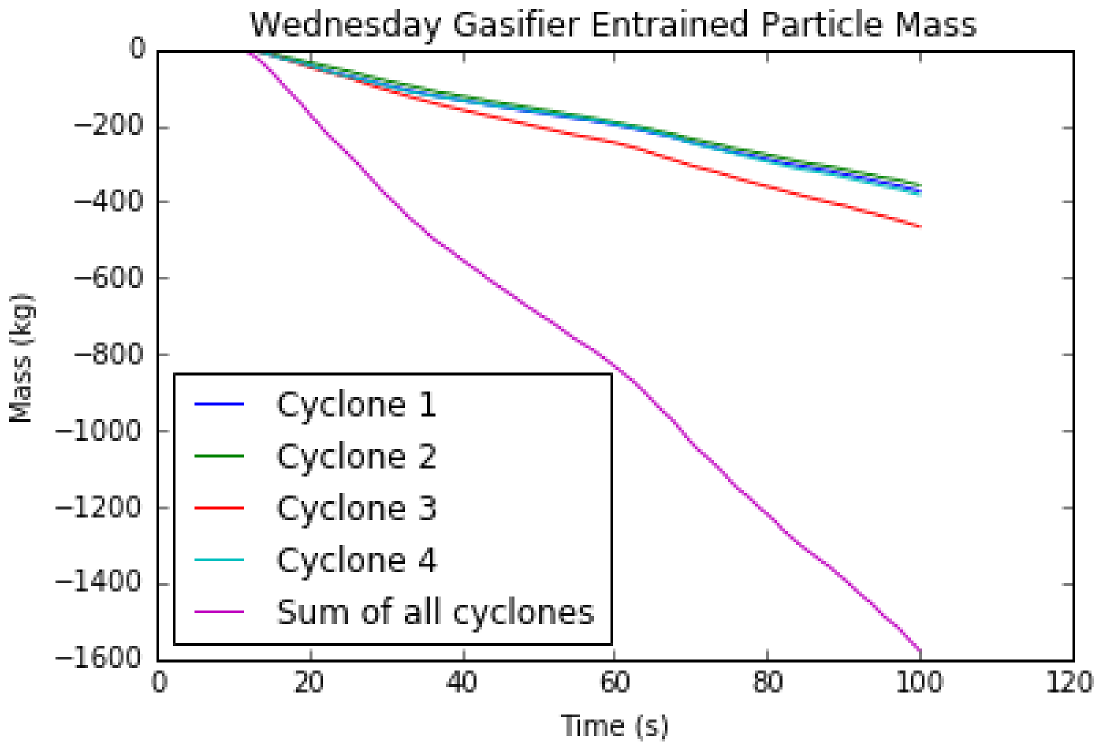
Jupyter Notebook – Time-integrated Particle Entrainment with Summation

- To sum the mass for all four cyclones, we will have to change the name for the Y value in each plot. We can easily do this by changing cyclone 1 y to y1, cyclone 2 y to y2, and so on.
- Define the summation equation.
- Add a new "ax.plot" command to include the calculated summation line to the plot.
- **Crtl→Enter** to run the cell and create the plot



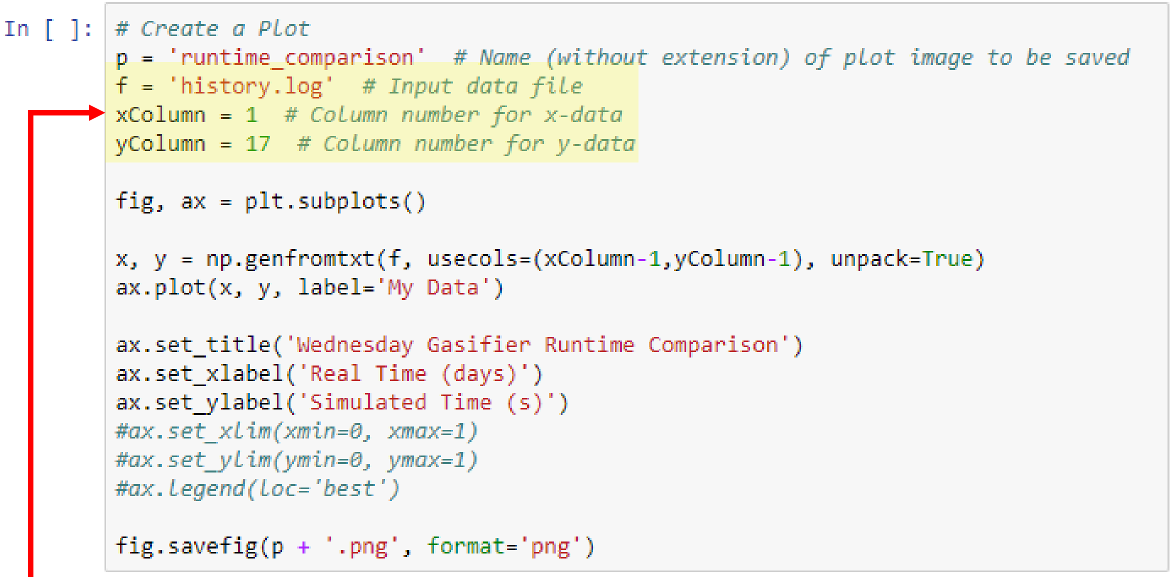
```
In [3]: # Create a Plot
        p = 'entrained_particle_mass'  # Name (without extension) of plot image to be saved
        xColumn = 1  # Column number for x-data
        yColumn = 8  # Column number for y-data

        fig, ax = plt.subplots()

        f = 'FLUXBC_cyclone_1_pressure'  # Input data file
        x, y1 = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y1, label='Cyclone 1')

        f = 'FLUXBC_cyclone_2_pressure'  # Input data file
        x, y2 = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y2, label='Cyclone 2')

        f = 'FLUXBC_cyclone_3_pressure'  # Input data file
        x, y3 = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y3, label='Cyclone 3')

        f = 'FLUXBC_cyclone_4_pressure'  # Input data file
        x, y4 = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
        ax.plot(x, y4, label='Cyclone 4')

        y = y1 + y2 + y3 + y4
        ax.plot(x, y, label='Sum of all cyclones')

        ax.set_title('Wednesday Gasifier Entrained Particle Mass')
        ax.set_xlabel('Time (s)')
        ax.set_ylabel('Mass (kg)')
        #ax.set_xlim(xmin=0, xmax=1)
        #ax.set_ylim(ymin=0, ymax=1)
        ax.legend(loc='best')

        fig.savefig(p + '.png', format='png')
```

Jupyter Notebook – Time-integrated Particle Entrainment with Summation



Jupyter Notebook – Compare Runtimes of Two Runs

- It is often useful to quantify how fast a simulation is running, or to compare two runs to see their relative runtimes. In our case, it is interesting to compare the runtimes of the Tuesday and Wednesday gasifiers. This will give us an idea of the combined impact of (1) refining the grid, and (2) adding thermal and chemistry calculations.
  - There are various ways to talk about runtime. For this example we will make a plot of the number of seconds simulated versus the number of hours of real time required. The cumulative real time spent by the solver is recorded in the history.log file under the column **"CPU (s)"**. This column will be a different number for the two simulations due to thermal and chemistry changes in history.log.
- Create a new cell below the entrainment plot cell. **Insert→Insert Cell Below**
- Copy and Paste the text from the system mass plot cell into the new cell.



```
# Create a Plot
p = 'runtime_comparison'  # Name (without extension) of plot image to be saved
f = 'history.log'  # Input data file
xColumn = 1  # Column number for x-data
yColumn = 17  # Column number for y-data

fig, ax = plt.subplots()

x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
ax.plot(x, y, label='My Data')

ax.set_title('Wednesday Gasifier Runtime Comparison')
ax.set_xlabel('Real Time (days)')
ax.set_ylabel('Simulated Time (s)')
#ax.set_xlim(xmin=0, xmax=1)
#ax.set_ylim(ymin=0, ymax=1)
#ax.legend(loc='best')

fig.savefig(p + '.png', format='png')
```

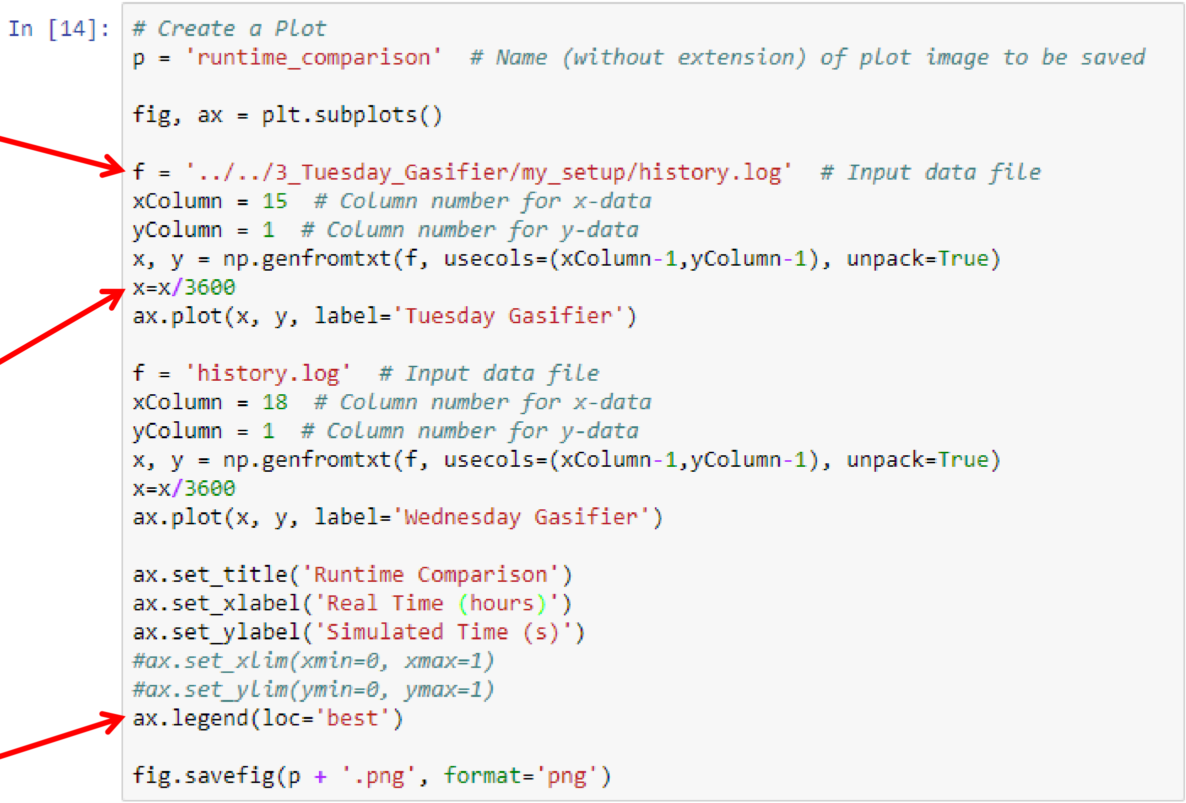
For the runtime comparison plot, the data file line and the X and Y column lines have to be moved because of differences between the two simulations. See next slide for details.

Jupyter Notebook – Compare Runtimes of Two Runs

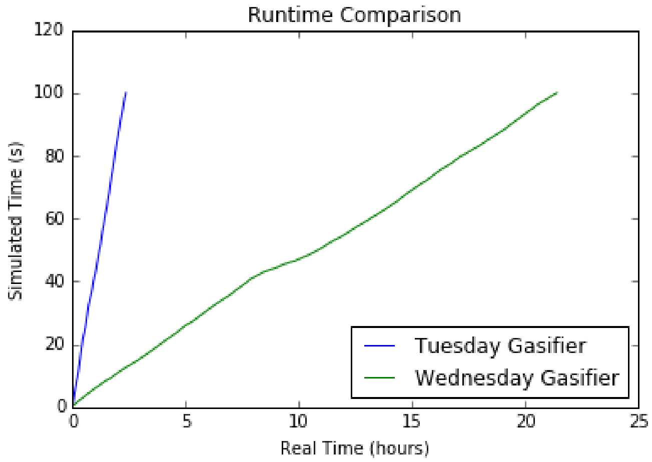
The file name for history.log for Tuesday gasifier, must have the correct path.

In order to show the x-axis in units of hours, we have to convert the history.log data from seconds to hours.

Be sure to uncomment the legend line in order to see the labels for each line.

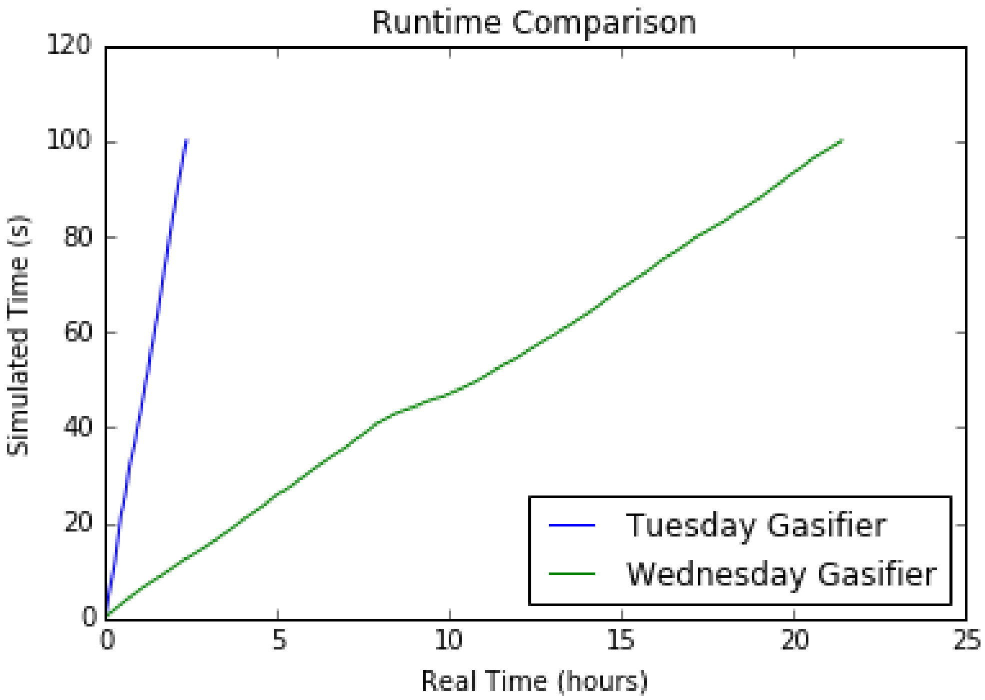


```
In [14]: # Create a Plot
         p = 'runtime_comparison'  # Name (without extension) of plot image to be saved

         fig, ax = plt.subplots()

         f = '../../3_Tuesday_Gasifier/my_setup/history.log'  # Input data file
         xColumn = 15  # Column number for x-data
         yColumn = 1  # Column number for y-data
         x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
         x=x/3600
         ax.plot(x, y, label='Tuesday Gasifier')

         f = 'history.log'  # Input data file
         xColumn = 18  # Column number for x-data
         yColumn = 1  # Column number for y-data
         x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
         x=x/3600
         ax.plot(x, y, label='Wednesday Gasifier')

         ax.set_title('Runtime Comparison')
         ax.set_xlabel('Real Time (hours)')
         ax.set_ylabel('Simulated Time (s)')
         #ax.set_xlim(xmin=0, xmax=1)
         #ax.set_ylim(ymin=0, ymax=1)
         ax.legend(loc='best')

         fig.savefig(p + '.png', format='png')
```



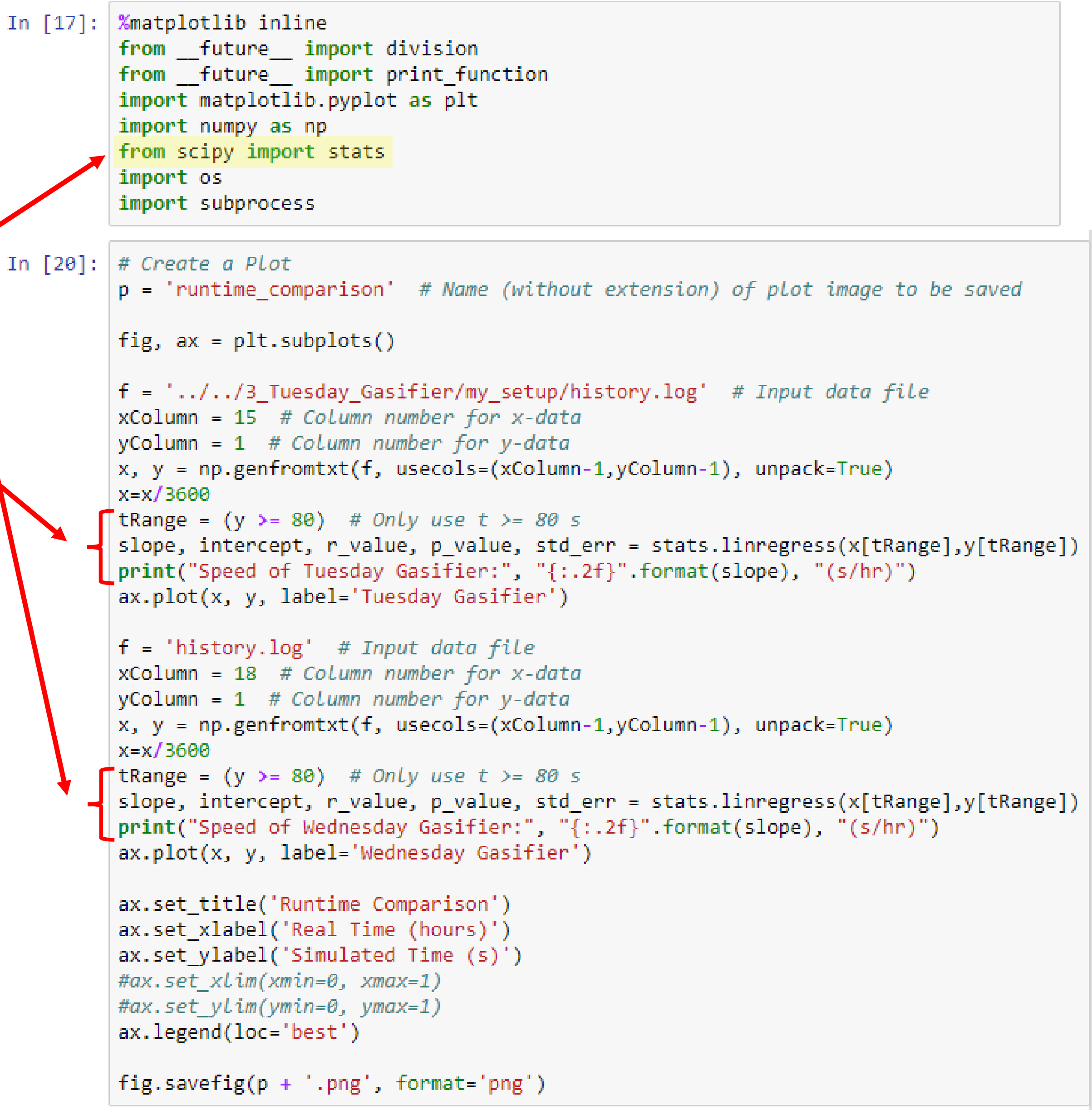
Interpreting the Runtime Plot

- One important new concept is that the first data file we read is actually in a different directory. The path `../../3_Tuesday_Gasifier/my_setup/history.log` tells the Jupyter notebook exactly which file we want.
- Another new idea is using mathematical operations on a set, and assigning the results back to that same set. The command `x=x/3600` changes the time scale of the x-axis from seconds (which is the default unit in the history.log file) to hours.
- An observation based on the plot:
  - We see that the Tuesday gasifier runs much faster than the Wednesday gasifier. It reached the 100 s end-time in a few hours, while the Wednesday gasifier took almost 24 hours to reach 100 s.
- It is important to realize the relative costs of the choices made during project setup.
  - Refining the grid can give more accurate answers, but increases the runtime.
  - Chemistry and thermal calculations are often necessary for realistic simulations, but they also add computational cost.



Determining the slope of Runtime plot

- To speak of the runtime data in terms of a rate of simulation seconds per hour of real time, we can use linear regression to calculate the slopes of the runtime lines.
- Add the highlighted line to the top cell and **Crtl→Enter** in that cell. This will import a module which will allows us to perform the regression on the lines.
- Add the following lines (remember: copy and paste is your friend).
    - The first line filters to include only the data equal to or after 80 seconds. You can change this number to see how the line regression changes as more of the data is included.
    - The second line says that you want a linear regression on the x and y data in the time range specified in the previous line.
    - The third line tells the notebook to print out the slope calculated by the linear regression, with some text before and after the numbers.
- After adding the lines, **Crtl→Enter**



In [17]:

```
%matplotlib inline
from __future__ import division
from __future__ import print_function
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import os
import subprocess
```

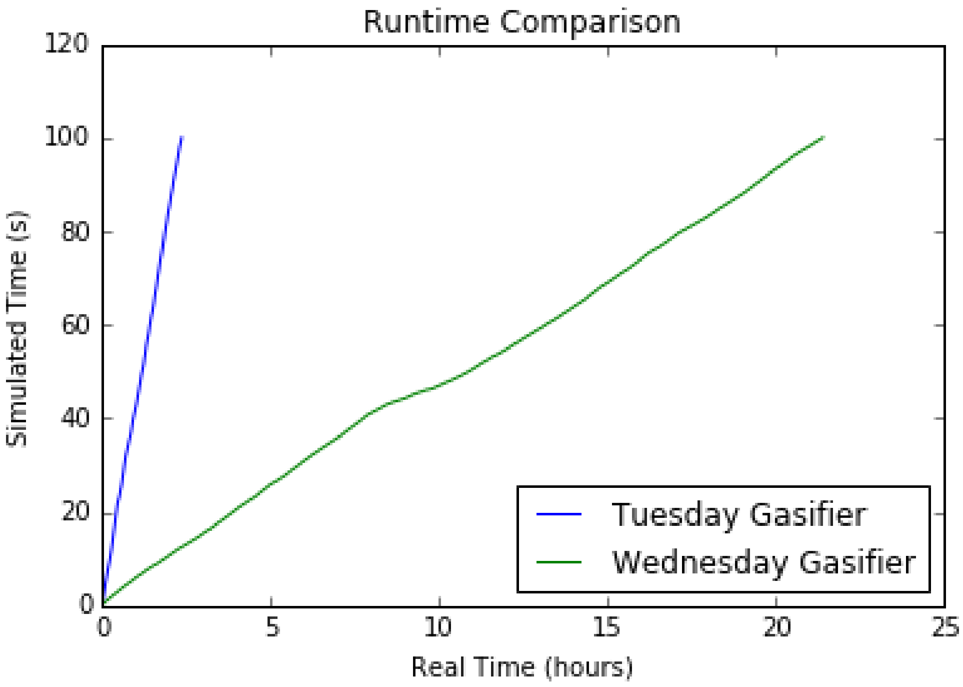
In [20]:

```
# Create a Plot
p = 'runtime_comparison'  # Name (without extension) of plot image to be saved

fig, ax = plt.subplots()

f = '../../3_Tuesday_Gasifier/my_setup/history.log'  # Input data file
xColumn = 15  # Column number for x-data
yColumn = 1  # Column number for y-data
x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
x=x/3600
tRange = (y >= 80)  # Only use t >= 80 s
slope, intercept, r_value, p_value, std_err = stats.linregress(x[tRange],y[tRange])
print("Speed of Tuesday Gasifier:", "{:.2f}".format(slope), "(s/hr)")
ax.plot(x, y, label='Tuesday Gasifier')

f = 'history.log'  # Input data file
xColumn = 18  # Column number for x-data
yColumn = 1  # Column number for y-data
x, y = np.genfromtxt(f, usecols=(xColumn-1,yColumn-1), unpack=True)
x=x/3600
tRange = (y >= 80)  # Only use t >= 80 s
slope, intercept, r_value, p_value, std_err = stats.linregress(x[tRange],y[tRange])
print("Speed of Wednesday Gasifier:", "{:.2f}".format(slope), "(s/hr)")
ax.plot(x, y, label='Wednesday Gasifier')

ax.set_title('Runtime Comparison')
ax.set_xlabel('Real Time (hours)')
ax.set_ylabel('Simulated Time (s)')
#ax.set_xlim(xmin=0, xmax=1)
#ax.set_ylim(ymin=0, ymax=1)
ax.legend(loc='best')

fig.savefig(p + '.png', format='png')
```

Interpreting the Runtime Plot

- The output from the notebook will now include the slope of each line with units of simulated seconds per hour of real time. Remember that this slope is only for 80-100s.

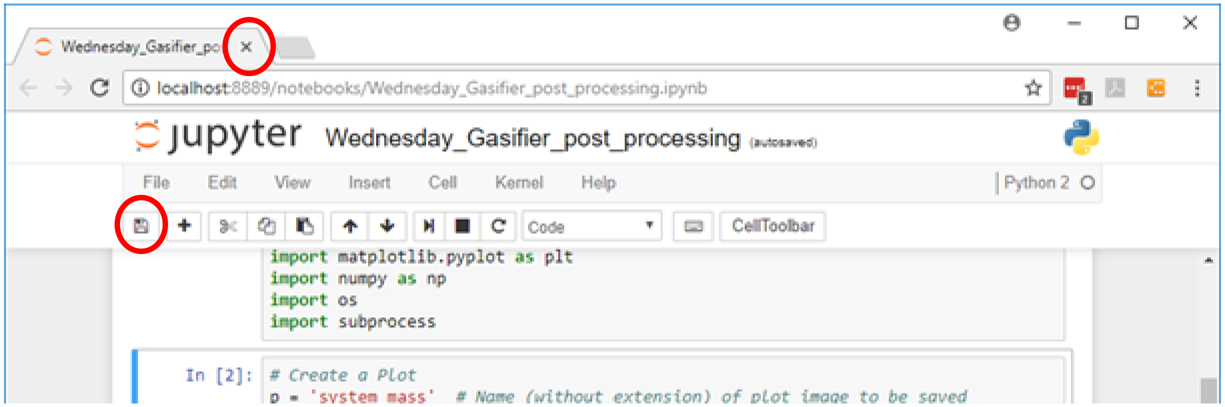
```
Speed of Tuesday Gasifier: 40.94 (s/hr)
Speed of Wednesday Gasifier: 4.93 (s/hr)
```



How to Close the Jupyter notebook

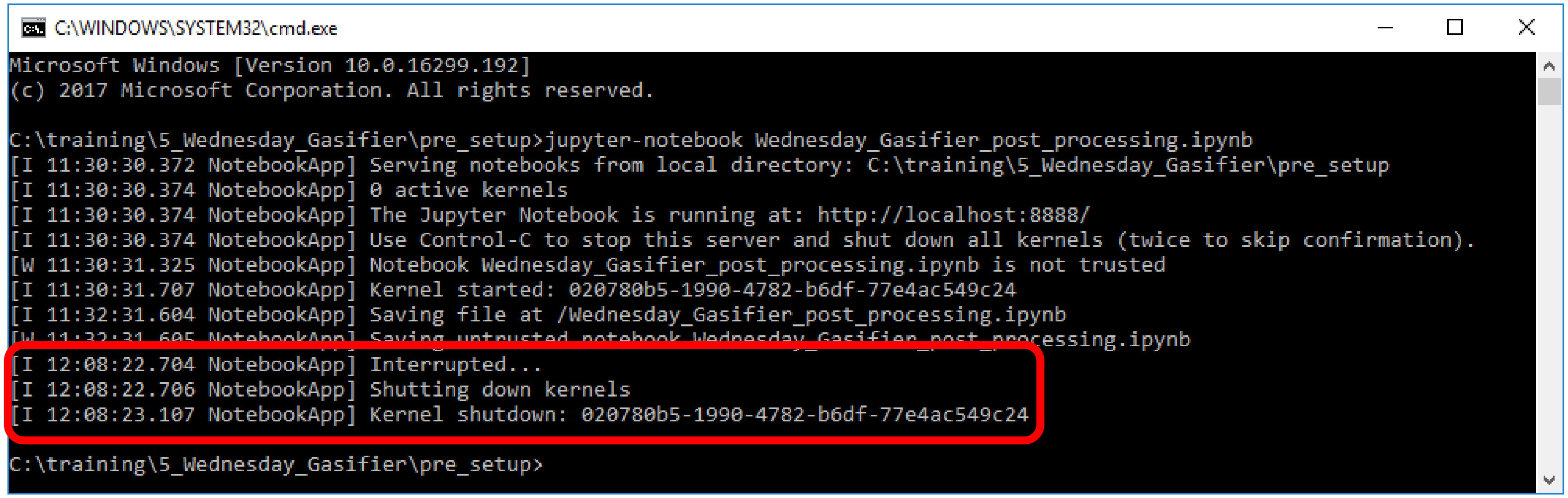
- Once you are finished using the Jupyter notebook, close it by following these steps:

1. Save the notebook, and close the browser tab


Wednesday_Gasifier_po
localhost:8889/notebooks/Wednesday_Gasifier_post_processing.ipynb
jupyter
Wednesday_Gasifier_post_processing (autosaved)
File
Edit
View
Insert
Cell
Kernel
Help
Python 2
Code
CellToolbar

```
import matplotlib.pyplot as plt
import numpy as np
import os
import subprocess

In [2]: # Create a Plot
        p = 'system mass'   # Name (without extension) of plot image to be saved
```

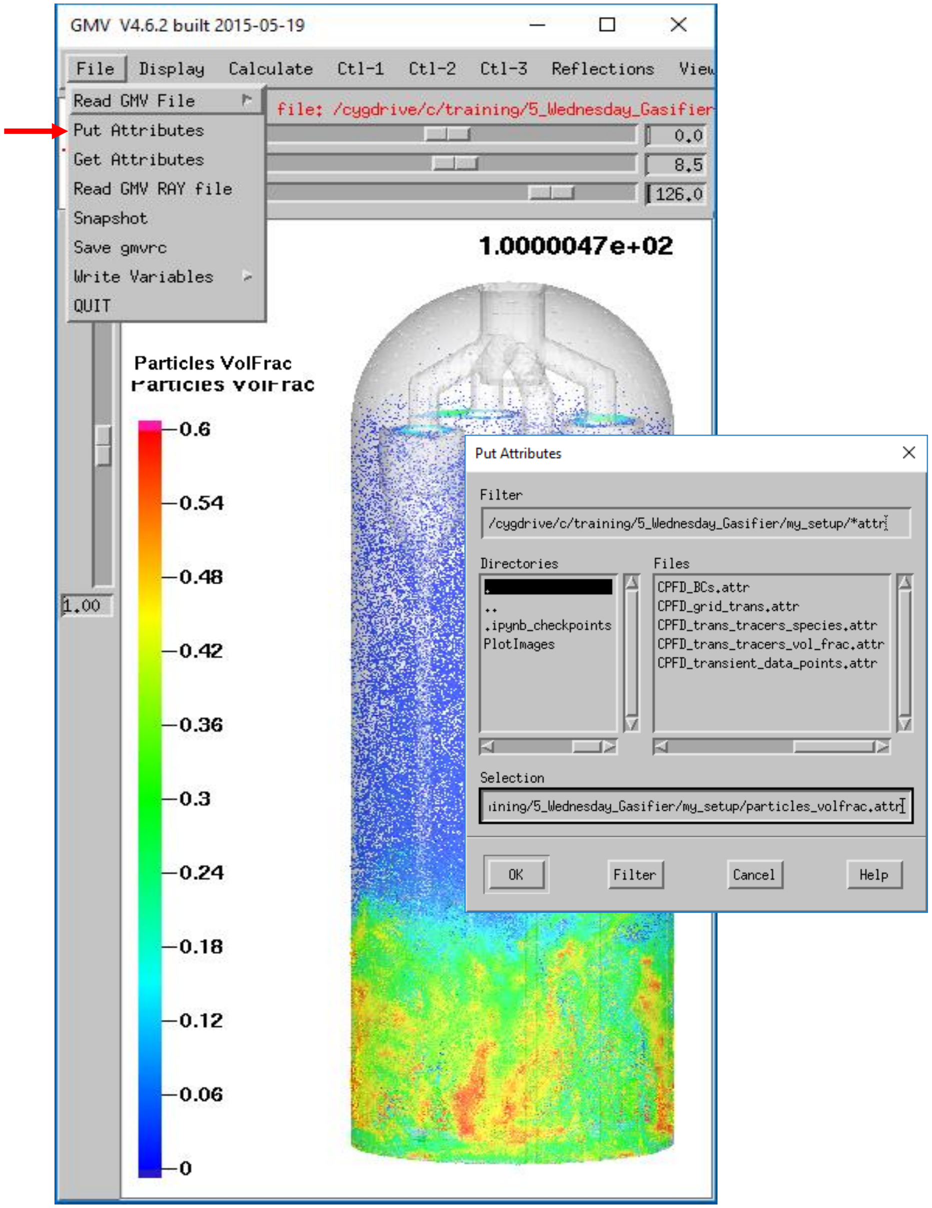
2. In the command terminal that was originally used to start the notebook, type **`Ctrl-C`** twice.



```
Microsoft Windows [Version 10.0.16299.192]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\training\5_Wednesday_Gasifier\pre_setup>jupyter-notebook Wednesday_Gasifier_post_processing.ipynb
[I 11:30:30.372 NotebookApp] Serving notebooks from local directory: C:\training\5_Wednesday_Gasifier\pre_setup
[I 11:30:30.374 NotebookApp] 0 active kernels
[I 11:30:30.374 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/
[I 11:30:30.374 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 11:30:31.325 NotebookApp] Notebook Wednesday_Gasifier_post_processing.ipynb is not trusted
[I 11:30:31.707 NotebookApp] Kernel started: 020780b5-1990-4782-b6df-77e4ac549c24
[I 11:32:31.604 NotebookApp] Saving file at /Wednesday_Gasifier_post_processing.ipynb
[W 11:32:31.605 NotebookApp] Saving untrusted notebook Wednesday_Gasifier_post_processing.ipynb
[I 12:08:22.704 NotebookApp] Interrupted...
[I 12:08:22.706 NotebookApp] Shutting down kernels
[I 12:08:23.107 NotebookApp] Kernel shutdown: 020780b5-1990-4782-b6df-77e4ac549c24

C:\training\5_Wednesday_Gasifier\pre_setup>
```

GMV Attribute Files

- GMV uses attribute files to store view settings.
- Open a GMV window with particles colored by volume fraction.
- Tip: whenever possible, start with the shortcut buttons in the Barracuda GUI. In this case use **Post-Run → Particle Volume Fraction**.
- Resize the GMV window so that the view “fits” the geometry. For the Wednesday Gasifier geometry, a window that is tall and not too wide works well.
- Save an attribute file of the view with a name such as “particles_volfrac.attr”
- Close GMV and open a command-line terminal in your current project directory for the next step in creating an animation from the attribute file just saved.



file: /cygdrive/c/training/5_Wednesday_Gasifier

Filter: /cygdrive/c/training/5_Wednesday_Gasifier/my_setup/*attr

Directories:
- ..
- .ipynb_checkpoints
- PlotImages

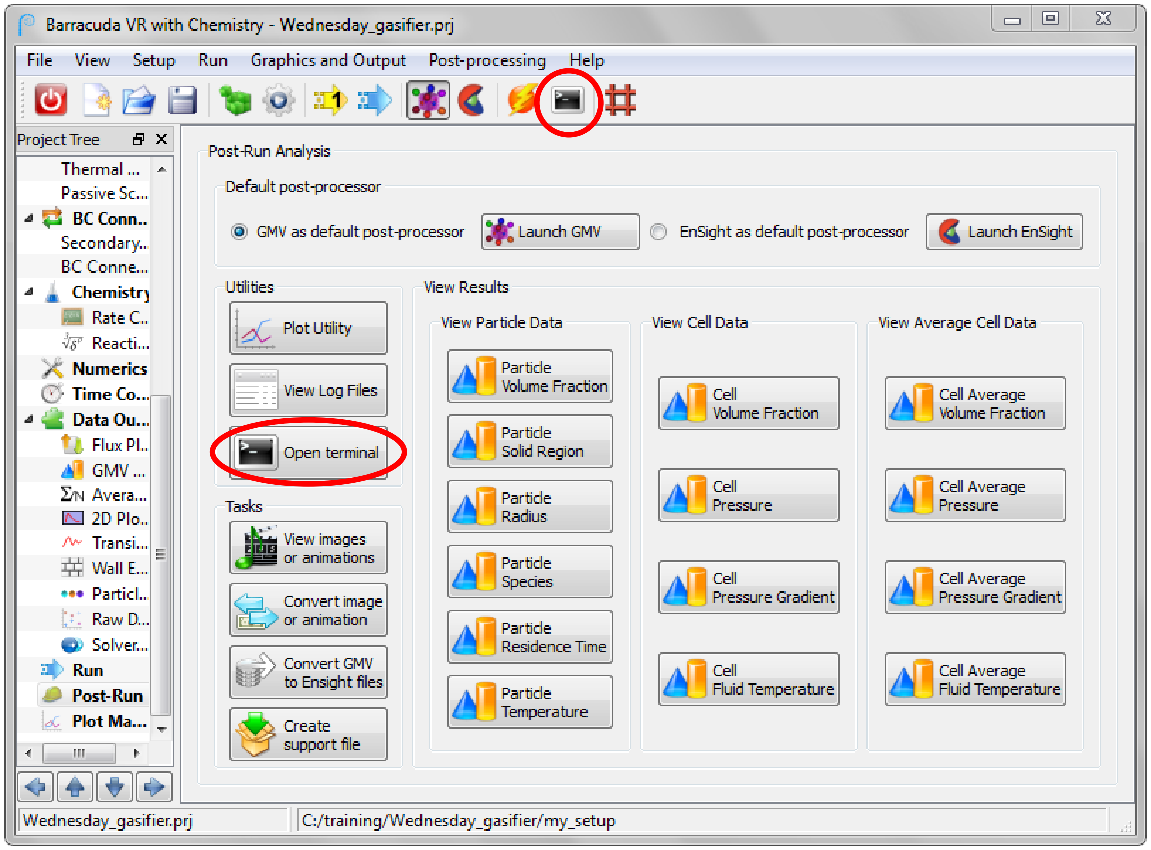
Files:
- CPFD_BCs.attr
- CPFD_grid_trans.attr
- CPFD_trans_tracers_species.attr
- CPFD_trans_tracers_vol_frac.attr
- CPFD_transient_data_points.attr

Selection: ining/5_Wednesday_Gasifier/my_setup/particles_volfrac.attr

Opening a Command-Line Terminal

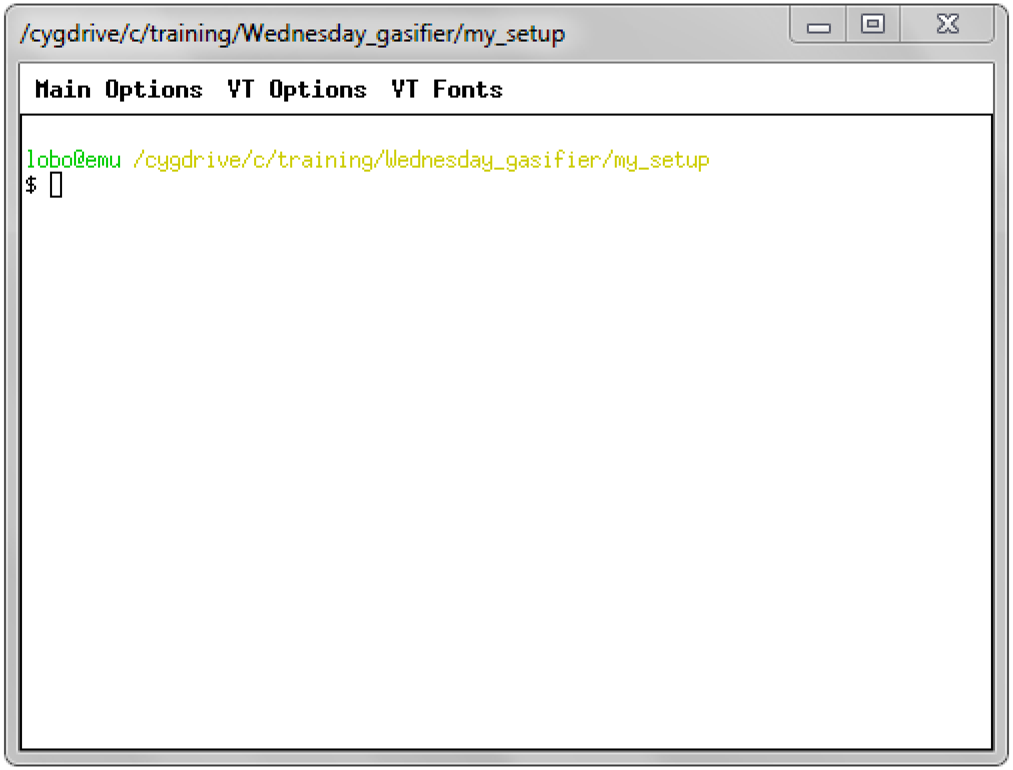
- The Barracuda VR GUI has several built-in ways to open a command-line terminal.
    - A shortcut button is always available on the top shortcut button bar.
    - The **Open Terminal** button is available in the **Post-Run** section of the GUI.
    - There is a menu item: **Post-processing → Open Terminal**
- When any of these is used, a new terminal is opened in the currently opened project's working directory.
- Depending on which operating system you are using, the following terminals will be opened by default when you use the GUI's **Open Terminal** functionality.
    - Linux: `xterm`
    - Windows: `cmd`
- **Tip**: On Windows, in order to run some scripts (*.sh) in this presentation, start an `xterm` from the `cmd` window by simply typing:

```
xterm
```



Getting to Know the Terminal

- Depending on whether you are using a native Linux version of `xterm`, or the version included in Cygwin on Windows, the prompt may look slightly different. However, the same basic information is usually presented:
    - Username
    - Machine name
    - Current working directory
    - A prompt symbol, indicating that the terminal is ready to accept input
- **Tip**: `xterm` supports tab-completion, which allows you to start typing commands or file names, then press the **Tab** key on your keyboard and the terminal will finish your command or file name if a match is available.
- **Tip**: `xterm` has command history built-in, which allows you to re-run previous commands easily. At the prompt, use the **Up** arrow key on your keyboard to cycle through previously typed commands.



```
lobo@emu /cygdrive/c/training/Wednesday_gasifier/my_setup
$ 
```

Using BATCHMOVIE.sh

- BATCHMOVIE.sh is a script that creates animations from GMV files, using attribute files such as the one we just saved.
- Run BATCHMOVIE.sh with the following command:

Commands needed in Windows are highlighted.

```
BATCHMOVIE.sh particles_volfrac.attr -mp4
```

Attribute file name

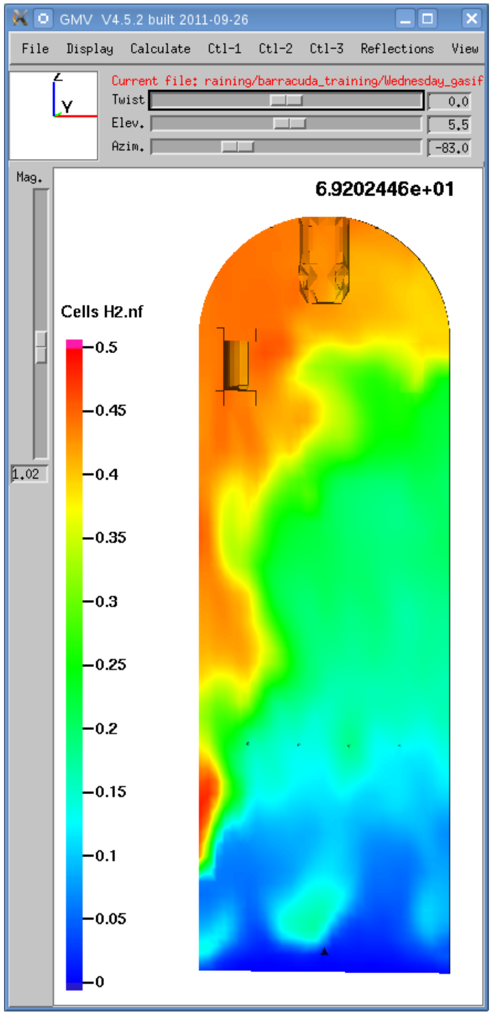
- You will see information scrolling in the terminal as each GMV file in the directory is read, and an image is made based on the attribute file specified.
- The command shown above is the simplest syntax of BATCHMOVIE.sh. It assumes that you want to create images from all Gmv* files in the current directory. If you do not want to use all Gmv* files, or if you want to do something else more advanced, see the BATCHMOVIE.sh usage information that is displayed when you call the script with no arguments:

```
BATCHMOVIE.sh
```

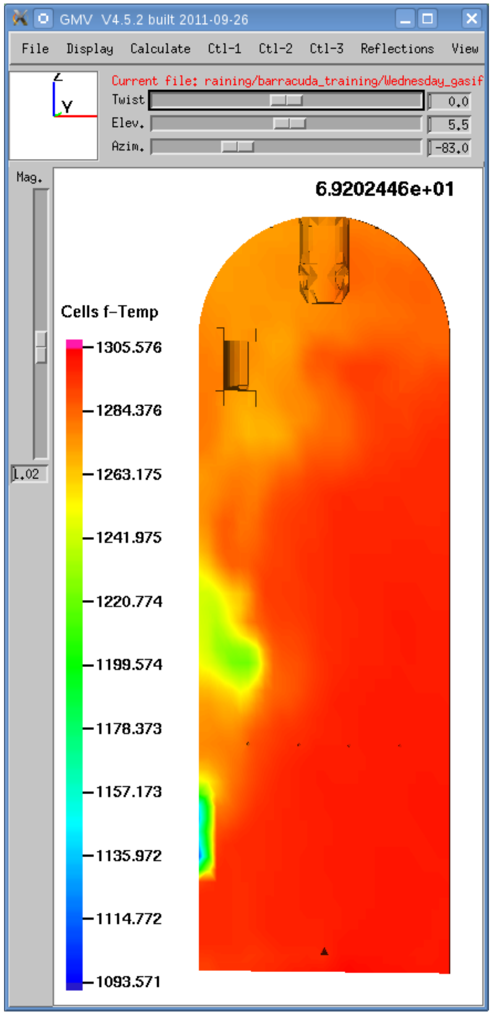
- At the end of the process, BATCHMOVIE.sh will create an animation with name <attribute>.mpg (**Linux**) or <attribute>.mp4 (**Windows**) and return control back to the terminal. To play the animation, use the command:
    - **Linux** - `xanim particles_volfrac.mpg`
    - **Windows** – play the file with Windows Media player or similar video player

Make Several Animations

- Create two more views of the simulation results. Shown at right are suggestions.
- **Tip**: remember to start with the built-in **Post-Run** shortcut buttons in the Barracuda VR GUI. These are the easiest way to open GMV with a view that is probably close to what you want.
- For each view, save an attribute file.
- Proceed to the next slide to create a script that makes animations from all of your available attribute files.
- **Tip:** It is a good idea to make all of your GMV windows the same size so that the side-by-side views are consistent. Use **Ctl-3 →Window size** to specify a window size that fits the geometry well and use that same dimension for each GMV window.



Cells colored by mole fraction of H2, half-section of vessel using subset. Attribute filename: cells_H2.nf.attr



Cells colored by fluid temperature, half-section of vessel using subset. Attribute filename: cells_f-Temp.attr

MAKE_ANIMATIONS.sh Script

- If you have a long-running simulation, it is useful to combine several BATCHMOVIE.sh calls into a single script. This way, when you want to update all of the animations, you run just the single script.
- BATCHMOVIE.sh was designed so that it only creates images from GMV files that were not present the last time it was run. This makes it very efficient at updating animations, since any frames that were previously created do not have to be made again.
- **Linux**: Create a new script using a command like this:

```
gedit MAKE_ANIMATIONS.sh
```

← Specifying the filename here creates a new file.

- **Windows**: Use Notepad++ to create a text file with the name MAKE_ANIMATIONS.sh (see next slide)
- In the script, put the following commands:

```
#!/bin/bash

BATCHMOVIE.sh cells_f-Temp.attr -mp4
BATCHMOVIE.sh cells_H2.nf.attr -mp4
BATCHMOVIE.sh particles_volfrac.attr -mp4
```

← Each line is executed in turn, so all three animations will be created when the script finishes running.

- For **Linux**, do not include the `-mp4` after each attribute file name.
- Save the file and close text editor.
- **Linux**: Make the script executable:

```
chmod +x MAKE_ANIMATIONS.sh
```

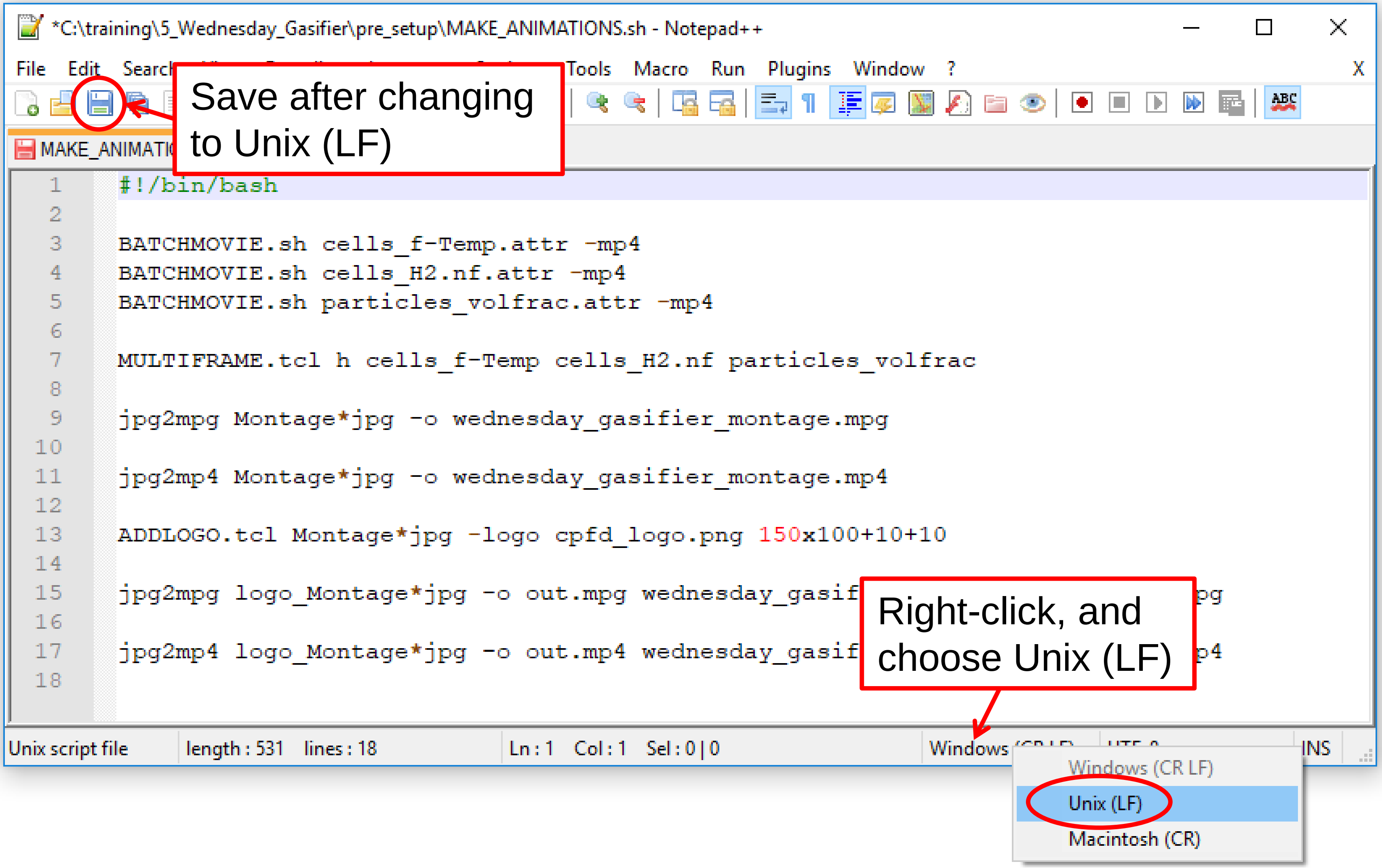
- Run the script by typing the script name in the terminal and pressing Enter:

```
MAKE_ANIMATIONS.sh
```

← Depending on how many GMV files are in your directory, and how big they are, the script could take several minutes to run. Even so, it is faster than using the GMV GUI auto-read feature.

Using Notepad++ for Writing Scripts in Windows

- When using Cygwin on Windows, scripts need to be saved with Unix (LF) line endings. You only need to perform this step once, the first time you save <u>each script</u>.



```
#!/bin/bash

BATCHMOVIE.sh cells_f-Temp.attr -mp4
BATCHMOVIE.sh cells_H2.nf.attr -mp4
BATCHMOVIE.sh particles_volfrac.attr -mp4

MULTIFRAME.tcl h cells_f-Temp cells_H2.nf particles_volfrac

jpg2mpg Montage*jpg -o wednesday_gasifier_montage.mpg

jpg2mp4 Montage*jpg -o wednesday_gasifier_montage.mp4

ADDLOGO.tcl Montage*jpg -logo cpfd_logo.png 150x100+10+10

jpg2mpg logo_Montage*jpg -o out.mpg wednesday_gasif

jpg2mp4 logo_Montage*jpg -o out.mp4 wednesday_gasif

```

Unix script file   length : 531   lines : 18   Ln : 1   Col : 1   Sel : 0 | 0   Windows

Save after changing to Unix (LF)

Right-click, and choose Unix (LF)

Using MULTIFRAME.tcl to Combine Animations

- Combining several animations into a single animation is very useful, especially when trying to understand and explain how different aspects of physics, thermal, and chemistry interact.
- MULTIFRAME.tcl is a script that operates on multiple sets of images, combining them into a new set of images starting with “Montage”.
- Using the three animations that we just created, add to your MAKE_ANIMATIONS.sh script so that it includes a MULTIFRAME.tcl command and a `jpg2mpg` (**Linux**) or a `jpg2mp4` (**Windows**) command to combine the resulting Montage*jpg files into a final animation.

```
 #!/bin/bash

BATCHMOVIE.sh cells_f-Temp.attr -mp4
BATCHMOVIE.sh cells_H2.nf.attr -mp4
BATCHMOVIE.sh particles_volfrac.attr -mp4

MULTIFRAME.tcl h cells_f-Temp cells_H2.nf particles_volfrac

jpg2mpg Montage*jpg -o wednesday_gasifier_montage.mpg

jpg2mp4 Montage*jpg -o wednesday_gasifier_montage.mp4
```

- Save, close, and run the script. Play the resulting animation.

MULTIFRAME.tcl to Combine Animations - explained

```
#!/bin/bash

BATCHMOVIE.sh cells_f-Temp.attr -mp4
BATCHMOVIE.sh cells_H2.nf.attr -mp4
BATCHMOVIE.sh particles_volfrac.attr -mp4

MULTIFRAME.tcl h cells_f-Temp cells_H2.nf particles_volfrac
```

Directories where images are stored.

Linux:

```
jpg2mpg Montage*jpg -o wednesday_gasifier_montage.mpg
```

`jpg2mpg` creates an output file named out.mpg by default. You can use the –o flag to specify a different output file name.

Windows:

```
jpg2mp4 Montage*jpg -o wednesday_gasifier_montage.mp4
```

`jpg2mp4` creates an output file named out.mp4 by default. You can use the –o flag to specify a different output file name.

New output file name.

Adding Logos to Animations

- Adding your company’s logo to an animation can be accomplished using the ADDLOGO.tcl script.
- Modify the MAKE_ANIMATIONS.sh script to add an ADDLOGO.tcl command, as well as another `jpg2mp4` call:

```
 #!/bin/bash

BATCHMOVIE.sh cells_f-Temp.attr -mp4
BATCHMOVIE.sh cells_H2.nf.attr -mp4
BATCHMOVIE.sh particles_volfrac.attr -mp4

MULTIFRAME.tcl h cells_f-Temp cells_H2.nf particles_volfrac

jpg2mpg Montage*jpg -o wednesday_gasifier_montage.mpg

jpg2mp4 Montage*jpg -o wednesday_gasifier_montage.mp4


ADDLOGO.tcl Montage*jpg -logo cpfd_logo.png 150x100+10+10


jpg2mpg logo_Montage*jpg -o out.mpg wednesday_gasifier_montage_with_logo.mpg

jpg2mp4 logo_Montage*jpg -o out.mp4 wednesday_gasifier_montage_with_logo.mp4
```

List of images to which logo is added.

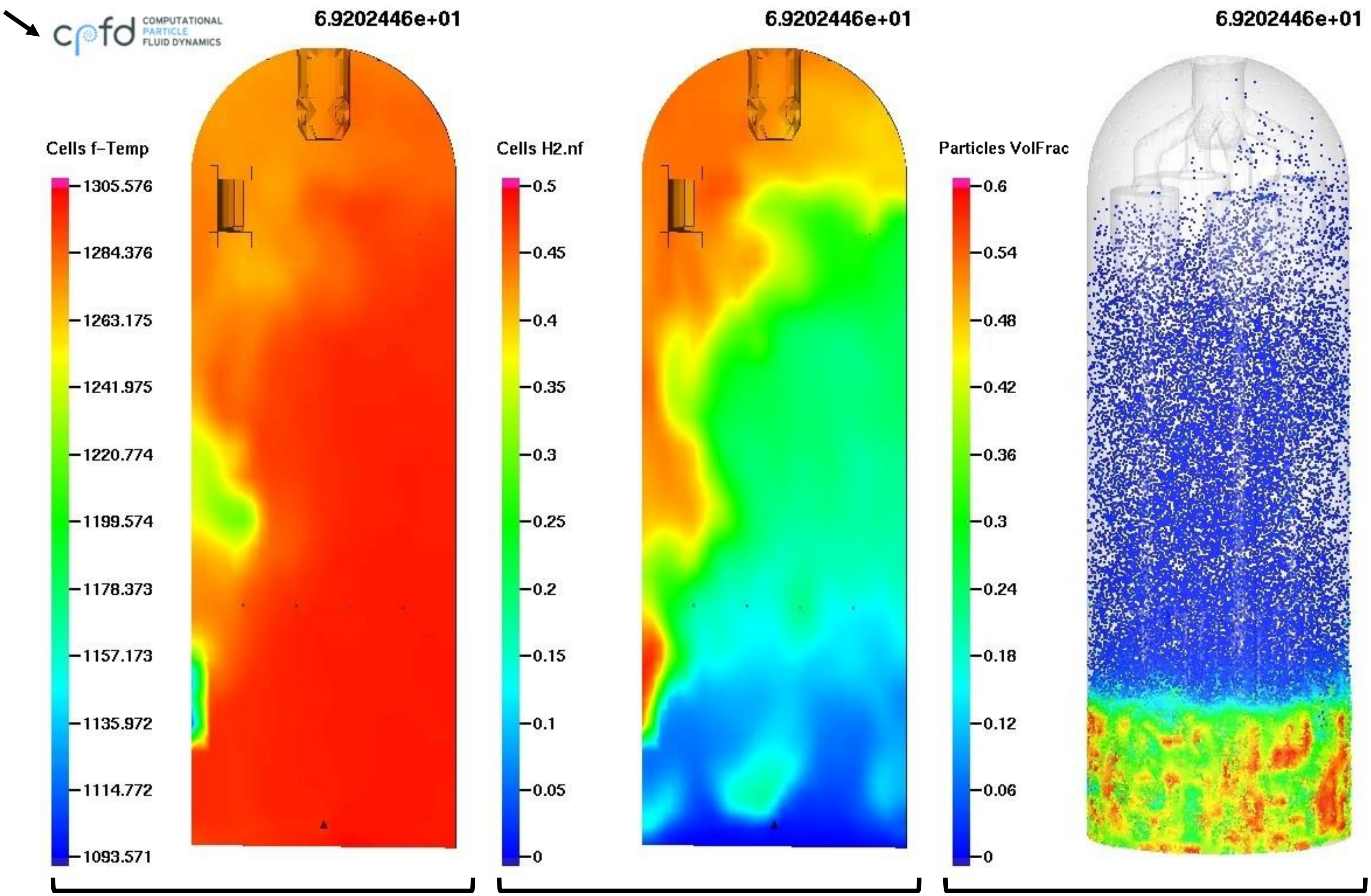
Size and position.

Logo image name.

- Save, close, and run the script. Play the resulting animation.

The Final Animation

Logo



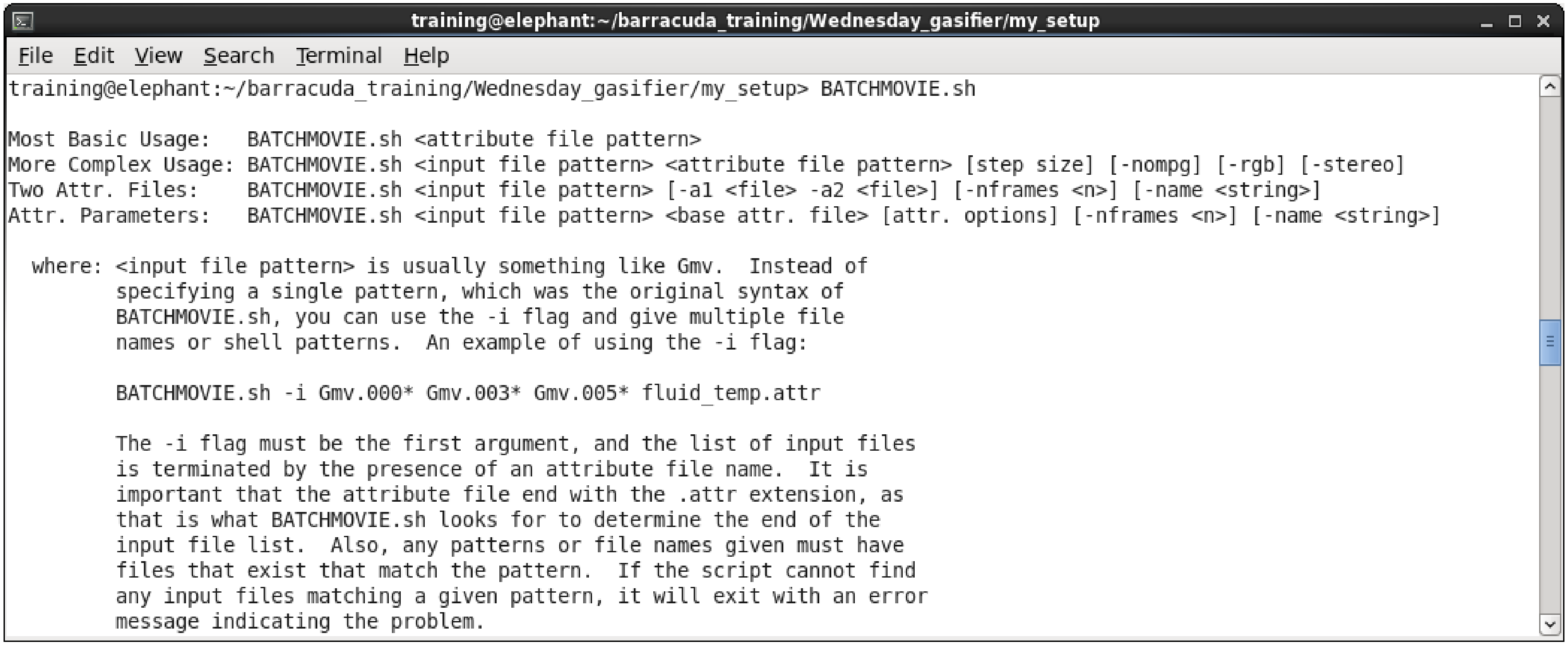
Three animations side-by-side.

Additional Post-Processing

- Using the skills learned for plotting and making animations, answer the following questions.
  - We set the cyclone dipleg flow BCs to feed particles at the same mass flow rate as they are being entrained into each cyclone. Are the feed BCs operating correctly? Are they able to keep up with the entrained mass flow rate?
  - Several transient data points were defined in the project setup to monitor fluid temperature. Plot the data from these data points, and determine whether there is a significant temperature difference between the bottom and top of the gasifier vessel. **Hint:** the data is written to a file named “trans.data00”.
  - Does the addition of thermal and chemistry calculations increase the particle entrainment? Plot the entrainment from the “early entrainment flux plane” for both the Tuesday and Wednesday gasifiers.
  - Make a combined animation of all gas species mole fractions. **Hint:** for large combined animations, make sure each individual animation is small enough in size. You want to make sure the final combined animation will fit on a typical screen.

Getting Help with the Scripts

- For information about all of the options available for Jupyter notebook, consult the Jupyter website: http://jupyter.org/index.html
- For help with Python plotting commands, see: https://matplotlib.org/gallery/index.html
- For help with GMV, consult the GMV user's manual, which is installed by default with Barracuda. Use the Barracuda GUI menu item **Help, GMV User's Manual**.
- For help with the utility scripts introduced in this presentation (BATCHMOVIE.sh, MULTIFRAME.tcl, jpg2mpg, and ADDLOGO.tcl), run the script without any arguments. A help message will be printed.


training@elephant:~/barracuda_training/Wednesday_gasifier/my_setup
File
Edit
View
Search
Terminal
Help

```
training@elephant:~/barracuda_training/Wednesday_gasifier/my_setup> BATCHMOVIE.sh

Most Basic Usage:   BATCHMOVIE.sh <attribute file pattern>
More Complex Usage: BATCHMOVIE.sh <input file pattern> <attribute file pattern> [step size] [-nompg] [-rgb] [-stereo]
Two Attr. Files:    BATCHMOVIE.sh <input file pattern> [-a1 <file> -a2 <file>] [-nframes <n>] [-name <string>]
Attr. Parameters:   BATCHMOVIE.sh <input file pattern> <base attr. file> [attr. options] [-nframes <n>] [-name <string>]

  where: <input file pattern> is usually something like Gmv.  Instead of
         specifying a single pattern, which was the original syntax of
         BATCHMOVIE.sh, you can use the -i flag and give multiple file
         names or shell patterns.  An example of using the -i flag:

         BATCHMOVIE.sh -i Gmv.000* Gmv.003* Gmv.005* fluid_temp.attr

         The -i flag must be the first argument, and the list of input files
         is terminated by the presence of an attribute file name.  It is
         important that the attribute file end with the .attr extension, as
         that is what BATCHMOVIE.sh looks for to determine the end of the
         input file list.  Also, any patterns or file names given must have
         files that exist that match the pattern.  If the script cannot find
         any input files matching a given pattern, it will exit with an error
         message indicating the problem.
```

Basic Commands for Survival

- The following commands are frequently used in the command-line terminal, and in scripts:
  - Change directory: `cd`
    - Entering `cd` without any arguments takes you to your home directory: `cd`
    - To go up one folder from where you are, use: `cd ../`
  - You can use either "relative" or "absolute" paths.
    - Relative: `cd documents/`
    - Absolute: `cd /home/lobo/project/`
  - List directory contents: `ls`
    - To list all details about files in your current directory: `ls -l`
    - To list all PNG files in your current directory: `ls *png`
  - "Change file mode" (permissions): `chmod`
    - Make "PLOTRESULTS.sh" executable: `chmod +x PLOTRESULTS.sh`
    - Make "myfolder" (and all its contents) read+write: `chmod -R +rw myfolder/`
  - Manual pages: `man`
    - Learn about the ls command: `man ls`
    - To get out of a man page, type "q".

Debugging Shell Scripts

- Like programs written in any other language, shell scripts will often have bugs. Here are some tips to help you in tracking down and fixing problems in your shell scripts.
  - Use a text editor with syntax highlighting. This will help to make sure that your scripts are syntactically correct.
  - Write small portions of the script at a time, testing it after each portion is done.
  - Use the shell. Any command that you are putting into a shell script can be run directly at a terminal. Does the command behave properly in a terminal?
  - Use `echo` to output messages at key points in the script. For example:

```
#!/bin/bash
echo “Entering run_01 directory”
cd run_01
echo “Leaving run_01 directory”
cd ../
```

  - Always test scripts on non-critical data first! You do not want to accidentally delete or overwrite important files with a mistyped command.

Barracuda Command-Line Reference

- You can start the solver from the command-line. This is useful in cases where command-line only access is available, or when you want to script multiple solver runs in sequence. If your project is named `my_project.prj`, you can run it from the command-line with this command:

  ```
  cpfd.x.17 my_project.prj
  ```

- If you have transient data files in the project directory already, and you run the above command, the solver will ask whether you want to overwrite, append, create new files, or quit. The most common options that people usually want are overwrite or append. You can specify an optional flag when calling the solver to automatically answer this question.

  Overwrite: `cpfd.x.17 -ow my_project.prj`
  Append: `cpfd.x.17 -aw my_project.prj`

- You can generate the grid from the command-line. You need to have already defined the grid through the Barracuda GUI first, and saved the project file so that the grid.i file is created by the GUI.

  ```
  cpfd.g.17 grid.i
  ```

- You can start the Interact utility from the command-line. You need to be in the project directory where the solver is running when starting the Interact utility. Otherwise, the solver will not receive the Interact signal.

  ```
  act.17
  ```

- You can start the Barracuda GUI from the command-line: `barracuda.17`

- You can see all solver options from the command-line: `cpfd.x.17 -help`

Note on Text Editors in Windows

- The default text editor in Windows is Notepad which only reads and writes DOS-based end-of-line (EOL) characters. Cygwin, however, requires UNIX-based EOL characters. This difference can create confusion when creating scripts in Notepad, which then need to be run in Cygwin.
- The best solution is to install a more advanced text editor, such as **Notepad++**, which will read and write Unix-based EOL characters.
- If basic Notepad must be used in Windows for creating scripts, two separate utilities are required for converting the EOL characters between DOS and UNIX.

- **Use unix2dos.exe to prepare a script for editing in Notepad:**

  `unix2dos.exe myscript.sh` ← Converts DOS EOL characters to UNIX

  `notepad.exe myscript.sh` ← Opens the script in notepad

- **Use dos2unix.exe to prepare a script for running in Cygwin:**

  `dos2unix.exe myscript.sh` ← Converts UNIX EOL characters to DOS

  `./myscript.sh` ← Executes the script