

Oily Python: a Reservoir Engineering Perspective

PyAr – November 17, 2012

Andrea Gavana
Maersk Oil

andrea.gavana@gmail.com
andrea.gavana@maerskoil.com



Outline

- ✓ What reservoir engineers do
- ✓ Data pre-processing and number crunching – *xlrd* and *numpy*
- ✓ 2D visualizations – *matplotlib*
- ✓ 3D visualizations – *VTK*, *mayavi* and *NetworkX*
- ✓ Integration with the reservoir numerical simulator – *f2py*
- ✓ Automation and N-D interpolation – Python and *scipy*
- ✓ Graphical user interfaces (GUIs) – *wxPython*

Presentation samples: <http://www.infinity77.net/pycon/oily.zip>



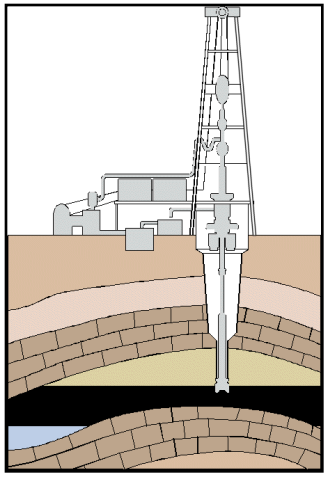
What We Do

- ✓ Using all sorts of real-life measurements:
 - Man-made seismic waves
 - Detailed record of the geologic formations penetrated by a well (*logs*)
 - Rock properties, oil/water/gas content in the reservoir rock
 - Pressure/temperature vs. depth in a well
 - Oil/water/gas production rates measured at the well
 - ... and many others ...

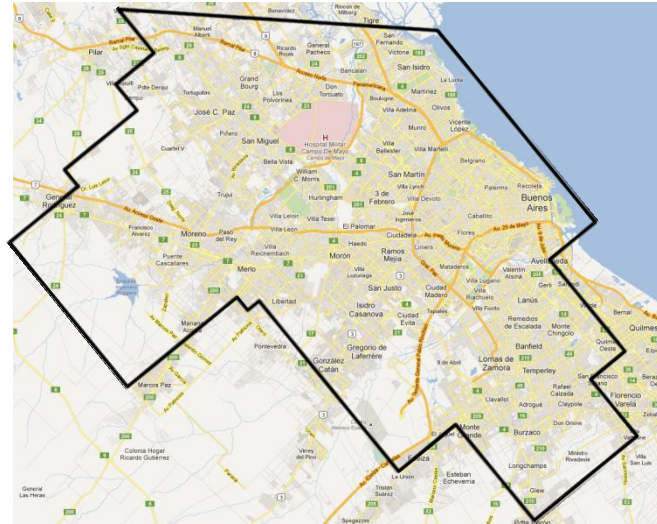
- ✓ A reservoir engineer:
 - Builds a 3D numerical model representing the reservoir and runs time-dependent fluid flow simulations
 - Tries to calibrate that model, i.e., match the simulated results with the real data
 - Using the calibrated model, tries to predict the future performances of the field



What We Do – Complications



1 – 10 Km



50 – 60 Km



20 – 100 GB

- ✓ Located underground: we can't go and see what's in there
- ✓ Sheer areal size – hard to accurately model numerically
- ✓ Huge amount of data to pre-process and integrate
- ✓ Each simulation can easily generate 100 GB of results to analyze

Data Pre-Processing

“When fed with garbage data, a simulator is a machine that calculates meaningless results with incredible precision.”

- ✓ A big part of the job is to ensure that the input data makes sense
 - Measurements come from many, unrelated sources
 - Data frequency – both in time and depth – varies wildly
 - Deep and thorough data checking needs to be carried out
- ✓ Dense visual representations of the input data are fundamental
 - Nothing beats seeing an image of your data to spot errors
 - Automatic filters and data adjustments (via Python code) are inherently limited
- ✓ Cleaned, sensible data can then be used to feed the simulation
 - One possible source of errors has been removed



Data Pre-Processing – *xlrd*

- ✓ Part of the data comes in Excel format (sigh...) – I am no friend with Excel
- ✓ *xlrd* is a great, multi-platform Python package to read Excel files
 - Fast as a rabbit – faster than Excel itself
 - Works around many Excel bugs (especially *datetime*-related)

```
# Open the Excel file
book = xlrd.open_workbook('example_1.xls')
# Get the first worksheet
sheet = book.sheet_by_index(0)

# Allocate an empty numpy array
values = numpy.zeros((sheet.nrows, 3))

# Loop over all the Excel sheet rows
for row in range(1, sheet.nrows):

    # Get the well name
    well_name = sheet.cell(row, 0).value

    # Column B should be a date...
    cell_type = sheet.cell(row, 1).ctype
    cell_value = sheet.cell(row, 1).value

    if cell_type == xlrd.XL_CELL_DATE:
        # It's a date!
        date = xlrd.xldate_as_tuple(cell_value, book.datemode)
        date = datetime.date(*date[0:3])

    # Store production data into a numpy array
    for col in range(3):
        values[row, col] = sheet.cell(row, col+2).value
```

- ✓ Smoothly handles different cell types (empty, text, number, boolean, etc...)
- ✓ Various Excel-errors handling (#REF!, #DIV/0!, #VALUE!, etc...)
- ✓ Info on cell fonts, formats, formulae
- ✓ It's the base of *XLSGrid* (an AGW widget in *wxPython*) ☺



Oily sample: *xlrd_1.py*



MAERSK
OIL

Number Crunching and I/O

Task of the day

- ✓ Quality check of the electrical measurements on a well (*logs*)
- ✓ Depth-based data at 15cm intervals (well length can be more than 10Km)
- ✓ Free format text file with variable-length headers
 - Data is organized in columns
- ✓ We only care about depth, rock property and water content
 - All other data is discarded
- ✓ Unphysical values must be filtered out ($X < 0$ or $X > 1$)
- ✓ Cleaned data is then exported in another format
 1. Keeping original depth intervals (15cm)
 2. Averaging rock property and water content every 6m



Number Crunching and I/O

```
TextPad - [C:\Users\AGA108\Desktop\PyAr\python\scientific\log.pm]
File Edit Search View Tools Macros Configure Window Help
# LAS format log file from PETREL
# Project units are specified as depth units
#-----
~Version information
VERS 2.0:
WRAP NO:
#-----
~Well
STRT .ft 2823.0000000 :
STOP .ft 25564.500000 :
STEP .ft 0.50000000 :
NULL -999.250000 :
COMP : COMPANY
WELL WELL : WELL
FLD : FIELD
LOC : LOCATION
SRVC : SERVICE COMPANY
DATE Monday, August 06 2012 13:46:51 : DATE
PROV : PROVINCE
UWI : UNIQUE WELL ID
API : API NUMBER
#-----
~Curve
DEPT .ft : DEPTH
BWV .RB/STB : BWV
SRVC : SRVC
```

Header

```
TextPad - [C:\Users\AGA108\Desktop\PyAr\python\scientific\log.pm]
File Edit Search View Tools Macros Configure Window Help
~Parameter
#-----
~Ascii
2823.0000000 0.2229000032 2.5000000000 -999.250000 12.810000420 -
2823.5000000 0.2229000032 2.5000000000 -999.250000 12.810000420 -
2824.0000000 0.2229000032 2.5000000000 -999.250000 9.5500001907 -
2824.5000000 0.2229000032 2.5000000000 -999.250000 10.140000343 -
2825.0000000 0.2229000032 2.5000000000 -999.250000 12.060000420 -
2825.5000000 0.2229000032 2.5000000000 -999.250000 10.550000191 -
2826.0000000 0.2229000032 2.5000000000 -999.250000 10.550000191 -
2826.5000000 0.2229000032 2.5000000000 -999.250000 9.8000001907 -
2827.0000000 0.2229000032 2.5000000000 -999.250000 9.0500001907 -
2827.5000000 0.2229000032 2.5000000000 -999.250000 8.6724996567 -
2828.0000000 0.2229000032 2.5000000000 -999.250000 8.2950000763 -
2828.5000000 0.2229000032 2.5000000000 -999.250000 7.9175000191 -
2829.0000000 0.2229000032 2.5000000000 -999.250000 7.5399999619 -
2829.5000000 0.2229000032 2.5000000000 -999.250000 9.0500001907 -
2830.0000000 0.2229000032 2.5000000000 -999.250000 8.7700004578 -
2830.5000000 0.2229000032 2.5000000000 -999.250000 7.9899997711 -
2831.0000000 0.2229000032 2.5000000000 -999.250000 8.9200000763 -
2831.5000000 0.2229000032 2.5000000000 -999.250000 8.9200000763 -
2832.0000000 0.2229000032 2.5000000000 -999.250000 10.680000305 -
2832.5000000 0.2229000032 2.5000000000 -999.250000 11.239999771 -
2833.0000000 0.2229000032 2.5000000000 -999.250000 10.930000305 -
2833.5000000 0.2229000032 2.5000000000 -999.250000 10.430000305 -
```

Data

Problem size and available resources

- ✓ 860 wells, 4.9 GB of data scattered over a network
- ✓ Python 2.7 on Windows Vista:
 - CPU @ 3.46 GHz, 64 bit architecture
 - 16 cores, 96 GB or RAM



Oily sample: ***numpy_1.py***



Number Crunching and I/O – *numpy*

```
# We skip the first 43 rows of the text file
skip = 43

# Column 0 = Depth
# Column 8 = Rock property
# Column 13 = Water content
columns = (0, 8, 13)

# 1. Load the data using numpy.loadtxt
data = numpy.loadtxt('log.prn', skiprows=skip, usecols=columns)
```

```
# 2. Filter out the bad values for rock property
#     and water content
rock_water = data[:, 1:]

rock_water[rock_water < 0] = -999
rock_water[rock_water > 1] = -999

data[:, 1:] = rock_water

# 3. Save the filtered data to a new file
numpy.savetxt('log_out.prn', data, fmt='%-15s')
```

```
# 4. Moving average every 20ft - 6m
# a. Set negative (default) values to NaN
averaged = numpy.where(data < 0, numpy.NaN, data)

# Pre-allocate a matrix for the averaged values
out_averaged = numpy.zeros((5, averaged.shape[1]))

for col in xrange(averaged.shape[1]):
    out_averaged[:, col] = moving_average(averaged[:, col], 40)
```

- ✓ *loadtxt* is very handy and fast
- ✓ Returns a 2D *numpy* array
- ✓ Supports a wide range of file formats by tweaking its keyword arguments
- ✓ Fast and intuitive operations on N-D arrays
- ✓ *savetxt* is as handy and as fast as *loadtxt*
- ✓ A moving average implementation is a 2-liner with *numpy*

Number Crunching and I/O – *numpy*

Final results and performances

- ✓ Looped through all the files in 6.5 minutes
- ✓ Can we do better?
 - Yes we can – go parallel with the *multiprocessing* module
 - The task is easily parallelizable: one file at a time

```
import numpy
from multiprocessing import Pool, cpu_count

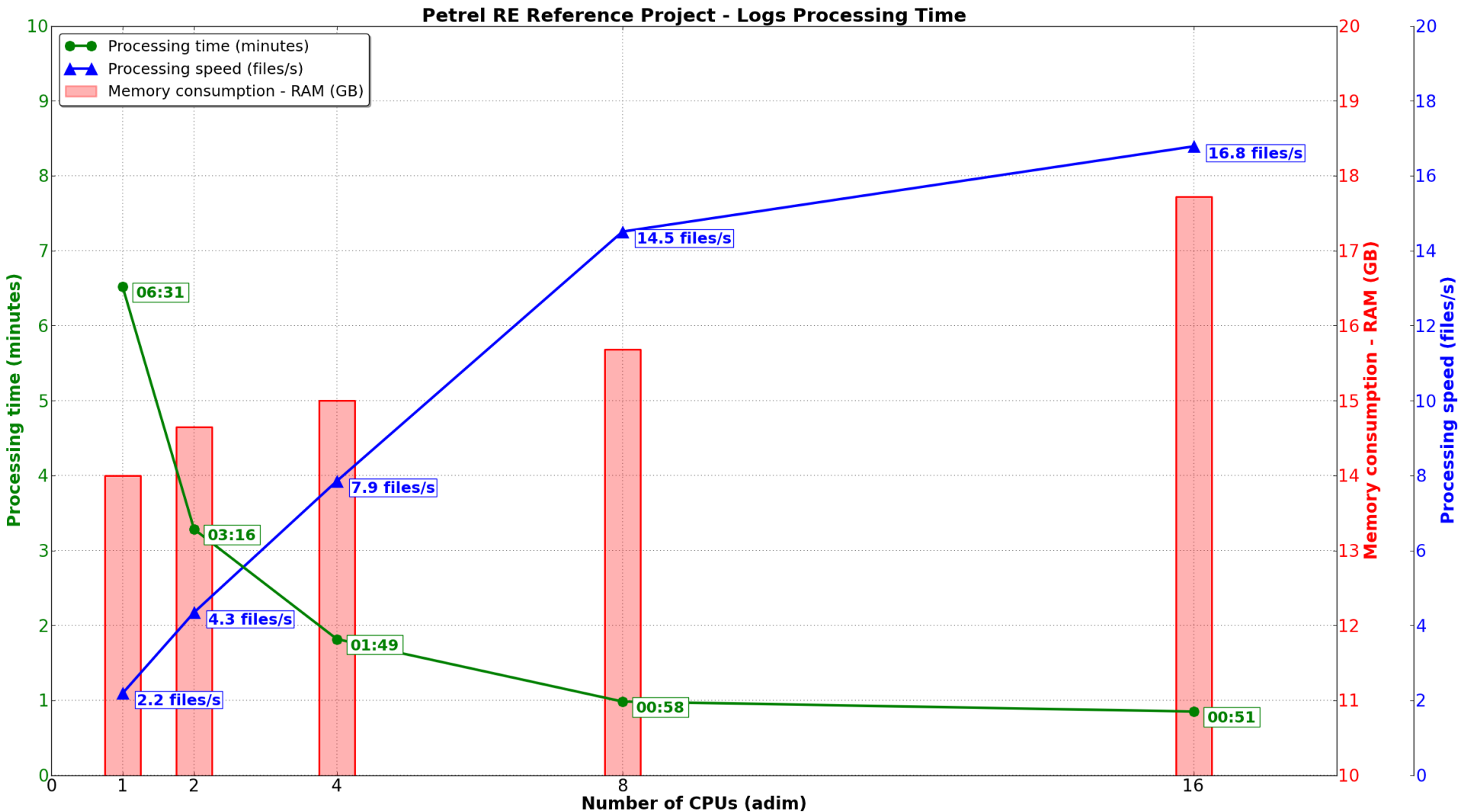
# Start a multiprocessing pool of processes
# Use all the available CPUs
pool = Pool(processes=cpu_count())

# prn_files is a list of all the text files
# Apply the function to every text file
pool.map(read_log_file, prn_files)
```

- ✓ Windows is less suited to parallel stuff than other platforms (no *os.fork()*)
- ✓ Nevertheless, this approach gives stupendous speed gains
- ✓ If I am I/O-bound... I don't care



Number Crunching and I/O – *numpy*



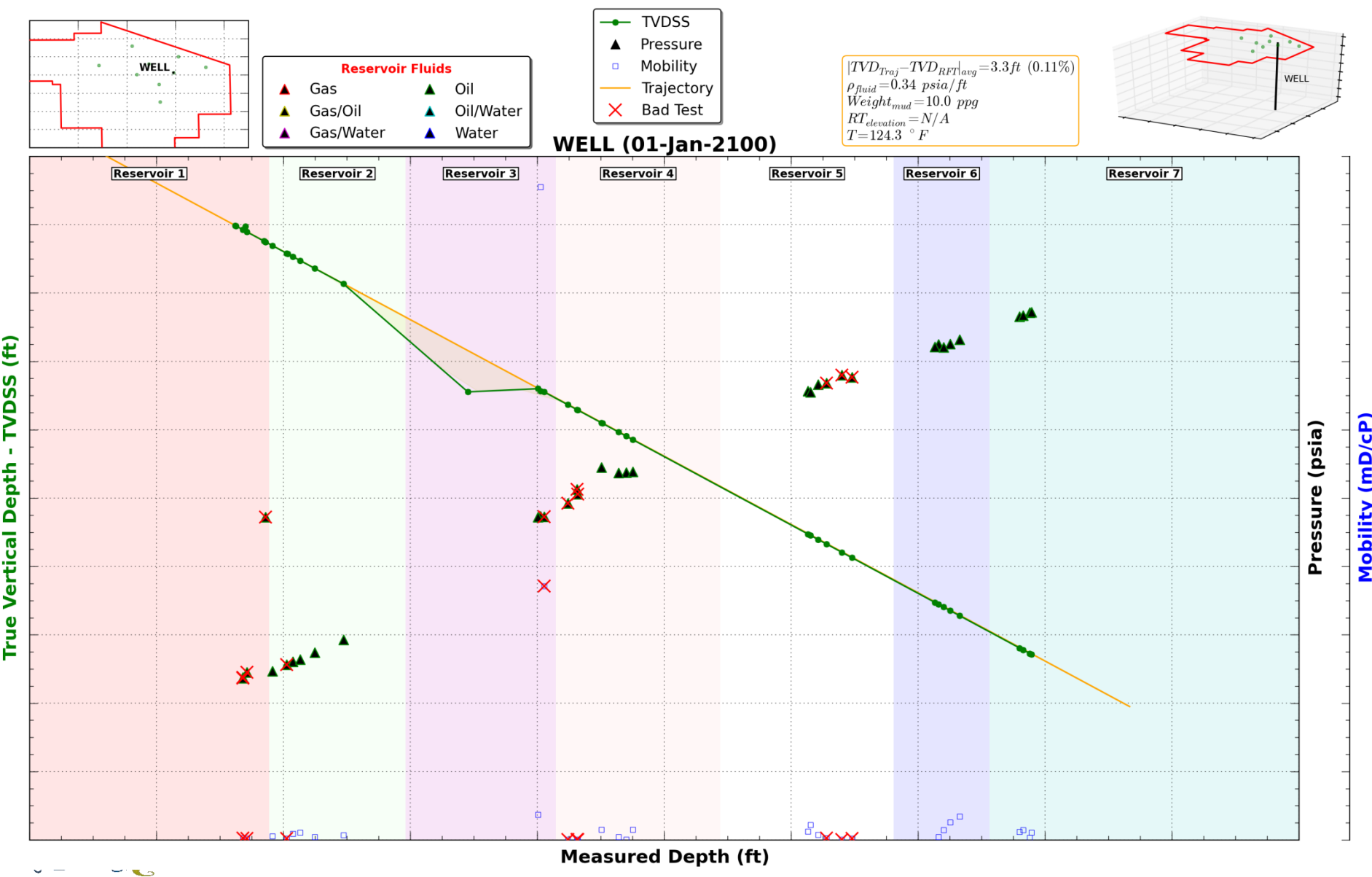
2D Visualizations

“A picture is worth a thousand words.”

- ✓ We produce visualizations for every data type in our datasets
 - Visual inspection is a powerful solution to spot errors
 - Everyone in the team has a chance to analyze the data
 - Often provide new insights on how to better integrate the data
- ✓ The generated plots contain as much information as possible
- ✓ *matplotlib* is the Python package of choice
 - Almost limitless customizations of plots
 - Very high plot quality and wide range of plot types
 - Easy integration with GUI toolkits (*wxPython, Qt, PyGtk, TkInter*)



2D Visualizations – *matplotlib*



2D Visualizations – *matplotlib*

```
from mpl_toolkits.axes_grid1 import host_subplot
import mpl_toolkits.axisartist as AA
import matplotlib.pyplot as plt

host = host_subplot(111, axes_class=AA.Axes)
plt.subplots_adjust(right=0.75)

par1 = host.twinx()
par2 = host.twinx()

new_fixed_axis = par2.get_grid_helper().new_fixed_axis
par2.axis['right'] = new_fixed_axis(loc='right',
                                   axes=par2,
                                   offset=(60, 0))

par2.axis['right'].toggle(all=True)
```

```
fig = plt.figure()
ax = fig.add_subplot(111)

colors = ['r', 'g', 'b', 'm', 'y']

for i in range(5):
    start, end = 10*i, 10*(i+1)
    ax.axvspan(start, end, color=colors[i], alpha=0.1)

    reservoir = 'Reservoir %d'%(i+1)

    ax.text(10*i+5, 8, reservoir, fontweight='bold',
           bbox=dict(fc='w', ec='k'), zorder=100,
           ha='center')

plt.show()
```

- ✓ Multiple independent Y-axis
- ✓ Axis location, ticks, colors, labels, etc... can be tweaked
- ✓ *axisartist* supports curvilinear axis as well



Oily sample: ***matplotlib_1.py***

- ✓ *axhspan* adds a horizontal span (rectangle) across the axis
- ✓ *axvspan* is its vertical friend



Oily sample: ***matplotlib_2.py***



MAERSK
OIL

2D Visualizations – *matplotlib*

WELL (Reservoir)

Event	Date	Top (ft)	Bottom (ft)
perforation	01-Jan-2100	4656.0	7034.0
perforation	01-Jan-2101	7840.0	8140.0
perforation	01-Jan-2102	8947.0	9247.0
perforation	01-Jan-2103	9625.0	17677.0
squeeze	01-Jan-2104	0.0	4656.0
squeeze	01-Jan-2105	7034.0	7840.0
squeeze	01-Jan-2106	8140.0	8947.0
squeeze	01-Jan-2107	9247.0	9625.0

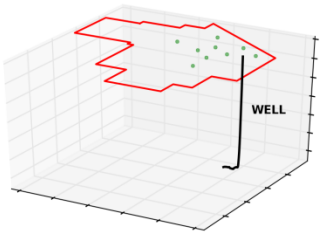
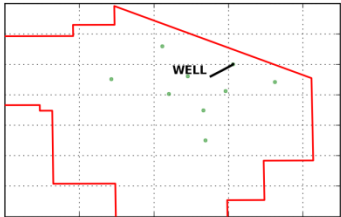
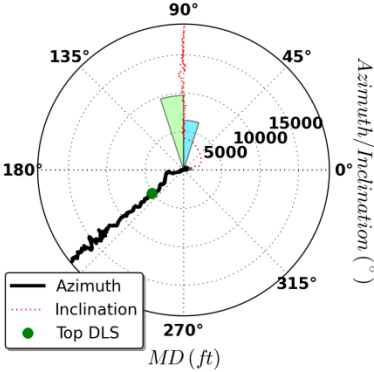
- Trajectory
- ▲ Squeeze
- ▲ Perforation
- ... Dogleg Severity
- Top 20 DLS

True Vertical Depth - TVDSS (ft)

Dogleg Severity (degrees / 100ft)

Measured Depth (ft)

MD vs. Azimuth / Inclination



2D Visualizations – *matplotlib*

```
fig = plt.figure()
ax = fig.add_subplot(111)

colLabels = ['Event', 'Date', 'Top (ft)', 'Bottom (ft)']

# No row labels
rowLabels = ['', '']

cellText = [['Perforation', '01-Jan-2020', '300', '400'],
             ['Squeeze', '01-Aug-2030', '0', '300']]

table = ax.table(cellText=cellText, rowLabels=rowLabels,
                 colLabels=colLabels, bbox=(0.1, 0.7, 0.8, 0.2))

table.auto_set_font_size(False)

plt.show()
```

```
# Make a square figure
fig = plt.figure(figsize=(6, 6))
# Add polar axes
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8], polar=True)

# Make some data up
r = numpy.arange(0, 3.0, 0.01)
theta = 2*numpy.pi*r
ax.plot(theta, r, color='#ee8d18', lw=3)
ax.set_rmax(2.0)
ax.grid(True)

plt.show()
```

- ✓ Tables are a useful addition to *matplotlib* plots
- ✓ Exact formatting, colors and font may sometimes be hard to get right



Oily sample: ***matplotlib_3.py***

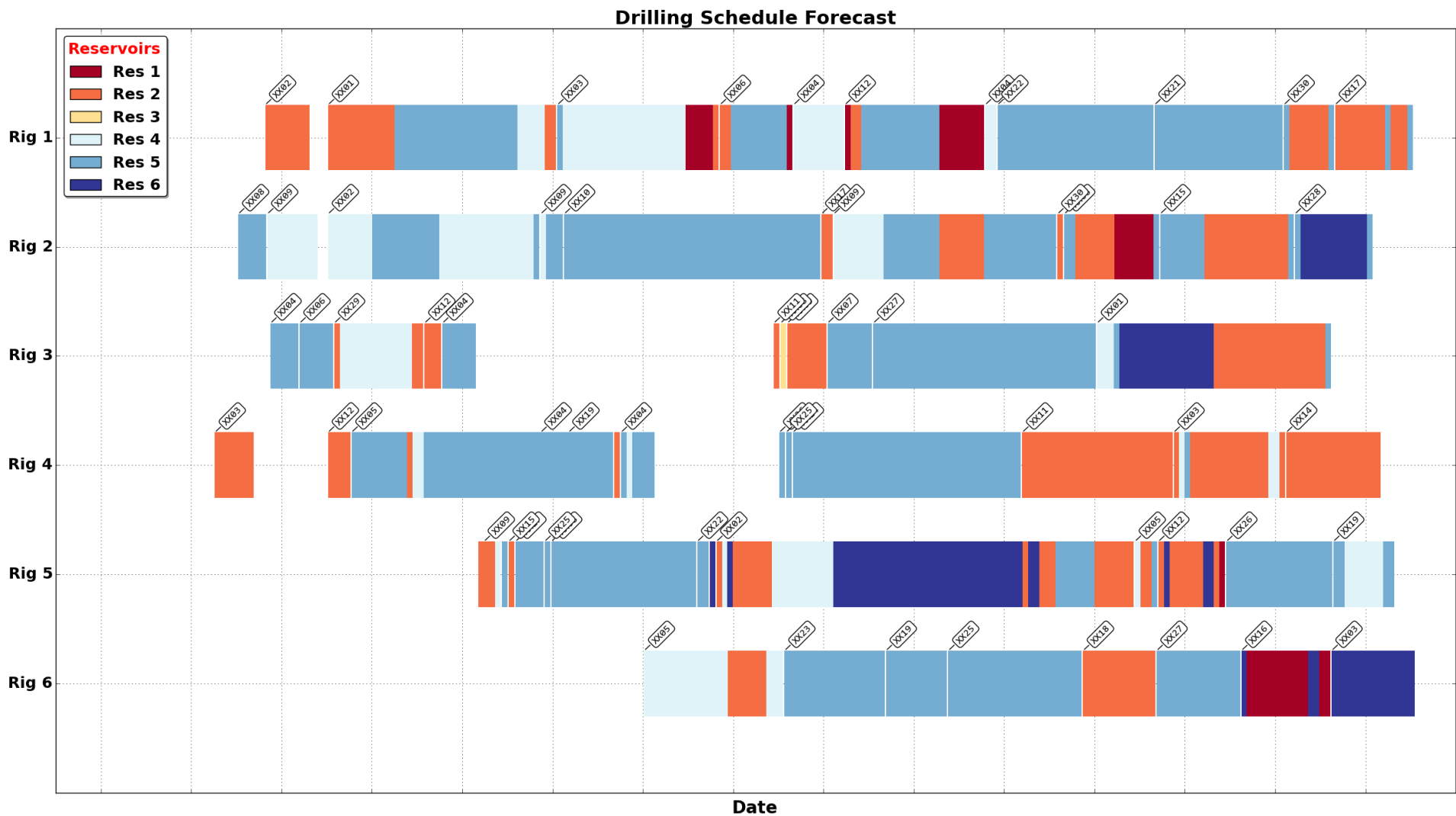
- ✓ Polar plots are not widely used in the oil industry
- ✓ They can be a great tool to analyze a well trajectory



Oily sample: ***matplotlib_4.py***



2D Visualizations – *matplotlib*



2D Visualizations – *matplotlib*

```
fig = plt.figure()
ax = fig.add_subplot(111)

ax.broken_barh([(110, 30), (150, 10)], (10, 9),
               facecolors='blue')
ax.broken_barh([(10, 50), (100, 20), (130, 10)], (20, 9),
               facecolors=('red', 'yellow', 'green'))

ax.set_ylim(5, 35)
ax.set_xlim(0, 200)
ax.set_xlabel('Drilling Time (days)')
ax.set_yticks([15, 25])
ax.set_yticklabels(['Rig 1', 'Rig 2'])
ax.grid(True)

ax.annotate('Rig stopped', (61, 25),
            xytext=(0.6, 0.9), textcoords='axes fraction',
            arrowprops=dict(facecolor='black', shrink=0.05),
            fontsize=16, ha='right', va='top')

plt.show()
```

- ✓ *broken_barh* is the perfect tool to draw drilling schedules
- ✓ Similar plots can be obtained by using multiple calls to *ax.barh()*
- ✓ Axis annotations add useful info about the data being displayed



Oily sample: *matplotlib_5.py*

I'll use this occasion to remember John Hunter, the creator of *matplotlib*
(1968-2012)



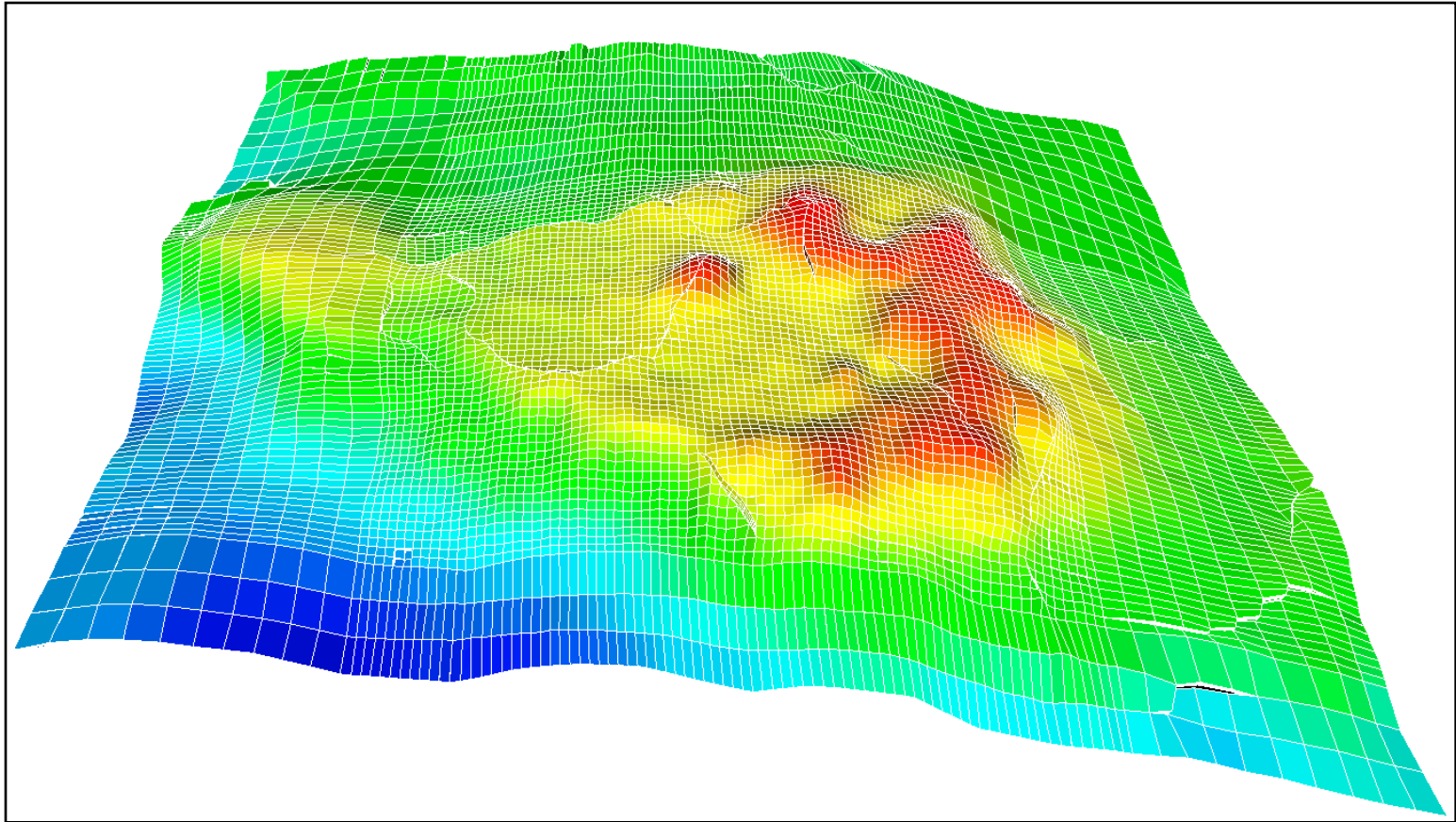
3D Visualizations

“There's something that 3D gives to the picture that takes you into another land and you stay there and it's a good place to be...”

- ✓ Most commercial software handle 3D stuff with no effort
- ✓ 3D visualization in Python is used only for specific, niche problems
 - Simulation results of well production at a specific depth
 - Double-checking input data for the simulation
 - Visualize a relationship between wells, area, reservoir and a project
- ✓ *VTK* and *mayavi* are the most widely used 3D rendering Python packages
 - Scale fairly well on big 3D datasets
 - *VTK* can easily be integrated in a GUI window (*wxPython*, *Qt*, *PyGtk*, etc...)
 - *VTK* figures can be saved as VRML files to let the colleagues play with them



3D Visualizations – VTK



- ✓ 3D reservoir model, 500,000 cells (VTK unstructured grid)
- ✓ We easily go up to 10 million cells, interaction is still smooth

3D Visualizations – VTK

- ✓ VTK unstructured grids require explicit point and cell representations
- ✓ 3D Cells can be seen as distorted hexahedrons

```
# matrix is a (8*Nx*Ny*Nz, 3) 2D numpy array
vtk_pts = array2vtkPoints(matrix)

# Create vtk data
grid = vtk.vtkUnstructuredGrid()
grid.SetPoints(vtk_pts)

# Create cells
ids = numpy.arange(0, 8*nx*ny*nz, dtype=numpy.float32)
ids = numpy.reshape(ids, (nx*ny*nz, 8))
cells = array2vtkCellArray(ids)

# Assign cells to unstructured grid
grid.SetCells(12, cells)

# Actually create the unstructured grid
ugrid = vtk.vtkExtractUnstructuredGrid()
ugrid.SetInput(grid)

ugrid = ugrid.GetOutput()
ugrid.Update()
```

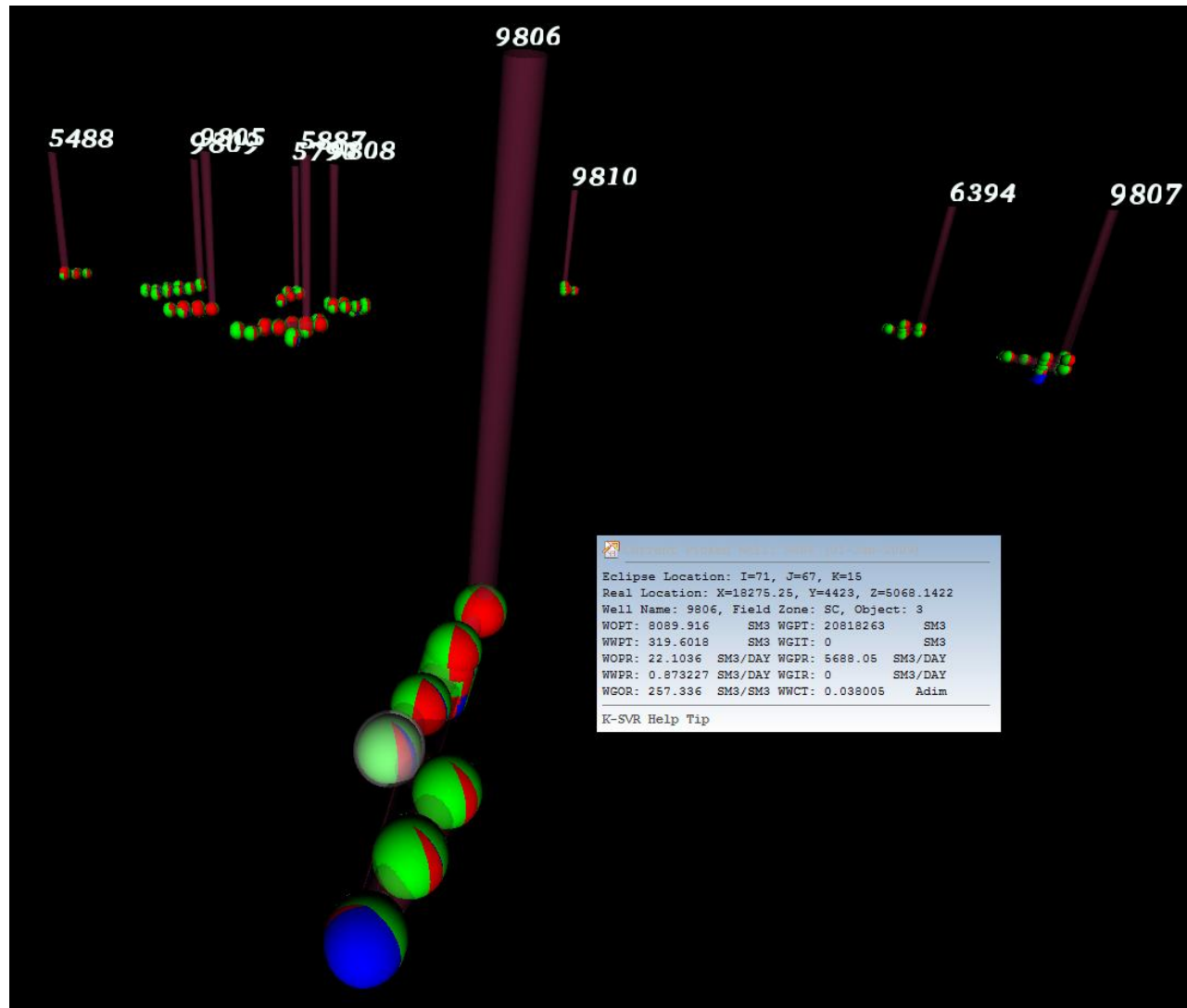
- ✓ Special techniques exist to handle very large datasets
- ✓ Coincident points can be merged (faster rendering)
- ✓ Highlighted functions are available in the *array_handler.py* module as part of the distributed samples
- ✓ These functions ease the transition between *numpy* arrays and VTK arrays



Oily sample: ***vtk_1.py***



3D Visualizations – VTK



- ✓ Spheres identify a producing interval in a well
- ✓ Colors represent the produced fluid (oil, water, gas)
- ✓ Spherical slices shows the relative abundance of each fluid
- ✓ Each sphere can be “picked”, i.e. selected with the mouse, to display more data
- ✓ Time based animation are possible

3D Visualizations – VTK

```
# x, y, z coordinates of a well trajectory
points = numpy.array(points)
line = [range(len(points))]

# Create the vtk data for the trajectory
vtk_pts = array2vtkPoints(points)
vtk_lines = array2vtkCellArray(line)

poly = vtk.vtkPolyData()
poly.SetPoints(vtk_pts)
poly.SetLines(vtk_lines)

# A filter that generates tubes around lines
profileTubes = vtk.vtkTubeFilter()
# Set the tube radius and resolution
profileTubes.SetRadius(radius)
profileTubes.SetNumberOfSides(20)
profileTubes.SetInput(poly)

# Map vtkPolyData to graphics primitives
wellMapper = vtk.vtkPolyDataMapper()
wellMapper.SetInput(profileTubes.GetOutput())

# Create an "actor" for the well
wellActor = vtk.vtkActor()
wellActor.SetMapper(wellMapper)

# Create a caption "actor" for the well name
textActor = vtk.vtkCaptionActor2D()
textActor.SetCaption(wellName)
```

- ✓ *vtkPolyData* can represent vertices, lines, polygons etc...
- ✓ *vtkTubeFilter* is a very good way to represent wells in a 3D space
- ✓ The well name caption “actor” follows the user view while she interacts with the VTK window
- ✓ Highlighted functions are available in the *array_handler.py* module as part of the distributed samples



Oily sample: ***vtk_2.py***



MAERSK
OIL

3D Visualizations – *NetworkX and mayavi*

- ✓ Visualize relationships between wells, areas, reservoirs and projects
- ✓ Shows dependencies between wells and undeveloped areas
- ✓ 3D version of a *GraphViz* inheritance diagram
- ✓ Particularly useful when a project contains 1000s of wells



Oily sample: *mayavi_1.py*



MAERSK
OIL

Integration with the Simulator

“Fast as a rabbit, dumb as a stone.”

- ✓ The reservoir simulator can easily generate 100 GB of results per simulation
- ✓ Each result set is made of 5-8 interesting files
 - Results are stored in heavily compressed, unformatted binary files
 - These files are generated by a Fortran-based simulator
 - File structure is relatively simple and straightforward
- ✓ We can use Python to extract the simulation results from these files
 - Performances are generally poor (code is slow)
 - Does not scale well when files are big
- ✓ Can we write a small Fortran routine and interface it with Python to read these large, binary files?
 - Enter *f2py*



Integration with the Simulator – *f2py*

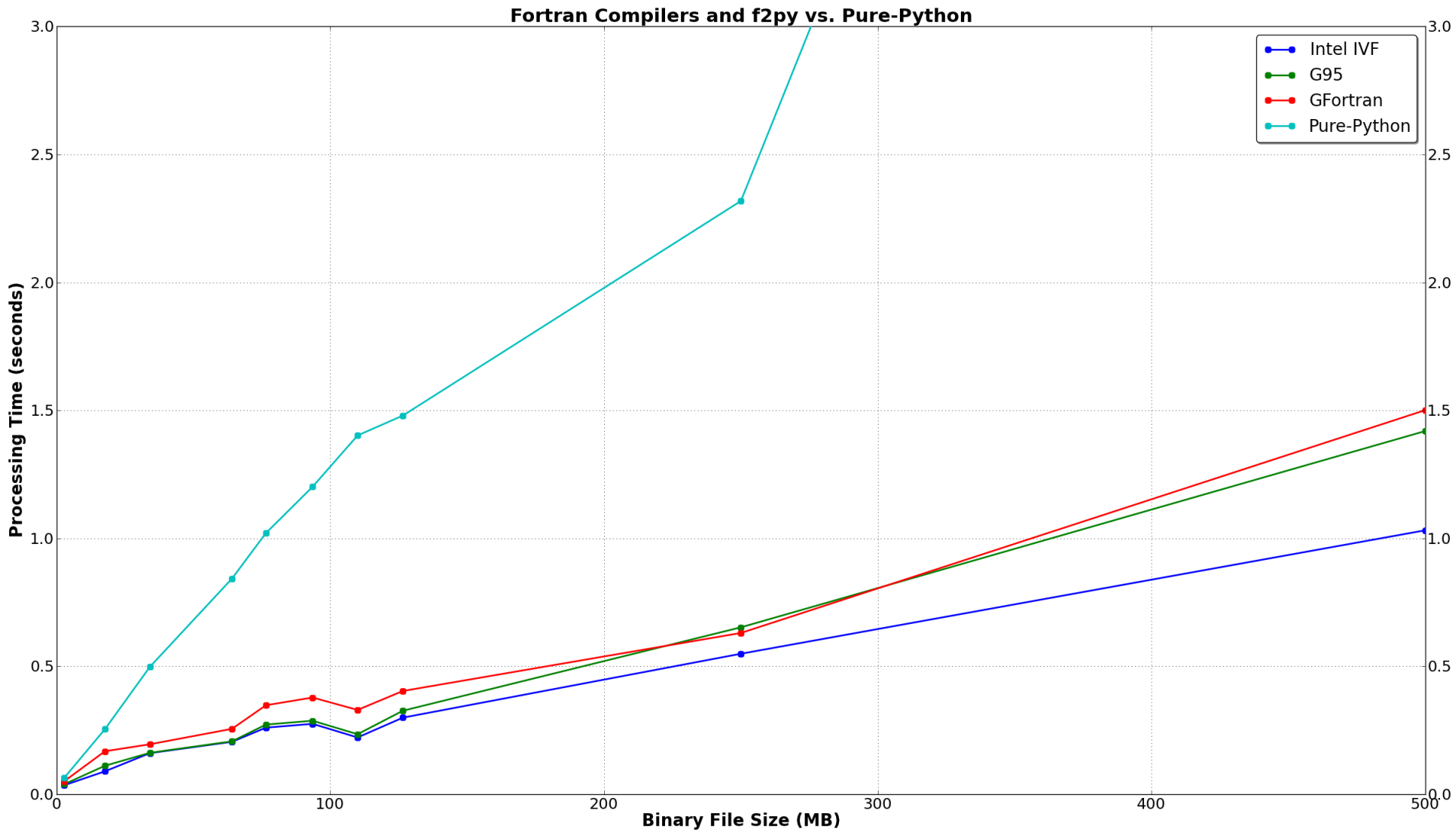
- ✓ Fortran to Python interface generator
- ✓ Connects the two languages:
 - Creates Python C/API modules from Fortran 77/90/95
 - Works directly on Fortran sources
 - Automatically handles the difference in the data storage order of multi-dimensional Fortran and *numpy* arrays
- ✓ Requires a Fortran compiler installed – supports many major compilers, such as gfortran, Intel IVF, Absoft, NAG, etc...

```
f2py -c fortran_file.f90 -m py_module
```

- ✓ Now every Fortran subroutine/function in *fortran_file.f90* is accessible in Python by importing *py_module*



Integration with the Simulator – *f2py*



Automation and N-D Interpolation

“Besides black art, there is only automation and mechanization.”

Task of the day

- ✓ We have 16,000 new simulations available (sensitivities)
 - Each of them represents a unique combination of 13 parameters (oil gravity, rock properties, distance between wells etc...)
 - Simulation results could give insights on the numerical model sensitivity to the parameters variations
- ✓ The 13 parameters form a discrete set of known data points
- ✓ Use a *f2py*-generated module to read results from all the simulations
- ✓ Use interpolation to estimate results at intermediate values of the parameters
 - *scipy* offers multi-dimensional interpolation/extrapolation capabilities
 - *scipy.interpolate.rbf*: uses Radial Basis Function interpolation of N-dimensional scattered data

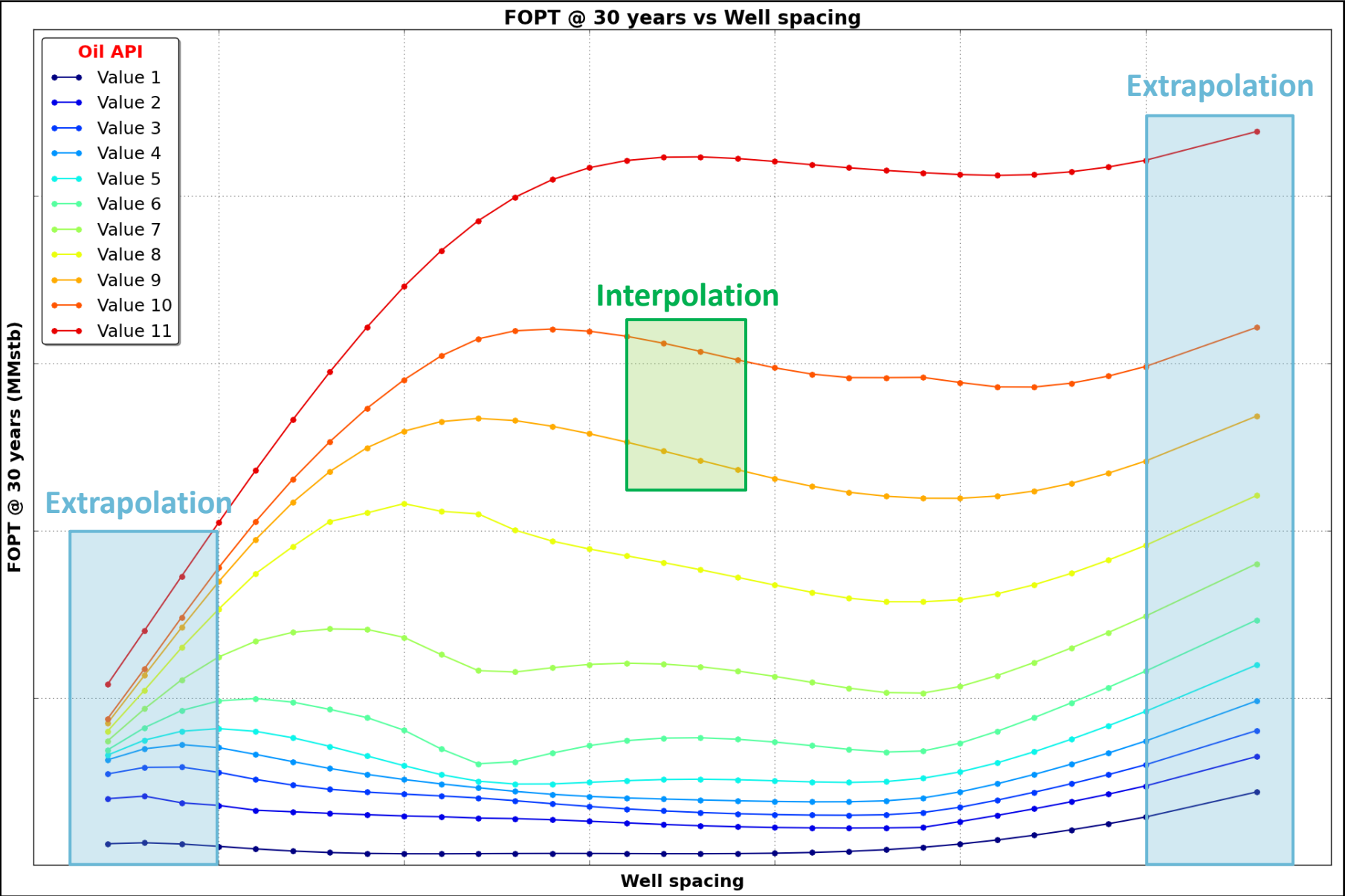


Oily sample: ***scipy_1.py***



MAERSK
OIL

Automation and N-D Interpolation – *scipy*



Graphical User Interfaces

“A picture is worth a thousand words. An interface is worth a thousand pictures.”

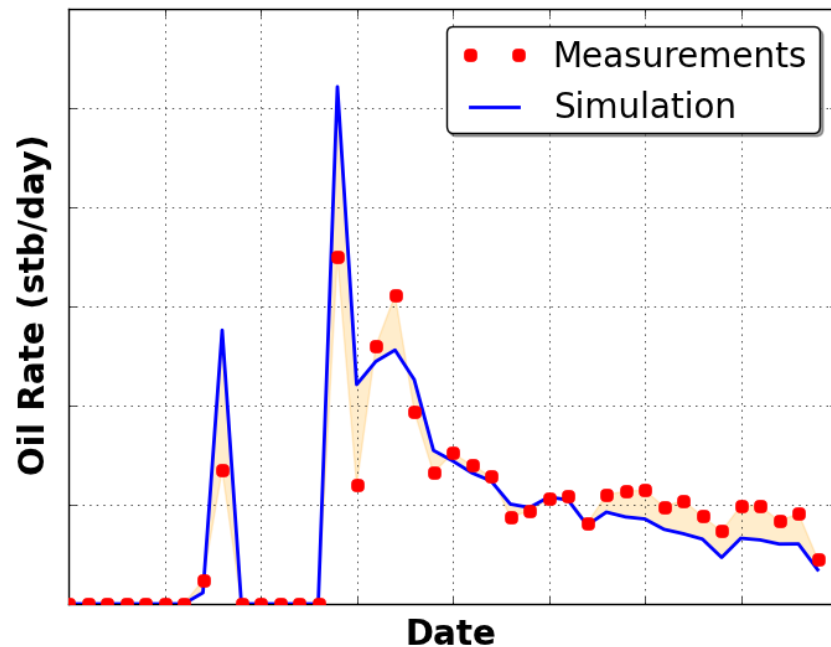
- ✓ User interfaces are an obvious choice when it comes to sharing your findings with non-Pythonistas colleagues
- ✓ Although many high quality GUI frameworks are available...
- ✓ *wxPython* is ***the*** tool I use
 - Almost effortlessly integrate with *matplotlib* and *VTK* (2D and 3D)
 - Easy to build practical, responsive and sexy user interfaces
 - GUIs look (and are) native, whatever the platform
 - Number of widgets available far surpass all other toolkits
- ✓ Distribution to colleagues is done via *py2exe* / *PyInstaller* and *InnoSetup* to generate a standard Windows installer



Graphical User Interfaces

Task of the week/month

- ✓ Create a GUI that evaluates the quality of a calibrated reservoir model



- ✓ Calibration is good when simulation results are close to measurements (shaded area)
- ✓ Errors in the calibration are measured by different formulas such as:

$$Error = \frac{1}{N} \sqrt{\sum_{i=1}^N \omega_i \left(\frac{s_i - o_i}{o_i} \right)^2}$$

- ✓ The GUI should allow the user to explore the numerical calculations and to quickly plot the simulation results against the measurements



Graphical User Interfaces

Complications

- ✓ Number of data points: 17 years of historical measurements
- ✓ Number of wells and simulation time steps (thousands)
- ✓ The user would like to be able to:
 - Filter out values outside a user-defined date window (per well)
 - Apply a custom multiplier to some of the measurements
 - Exclude some values if a well has been closed for more than X days in a month
 - Modify the error function if a well has been using some gas to ease production
 - Many, many other customizations...
- ✓ The GUI puts together the power of *numpy*, *f2py*, *matplotlib*, *scipy*, *multiprocessing* and *wxPython* to deliver all that and much more 😊



Graphical User Interfaces

HM_Evaluation v1.1.8

FileOptionsRun ConfigurationInterfaceHelp

Well Info

Available Wells

Well 1

Well 2

Well 3

Well 4

Well 5

Well 6

Well 7

Well 8

Well 9

Well 10

Well 11

Well 12

Well 13

Well 14

Well 15

Well 16

Well 17

Well 18

Well 19

Well 20

Well 21

Well 22

Well 23

Well 24

Well 25

Well 26

Well 27

Well 28

Well 29

Input Data

ECLIPSE Summary File (Production Output):

MY_SUMMARY.SMSPEC

Back-Calculated BHPs File:

THP_BHP_Values.xls

Output Folder for Images and Documents:

c:\MyProjects\HM_Evaluation

Simulation

HM Quality

Well 7 Profiles

Well 7 Raw Data

Uptime Filter (%):

80

Cumulative Filter (STB/MSCF):

0

Custom End Date:

7/ 1/2011

Match Deviations

Wells	WOPT	WGPT	WWPT	WBHP All Data	WOPR	WGPR	WWCT	Grade
Well 1								
Well 2	100%	100%	100%	1095	100%	100%	55%	6.95
Well 3								
Well 4								
Well 5								
Well 6								
Well 7	8%	8%		51	11%	11%	5%	0.87
Well 8	0%	14%	22%	84	10%		4%	0.98
Well 9	11%			57	4%		2%	1.19
Well 10	3%	4%	21%	114	8%		4%	0.85
Well 11	6%	0%	10%	98	12%		4%	0.96
Well 12	16%	16%		510	23%	19%	23%	2.27
Well 13	18%	29%	28%	104	11%		6%	1.37
Well 14	5%	14%	46%	165	17%	14%	17%	1.37
Well 15	8%	10%	32%	84	12%		5%	1.19
Well 16	0%	25%	48%	128	8%	11%	5%	1.24
Well 17	5%	10%	7%	128	10%	33%	5%	1.04
Well 18	10%	8%	14%	129	8%	4%	2%	0.98
Well 19								
Well 20								
Well 21								
Well 22	9%	15%	28%	61	7%	17%	4%	1.09
Well 23	10%	28%	62%		14%	30%	12%	2.00
Well 24	4%	4%		52	6%		7%	0.63
Well 25	7%	15%	41%	102	22%		5%	1.52
Well 26	25%	10%			19%		3%	1.63
Well 27	31%	28%		37	20%	31%	12%	1.35
Well 28	33%	13%		227	35%	15%	6%	2.18
Well 29				52				0.62
Avg Dev	15%	18%	35%	173	18%	26%	9%	1.58
Grade	Poor	Fair	Poor	Poor	Fair	Good	Fair	

(*) Values come from the "Back-Calculated BHPs File". Not included in the grading.

Match Quality

Wells	WOPT	WGPT	WWPT	WBHP All Data	WOPR	WGPR	WWCT	Overall
Well 1								
Well 2	Poor	Poor	Poor	Poor	Poor	Poor	Poor	Poor
Well 3								
Well 4								
Well 5								
Well 6								
Well 7	Fair	Good		Good	Fair	Good	Fair	Good
Well 8	Good	Good	Poor	Good	Fair		Good	Good
Well 9	Poor			Good	Good		Good	Fair
Well 10	Good	Good	Poor	Fair	Good		Good	Good
Well 11	Fair	Good	Good	Fair	Fair		Good	Good
Well 12	Poor	Fair		Poor	Poor	Good	Poor	Poor
Well 13	Poor	Poor	Poor	Fair	Fair		Fair	Fair
Well 14	Good	Good	Poor	Fair	Fair	Good	Poor	Fair
Well 15	Fair	Good	Poor	Good	Fair		Good	Fair
Well 16	Good	Poor	Poor	Fair	Good	Good	Fair	Fair
Well 17	Good	Good	Good	Fair	Fair	Fair	Fair	Fair
Well 18	Fair	Good	Fair	Fair	Good	Good	Good	Good
Well 19								
Well 20								
Well 21								
Well 22	Fair	Good	Poor	Good	Good	Good	Good	Fair
Well 23	Fair	Poor	Poor		Fair	Good	Poor	Fair
Well 24	Good	Good		Good	Good		Fair	Good
Well 25	Fair	Good	Poor	Fair	Poor		Fair	Fair
Well 26	Poor	Good			Fair		Good	Fair
Well 27	Poor	Poor		Good	Poor	Fair	Poor	Fair
Well 28	Poor	Good		Poor	Poor	Good	Fair	Poor
Well 29				Good				Good
Good	5%	15%	10%	85	10%	30%	5%	
Fair	10%	20%	20%	170	20%	40%	10%	
Good	30%	68%	15%	42%	30%	73%	40%	33%
Fair	35%	5%	8%	42%	45%	18%	35%	52%

Messages & Actions

Time

HM_Evaluation Messages

11:36:51

Production results processed correctly

Dry Run

Start

Kill

Computer Name: WCPH00216, User Logon: aga108

Thursday, 11 October 2012 @ 11:37:48

RAM: 0 Mb

CPU: 8%

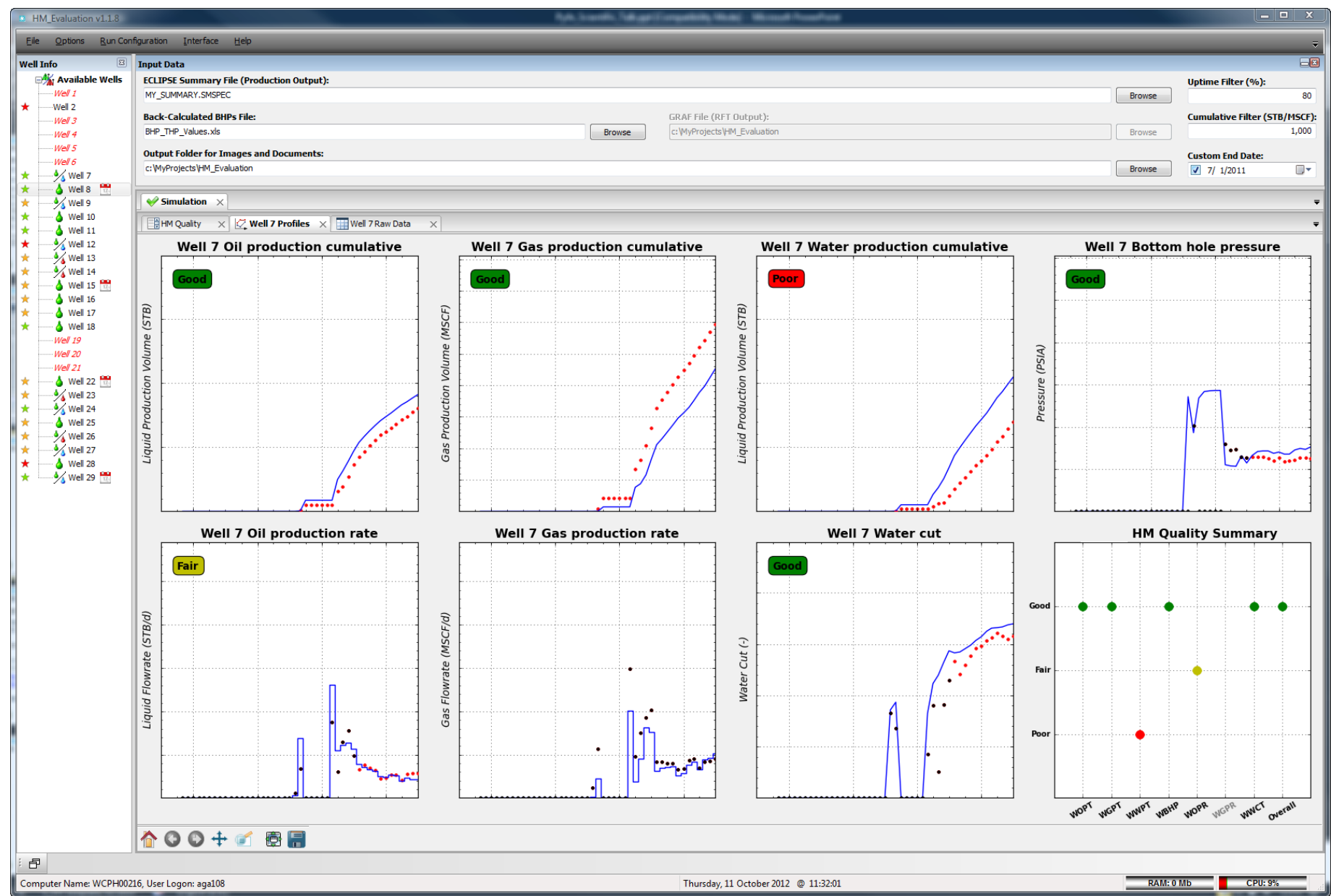
Graphical User Interfaces

Final outcome

- ✓ We have a fast, practical and nice GUI to examine the quality of model calibration
- ✓ Colleagues can independently run the GUI and examine the results
- ✓ Multiple simulations can be analyzed and compared
- ✓ The interface automagically exports *matplotlib* figures for all the wells and Excel reports (and it does it on multiple processors...)
 - Findings and insights can easily be shared outside the team
 - Consistent, fixed (and beautiful) format for pictures in reports and documents
- ✓ We have the source code 😊 – any modification is embarrassingly fast



Graphical User Interfaces



Graphical User Interfaces

Task of the week/month

- ✓ The reservoir simulator we use is called ECLIPSE
 - It's keyword-based – you enter inputs in a text file with keywords and sub-keywords
 - 1983: first release of ECLIPSE (**ECL**'s **I**mplicit **P**rogram for **S**imulation **E**ngineering)
 - ECLIPSE currently handles $\approx 1,600$ keywords
 - On average, each keyword has 3 switches/sub-keywords ($\approx 4,200$ in total)
 - No editor with syntax highlighting, error checking capabilities and integrated help system exists for the input files (after 30 years!!)
- ✓ How about a *wxPython*-based editor with all these capabilities?
 - The *wx.StyledTextCtrl* (Scintilla-based) already provides excellent syntax highlighting for various programming languages
 - *wxPython* 2.9 contains powerful HTML viewing capabilities (via *wx.html2* module)
 - The ECLIPSE input files syntax is very similar to the programming language *Lua*



Graphical User Interfaces

Another GUI: *DeckEd*

- ✓ *DeckEd* is a text editor based on `wx.StyledTextCtrl`
- ✓ Syntax highlighting for the reservoir simulator ECLIPSE and more than 60 other programming languages (Python, C++, Java, HTML, PHP, Ruby, etc...)
- ✓ Integrated help for the reservoir simulator keywords and sub-keywords
- ✓ Runtime monitoring of simulation status and progress
- ✓ Runtime error checking for ECLIPSE input files keywords
- ✓ Plugin-based architecture – you can add a Python debugger, a spell checker, a code browser, etc...



Graphical User Interfaces

Keyword Tree

Open Files List

Real-Time Error Checking

Integrated Help

The screenshot displays the Eclipse graphical user interface for reservoir simulation. The interface is divided into several panes:

- Keywords Pane (Left):** A tree view showing the structure of the input file. It includes sections like NOECHO, MESSAGES, RUNSPEC, and various well and field definitions. A blue arrow points from the 'Keyword Tree' label to this pane.
- Open Files List (Right):** A list of files currently open in the application, including 'KB_SOUTH_WF.DATA', 'BP_CASES_11042011.D...', 'BRINETRACER.DATA', and 'IX_P1_ECL2DX.Dxf'. A blue arrow points from the 'Open Files List' label to this pane.
- Real-Time Error Checking (Center):** The main text area displaying the input file's content. It shows keywords like 'NOECHO', 'MESSAGES', 'RUNSPEC', 'DIMENS', 'START', 'WELLDIM', and 'FIELD'. A blue arrow points from the 'Real-Time Error Checking' label to this pane.
- Integrated Help (Bottom):** A pane showing help documentation for the 'WELLDIM' keyword. It includes a table of well dimension data and a description of the data format. A blue arrow points from the 'Integrated Help' label to this pane.

The main text area contains the following input file content:

```
1 NOECHO
2
3 MESSAGES
4 -- PRINT LIMIT ----->| STOP LIMIT ----->
5 -- mess comm warn prob erro bug mess comm warn prob
6 2*      100 10000 4*      1000000 100000 /
7
8 -----
9
10 RUNSPEC
11
12
13 --NOSIM
14
15 -- model name
16 TITLE
17 KB_SOUTH
18
19 -- model dimensions
20 DIMENS
21 -- iii jjj kkk
22 480 444 14 /
23
24 HELLO
25
26 -- three phase with dissolved gas
27 OIL
28 WATER
29 GAS
30 DISGAS
31
32 PERMX
33
34 -- unit conversion
35 FIELD
36
37 -- simulation start date
38 START
39 1 JAN 2012 /
40
41 -- well dimensions
42 WELLDIMS
43 -- mxwel mxcon mxgrp mxwpg
44 400 300 25 300 /
45
46 -----
47 -- other runspec options
```

The bottom pane shows the 'WELLDIM' help section, which includes a table of well dimension data and a description of the data format.

Well Dimension	Value
ECLIPSE 100	
ECLIPSE 300	

The data consists of up to 10 items, describing the dimensions of the well data to be used in the run. The data must be terminated by a slash (/).

Graphical User Interfaces

Alphabetical
Keyword List

Real-Time
Keyword Help

Directory Tree

Keyword Usage
Examples

Help & Examples

Full Help Examples (2) Sub-Items (3)

Example 1

```
AQANTRC
1 WT1 1.0 /
1 WT2 0.0 /
2 WT1 0.0 /
2 WT2 1.0 /
/
```

Example 2

```
AQANTRC
1 TR6 0.6 /
/
```

BRINETRACER.DATA - file://C:/MyProjects/D

File Edit View Format Settings Tools

Keywords

Keywords Entry

Line Entry

511

512

513 RPTPROPS

514 -- PROPS Reporting Options

515

516 'PVTO' 'PVDO' 'PVTW'

517 /

518

519 SOLUTION

520

521

522 EQUIL

523

524 SALTVD

525 5000.0 9.0

526 5500.0 9.0 /

527

528 AQUFETP

529 1 5400 1* 2.0E9 3.0E-5 540.96 1 10.0 /

530 /

531

532 AQUCT

533 2 5400.0 1* 20.0 0.10 3.0E-5 2400.0 140.0 4.8 1 1 9.5 /

534 /

535

536

537

538

539

540

541 AQANCON

542 1 15 1

543 2 15 1

544 /

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1445

1446

1447

1448

1449

1450

1451

1452

1453

1454

1455

1456

1457

1458

1459

1460

1461

1462

1463

1464

1465

1466

1467

1468

1469

1470

1471

1472

1473

1474

1475

1476

1477

1478

1479

1480

1481

1482

1483

1484

1485

1486

1487

1488

1489

1490

1491

1492

1493

1494

1495

1496

1497

1498

1499

1500

1501

1502

1503

1504

1505

1506

1507

1508

1509

1510

1511

1512

1513

1514

1515

1516

1517

1518

1519

1520

1521

1522

1523

1524

1525

1526

1527

1528

1529

1530

1531

1532

1533

1534

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545

1546

1547

1548

1549

1550

1551

1552

1553

1554

1555

1556

1557

1558

1559

1560

1561

1562

1563

1564

1565

1566

1567

1568

1569

1570

1571

1572

1573

1574

1575

1576

1577

1578

1579

1580

1581

1582

1583

1584

1585

1586

1587

1588

1589

1590

1591

1592

1593

1594

1595

1596

1597

1598

1599

1600

1601

1602

1603

1604

1605

1606

1607

1608

1609

1610

1611

1612

1613

1614

1615

1616

1617

1618

1619

1620

1621

1622

1623

1624

1625

1626

1627

1628

1629

1630

1631

1632

1633

1634

1635

1636

1637

1638

1639

1640

1641

1642

1643

1644

1645

1646

1647

1648

1649

1650

1651

1652

1653

1654

1655

1656

1657

1658

1659

1660

1661

1662

1663

1664

1665

1666

1667

1668

1669

1670

1671

1672

1673

1674

1675

1676

1677

1678

1679

1680

1681

1682

1683

1684

1685

1686

1687

1688

1689

1690

1691

1692

1693

1694

1695

1696

1697

1698

1699

1700

1701

1702

1703

1704

1705

1706

1707

1708

1709

1710

1711

1712

1713

1714

1715

1716

1717

1718

1719

1720

1721

1722

1723

1724

1725

1726

1727

1728

1729

1730

1731

1732

1733

1734

1735

1736

1737

1738

1739

1740

1741

1742

1743

1744

1745

1746

1747

1748

1749

1750

1751

1752

1753

1754

1755

1756

1757

1758

1759

1760

1761

1762

1763

1764

1765

1766

1767

1768

1769

1770

1771

1772

1773

1774

1775

1776

1777

1778

1779

1780

1781

1782

1783

1784

1785

1786

1787

1788

1789

1790

1791

1792

1793

1794

1795

1796

1797

1798

1799

1800

1801

1802

1803

1804

1805

1806

1807

1808

1809

1810

1811

1812

1813

1814

1815

1816

1817

1818

1819

1820

1821

1822

1823

1824

1825

1826

1827

1828

1829

1830

1831

1832

1833

1834

1835

1836

1837

1838

1839

1840

1841

1842

1843

1844

1845

1846

1847

1848

1849

1850

1851

1852

1853

1854

1855

1856

1857

1858

1859

1860

1861

1862

1863

1864

1865

1866

1867

1868

1869

1870

1871

1872

1873

1874

1875

18

Conclusions

- ✓ Many, many more examples of the usage of Python in the oil industry that I couldn't show
- ✓ Python is becoming increasingly popular amongst reservoir engineers
 - Automation improves working effectiveness a hundredfold
 - Beauty and elegance of the language – easy to grasp even for newcomers
- ✓ Third-party packages add great value to the standard library:
 - *matplotlib* – plot customization and unbeatable figure quality
 - *numpy* and *scipy* – fast numerical manipulation of multi-dimensional arrays
 - *f2py* – when you need Fortran raw speed with Python elegance
 - *VTK* and *mayavi* – scalable 3D visualization
 - *wxPython* – the glue to keep all the above together in a nice, point-and-click GUI
- ✓ Presentation samples: <http://www.infinity77.net/pycon/oily.zip>



Thank You

Questions?



Comments?

