OMNeT++ Community Summit 2017, University of Bremen, Sept 7-8.

OMNeT++ Best Practices Reloaded

András Varga

(Result Analysis)

A Little History

Recent improvements in OMNeT++ (versions 5.0, 5.1):

- Run filtering
- Handling of weighted statistics
- Data export revised
- Scavetool revised

Last year's OMNeT++ Summit:

- Presentation: SQLite result file format
- Panel discussion: Python, Pandas and Jupyter recommended

Recommendation

- Browsing, casual plotting:
  Analysis Tool in the OMNeT++ IDE

- Serious analysis:
  Python (with the right packages)

Result File Analysis using Python

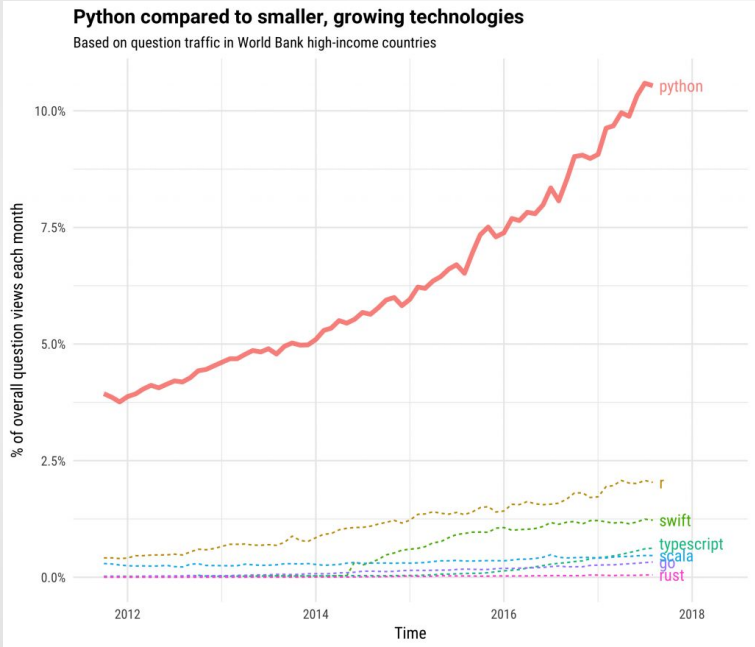
Python

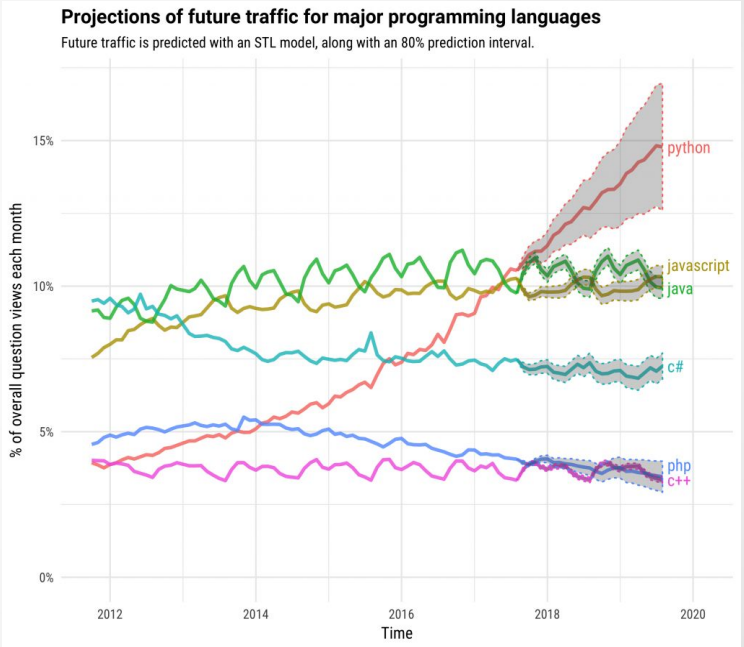
Python is a very nice programming language for {….}*

* Big Data, Machine Learning / AI, Statistics, GUIs, Sysadmin tools, Integration, etc.

Artificial Intelligence
Cryptography
Database
Foreign Function Interface
Game Development
GIS (Geographic Information System)
GUI
Audio / Music
ID3 Handling
Image Manipulation
Indexing and Searching
Machine Learning
Natural Language Processing
Networking
Neural Networks
Platform-Specific
Plotting
Presentation
RDF Processing
Scientific
Standard Library Enhancements
Threading
System administration
Web Development
Workflow
XML Processing
Flow Based Programming
...

Python Popularity



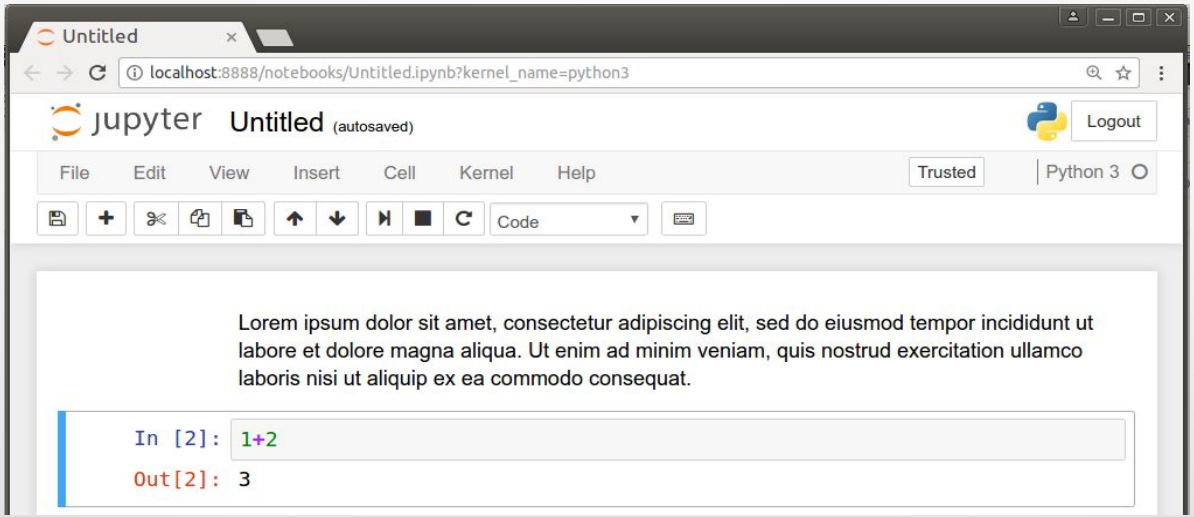


Page view statistics on stackoverflow.com

Jupyter Notebook

Jupyter is a web application that allows you to create documents that contain live code, equations, visualizations and explanatory text. It can also be used just as a comfortable Python shell.

Notebook “cells” can be text cells (markdown), or executable “code” cells. Normally used with local server process. (Start server with “jupyter notebook”, then open displayed URL in browser.)



Python / Pandas

Data analysis library

Central concept: **data frame**

- Extremely powerful, multi-dimensional, indexed, high-performance table
- Cells may contain arbitrary objects (e.g. arrays or matrices), not only numbers and strings
- Built on top of NumPy, a package for numerical computing; includes **ndarray**, a powerful N-dimensional array object)

Some key features:

- Querying, modifying, filtering, joining/merging, pivot table creation, statistics
- Import/export in CSV, JSON, Excel, HDF5 and other formats
- Integration with Matplotlib (plotting library)

Pandas Minimal Example

```
import pandas as pd

df = pd.DataFrame( {
        'AAA' : ['foo','bar','foo', 'bar'],
        'BBB' : [10,20,30,40],
        'CCC' : [100,50,-30,-50]
} )
```

	AAA	BBB	CCC
0	foo	10	100
1	bar	20	50
2	foo	30	-30
3	bar	40	-50

df.describe() → *statistical summaries of numerical columns*

df['BBB'].mean() → 25.0

df['BBB'] + df['CCC'] → 110, 70, 0, -10

print(df.to_csv()) → *prints table in CSV format*

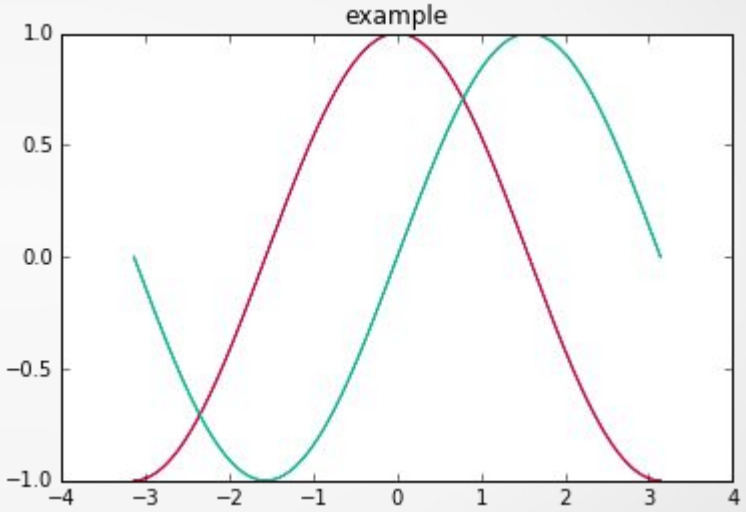
Python / Matplotlib

Scientific plotting package for Python

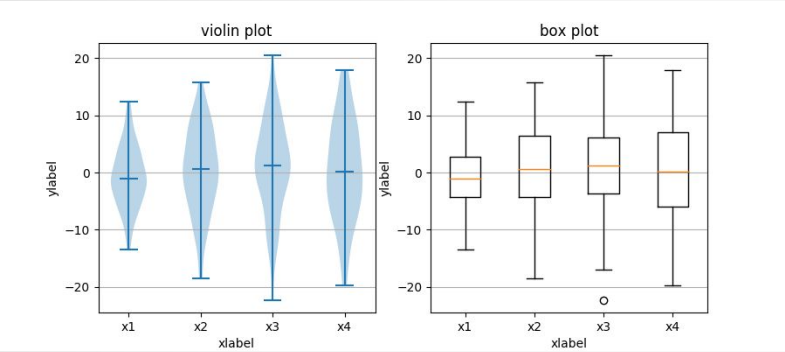
- Modeled after Matlab's plotting package
- Interactive and "batch" (image export) mode
- Line plots, line plots with error bars, scatter plots, bar charts, pie charts, box plots, violin plots, polar charts, contour plots, field plots, ...
- Subplots, styling, markers, annotations, …
- Two APIs: an object-oriented one, and a stateful one built on top of it, optimized for interactive use
- Backends (Qt, GTK, Agg, etc.)

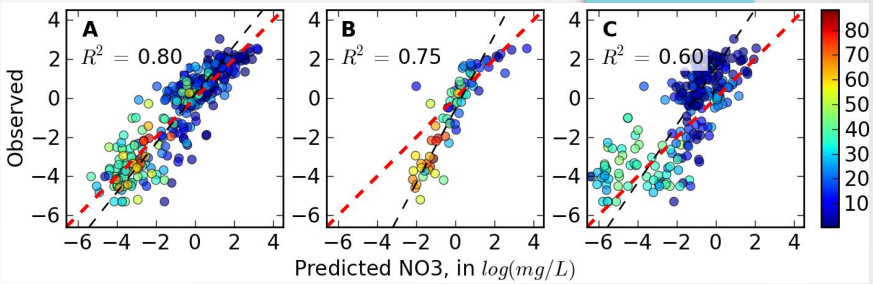
Matplotlib Basic Example

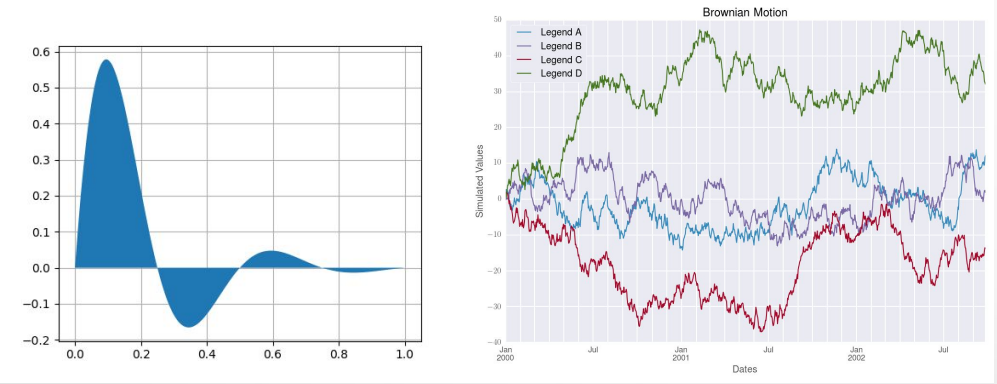
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-np.pi, np.pi, 300)
cosx, sinx = np.cos(x), np.sin(x)

plt.plot(x, cosx)
plt.plot(x, sinx)
plt.title("example")

plt.show()
```

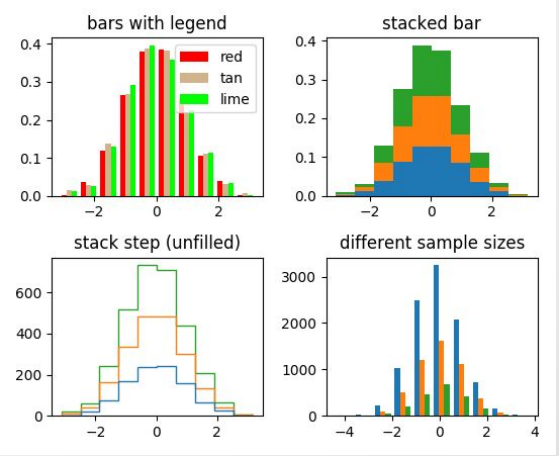


Matplotlib Gallery





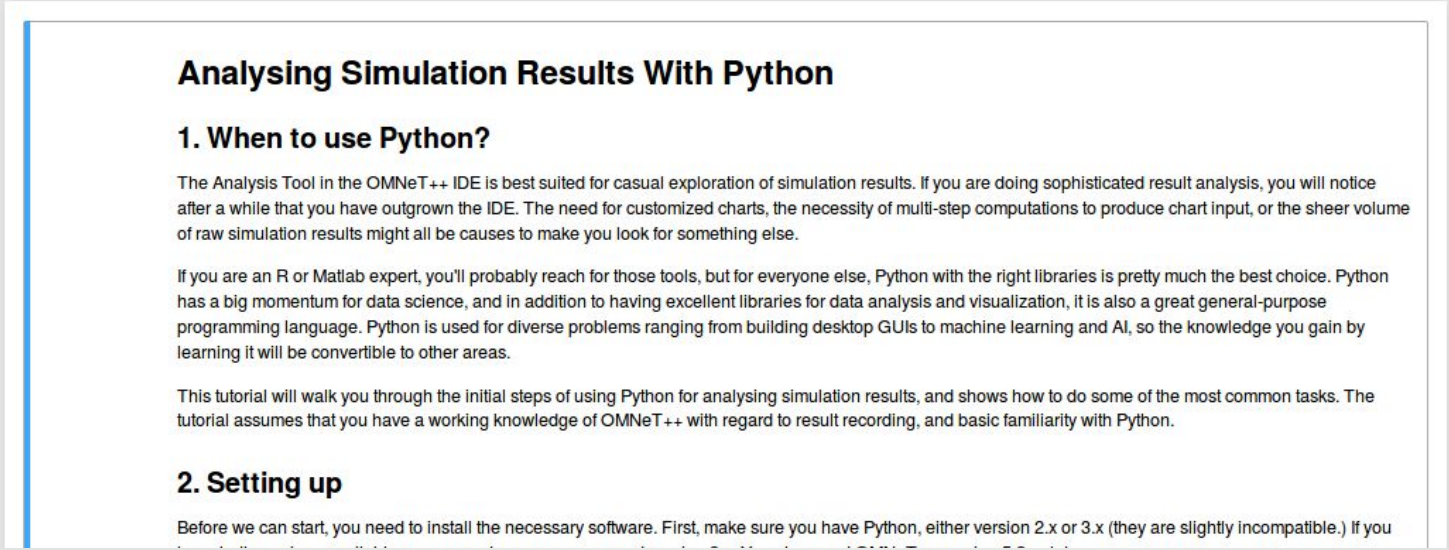




How To Use Them?

A detailed tutorial on processing and plotting OMNeT++ results using Python, Pandas and Matplotlib is in preparation, and will be posted on omnetpp.org shortly.

Latest draft available at: https://omnetpp.org/doc/pandas-tutorial



Analysing Simulation Results With Python

1. When to use Python?

The Analysis Tool in the OMNeT++ IDE is best suited for casual exploration of simulation results. If you are doing sophisticated result analysis, you will notice after a while that you have outgrown the IDE. The need for customized charts, the necessity of multi-step computations to produce chart input, or the sheer volume of raw simulation results might all be causes to make you look for something else.

If you are an R or Matlab expert, you'll probably reach for those tools, but for everyone else, Python with the right libraries is pretty much the best choice. Python has a big momentum for data science, and in addition to having excellent libraries for data analysis and visualization, it is also a great general-purpose programming language. Python is used for diverse problems ranging from building desktop GUIs to machine learning and AI, so the knowledge you gain by learning it will be convertible to other areas.

This tutorial will walk you through the initial steps of using Python for analysing simulation results, and shows how to do some of the most common tasks. The tutorial assumes that you have a working knowledge of OMNeT++ with regard to result recording, and basic familiarity with Python.

2. Setting up

Before we can start, you need to install the necessary software. First, make sure you have Python, either version 2.x or 3.x (they are slightly incompatible.) If you

Getting Simulation Results into Python/Pandas

1. Export from the IDE (in CSV or JSON)
   - Read CSV into Python e.g. with Pandas' read_csv() function
   - Convenient for casual use, but cumbersome if needs to be repeated often
2. Export using scavetool
   - Advantage: automation via shell scripts
3. Use specialized Python lib for reading OMNeT++ result files
   - Eliminates conversion step, integrates into Python workflow
4. The SQLite way: record in SQLite format, then use SQL queries in Python
   - Advantage: power of SQL (easy to make complex queries)
   - Issue: cannot join data from multiple files in one query
5. (Custom result recording in a format well supported in Python, e.g. CSV)
   - Eliminates conversion step and custom loaders, but more difficult to implement

Exporting in CSV

$ scavetool x *.sca *.vec -o aloha.csv

Exported CSV:

```
run,type,module,name,attrname,attrvalue,value,count,[...],binedges,binvalues,vectime,vecvalue
PureAlohaExperiment-4-20170627-20:42:20-22739,runattr,,,configname,PureAlohaExperiment,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,runattr,,,datetime,20170627-20:42:20,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,runattr,,,experiment,PureAlohaExperiment,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,runattr,,,inifile,omnetpp.ini,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,itervar,,,iaMean,3,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,itervar,,,numHosts,10,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,param,,,Aloha.numHosts,10,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,param,,,Aloha.host[*].iaTime,exponential(3s),,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,param,,,Aloha.numHosts,20,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,param,,,Aloha.slotTime,0,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,param,,,Aloha.txRate,9.6kbps,,,,,,,,,,
PureAlohaExperiment-4-20170627-20:42:20-22739,param,,,Aloha.host[*].pkLenBits,952b,,,,,,,,,,
...
```

CSV Structure

Exported CSV contains one item per row.

Different columns are filled in for different item types:

- **For scalars:** run, type='scalar', module, name, value
- **For vectors:** run, type='vector', module, name, vectime*, vecvalue*
- **For statistics:** run, type='statistic', module, name, count, mean, stddev, min, max, etc.
- **For histograms:** run, type='histogram', module, name, <statistic columns>, binedges*, binvalues*
- **For result attributes:** run, type='attr', module, name, attrname, attrvalue
- **For iteration variables:** run, type='itervar', attrname, attrvalue
- **For run attributes:** run, type='runattr', attrname, attrvalue
- **For param assignments:** run, type='param', attrname, attrvalue

* field contains space-separated numbers as a string

Importing into Python

Reading the CSV file into a Pandas data frame:

```
import pandas as pd
df = pd.read_csv('aloha.csv')
```

Extra conversions necessary:

- “true”/”false” -> True/False
- “0.6 1.3 5.2” -> [0.6, 1.3, 5.2]
- Both read-time or post-read conversion possible
- Details available in the tutorial

Selecting Data

Selecting columns:

df[“name”], df.name

df[[“run”, “attrname”, “attrvalue”]]

Filtering by rows:

df[(df.type==”scalar”) & (df.name==”pkdrop:count”)]

(elementwise comparisons, resulting in Boolean arrays; data frame indexed with a Boolean array selects rows that correspond to True)

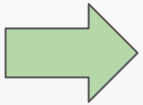
Arranging Data

Use pivot() to “reshape” data based on column values

df.pivot(index='run', columns='name', values='value')

- turns unique values in the “name” column into separate columns
- numbers in the “value” column become cell values

run	name	value
run1	throughput	1204
run1	delay	0.012
run2	throughput	1535
run2	delay	0.018
run3	throughput	2321
run3	delay	0.027



run	throughput	delay
run1	1204	0.012
run2	1535	0.018
run3	2321	0.027

Arranging Data, cont'd

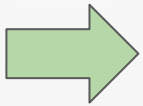
pivot_table(): a more powerful variant that can aggregate numerical data

df.pivot_table(index='iaMean', columns='numHosts', values='utilization', aggfunc='mean')

- unique values of "iaMean" defines rows
- turns unique values in the "numHosts" column into separate columns
- the mean of the numbers for the same *(iaMean,numHosts)* pairs become cell values
- aggregation function is a parameter (default is mean()).

run*	numHosts	iaMean	utilization
run1	10	1.0	0.156013
run2	10	1.0	0.156219
run3	10	2.0	0.194817
...	...	...	...

* multiple repetitions for each *(numHosts, iaMean)* pair



numHosts	10.0	15.0	20.0
iaMean			
1.0	0.156116	0.089539	0.046586
2.0	0.194817	0.178159	0.147564
3.0	0.176321	0.191571	0.183976
4.0	0.153569	0.182324	0.190452
5.0	0.136997	0.168780	0.183742

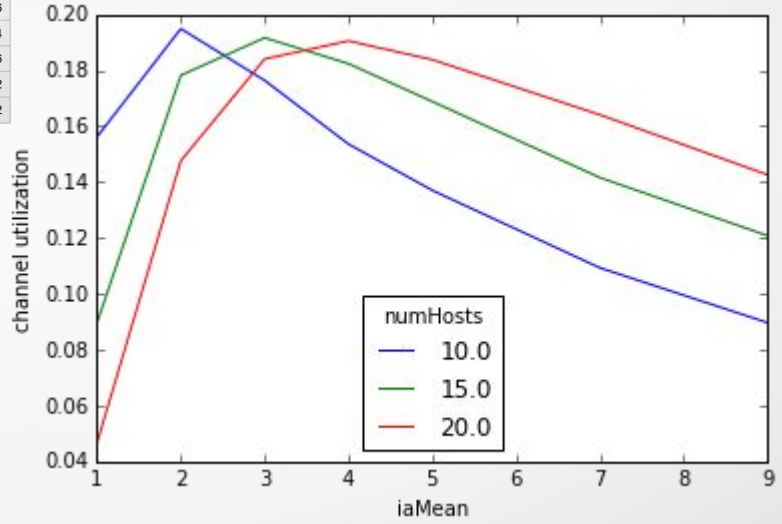
Plotting Scalars

Data frame has a plotting function that understands the previous table

pivot_df = ...

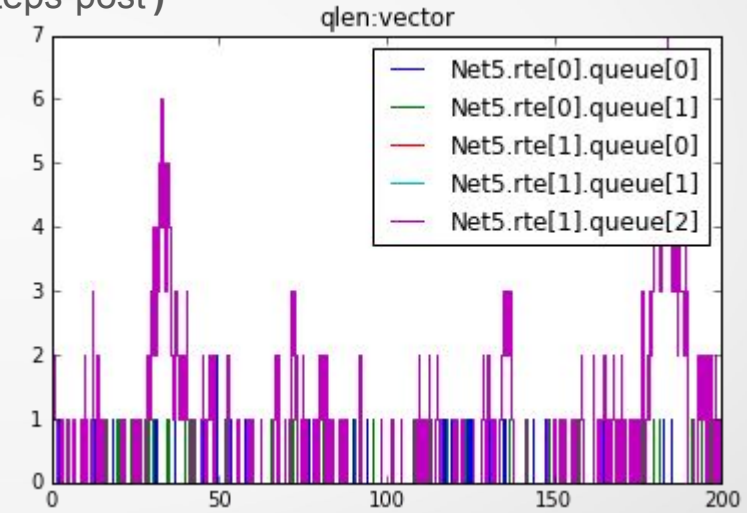
pivot_df.plot.line()
plt.ylabel('channel utilization')
plt.show()

numHosts	10.0	15.0	20.0
iaMean			
1.0	0.156116	0.089539	0.046586
2.0	0.194817	0.178159	0.147564
3.0	0.176321	0.191571	0.183976
4.0	0.153569	0.182324	0.190452
5.0	0.136997	0.168780	0.183742



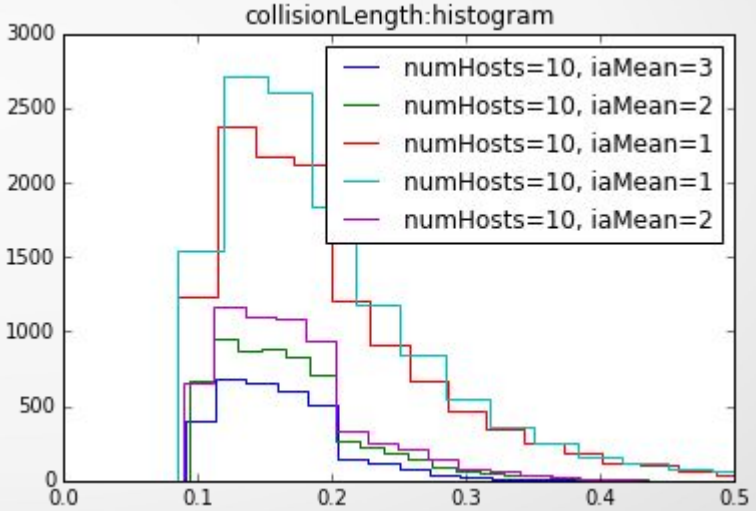
Plotting Vectors

```
vectors_df = …
for row in vectors_df.itertuples():
    plt.plot(row.vectime, row.vecvalue, drawstyle='steps-post')
plt.title(vectors_df.name.values[0])
plt.legend(vectors_df.module)
plt.show()
```



Plotting Histograms

```
histograms_df = …
for row in histograms_df.itertuples():
    plt.plot(row.binedges, np.append(row.binvalues, 0), drawstyle='steps-post')
plt.title('collisionLength:histogram')
plt.legend(histograms_df.iterationvars)
plt.xlim(0, 0.5)
plt.show()
```



IDE Analysis Tool Redesign (Work in Progress)

Motivation to Improve the Analysis Tool

Expressiveness / power:

- Limited computational power (arbitrary computations cannot be expressed)
- Limited charting options (only the most common chart types are supported)

Usability:

- The “Dataset” UI concept feels unnatural to most users as a way to describe data selection, computational and charting steps
- “Datasets” are cumbersome to create and edit in the tree-based UI

No transition path:

- No support for migrating existing analysis to R or Python, user needs to reimplement everything from scratch

Goals

What we'd like:

- Be able to express arbitrary computations, concisely
- More charting options (possibly unlimited)
- Smooth transition towards standalone, script-based result analysis
- Retain ease-of-use for casual result exploration and plotting (point-and-click UI, dialogs, etc.)

"Simple things should be easy, complicated things should be possible"

Solution Draft

"Datasets" UI replaced by "Charts" UI

A "Chart":

- Created by selecting result items and hitting "Plot" button, or directly ("New blank chart" command)
- Encapsulates a user-editable script to select data, perform computations and plot the result
- Runs script and displays the plot when opened
- Chart still configurable via its "Properties"dialog

"Charts" page may also contain:

- Folders (for grouping items)
- Chart Sheets
- Scripts (reusable across Charts)

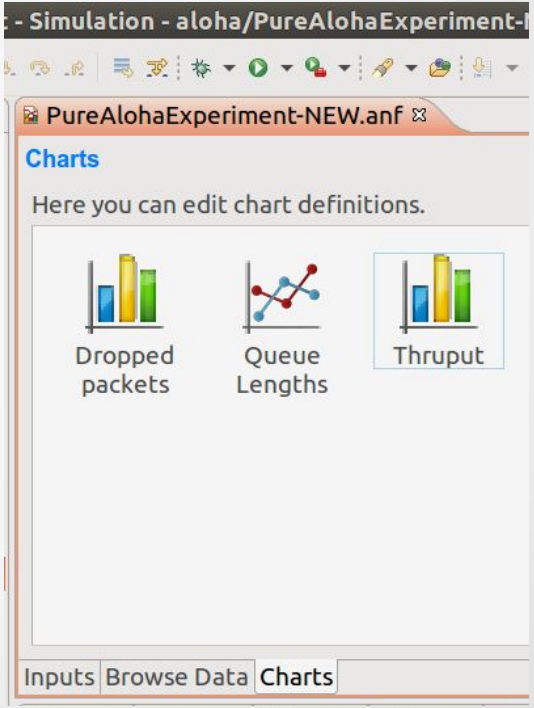


Chart Scripting

Envisioned solution: use Python/Pandas/Matplotlib

- Great potential: boosts computational and plotting abilities of the IDE
- Provides smooth migration path towards standalone, scripted result analysis

Details:

- On creation of a Chart, initial script is copied from a template
- Script has API access to the content of result files specified on the Inputs page and loaded by the Analysis Tool
- Plotting: via Matplotlib, or instantiating Analysis Tool's built-in chart types via Python API

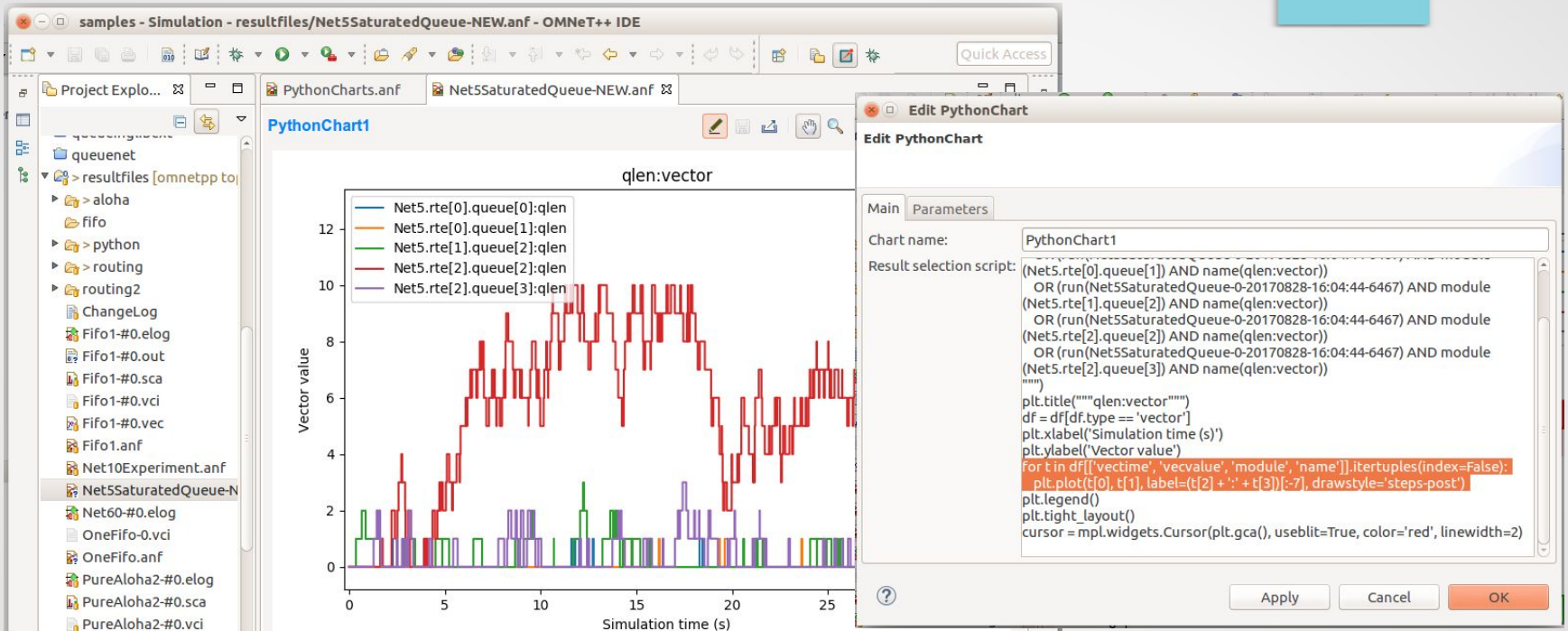
Technology

- Java-Python bridge (separate process, socket-based communication)
  - Running processing in separate process isolates IDE from potential crashes, makes it easier to deal with out-of-memory conditions and to abort long-running (or runaway) computations
- Matplotlib custom back-end, based on the “Agg” raster backend
- Status: proof-of-concept prototype exists, looks good

Other UI Improvements

- Inputs page redesigned for simplicity
- Pages now contain local toolbar for most frequently used actions
- Increase/decrease display precision for numbers
- Performance improvements

Prototype UI



Roadmap

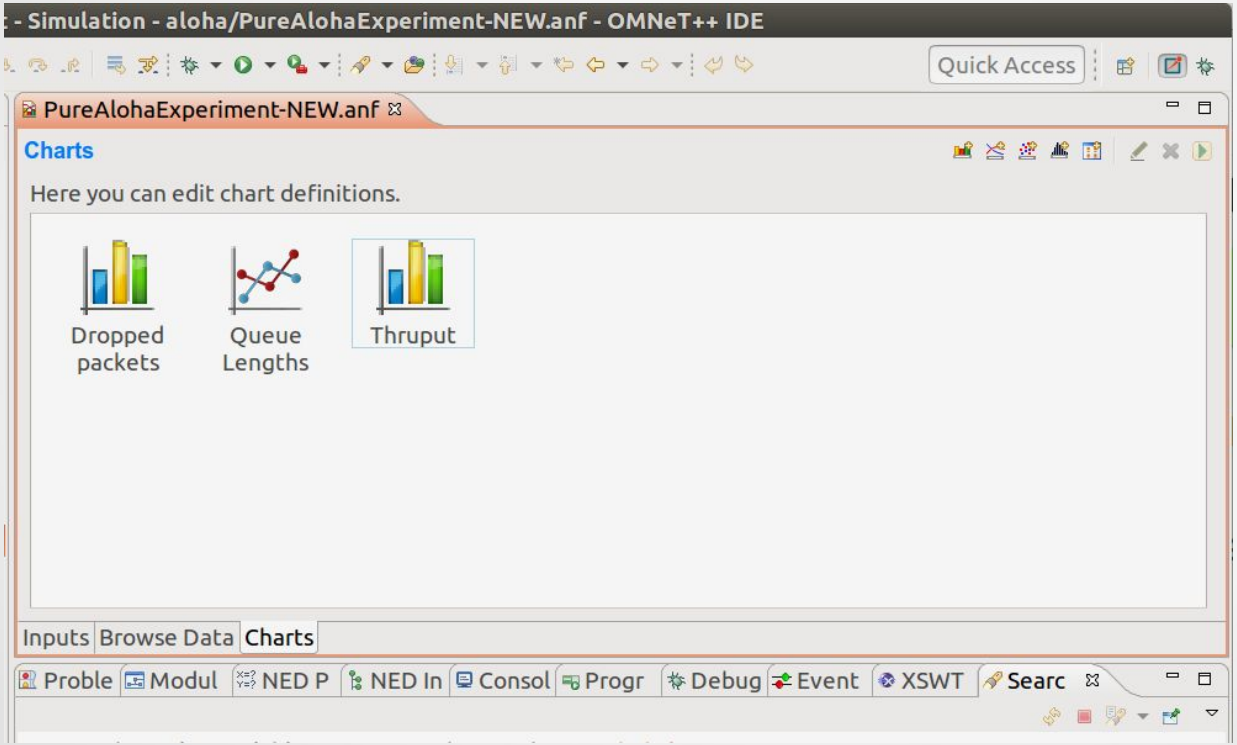
OMNeT++ 5.2 release: by end September

Python/Pandas tutorial for OMNeT++: at the same time as OMNeT++ 5.2

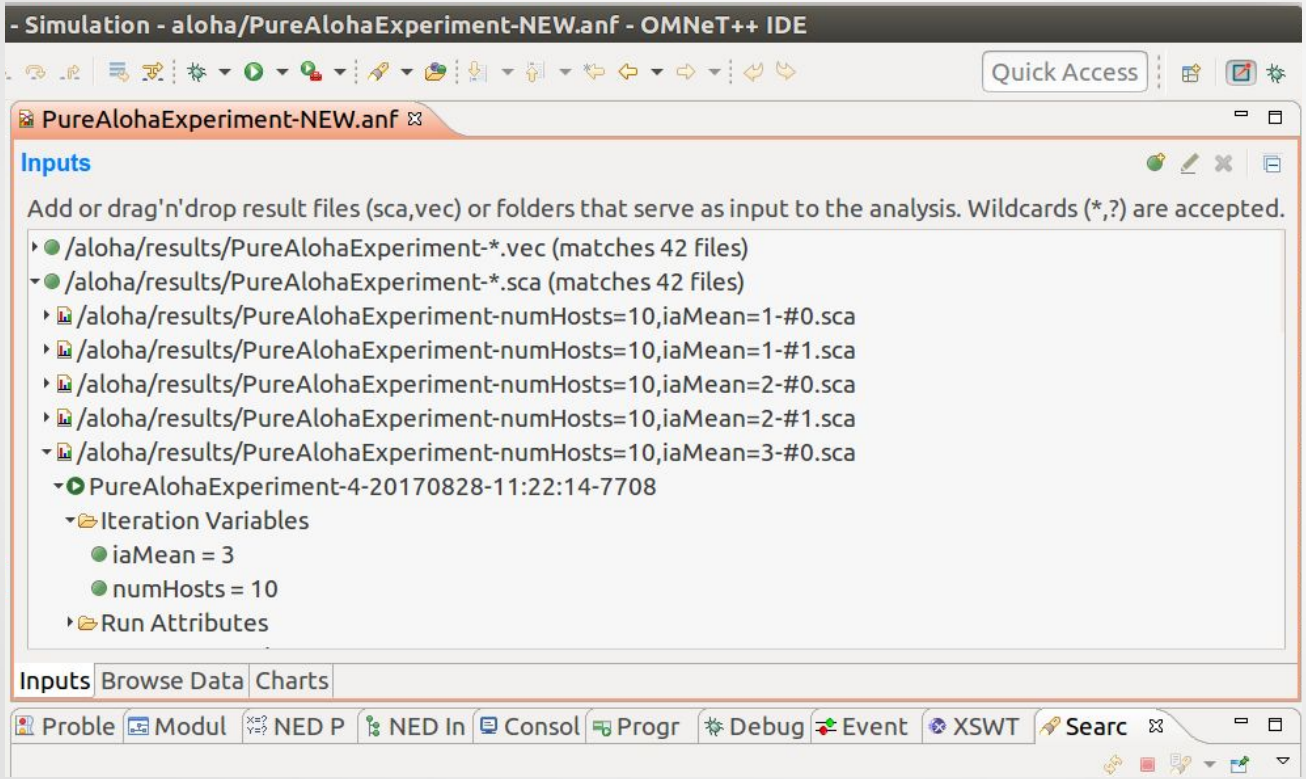
Enhanced Analysis Tool in the IDE, with Python integration: OMNeT++ 5.3
(release planned for March 2018)

Thank you for your attention!

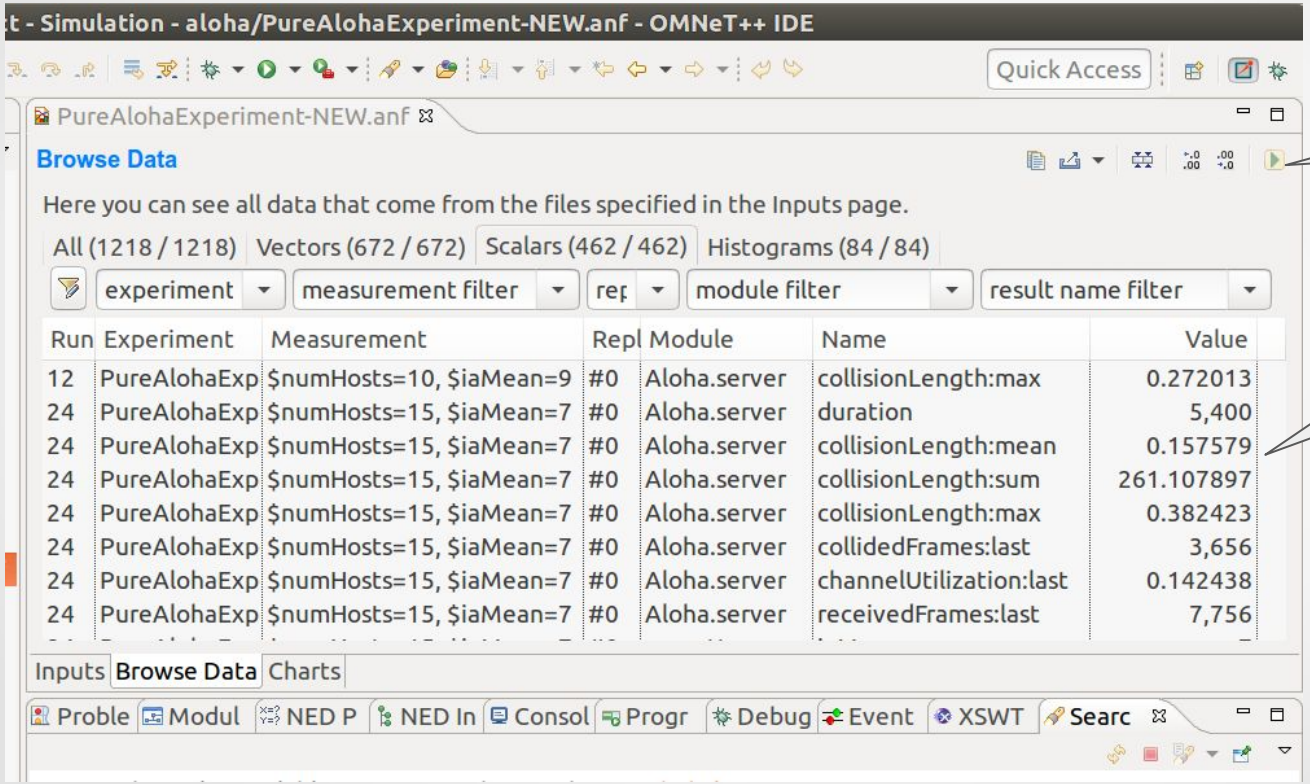
“Charts” Page


- Simulation - aloha/PureAlohaExperiment-NEW.anf - OMNeT++ IDE
Quick Access
PureAlohaExperiment-NEW.anf
Charts
Here you can edit chart definitions.
Dropped packets
Queue Lengths
Thruput
Inputs
Browse Data
Charts
Proble
Modul
NED P
NED In
Consol
Progr
Debug
Event
XSWT
Searc

Redesigned “Inputs” Page



Refined “Browse Data” Page



Local toolbar w/ most common actions

Right-align, display precision adjustable