Python in Computational Neuroscience & Modular toolkit for Data Processing (MDP)

Tiziano Zito, Pietro Berkes, Niko Wilbert
Python in Computational Neuroscience
Python

Created in 1991 by Guido van Rossum as a scripting language. Its characteristics are:

- very concise and readable code, almost like pseudo-code:
  
  ```python
  print "hello world"
  ```

- garbage collection (memory is freed automatically)

- dynamically typed

  ```python
  a = "test"
  
  def increment(value):
      return value + 1
  
  increment(a)  # error at runtime
  ```

Allows fast implementation, relies on conventions and documentation.

Note that static typing often only catches simple bugs, but not subtle ones (e.g., division by zero).
whitespaces indicate code blocks

```python
for i in range(4):
    line = "Happy Birthday"
    if i % 3:
        line += " to you"
    else:
        line += " dear Guido"
print line
```

supports Object Oriented and to some extend Functional programming

```python
class Test(object):
    def info(self):
        print "test here"

test = Test() # create class instance
test.info() # prints "test here"
```

dynamic nature also enables metaprogramming
Python

- Python code is interpreted by a virtual machine (after being compiled to byte code) or can be written in an interactive interpreter (REPL). Python can be a 100 times slower than C, but relies on external libraries where performance matters (e.g. numerics in Fortran).

- Python is maybe the leading modern dynamic language. (according to TIOBE, when PHP, Pearl and VB are ignored) It is one of three official languages at Google (e.g. Youtube is implemented in Python).

- Python is generally considered to have hit a sweet spot in language design, people just like it.

- Open source ecosystem (language, libraries, IDEs).

Bottom line: Python allows very rapid and enjoyable development.
Python has gained much popularity in science, thanks to its available libraries and language quality.

- Python is now competitor to Matlab in data analysis and smaller simulations.
- Python is increasingly used to interface with the standard neural simulators (like NEURON, e.g. via PyNN).

Examples of Research groups migrating their code bases to Python.

So let’s compare Python and Matlab (hopefully in an objective way) ;-)
### Python vs. Matlab

<table>
<thead>
<tr>
<th>Python</th>
<th>Matlab</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ free and open source</td>
<td>- expensive and proprietary</td>
</tr>
<tr>
<td>+ very good language design</td>
<td>- poor language design</td>
</tr>
<tr>
<td>+ advanced programming tools, scales well</td>
<td>- lack of tools, problems with large projects</td>
</tr>
<tr>
<td>- scientific libraries are good and improving</td>
<td>+ scientific libraries superior in several areas (e.g. statistics)</td>
</tr>
<tr>
<td>+ huge range of libraries from all areas, very diverse</td>
<td>- restricted to numerical applications</td>
</tr>
<tr>
<td>+ easy integration of C code for performance</td>
<td>- interfacing C code problematic</td>
</tr>
</tbody>
</table>

Python is not just a low-cost Matlab clone!
Modular Toolkit for Data Processing
Open Source library (LGPL)

first release 2004

15k+ downloads, available in Debian, Ubuntu, MacPorts, Python(x,y)

originated in and supported by research group of Prof. Wiskott, but used outside neuroscience as well
1. Introducing the basic building blocks of MDP
2. Example
3. Outlook
**Building blocks: Node**

**Node**: fundamental data processing element, node classes represent algorithms, public API:

- **train** (optional)
  - support for multiple phases, batch, online, chunks, supervised, unsupervised

- **execute**
  - map $n$ dimensional input to $m$ dimensional output

- **inverse** (optional)
  - inverse of execute mapping

Data format: 2d numpy arrays
- (1st index for samples, 2nd index for channels)
- automatic checks and conversions (dimensions, dtype).
Example: Principal Component Analysis (PCA)
reduce dimension of data from 10 to 5:

```python
>>> import mdp
>>> import numpy as np
>>> data = np.random.random((50,10))  # 50 data points
>>> node = mdp.nodes.PCANode(output_dim=5,
...                           dtype='float32')
>>> node.train(data)
>>> proj_data = node.execute(data)
```

shortcut:

```python
>>> import mdp
>>> import numpy as np
>>> data = np.random.random((50,10))  # 50 data points
>>> proj_data = mdp.pca(data, output_dim=5, dtype='float32')
```
Some available nodes:

- PCA (standard, NIPALS)
- ICA (FastICA, CuBICA, JADE, TDSEP)
- Locally Linear Embedding
- Hessian Locally Linear Embedding
- Fisher Discriminant Analysis
- Slow Feature Analysis
- Independent Slow Feature Analysis
- Restricted Boltzmann Machine
- Growing Neural Gas
- Factor Analysis
- Gaussian Classifiers
- Polynomial Expansion
- Time Frames
- Hit Parades
- Noise
- ...

Or write your own node (and contribute it :-).
Combine nodes in a **Flow** (data processing pipeline):

```python
>>> flow = PCANode() + SFANode() + FastICANode()
>>> flow.train(train_data)
>>> test_result = flow.execute(test_data)
>>> rec_test_data = flow.invert(test_result)
>>> flow += HitParadeNode()
```

- automatic organization: training, execution, inversion
- automatic checks: dimensions and data formats
- use arrays or iterators
- crash recovery, checkpoints
Building blocks: Network

mdp.hinet package for hierarchical networks

**Layer** (combine nodes horizontally in parallel)

**Switchboard** (routing between layers)

**FlowNode** (combine nodes into a “supernode”)

All these classes are nodes, combine them as you want.
HTML representation of your network:

```python
>>> mdp.hinet.show_flow(flow)
```

Use this for debugging, reports or GUI.
Write your own node class:

```python
>>> class MyNode(Node):
...     def _train(self, x):
...         ... training code ...
...     def _execute(self, x):
...         ... execution code ...
...     ...
>>> flow = PCANode() + MyNode()
```

- integrate with the existing library
- benefit from automatic checks and conversions
- contribute your node to make it available to a broader audience
Parallelization

- for “embarrassingly parallel” problems: data chunks for node training can be processed independently (fork node), combine results in the end (join node)
- parallel training and execution is automatically handled by ParallelFlow
- easy to implement for your own nodes (implement _fork and _join methods)

Example:

```python
>>> pflow = mdp.parallel.ParallelFlow([PCANode(), SFANode()])
>>> scheduler = mdp.parallel.ProcessScheduler(n_processes=4)
>>> pflow.train(data, scheduler)
```
Real World Example

- object recognition system, working on 155x155 pixel image sequences
- several GB of training data for each training phase.
- hierarchical network with nested nodes, 900 “supernodes” on lowest layer
- training is parallelized, takes multiple hours on network

[Franzius, M., Wilbert, N., and Wiskott, L., 2008]
Upcoming: BiNet package

mdp.binet package will allow data flow in both directions, enabling for example error backpropagation and loops.

compatible with both the mdp.parallel and mdp.hinet packages.

HTML+JS based inspector for debugging and analysing

scheduled for inclusion in MDP 3.0 (maybe end of 2009)
Embedding / Using MDP

- Comprehensive documentation:
  tutorial covering basic and advanced usage,
  detailed doc-strings,
  PEP8 compliant, commented, and pylint-clean code

- API is stable and designed for straightforward embedding

- Unittest coverage (400+ unit tests)

- Minimal dependencies: Python + NumPy

- Used by:
  PyMCA (X-ray fluorescence mapping),
  PyMVPA (ML framework for neuroimaging data analysis),
  Chandler (personal organizer application)
Thank you!

mdp-toolkit.sourceforge.net