NeXpy: A Python Toolbox for Interactive Data Analysis

http://www.nexusformat.org/NeXpy

Ray Osborn, Stephan Rosenkranz, John-Paul Castellan
Materials Science Division, Argonne National Laboratory, Argonne, IL
Boyana Norris, Jason Sarich, Dmitry Karpeev
Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL

with special thanks to
Paul Kienzle
NIST Center for Neutron Research, NIST, Gaithersburg, MD
NeXpy - a Python-based approach

- What NeXpy is not:
  - a comprehensive solution to all the issues so far raised
  - particularly sophisticated or novel

- What NeXpy is:
  - a toolbox for manipulating and visualizing arbitrary NeXus data
  - a possible scripting engine for GUI applications
  - a demonstration of the value of combining:
    - a flexible data model
    - a powerful scripting language

[Images of NeXus and NumPy logos]
Python API

- There are two levels to the Python API
  - A one-to-one mapping of the C-API returning Numpy arrays (napi.py)
  - A one-to-one mapping of the NeXus data objects into Python classes (tree.py)

```python
>>> f=nexus.open("data/chopper.nxs")
>>> f.opengroup("entry","NXentry")
>>> f.opendata("title")
>>> f.getdata()
'MgB2 PDOS 43.37g 8K 120meV E0@240Hz T0@120Hz'
>>> f.closedata()
>>> f.closegroup()
>>> f.close()
```

```python
>>> a=nexus.load("data/chopper.nxs")
>>> a.entry.data.nxtree()
data:NXdata
  data = int32(148x750)
    @axes = polar_angle:time_of_flight
    @long_name = Neutron Counts
    @signal = 1
    @units = counts
  polar_angle = float32(148)
    @long_name = Polar Angle [degrees]
    @units = degrees
  time_of_flight = [ 1900.  1902.  1904. ...,  3396.  3398.  3400.]
    @long_name = Time-of-Flight [microseconds]
    @units = microseconds
  title = MgB2 PDOS 43.37g 8K 120meV E0@240Hz T0@120Hz
>>> print a.entry.data.title
MgB2 PDOS 43.37g 8K 120meV E0@240Hz T0@120Hz
```
ARCS/SNS Data
Features of NeXus tree interface

- The entire tree structure of a NeXus file can be loaded with a single command
  - The data values are not read until directly referenced
- NeXus objects can be created by simple assignments
  ```python
  >>> sample = NXsample()
  >>> sample.temperature=40.0
  >>> sample.temperature.units='K'
  >>> entry = NXentry(sample)
  >>> entry.nxtree()
  entry:NXentry
  sample:NXsample
    temperature = 40.0
      @units = K
  ```
- Note: Command-line assignments automatically convert Numpy data into NeXus objects
  - e.g., typing “entry.sample.temperature=40.0” converts the temperature into valid NeXus data
- All objects of the same class can easily be listed
  - e.g., entry.NXdata[0], entry.NXdata[1], ...
Interactive manipulation of NeXus data

- The syntax makes the creation of standard-conforming NeXus data structures simple

```python
>>> entry=NXentry()
>>> x=np.linspace(0,2.*np.pi,101)
>>> y=x
>>> X,Y=np.meshgrid(x,y)
>>> z=np.sin(X)*np.sin(Y)
>>> entry.data=NXdata(z,(x,y))
>>> entry.nxtree()
entry:NXentry
data:NXdata
    axis1 = float64(101)
    axis2 = float64(101)
    signal = float64(101x101)
      @axes = axis1:axis2
      @signal = 1
>>> entry.nxplot()
```

![Figure 1](image)
**NXdata group manipulations**

- The NXdata objects are designed to be manipulated:
  - Diced and sliced
  - Scaled
  - Added and subtracted

```python
>>> data[1:2.].nxsum(0).nxtree()
data:NXdata
  axis2 = float64(101)
  signal = float64(101)
    @axes = axis2
    ... ...,  
-1.00000000e+00 -7.07106781e-01 -2.44929360e-16  
    @axes = x
    @signal = 1
  x = float64(9)
```

```python
>>> (2*data).nxtree()  # or (data+data).nxtree()
data:NXdata
  signal = [ 0.00000000e+00  1.41421356e+00  2.00000000e+00 ...,  
-2.00000000e+00 -1.41421356e+00 -4.89858720e-16  
    @axes = x
    @signal = 1
  x = float64(9)
```

```python
>>> (data+1).nxtree()
data:NXdata
  signal = [ 1.          1.70710678  2.         ...,  0.          0.29289322  1.        
    @axes = x
    @signal = 1
  x = float64(9)
```
NeXpy GUI
Features of NeXpy GUI

- Persistent data
- Comprehensive access to metadata
- File-based memory management
  - Each data object maps directly to a valid NeXus file
- Non-proprietary language (*i.e.*, Python)
- Flexibility to do whatever you want to the data
  - No well-defined algorithms
On the To-Do List (near-term)

- Adding a data editor
- Incorporating generalized coordinate transformations
- Incorporating Open GL modules to improve current Matplotlib speeds
- Expanded use of parallelization, including GPUs
- Adding full 3D plotting
  - Using Mayavi
- Adding a fitting pane
  - Using the DANSE MYSTIC/PARK framework
- Installing NeXpy as a standard part of the NeXus distribution
- Create a version for the iPad

http://www.nexusformat.org/NeXpy