Advanced Astronomical Observations 2022
Session 07: Using Masked Arrays

Kinoshita Daisuke

25 March 2022
publicly accessible version

For this session, we try masked arrays. Astronomical array data may contain inappropriate pixel values. For example, a hot pixel always gives a saturated value. To deal with invalid pixel values, masked arrays are extremely useful.

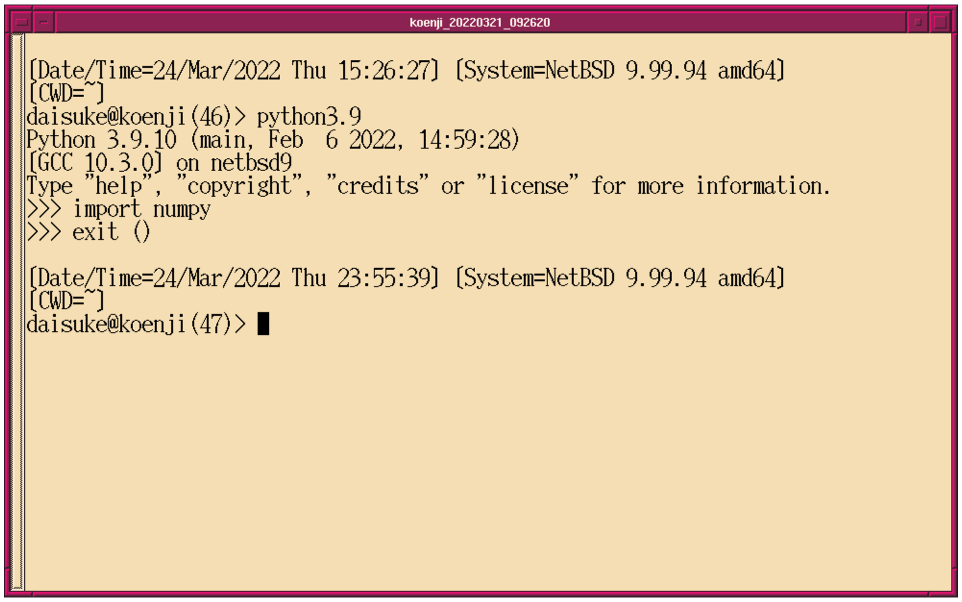
1 Numpy

Make sure to have Numpy on your computer. Try following to check whether you have Numpy properly installed on your computer. (Fig. 1)

```
% python3.9
Python 3.9.10 (main, Feb  6 2022, 14:59:28)
[GCC 10.3.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> exit ()
```

If you do not have Numpy on your computer, you see an error message like below. Visit the official website of Numpy (`https://numpy.org/`, Fig. 2), read the documentation, and install Numpy.

```
% python3.9
Python 3.9.10 (main, Jan 31 2022, 11:49:56)
[GCC 10.3.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```



```
[Date/Time=24/Mar/2022 Thu 15:26:27] [System=NetBSD 9.99.94 amd64]
[CWD=~]
daisuke@koenji(46)> python3.9
Python 3.9.10 (main, Feb  6 2022, 14:59:28)
[GCC 10.3.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> exit ()

[Date/Time=24/Mar/2022 Thu 23:55:39] [System=NetBSD 9.99.94 amd64]
[CWD=~]
daisuke@koenji(47)> 
```

Figure 1: Importing Numpy using interactive mode of Python.

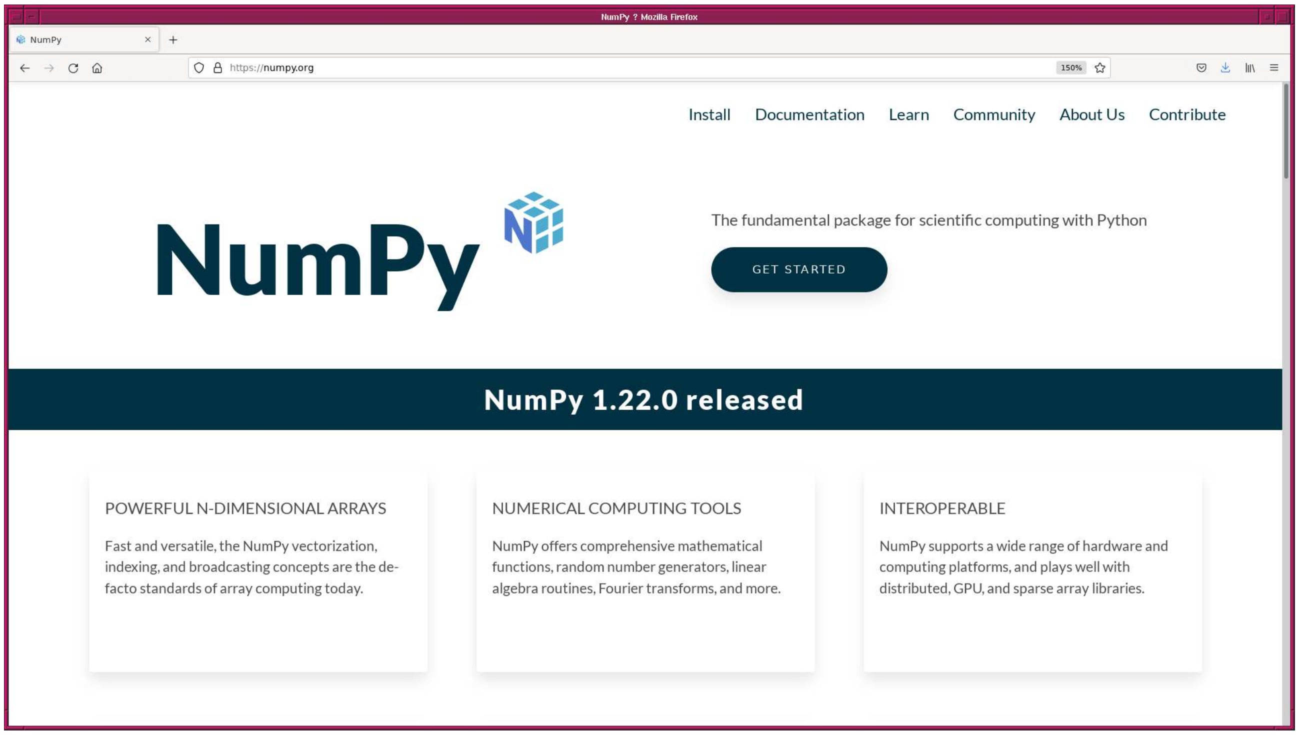
```
ModuleNotFoundError: No module named 'numpy'
>>> exit ()
```

2 Numpy arrays

Here is an example of a calculation using Numpy arrays.

Python Code 1: advobs202202_s07_01.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy

# creating a numpy array
a = numpy.array ([ [100.0, 100.5, 99.0], \
                   [101.0, 99.5, 101.5], \
                   [98.5, 102.0, 99.9] ])

# printing numpy array "a"
print ("a")
print (a)

# creating one more numpy array
b = numpy.array ( [ [50.0, 49.5, 50.5], \
                    [49.0, 51.0, 48.5], \
                    [51.5, 48.0, 49.9] ])

# printing numpy array "b"
print ("b")
print (b)

# calculation
c = a - b
```



The fundamental package for scientific computing with Python

NumPy 1.22.0 released

POWERFUL N-DIMENSIONAL ARRAYS

Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

NUMERICAL COMPUTING TOOLS

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

INTEROPERABLE

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

Figure 2: The official website of Numpy.

```
# printing numpy array "c"
print ("c = a - b")
print (c)
```

Run the script.

```
% chmod a+x advobs202202_s07_01.py
% ./advobs202202_s07_01.py
a
[[100.   100.5   99. ]
 [101.    99.5  101.5]
 [ 98.5  102.    99.9]]
b
[[50.   49.5  50.5]
 [49.   51.   48.5]
 [51.5  48.   49.9]]
c = a - b
[[50.   51.   48.5]
 [52.   48.5  53. ]
 [47.   54.   50. ]]
```

Calculate a mean.

Python Code 2: advobs202202_s07_02.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
```

```
# creating a numpy array
data = numpy.array ([ 100.0, 99.5, 100.5, 99.0, 101.0, \
                      98.5, 101.5, 98.0, 102.0, 200.0 ])

# creating a numpy array for error
error = numpy.array ([ 1.0, 1.0, 1.0, 1.0, 1.0, \
                       1.0, 1.0, 1.0, 1.0, 100.0 ])

# calculation of a mean
mean = numpy.mean (data)

# printing data and errors
print ("data  =", data)
print ("error =", error)

# printing mean
print ("mean =", mean)

# the other way to calculate a mean
average = numpy.average (data)

# printing mean
print ("average =", average)

# weighted average
weighted_average = numpy.average (data, weights=1.0/error)

# printing weighted average
print ("weighted average =", weighted_average)
```

Execute the script.

```
% chmod a+x advobs202202_s07_02.py
% ./advobs202202_s07_02.py
data  = [100.    99.5 100.5   99.   101.    98.5 101.5   98.   102.   200. ]
error = [  1.    1.    1.    1.    1.    1.    1.    1.    1. 100.]
mean = 110.0
average = 110.0
weighted average = 100.11098779134295
```

Calculate standard deviation.

Python Code 3: advobs202202_s07_03.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy

# creating a numpy array
data = numpy.array ([ 100.0, 99.5, 100.5, 99.0, 101.0, \
                      98.5, 101.5, 98.0, 102.0, 200.0 ])

# creating a numpy array for error
error = numpy.array ([ 1.0, 1.0, 1.0, 1.0, 1.0, \
                       1.0, 1.0, 1.0, 1.0, 100.0 ])
```

```
# weighted average
weighted_average = numpy.average (data, weights=1.0/error)

# printing weighted average
print ("weighted average =", weighted_average)

# standard deviation
stddev = numpy.std (data)

# printing standard deviation
print ("stddev =", stddev)

# examining data
for datum in data:
    if ( (datum < weighted_average + 3.0 * stddev) \
         and (datum > weighted_average - 3.0 * stddev) ):
        print ("%5.1f: within average +/- 3.0 * sigma" % datum)
    else:
        print ("%5.1f: outside of average +/- 3.0 * sigma" % datum)
```

Run the script.

```
% chmod a+x advobs202202_s07_03.py
% ./advobs202202_s07_03.py
weighted average = 100.11098779134295
stddev = 30.024989592004857
100.0: within average +/- 3.0 * sigma
 99.5: within average +/- 3.0 * sigma
100.5: within average +/- 3.0 * sigma
 99.0: within average +/- 3.0 * sigma
101.0: within average +/- 3.0 * sigma
 98.5: within average +/- 3.0 * sigma
101.5: within average +/- 3.0 * sigma
 98.0: within average +/- 3.0 * sigma
102.0: within average +/- 3.0 * sigma
200.0: outside of average +/- 3.0 * sigma
```

3 Using masked arrays

Visit following web page to learn about masked arrays of Numpy. (Fig. 3)

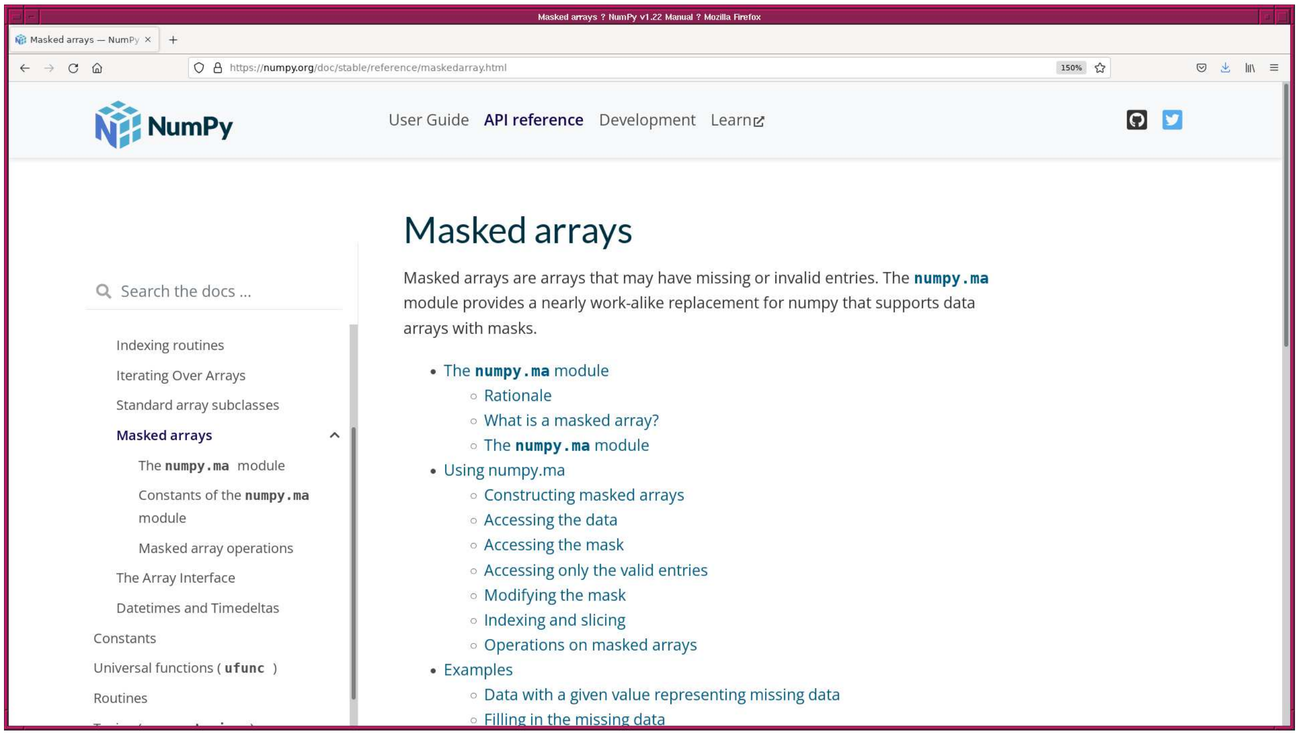
- https://numpy.org/doc/stable/reference/maskedarray.html

3.1 A simple example

Try to use a masked array. Here is an example.

Python Code 4: advobs202202_s07_04.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# creating a masked array
data = numpy.array ([ [1, 2, 3], \
```



Masked arrays

Masked arrays are arrays that may have missing or invalid entries. The **numpy.ma** module provides a nearly work-alike replacement for numpy that supports data arrays with masks.

- The **numpy.ma** module
  - Rationale
  - What is a masked array?
  - The **numpy.ma** module
- Using numpy.ma
  - Constructing masked arrays
  - Accessing the data
  - Accessing the mask
  - Accessing only the valid entries
  - Modifying the mask
  - Indexing and slicing
  - Operations on masked arrays
- Examples
  - Data with a given value representing missing data
  - Filling in the missing data

Figure 3: The official document of masked array of Numpy.

```
                           [4 , 5 , 6] , \
                           [7 , 8 , 9] ])
mask = numpy.array ([ [0 , 0 , 0] , \
                           [0 , 0 , 1] , \
                           [1 , 0 , 0] ])
masked_data = numpy.ma.masked_array (data , mask=mask)

# masked array?
is_masked_data = numpy.ma.is_masked (data)
print ("Is \"data\" masked? ==> %s" % is_masked_data)
is_masked_masked_data = numpy.ma.is_masked (masked_data)
print ("Is \"masked_data\" masked? ==> %s" % is_masked_masked_data)

# printing original data and masked array
print ("original data:")
print (data)
print ("mask:")
print (mask)
print ("masked_data:")
print (masked_data)
```

Execute the script.

```
% chmod a+x advobs202202_s07_04.py
% ./advobs202202_s07_04.py
Is "data" masked? ==> False
Is "masked_data" masked? ==> True
original data:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
mask:
[[0 0 0]
 [0 0 1]
 [1 0 0]]
masked_data:
[[1 2 3]
 [4 5 --]
 [-- 8 9]]
```

3.2 Making a mask using Boolean data

A mask can be an array of Boolean data type.

Python Code 5: advobs202202_s07_05.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# creating a masked array
data = numpy.array ([ [1, 2, 3], \
                      [4, 5, 6], \
                      [7, 8, 9] ])
mask = numpy.array ([ [False, False, False], \
                      [False, False, True], \
                      [True, False, False] ])
masked_data = numpy.ma.array (data, mask=mask)

# printing masked array
print ("data:")
print (data)

# printing masked array
print ("mask:")
print (mask)

# printing masked array
print ("masked_data:")
print (masked_data)
```

Run the script.

```
% chmod a+x advobs202202_s07_05.py
% ./advobs202202_s07_05.py
data:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
mask:
[[False False False]
 [False False  True]
 [ True False False]]
masked_data:
[[1 2 3]
 [4 5 --]
 [-- 8 9]]
```

3.3 Extracting data and mask from masked array

Here is one more example of making a masked array.

Python Code 6: advobs202202_s07_06.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# creating a numpy array
data = numpy.array ([ 100.0, 99.5, 100.5, 99.0, 101.0, \
                      98.5, 101.5, 98.0, 102.0, 200.0 ])

# creating a numpy array for error
error = numpy.array ([ 1.0, 1.0, 1.0, 1.0, 1.0, \
                       1.0, 1.0, 1.0, 1.0, 100.0 ])

# initialisation of a numpy array for a mask
mask = numpy.array ([])

# weighted average
weighted_average = numpy.average (data, weights=1.0/error)

# printing weighted average
print ("weighted average =", weighted_average)

# standard deviation
stddev = numpy.std (data)

# printing standard deviation
print ("stddev =", stddev)

# examining data
for datum in data:
    if ( (datum < weighted_average + 3.0 * stddev) \
         and (datum > weighted_average - 3.0 * stddev) ):
        print ("%5.1f: within average +/- 3.0 * sigma" % datum)
        # appending a value to the array "mask"
        mask = numpy.append (mask, 0)
    else:
        print ("%5.1f: outside of average +/- 3.0 * sigma" % datum)
        # appending a value to the array "mask"
        mask = numpy.append (mask, 1)

# printing the array "data"
print ("data:", data)

# printing the array "mask"
print ("mask:", mask)

# creating a masked array
masked_data = numpy.ma.MaskedArray (data, mask=mask)

# printing masked array
print ("masked_data =", masked_data)
print ("masked_data.data =", masked_data.data)
print ("masked_data.mask =", masked_data.mask)
```

Execute the script.

```
% chmod a+x advobs202202_s07_06.py
% ./advobs202202_s07_06.py
weighted average = 100.11098779134295
stddev = 30.024989592004857
100.0: within average +/- 3.0 * sigma
 99.5: within average +/- 3.0 * sigma
100.5: within average +/- 3.0 * sigma
 99.0: within average +/- 3.0 * sigma
101.0: within average +/- 3.0 * sigma
 98.5: within average +/- 3.0 * sigma
101.5: within average +/- 3.0 * sigma
 98.0: within average +/- 3.0 * sigma
102.0: within average +/- 3.0 * sigma
200.0: outside of average +/- 3.0 * sigma
data: [100.   99.5 100.5  99.  101.   98.5 101.5  98.  102.  200. ]
mask: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
masked_data = [100.0 99.5 100.5 99.0 101.0 98.5 101.5 98.0 102.0 --]
masked_data.data = [100.   99.5 100.5  99.  101.   98.5 101.5  98.  102.  200. ]
masked_data.mask = [False False False False False False False False False  True]
```

4 Calculation using masked arrays

4.1 Calculation using a masked array and a Numpy array

Here is an example of calculation using a masked array.

Python Code 7: advobs202202_s07_07.py

```python
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# masked array
a = numpy.ma.array ([30, 31, 32, 33, 34, 35], mask=[0, 0, 0, 0, 0, 1])

# numpy array
b = numpy.array ([12, 13, 14, 15, 16, 17])

# calculation
c = a - b

# result of calculation
print ("a =", a)
print ("b =", b)
print ("c = a - b =", c)
```

Run the script, and show the result of calculation.

```
% chmod a+x advobs202202_s07_07.py
% ./advobs202202_s07_07.py
a = [30 31 32 33 34 --]
b = [12 13 14 15 16 17]
c = a - b = [18 18 18 18 18 --]
```

4.2 Calculation using two masked arrays

Following is an example of calculations using two masked arrays.

Python Code 8: advobs202202_s07_08.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# masked arrays
a = numpy.ma.array ([30, 31, 32, 33, 34, 35], mask=[0, 0, 0, 0, 0, 1])
b = numpy.ma.array ([12, 13, 14, 15, 16, 17], mask=[1, 0, 0, 0, 0, 0])

# calculation
c = a + b

# result of calculation
print ("a =", a)
print ("b =", b)
print ("c = a + b =", c)
```

Execute the script.

```
% chmod a+x advobs202202_s07_08.py
% ./advobs202202_s07_08.py
a = [30 31 32 33 34 --]
b = [-- 13 14 15 16 17]
c = a + b = [-- 44 46 48 50 --]
```

5 Calculating statistical values

Calculate average and standard deviation.

Python Code 9: advobs202202_s07_09.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# creating a numpy arrays
data = numpy.array ([100.0, 100.1, 99.9, 100.2, 99.8, \
                     100.3, 99.7, 100.4, 99.6, 300.0])
mask = numpy.array ([0, 0, 0, 0, 0, 0, 0, 0, 0, 1])

# creating a masked array
mdata = numpy.ma.array (data, mask=mask)

# printing data
print ("original data:", data)
print ("masked data:  ", mdata)

# calculation of average
average_data  = numpy.average (data)
```

```
average_mdata = numpy.ma.average (mdata)

# printing average
print ("average of original data:", average_data)
print ("average of masked data:  ", average_mdata)

# calculation of standard deviation
stddev_data  = numpy.std (data)
stddev_mdata = numpy.ma.std (mdata)

# printing stddev
print ("stddev of original data:", stddev_data)
print ("stddev of masked data:  ", stddev_mdata)
```

Run the script.

```
% chmod a+x advobs202202_s07_09.py
% ./advobs202202_s07_09.py
original data: [100.   100.1   99.9 100.2   99.8 100.3   99.7 100.4   99.6 300. ]
masked data:   [100.0 100.1 99.9 100.2 99.8 100.3 99.7 100.4 99.6 --]
average of original data: 120.0
average of masked data:   100.00000000000001
stddev of original data: 60.000499997916684
stddev of masked data:   0.2581988897471623
```

6 Sorting

6.1 Using `numpy.ma.sort` function

Try sorting.

Python Code 10: advobs202202_s07_10.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# masked array
a = numpy.ma.array ([35, 31, 34, 32, 33, 30], mask = [0, 0, 0, 1, 0, 0])

# sorting
b = numpy.ma.sort (a)

# printing result
print ("original data:", a)
print ("sorted data:   ", b)
```

Run the script and check the result of sorting.

```
% chmod a+x advobs202202_s07_10.py
% ./advobs202202_s07_10.py
original data: [35 31 34 -- 33 30]
sorted data:   [30 31 33 34 35 --]
```

6.2 Using sort method

In-place sorting is carried out by sort() method.

Python Code 11: advobs202202_s07_11.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# masked array
a = numpy.ma.array ([35, 31, 34, 32, 33, 30], mask = [0, 0, 0, 1, 0, 0])

# printing masked array
print ("original data:", a)

# in-place sorting
a.sort ()

# printing result
print ("data after sorting:", a)
```

Execute the script.

```
% chmod a+x advobs202202_s07_11.py
% ./advobs202202_s07_11.py
original data: [35 31 34 -- 33 30]
data after sorting: [30 31 33 34 35 --]
```

7 Sigma-clipping using Astropy

The function "`sigma_clip`" of astropy.stats returns a masked array if an option `masked=True` is given. Try following.

Python Code 12: advobs202202_s07_12.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# importing astropy module
import astropy
import astropy.stats

# raw data
raw = numpy.array ([100.0, 100.1, 100.2, 100.3, 100.4, \
                    100.5, 100.6, 100.7, 100.8, 100.9, \
                    500.0, 1000.0, 3000.0])

# printing raw data
print ("raw data:")
print (raw)

# statistical information of raw data
print ("mean    =", numpy.mean (raw) )
print ("median =", numpy.median (raw) )
print ("stddev =", numpy.std (raw) )

# sigma clipping with masked=False
clipped = astropy.stats.sigma_clip (raw, sigma=3.0, maxiters=5, \
                                    cenfunc='median', masked=False)

print ("Is \"clipped\" masked? %s" % numpy.ma.is_masked (clipped) )
print ("clipped data:")
print (clipped)

# sigma clipping with masked=True
mdata = astropy.stats.sigma_clip (raw, sigma=3.0, maxiters=5, \
                                  cenfunc='median', masked=True)

print ("Is \"mdata\" masked? %s" % numpy.ma.is_masked (mdata) )
print ("masked data:")
print (mdata)
```

Run the script.

```
% chmod a+x advobs202202_s07_12.py
% ./advobs202202_s07_12.py
raw data:
[ 100.    100.1   100.2   100.3   100.4   100.5   100.6   100.7   100.8   100.9
  500.   1000.   3000. ]
mean   = 423.4230769230769
median = 100.6
stddev = 785.4528592365299
Is "clipped" masked? False
clipped data:
[100.  100.1 100.2 100.3 100.4 100.5 100.6 100.7 100.8 100.9]
Is "mdata" masked? True
masked data:
[100.0 100.1 100.2 100.3 100.4 100.5 100.6 100.7 100.8 100.9 -- -- --]
```

8 Making a mask using logic functions

Masks can be generated conveniently using logic functions. Here are some examples.

8.1 Making a mask using `numpy.greater`

Use a function `numpy.greater` to make a mask.

Python Code 13: advobs202202_s07_13.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# data
data = numpy.array ([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, \
                     5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5])

# printing data
```

```
print ("data:")
print (data)

# making a mask using a function numpy.greater
mask = numpy.greater (data, 7.0)

# printing mask
print ("mask:")
print (mask)

# making a masked array
masked_data = numpy.ma.array (data, mask=mask)

# printing a masked array
print ("masked_data:")
print (masked_data)
```

Execute the script. All the data greater than 7.0 are masked.

```
% chmod a+x advobs202202_s07_13.py
% ./advobs202202_s07_13.py
data:
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
mask:
[False False False False False False False False False False False False
 False False False  True  True  True  True  True]
masked_data:
[0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 -- -- -- --
 --]
```

You can have the same mask using the operator ">" instead of using the function `numpy.greater`.

Python Code 14: advobs202202_s07_14.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# data
data = numpy.array ([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, \
                     5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5])

# printing data
print ("data:")
print (data)

# making a mask using an operator ">"
mask = data > 7.0

# printing mask
print ("mask:")
print (mask)

# making a masked array
masked_data = numpy.ma.array (data, mask=mask)
```

```
# printing a masked array
print ("masked_data:")
print (masked_data)
```

Try above script.

```
% chmod a+x advobs202202_s07_14.py
% ./advobs202202_s07_14.py
data:
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
mask:
[False False False False False False False False False False False False
 False False False  True  True  True  True  True]
masked_data:
[0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 -- -- -- --
 --]
```

8.2 Trying a function `numpy.less`

Try the function `numpy.less`.

Python Code 15: advobs202202_s07_15.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# data
data = numpy.array ([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, \
                     5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5])

# printing data
print ("data:")
print (data)

# making a mask using a function numpy.less
mask = numpy.less (data, 3.0)

# printing mask
print ("mask:")
print (mask)

# making a masked array
masked_data = numpy.ma.array (data, mask=mask)

# printing a masked array
print ("masked_data:")
print (masked_data)
```

Run the script.

```
% chmod a+x advobs202202_s07_15.py
% ./advobs202202_s07_15.py
data:
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
mask:
[ True  True  True  True  True  True False False False False False False
 False False False False False False False False]
masked_data:
[-- -- -- -- -- -- 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5]
```

You can also use the operator "<" instead of the function `numpy.ma.less`.

Python Code 16: advobs202202_s07_16.py

```python
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# data
data = numpy.array ([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, \
                     5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5])

# printing data
print ("data:")
print (data)

# making a mask using an operator "<"
mask = data < 3.0

# printing mask
print ("mask:")
print (mask)

# making a masked array
masked_data = numpy.ma.array (data, mask=mask)

# printing a masked array
print ("masked_data:")
print (masked_data)
```

Execute the script.

```
% chmod a+x advobs202202_s07_16.py
% ./advobs202202_s07_16.py
data:
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
mask:
[ True  True  True  True  True  True False False False False False False
 False False False False False False False False]
masked_data:
[-- -- -- -- -- -- 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5]
```

8.3 Using greater_equal and less_equal

Try functions greater_equal and less_equal.

Python Code 17: advobs202202_s07_17.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# data
data = numpy.array ([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, \
                     5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5])

# printing data
print ("data:")
print (data)

# making a mask using a function numpy.greater_equal
mask = numpy.greater_equal (data, 7.0)

# printing mask
print ("mask:")
print (mask)

# making a masked array
masked_data = numpy.ma.array (data, mask=mask)

# printing a masked array
print ("masked_data:")
print (masked_data)

# making a mask using a function numpy.less_equal
mask2 = numpy.less_equal (data, 3.0)

# printing mask
print ("mask2:")
print (mask2)

# making a masked array
masked_data2 = numpy.ma.array (data, mask=mask2)

# printing a masked array
print ("masked_data:")
print (masked_data2)
```

Run the script.

```
% chmod a+x advobs202202_s07_17.py
% ./advobs202202_s07_17.py
data:
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
mask:
[False False False False False False False False False False False False
 False False  True  True  True  True  True  True]
masked_data:
```

```
[0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 -- -- -- -- -- --]
mask2:
[ True  True  True  True  True  True  True False False False False False
 False False False False False False False False]
masked_data:
[-- -- -- -- -- -- -- 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5]
```

Operators ">=" and "<=" can be used instead of functions numpy.greater_equal and numpy.less_equal.

Python Code 18: advobs202202_s07_18.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# data
data = numpy.array ([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, \
                     5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5])

# printing data
print ("data:")
print (data)

# making a mask using an operator ">="
mask = data >= 7.0

# printing mask
print ("mask:")
print (mask)

# making a masked array
masked_data = numpy.ma.array (data, mask=mask)

# printing a masked array
print ("masked_data:")
print (masked_data)

# making a mask using an operator "<="
mask2 = data <= 3.0

# printing mask
print ("mask2:")
print (mask2)

# making a masked array
masked_data2 = numpy.ma.array (data, mask=mask2)

# printing a masked array
print ("masked_data:")
print (masked_data2)
```

Execute the script.

```
% chmod a+x advobs202202_s07_18.py
% ./advobs202202_s07_18.py
data:
```

```
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
mask:
[False False False False False False False False False False False False
 False False  True  True  True  True  True  True]
masked_data:
[0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 -- -- -- -- -- --]
mask2:
[ True  True  True  True  True  True  True False False False False False
 False False False False False False False False]
masked_data:
[-- -- -- -- -- -- -- 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5]
```

8.4 Making a mask using two conditions

Use two conditions to make a mask.

Python Code 19: advobs202202_s07_19.py

```python
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# data
data = numpy.array ([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, \
                     5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5])

# printing data
print ("data:")
print (data)

# making a mask using two conditions
mask = numpy.logical_or ( (data < 3.0), (data > 7.0) )

# printing mask
print ("mask:")
print (mask)

# making a masked array
masked_data = numpy.ma.array (data, mask=mask)

# printing a masked array
print ("masked_data:")
print (masked_data)
```

Execute the script.

```
% chmod a+x advobs202202_s07_19.py
% ./advobs202202_s07_19.py
data:
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
mask:
[ True  True  True  True  True  True False False False False False False
 False False False  True  True  True  True  True]
masked_data:
```

```
[-- -- -- -- -- -- 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 -- -- -- -- --]
```

The operator "|" can be used instead of the function `numpy.logical_or`.

Python Code 20: advobs202202_s07_20.py

```python
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# data
data = numpy.array ([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, \
                     5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5])

# printing data
print ("data:")
print (data)

# making a mask using two conditions
mask = (data < 3.0) | (data > 7.0)

# printing mask
print ("mask:")
print (mask)

# making a masked array
masked_data = numpy.ma.array (data, mask=mask)

# printing a masked array
print ("masked_data:")
print (masked_data)
```

Execute the script.

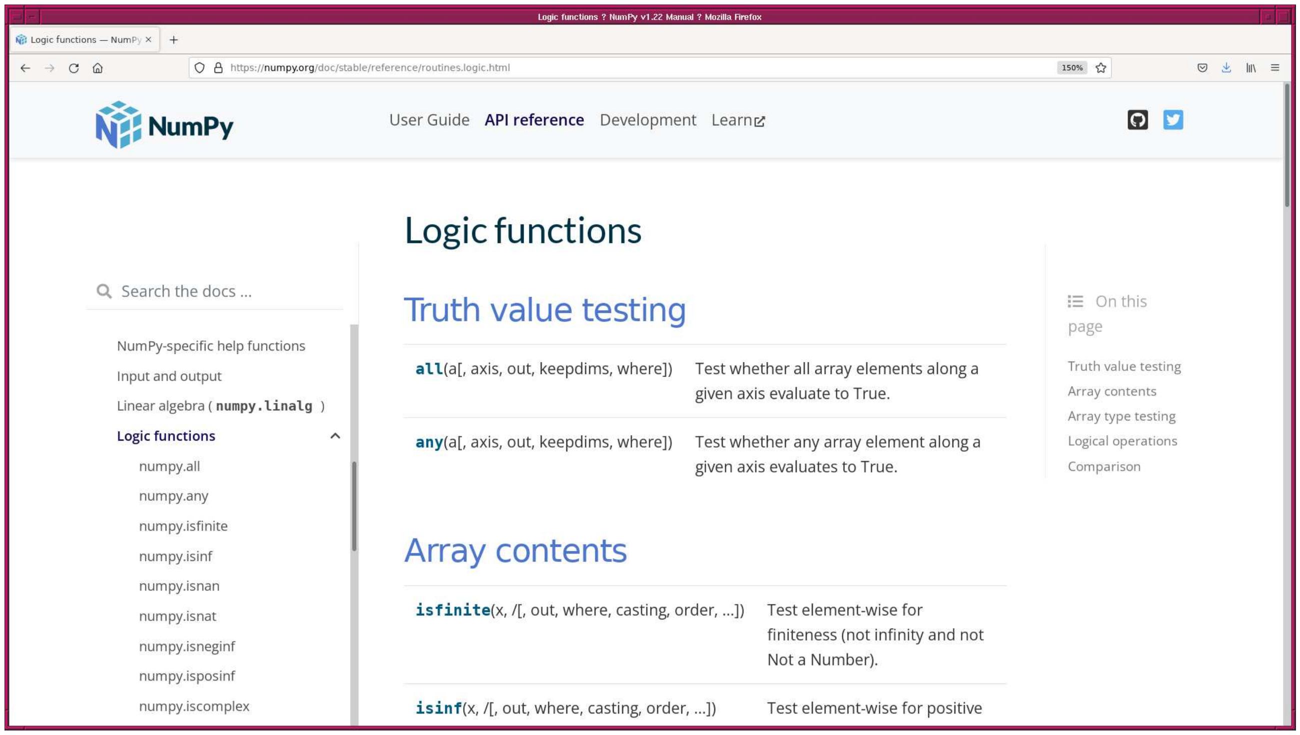
```
% chmod a+x advobs202202_s07_20.py
% ./advobs202202_s07_20.py
data:
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
mask:
[ True  True  True  True  True  True False False False False False False
 False False False  True  True  True  True  True]
masked_data:
[-- -- -- -- -- -- 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 -- -- -- -- --]
```

To learn more about logic functions and operators, visit following web page. (Fig. 4)

- `https://numpy.org/doc/stable/reference/routines.logic.html`

9 Sigma-clipping without using `astropy.stats`

Here is an example of sigma-clipping without using `astropy.stats` module.



Logic functions

Truth value testing

all(a[, axis, out, keepdims, where])	Test whether all array elements along a given axis evaluate to True.
any(a[, axis, out, keepdims, where])	Test whether any array element along a given axis evaluates to True.

Array contents

isfinite(x, /[, out, where, casting, order, ...])	Test element-wise for finiteness (not infinity and not Not a Number).
isinf(x, /[, out, where, casting, order, ...])	Test element-wise for positive

Figure 4: The official document about logic functions of Numpy.

Python Code 21: advobs202202_s07_21.py

```
#!/usr/pkg/bin/python3.9

# importing numpy module
import numpy
import numpy.ma

# data
data = numpy.array ([100.0, 99.9, 100.1, 99.8, 1500.0, \
                     100.2, 99.7, 3000.0, 100.3, 99.6, \
                     100.4, 99.5, 100.5, 0.0, 99.4, \
                     100.6, 5000.0, 99.3, 100.7, 99.2, \
                     100.8, 99.1, 300.0, 100.9, 99.0, \
                     101.0, 98.9, 101.1, 150.0, 98.8])

# printing data
print ("data:")
print (data)

# a function to carry out sigma-clipping
def sigma_clip (data, sigma, maxiters):
    # making a mask
    mask = numpy.array ([False] * len (data))
    # making a masked data
    mdata  = numpy.ma.array (data, mask=mask)
    # iterations
    for i in range (maxiters):
        # calculation of median
        median = numpy.ma.median (mdata)
        # calculation of standard deviation
        stddev = numpy.ma.std (mdata)
        # if stddev is the same as the value of previous iteration,
        # then stop the iteration
        if ( (i > 0) and (stddev == stddev_prev) ):
            break
        # higher limit
        high   = median + sigma * stddev
        # lower limit
        low    = median - sigma * stddev
        # making a new mask
        mask   = ( (mdata < low) | (mdata > high) )
        # making a new masked data
        mdata  = numpy.ma.array (mdata, mask=mask)
        # copying median and stddev for next iteration
        median_prev = median
        stddev_prev = stddev
        # printing information
        print ("%d-th iteration" % (i + 1) )
        print ("  median = %f" % median)
        print ("  stddev = %f" % stddev)
        print ("  number of rejected data = %d" \
                  % ( len (mdata) - len (mdata.compressed () ) ) )
    # returning masked array
    return (mdata)

# carrying out sigma-clipping
mdata = sigma_clip (data, 3.0, 10)

# printing result
print ("mdata:")
print (mdata)
```

Run the script, and check the result.

```
% chmod a+x advobs202202_s07_21.py
% ./advobs202202_s07_21.py
data:
[ 100.     99.9   100.1    99.8  1500.    100.2    99.7  3000.    100.3    99.6
  100.4    99.5   100.5     0.    99.4   100.6  5000.     99.3   100.7    99.2
  100.8    99.1   300.    100.9    99.    101.     98.9   101.1   150.     98.8]
1-th iteration
  median = 100.150000
  stddev = 1025.006930
  number of rejected data = 1
2-th iteration
  median = 100.100000
  stddev = 579.537121
  number of rejected data = 2
3-th iteration
  median = 100.050000
  stddev = 262.327548
  number of rejected data = 3
4-th iteration
  median = 100.000000
  stddev = 43.755003
  number of rejected data = 4
5-th iteration
  median = 99.950000
  stddev = 21.848020
  number of rejected data = 5
6-th iteration
  median = 100.000000
  stddev = 9.831180
  number of rejected data = 6
7-th iteration
  median = 99.950000
  stddev = 0.692219
  number of rejected data = 6
mdata:
[100.0 99.9 100.1 99.8 -- 100.2 99.7 -- 100.3 99.6 100.4 99.5 100.5 --
 99.4 100.6 -- 99.3 100.7 99.2 100.8 99.1 -- 100.9 99.0 101.0 98.9 101.1
 -- 98.8]
```

10 For your further reading

- Visit the official web page of masked array module of Numpy to learn about masked array.
  - `https://numpy.org/doc/stable/reference/maskedarray.html`
  - `https://numpy.org/doc/stable/reference/maskedarray.generic.html`
  - `https://numpy.org/doc/stable/reference/routines.ma.html`
- Learn about Boolean and logic functions.
  - `https://docs.python.org/3/library/stdtypes.html`
  - `https://numpy.org/doc/stable/reference/routines.logic.html`

11 Exercise

1. What is Boolean? What are "`True`" and "`False`" of Python? How useful is it?
2. Make five Python scripts which use Boolean and logic functions. Show the source codes of your Python scripts. Show the result of the execution of those Python script.
3. What is a masked array of Numpy? How useful is it? Give some examples and brief descriptions of astronomical applications of masked arrays.
4. Make five Python scripts which use masked arrays. Show the source codes of your Python scripts. Show the result of the execution of those Python script.
5. Make a function which works like `astropy.stats.sigma_clip`. Make sure to design a function to receive a 2-dimensional array, and return a 2-dimensional masked array. Describe the design of your function. Show the source code of your function. Execute the function and show the result. Mention which capabilities of `astropy.stats.sigma_clip` are implemented for your function, and which capabilities are not implemented.
6. Make a function which works like `astropy.stats.sigma_clip`. Make sure to design a function to receive a 3-dimensional data cube, and return a 3-dimensional masked data cube. Describe the design of your function. Show the source code of your function. Execute the function and show the result. Mention which capabilities of `astropy.stats.sigma_clip` are implemented for your function, and which capabilities are not implemented.
7. Make a function which works like `astropy.stats.sigma_clipped_stats`. Describe the design of your function. Show the source code of your function. Execute the function and show the result. Mention which capabilities of `astropy.stats.sigma_clipped_stats` are implemented for your function, and which capabilities are not implemented.
8. Download a DSS or SDSS image including a bright star using `Astroquery` module. Make a Python script to mask saturated pixels, replace the pixel values of saturated pixels with zeros, and write a new image into a FITS file. Show the source code of your Python script. Execute the script, and show the result. Display both original image and masked image using Ginga.