Numpy Arrays

Thomas Schwarz, SJ

NumPy Fundamentals

- Numpy is a module for faster vector processing with numerous other routines
- Scipy is a more extensive module that also includes many other functionalities such as machine learning and statistics

NumPy Fundamentals

- Why Numpy?
  - Remember that Python does not limit lists to just elements of a single class
  - If we have a large list $[a_1, a_2, a_3, \ldots, a_n]$ and we want to add a number to all of the elements, then Python will asks for each element:
    - What is the type of the element
    - Does the type support the + operation
    - Look up the code for the + and execute
- This is slow

NumPy Fundamentals

- Why Numpy?
  - Primary feature of Numpy are arrays:
    - List like structure where all the elements have the same type
      - Usually a floating point type
  - Can calculate with arrays much faster than with list
  - Implemented in C / Java for Cython or Jython

NumPy Arrays

- NumPy Arrays are containers for numerical values
- Numpy arrays have dimensions
  - Vectors: one-dimensional
  - Matrices: two-dimensional
  - Tensors: more dimensions, but much more rarely used
- Nota bene: A matrix can have a single row and a single column, but has still two dimensions

NumPy Arrays

- After installing, try out `import numpy as np`
- Making arrays:
  - Can use lists, though they better be of the same type

```
import numpy as np
my_list = [1,5,4,2]
my_vec = np.array(my_list)
my_list = [[1,2],[4,3]]
my_mat = np.array(my_list)
```

Array Creation

- Numpy can generate arrays:
  - From disks or the net,
    - using various libraries
      - using loadtxt and similar functions
  - From lists and similar data structures
  - Generate them natively

Array Creation

- Numpy has a number of ways to create an array
  - Import numpy as np
  - `np.zeroes((2,3))`
    - `array([[ 0., 0., 0.], [ 0., 0., 0.]])`
  - `np.ones(5)`
    - `array([1., 1., 1., 1. , 1.])`
  - `np.eye(3)` generates the identity matrix
    - `array([[1., 0., 0.], [0., 1., 0.],[0., 0., 1.]])])`

Array Creation

- Numpy has a number of ways to create arrays
  - `np.linspace(1., 4., 6)` creates an array of 6 elements between 1.0 and 4.0 evenly spaced out
    - `array([ 1. ,  1.6,  2.2,  2.8,  3.4,  4. ])`
  - `np.arange(2, 3, 0.1)` a more generalized version of Python's range function (with float step)
    - `array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])`
  - `np.arange(2,5)`
    - `array([2, 3, 4])`

Array Creation

- Can generate using lists, tuples, etc. even with a mix of types
  - `np.array([[1,2,3], (1,0,0.5)])`
    - `array([[1. , 2. , 3. ], [1. , 0. , 0.5]])`

Array Creation

- Creating arrays:
  - np.full to fill in with a given value

```
np.full(5, 3.141)

array([3.141, 3.141, 3.141, 3.141, 3.141])
```

Array Creation

- Can also use random values.
  - Uniform distribution between 0 and 1

```
>>> np.random.random((3,2))
array([[0.39211415, 0.50264835],
       [0.95824337, 0.58949256],
       [0.59318281, 0.05752833]])
```

Array Creation

- Or random integers

```
>>> np.random.randint(0,20,(2,4))
array([[ 5,  7,  2, 10],
       [19,  7,  1, 10]])
```

Array Creation

- Or other distributions, e.g. normal distribution with mean 2 and standard deviation 0.5

```
>>> np.random.normal(2,0.5, (2,3))
array([[1.34857621, 1.34419178, 1.977698  ],
       [1.31054068, 2.35126538, 3.25903903]])
```

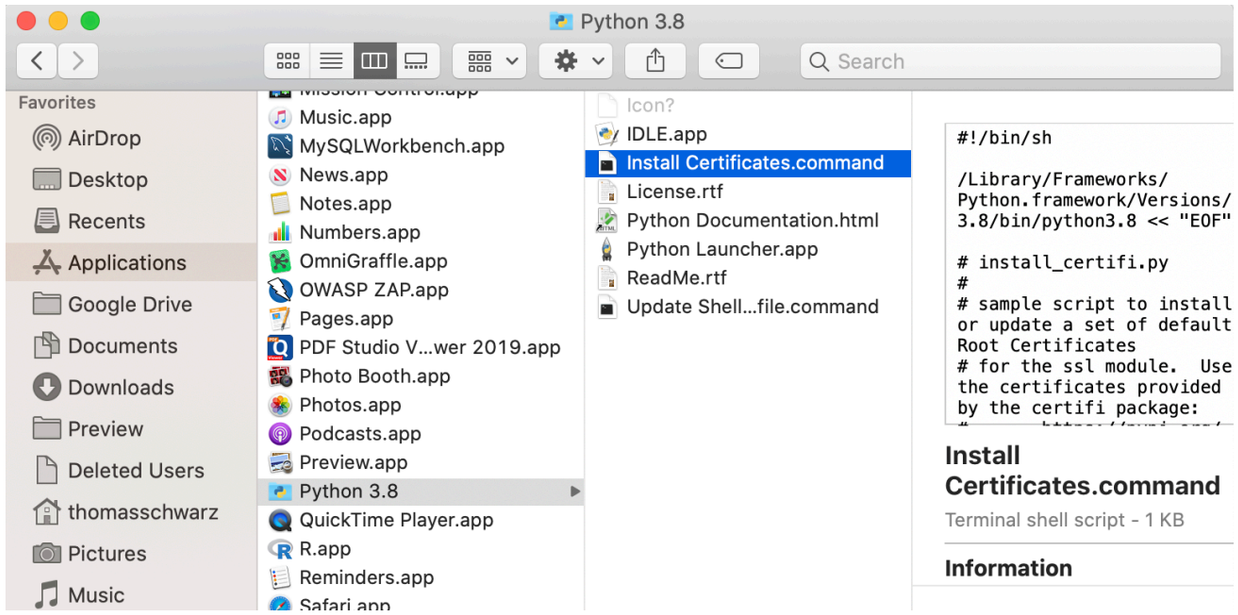
Array Creation

- fromfunction

```
>>> x = np.fromfunction(lambda i,j: (i**2+j**2)//2, (4,5) )

>>> x.astype(int)

array([[ 0,  0,  2,  4,  8],
       [ 0,  1,  2,  5,  8],
       [ 2,  2,  4,  6, 10],
       [ 4,  5,  6,  9, 12]])

>>> x.shape
(4,5)
```

Array Creation

- Creating from download / file
  - We use urllib.request module
    - If you are on Mac, you need to have Python certificates installed
      - Go to your Python installation in Applications and click on "Install Certificates command"



Array Creation

- Use urllib.request.urlretrieve with website and file name
  - Remember: A file will be created, but the directory needs to exist

```
urllib.request.urlretrieve(
    url = "https://ndownloader.figshare.com/files/12565616",
    filename = "avg-monthly-precip.txt"
)
```

  - This is a text file, with one numerical value per line
  - Then create the numpy array using

```
avgmp = np.loadtxt(fname = 'avg-monthly-precip.txt')
print(avgmp)
```

Array Creation

- Example: Get an account at openweathermap.org/appid
- Install requests and import json
  - Use the openweathermap.org api to request data on a certain town
- Result is send as a JSON dictionary

Array Creation

```
import numpy as np
import requests
import json

mumbai=json.loads(requests.get('http://api.openweathermap.org/data/
2.5/weather?
q=mumbai,india&APPID=4561e0cd15ec2ee307bdcfe19ec22ab9').text)
vasai = json.loads(requests.get('http://api.openweathermap.org/data/
2.5/weather?
q=vasai,india&APPID=4561e0cd15ec2ee307bdcfe19ec22ab9').text)
navi_mumbai = json.loads(requests.get('http://api.openweathermap.org
data/2.5/weather?
q=navi%20mumbai,india&APPID=4561e0cd15ec2ee307bdcfe19ec22ab9').text)
```

Array Creation

- Can use np.genfromtext
  - Very powerful and complicated function with many different options

Array Creation

- Example

```
converters = {5: lambda x: int(0 if 'Iris-setosa' else 1 if 'Iris-
virginica' else 2) }
my_array = np.genfromtxt('../Classes2/Iris.csv',
                         usecols=(1,2,3,4,5),
                         dtype =[float, float, float, float, float]
                         delimiter = ',',
                         converters = converters,
                         skip_header=1)
```

- Need a source (the iris file)
- Can specify the columns we need
- Give the dtype U20-unicode string, S20-byte string
- Delimiter
- Skipheader, skipfooter
- converters to deal with encoding

Array Creation

- This is an array of 150 tuples
- Use comprehension to convert to a two-dimensional array

```
m = np.array( [ [row[0], row[1], row[2], row[3], row[4]]
for row in my_array ])
```

NumPy Array Attributes

- The number of dimensions: ndim
- The values of the dimensions as a tuple: shape
- The size (number of elements)

```
>>> tensor
array([[[2.11208424, 2.01510638, 2.03126777, 1.89670846],
        [1.94359036, 2.02299445, 2.08515919, 2.05402626],
        [1.8853457 , 2.01236192, 2.07019962, 1.93713157]],

       [[1.84275427, 1.99537922, 1.96060154, 1.90020305],
        [2.00270166, 2.11286224, 2.03144254, 2.06924855],
        [1.95375653, 2.0612986 , 1.82571628, 1.86067971]]])
>>> tensor.ndim
3
>>> tensor.shape
(2, 3, 4)
>>> tensor.size
24
```

NumPy Array Attributes

- The data type: dtype
  - can be bool, int, int64, uint, uint64, float, float64, complex ...
- The size of a single element in bytes: itemsize
- The size of the total array: nbytes

NumPy Array Indexing

- Single elements
  - Use the bracket notation [ ]
    - Single array: Same as in standard python

```
>>> vector = np.random.normal(10,1,(5))
>>> print(vector)
[10.25056641 11.37079651 10.44719557 10.54447875 10.43634562]
>>> vector[4]
10.436345621654919
>>> vector[-2]
10.544478746079845
```

NumPy Arrays Indexing

- Matrix and tensor elements: Use a single bracket and a comma separated tuple

```
>>> tensor
array([[[2.11208424, 2.01510638, 2.03126777, 1.89670846],
        [1.94359036, 2.02299445, 2.08515919, 2.05402626],
        [1.8853457 , 2.01236192, 2.07019962, 1.93713157]],

       [[1.84275427, 1.99537922, 1.96060154, 1.90020305],
        [2.00270166, 2.11286224, 2.03144254, 2.06924855],
        [1.95375653, 2.0612986 , 1.82571628, 1.86067971]]])
>>> tensor[0,0,1]
2.015106376191313
```

NumPy Arrays Indexing

- Multiple bracket notation
  - We can also use the Python indexing of multi-dimensional lists using several brackets

```
>>> tensor[0][1][2]
2.085159191502853
```

- It is more writing and more error prone than the single bracket version

NumPy Arrays Indexing

- We can also define slices

```
>>> vector = np.random.normal(10,1,(3))
>>> vector
array([10.61948855,  7.99635252,  9.05538706])
>>> vector[1:3]
array([7.99635252, 9.05538706])
```

NumPy Arrays Indexing

- In Python, slices are new lists
- In NumPy, slices are **not** copies
  - Changing a slice changes the original

NumPy Arrays Indexing

- Example:
  - Create an array

```
>>> vector = np.random.normal(10,1,(3))
>>> vector
array([10.61948855,  7.99635252,  9.05538706])
```

  - Define a slice

```
>>> x = vector[1:3]
```

NumPy Arrays Indexing

- Example (cont.)
  - Change the first element in the slice

```
>>> x[0] = 5.0
```

  - Verify that the change has happened

```
>>> x
array([5.          , 9.05538706])
```

  - But the original has also changed:

```
>>> vector
array([10.61948855,  5.          ,  9.05538706])
```

NumPy Arrays Indexing

- Slicing does **not** makes copies
  - This is done in order to be efficient
    - Numerical calculations with a large amount of data get slowed down by unnecessary copies

NumPy Arrays Indexing

- If we want a copy, we need to make one with the copy method
- Example:
- Make an array

```
>>> vector = np.random.randint(0,10,5)
>>> vector
array([0, 9, 5, 7, 8])
```

- Make a copy of the array

```
>>> my_vector_copy = vector.copy()
```

NumPy Arrays Indexing

- Example (continued)
  - Change the middle elements in the copy

```
>>> my_vector_copy[1:-2]=100
```

  - Check the change

```
>>> my_vector_copy
array([  0, 100, 100,   7,   8])
```

  - Check the original

```
>>> vector
array([0, 9, 5, 7, 8])
```

  - No change!

NumPy Arrays Indexing

- Multi-dimensional slicing
  - Combines the slicing operation for each dimension

```
>>> slice = tensor[1:, :2, :1]
>>> slice
array([[[1.84275427],
        [2.00270166]]])
```

NumPy Arrays
Conditional Selection

- We can create an array of Boolean values using comparisons on the array

```
>>> array = np.random.randint(0,10,8)
>>> array
array([2, 4, 4, 0, 0, 4, 8, 4])
>>> bool_array = array > 5
>>> bool_array
array([False, False, False, False, False,
       False,  True, False])
```

NumPy Arrays
Conditional Selection

- We can then use the Boolean array to create a selection from the original array

```
>>> selection=array[bool_array]
>>> selection
array([8])
```

- The new array only has one element!

Selftest

- Can you do this in one step?
  - Create a random array of 10 elements between 0 and 10
  - Then select the ones larger than 5

Selftest Solution

- Solution:
  - Looks a bit cryptic
    - First, we create an array

```
>>> arr = np.random.randint(0,10,10)
>>> arr
array([3, 2, 7, 8, 7, 2, 1, 0, 4, 8])
```

  - Then we select in a single step

```
>>> sel = arr[arr>5]
>>> sel
array([7, 8, 7, 8])
```

NumPy Arrays
Conditional Selection

- Let's try this out with a matrix
    - We create a vector, then use **reshape** to make the array into a vector
        - Recall: the number of elements needs to be the same

```
>>> mat = np.arange(1,13).reshape(3,4)
>>> mat
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

NumPy Arrays
Conditional Selection

- Now let's select:

```
>>> mat1 = mat[mat>6]
>>> mat1
array([ 7,  8,  9, 10, 11, 12])
```

- This is no longer a matrix, which makes sense

Slicing

- Photo Manipulation
  - Need to install imageio and matplotlib

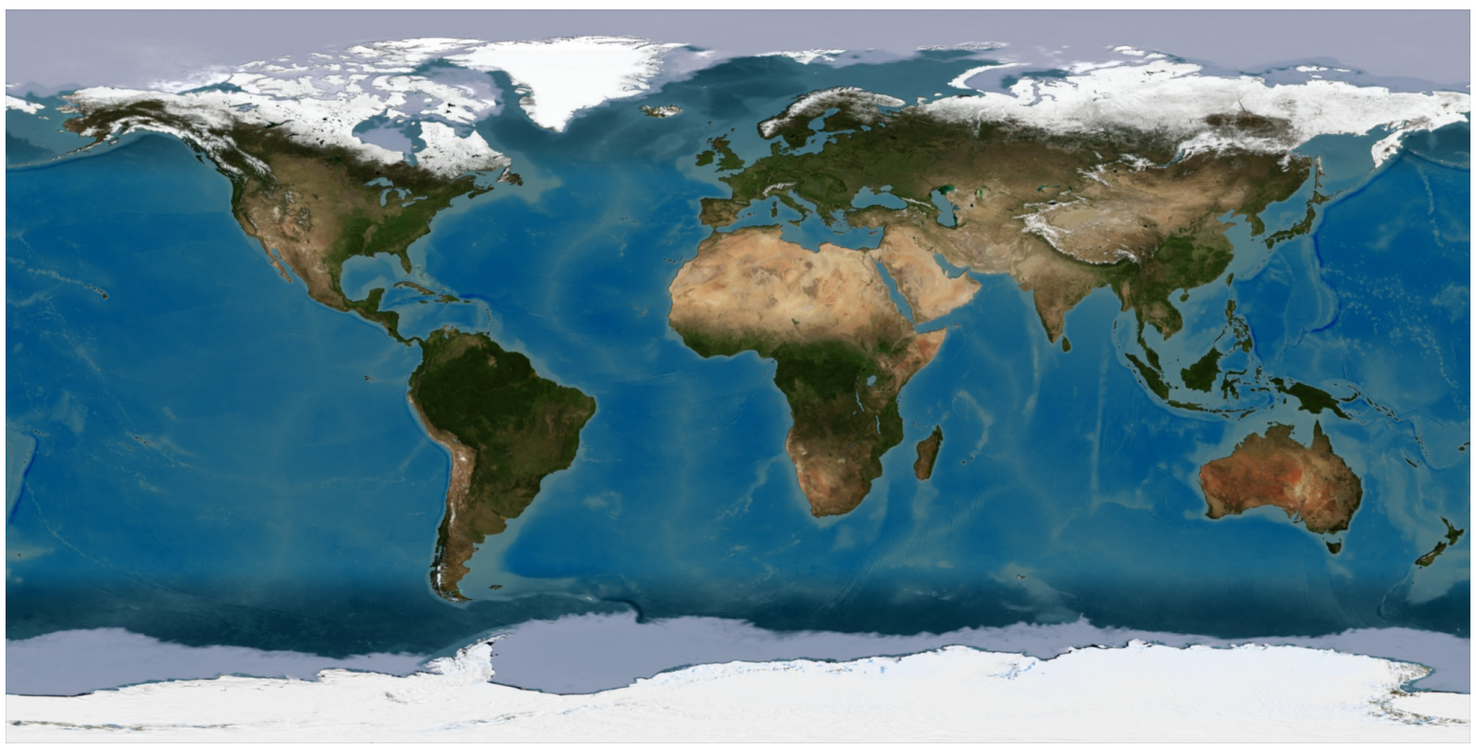
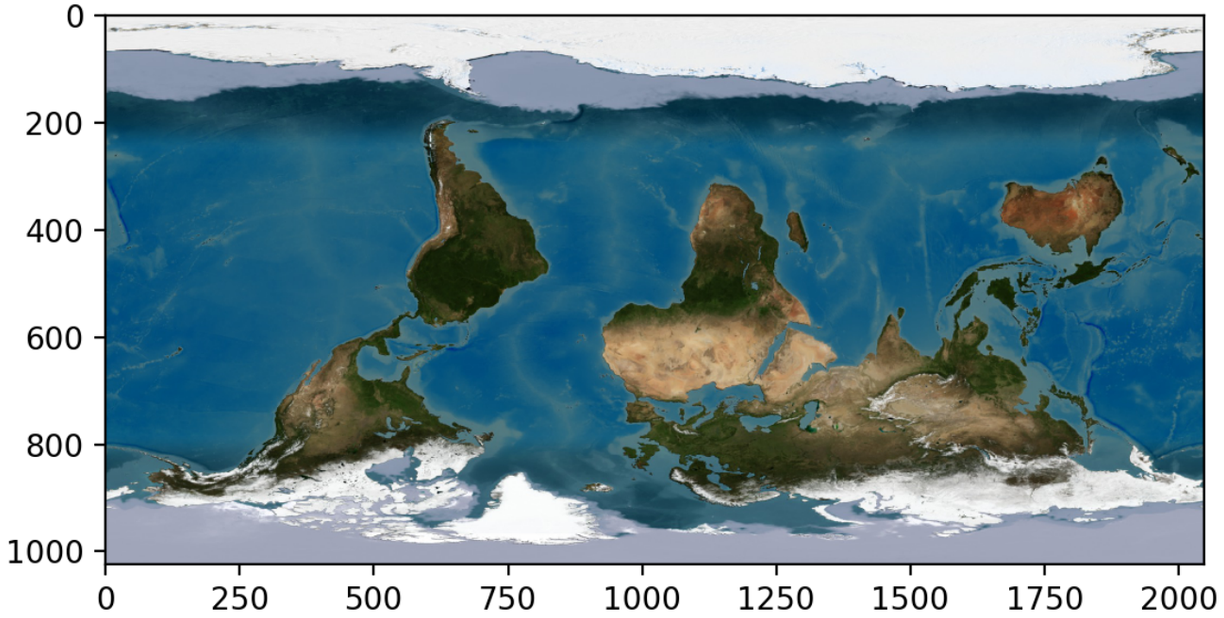
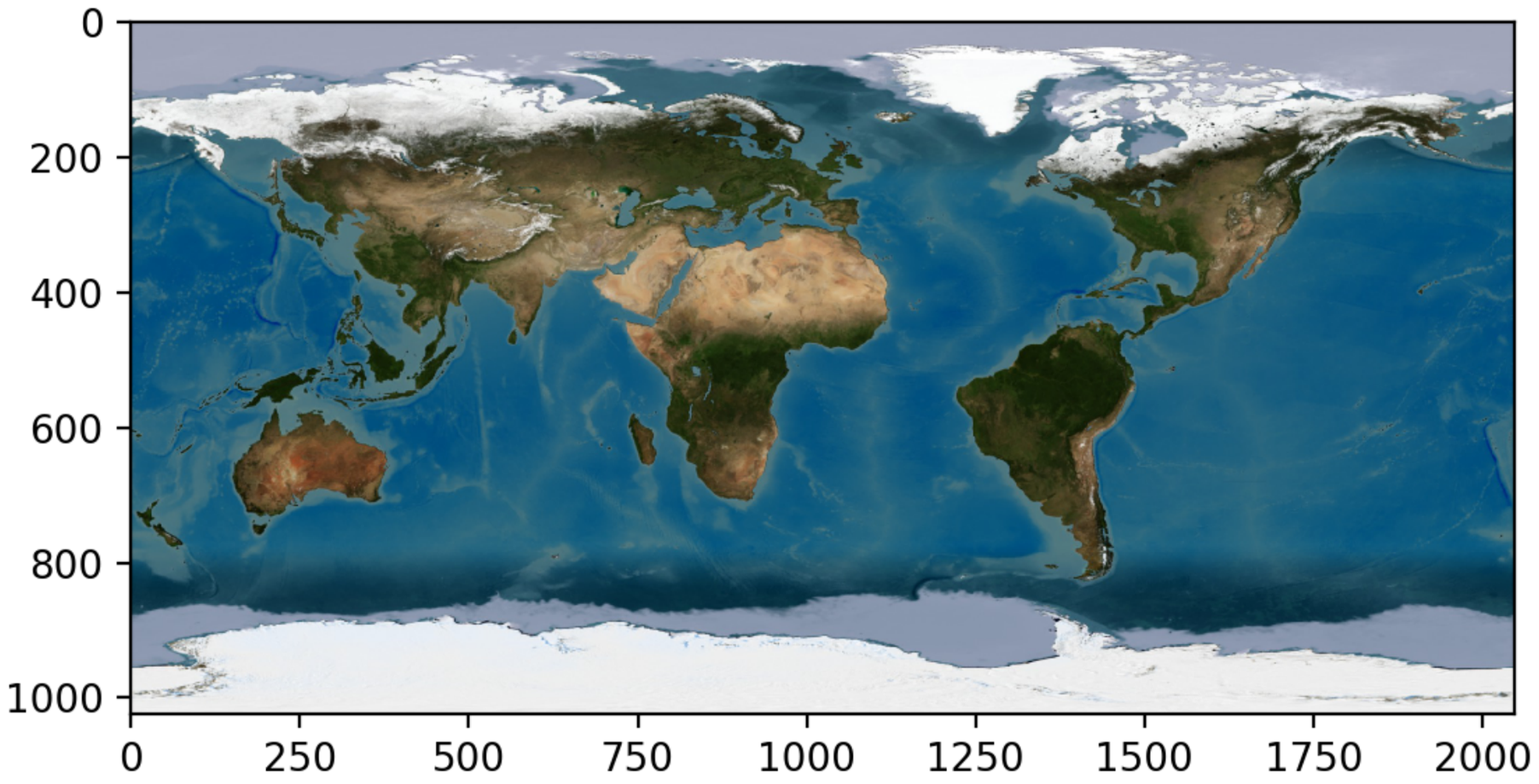
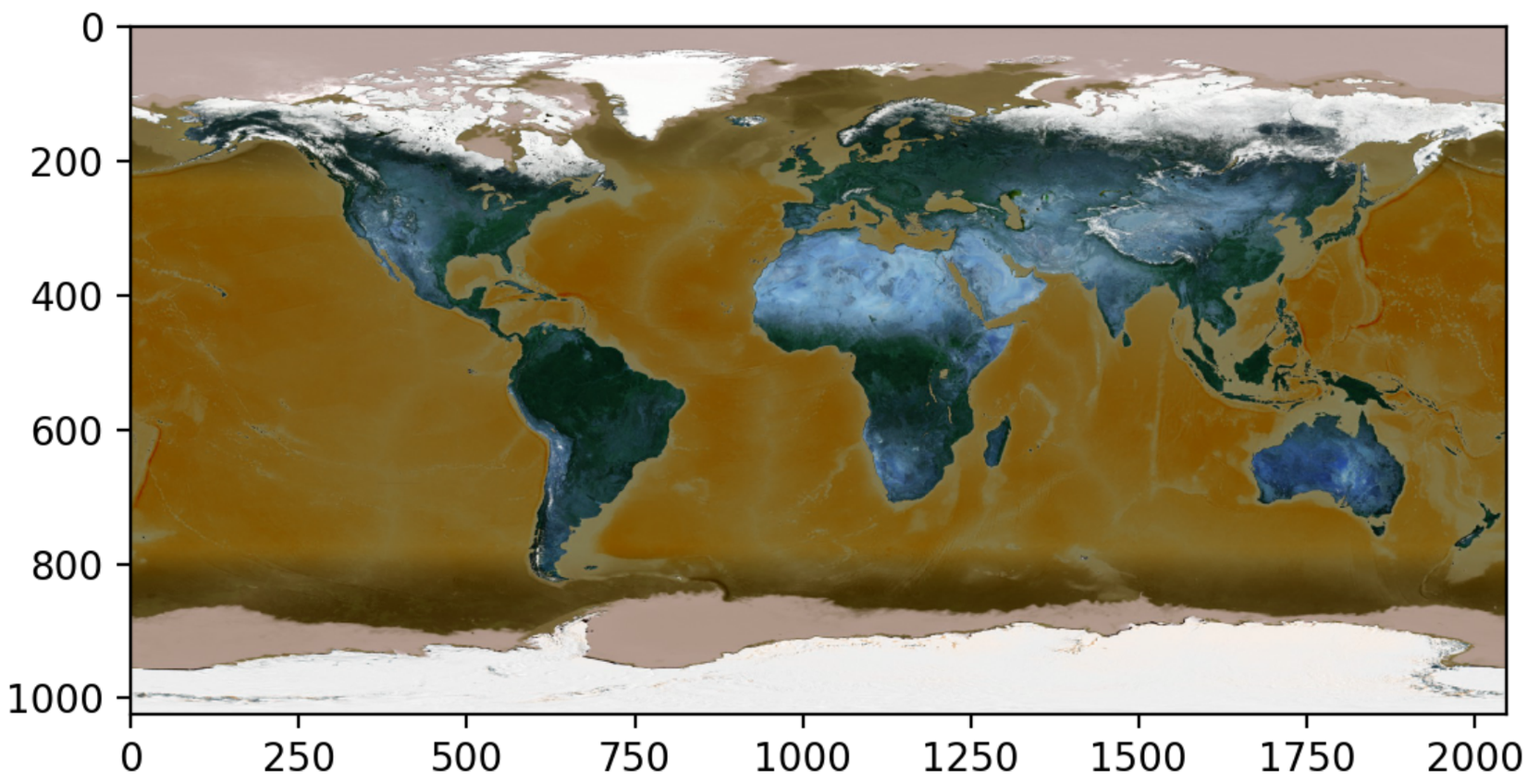
```
import imageio
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

- Get a photo as a 3-dimensional array
  -

```
im = mpimg.imread('earth.jpg')
print(im.shape)
```

Slicing

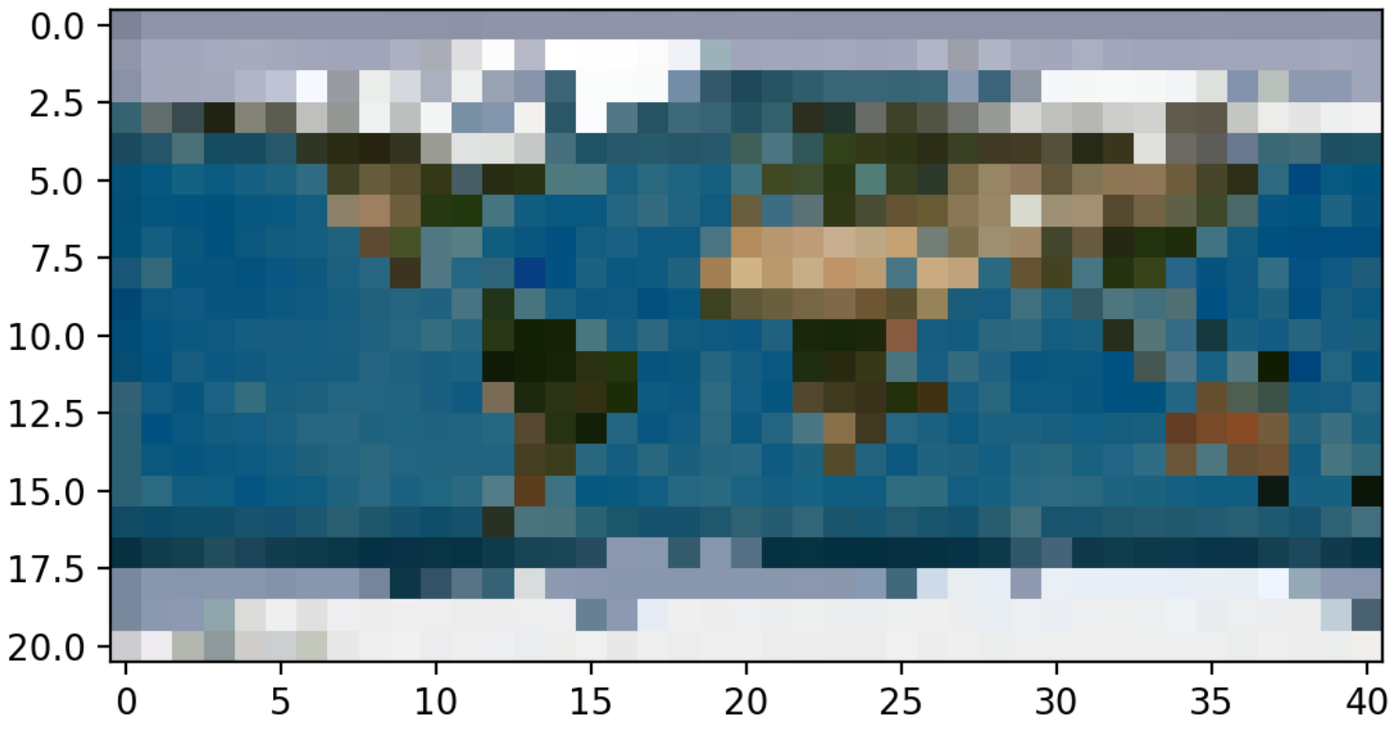
- Display the photo
- ```
  plt.imshow(im)
  plt.show()
  ```


  ```

Slicing

- Swap first coordinate

```
plt.imshow(im[::-1,])
plt.show()
```



Slicing

- Swap second coordinate
-

```
plt.imshow(im[:,::-1,])
plt.show()
```



Slicing

- Swap third coordinate (color coordinate)
  - ```
    plt.imshow(im[:,:,::-1])
    plt.show()
    ```


    ```

Slicing

- Take every 50th line and column

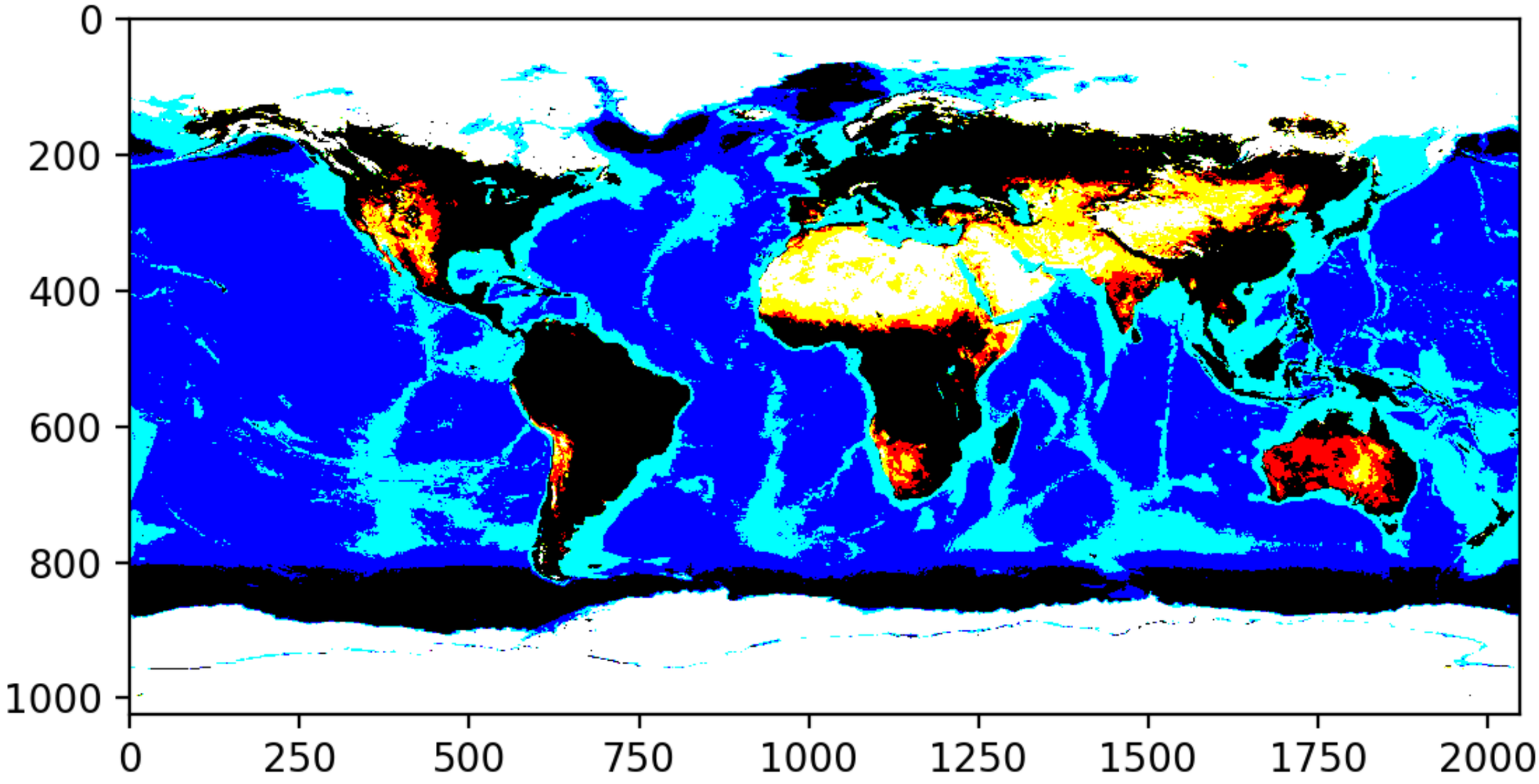
```
plt.imshow(im[::50, ::50,])
plt.show()
```



Slicing

- We can also apply functions
  - np.where allows us to replace values
    - `image = np.where(im>100, 255, 0)`
      - Where-ever the value of the image is less than 100, replace it with 0
      - Otherwise, replace it with 255

Slicing

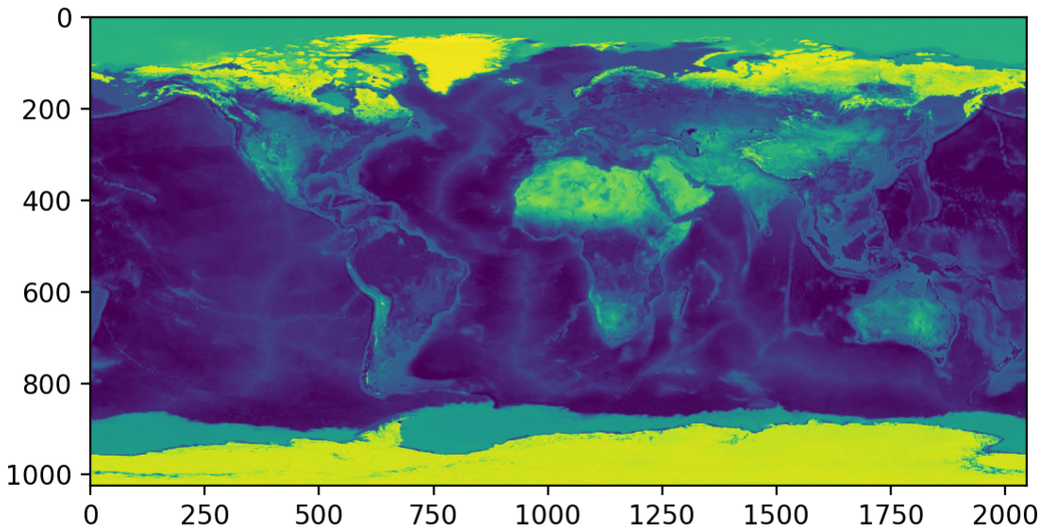


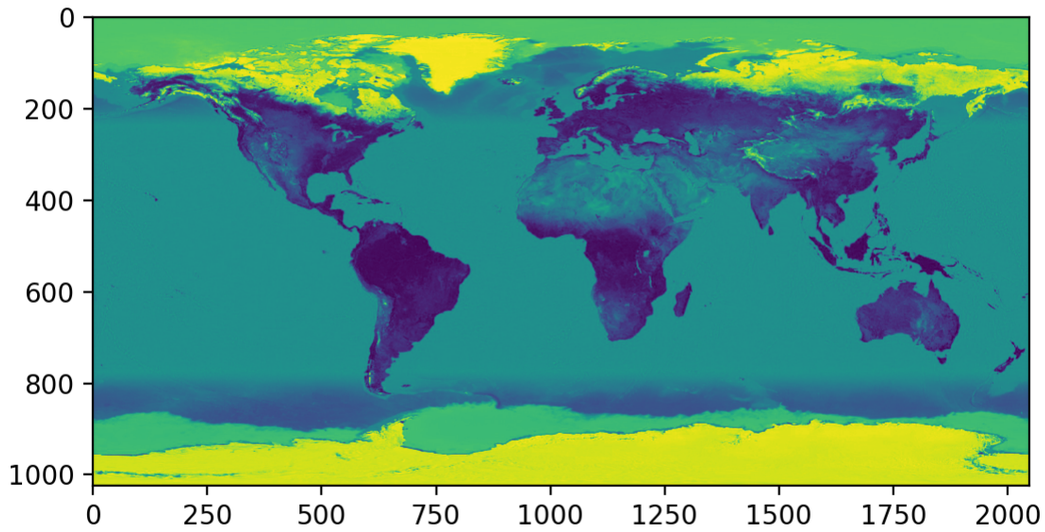
Slicing

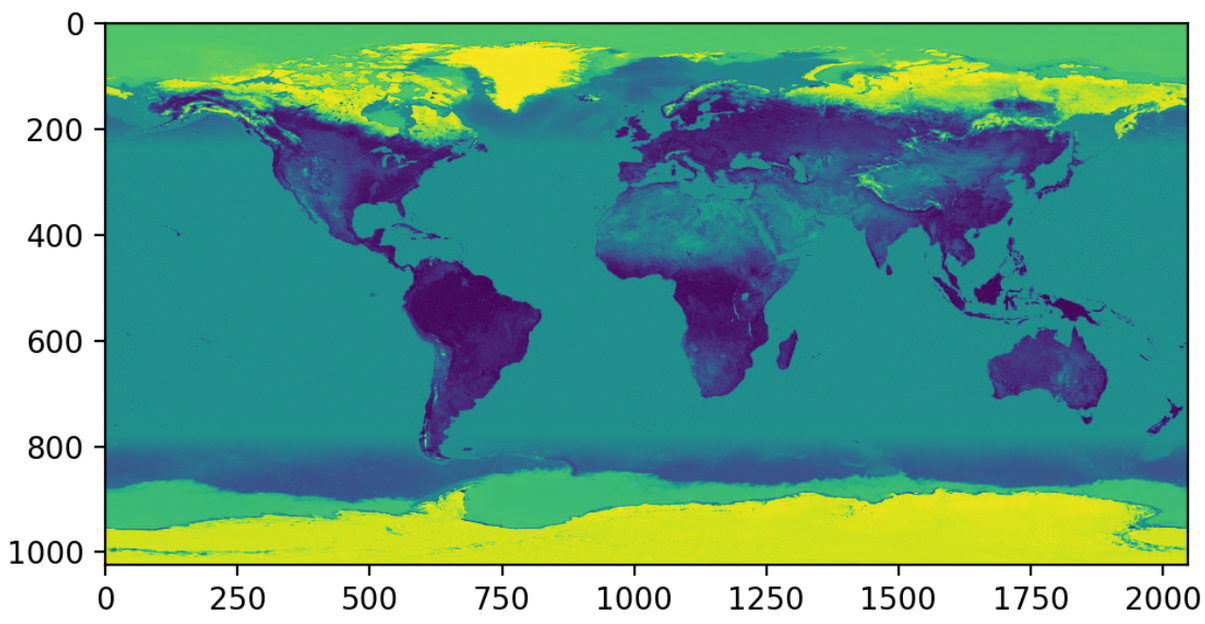
- Can use a sub-image

```
plt.imshow(im[:,:,0])
plt.show()

plt.imshow(im[:,:,1])
plt.show()

plt.imshow(im[:,:,2])
plt.show()
```

Slicing


0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750
2000

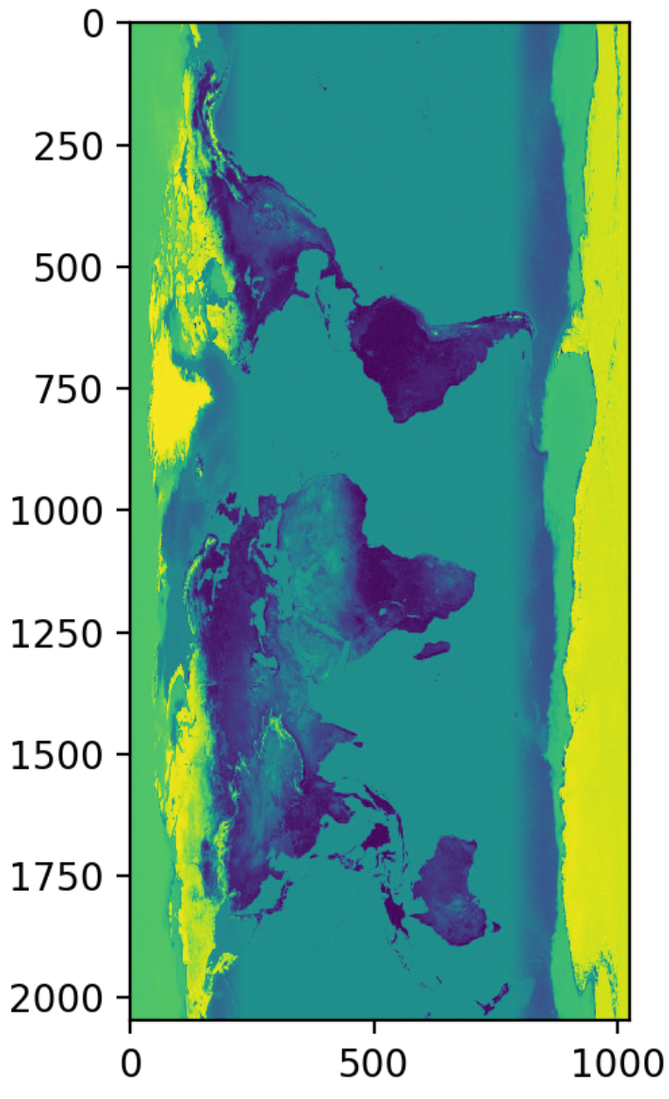

0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750
2000


0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750
2000

Slicing

- Can use the transpose

```
plt.imshow(im[:,:,2].T)
plt.show()
```



Slicing

- Or just concentrate on the essential

```
plt.imshow(im[250:500,1350:1600, : ])
```

Slicina