Python Tutorial Series 2019
Data Analysis & Visualization 2
SciPy, Image display and Manipulation

Ph.D. Bruno C. Quint - bquint@gemini.edu
SUSD - GEMINI South, La Serena, Chile

Zen of Python

```
In [1]: import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Previously...

Basics I

What is Python?

What will you need?

Python as a terminal

Python as a script

Types of variables

Basics II

Code Flow

Functions

Classes

Methods

Attributes

Sub-Classes

Basics III

(Re)using code

Modules

Packages

Python2 to Python3

External packages

Virtual Environments

Conda and Anaconda

Data Analysis and Visualization

DataViz 1	DataViz 2
iPython / Jupyter	NumPy Tricks
NumPy	Image Display
MatPlotLib	Image Analysis
2D Plots	Image Manipulation

NumPy Tricks

Broadcasting

```
x = [[1, 1, 1],
     [2, 2, 2]]

x = np.array(x)

print(x.shape)
```

```
(2, 3)
```

NumPy Tricks

Broadcasting

```
x = [[1, 1, 1],
     [2, 2, 2]]

x = np.array(x)

print(x.shape)
```

```
(2, 3)
```

```
y1 = 3

z1 = x * y1
print(z1)
```

```
[[3 3 3]
 [6 6 6]]
```

NumPy Tricks

Broadcasting

```
x = [[1, 1, 1],
     [2, 2, 2]]

x = np.array(x)

print(x.shape)
```

```
(2, 3)
```

```
y2 = [3, 4, 5]

y2 = np.array(y2)

print(y2.shape)
```

```
(3,)
```

```
y1 = 3

z1 = x * y1
print(z1)
```

```
[[3 3 3]
 [6 6 6]]
```

NumPy Tricks

Broadcasting

```
x = [[1, 1, 1],
     [2, 2, 2]]

x = np.array(x)

print(x.shape)
```

```
(2, 3)
```

```
y2 = [3, 4, 5]

y2 = np.array(y2)

print(y2.shape)
```

```
(3,)
```

```
y1 = 3

z1 = x * y1
print(z1)
```

```
[[3 3 3]
 [6 6 6]]
```

```
z2 = x * y2
print(z2)
```

```
[[3  4  5]
 [6  8 10]]
```

NumPy Tricks

Broadcasting

```
x = [[1, 1, 1],
     [2, 2, 2]]

x = np.array(x)

print(x.shape)
```

```
(2, 3)
```

```
y2 = [3, 4, 5]

y2 = np.array(y2)

print(y2.shape)
```

```
(3,)
```

```
y3 = [6, 7]

y3 = np.array(y3)
y3 = y3[:, np.newaxis]

print(z3.shape)
```

```
(2,)
```

```
y1 = 3

z1 = x * y1
print(z1)
```

```
[[3 3 3]
 [6 6 6]]
```

```
z2 = x * y2
print(z2)
```

```
[[3  4  5]
 [6  8 10]]
```

NumPy Tricks

Broadcasting

```
x = [[1, 1, 1],
     [2, 2, 2]]

x = np.array(x)

print(x.shape)
```

```
(2, 3)
```

```
y2 = [3, 4, 5]

y2 = np.array(y2)

print(y2.shape)
```

```
(3,)
```

```
y3 = [6, 7]

y3 = np.array(y3)
y3 = y3[:, np.newaxis]

print(z3.shape)
```

```
(2,)
```

```
y1 = 3

z1 = x * y1
print(z1)
```

```
[[3 3 3]
 [6 6 6]]
```

```
z2 = x * y2
print(z2)
```

```
[[3  4  5]
 [6  8 10]]
```

```
z3 = x * y3
print(z3)
```

```
ValueError: operands could not be broadcast together with shapes (2,3) (2,)
```

NumPy Tricks

Broadcasting

```
x = [[1, 1, 1],
     [2, 2, 2]]

x = np.array(x)

print(x.shape)
```

```
(2, 3)
```

```
y2 = [3, 4, 5]

y2 = np.array(y2)

print(y2.shape)
```

```
(3,)
```

```
y3 = [6, 7]

y3 = np.array(y3)
y3 = y3[:, np.newaxis]

print(z3.shape)
```

```
(2, 1)
```

```
y1 = 3

z1 = x * y1
print(z1)
```

```
[[3 3 3]
 [6 6 6]]
```

```
z2 = x * y2
print(z2)
```

```
[[3  4  5]
 [6  8 10]]
```

```
z3 = x * y3
print(z3)
```

```
[[ 6  6  6]
 [14 14 14]]
```

NumPy Tricks

Meshgrid

Let us build two matrices with X and Y coordinates to be used later to manipulate images

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

NumPy Tricks

Meshgrid

First, we create an array with the column indexes (our X)

```
n = 5
col_indexes = np.arange(n)
print(col_indexes)
```

```
[0 1 2 3 4]
```

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

NumPy Tricks

Meshgrid

Then we create an array with the row indexes (our Y)

```
n = 5
col_indexes = np.arange(n)
print(col_indexes)
```

```
[0 1 2 3 4]
```

```
m = 10
row_indexes = np.arange(m)
print(row_indexes)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

NumPy Tricks

Meshgrid

We use meshgrid to create the grids with the indexes

```
n = 5
col_indexes = np.arange(n)
print(col_indexes)
```

```
[0 1 2 3 4]
```

```
m = 10
row_indexes = np.arange(m)
print(row_indexes)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```
cols, rows =
    np.meshgrid(
        col_indexes,
        row_indexes
    )
```

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

NumPy Tricks

```
n = 5
col_indexes = np.arange(n)
print(col_indexes)
```

```
[0 1 2 3 4]
```

```
m = 10
row_indexes = np.arange(m)
print(row_indexes)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```
cols, rows =
    np.meshgrid(
        col_indexes,
        row_indexes
    )
```

Meshgrid

You can print and check that the output matches our expected matrices

cols

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

rows

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

NumPy Tricks

```
n = 5
col_indexes = np.arange(n)
print(col_indexes)
```

```
[0 1 2 3 4]
```

```
m = 10
row_indexes = np.arange(m)
print(row_indexes)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```
cols, rows =
    np.meshgrid(
        col_indexes,
        row_indexes
    )
```

Meshgrid

Rename it and we are done!

```
X = cols
Y = rows
```

X

cols

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

Y

rows

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

NumPy Tricks

mgrid

mgrid allows even quicker creation of indexes grids

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

NumPy Tricks

mgrid

Remember that our output grid has 5 rows (n) and 10 columns (m)

```
n = 5
m = 10
```

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

NumPy Tricks

mgrid

The syntax to use mgrid is a bit different:
It uses square brackets instead

```
n = 5
m = 10
```

```
rows2, cols2 = np.mgrid[0:m, 0:n]
```

cols2

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

rows2

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

NumPy Tricks

mgrid

```
n = 5
m = 10
```

```
rows2, cols2 = np.mgrid[0:m, 0:n]
```

Note that the order is different from `numpy.meshgrid`

cols2

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
```

rows2

```
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [7 7 7 7 7]
 [8 8 8 8 8]
 [9 9 9 9 9]]
```

Image Display

My First Image

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

Import libraries

Image Display

My First Image

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

np.random.seed(0)
data = np.random.uniform(size=(4, 5))
```

Create random 2D array

Image Display

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

np.random.seed(0)
data = np.random.uniform(size=(4, 5))

fig, ax = plt.subplots(num='my_fig')
ax.imshow(data)
plt.show()
```

Create figure, axis and display

My First Image

Image Display

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

np.random.seed(0)
data = np.random.uniform(size=(4, 5))

fig, ax = plt.subplots(num='my_fig')
ax.imshow(data)
plt.show()
```

Create figure, axis and display

My First Image

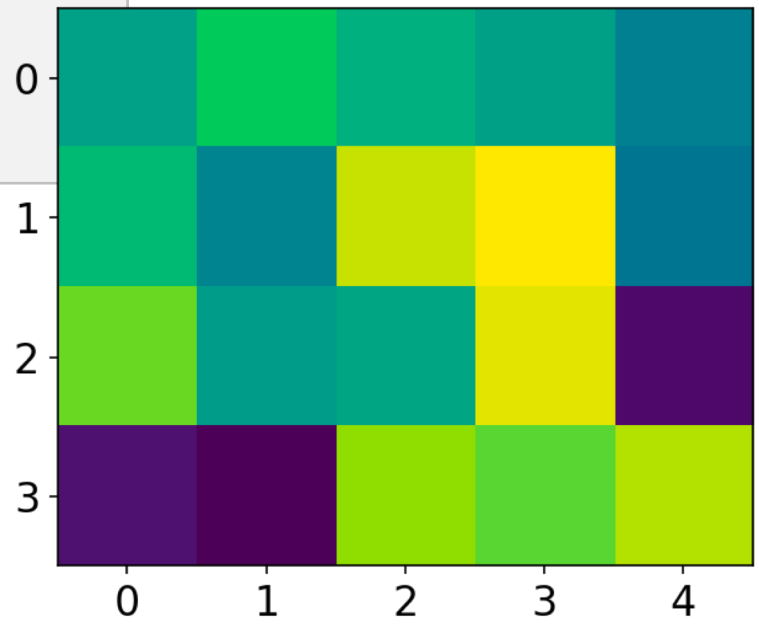


Image Display

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

np.random.seed(0)
data = np.random.uniform(size=(4, 5))

fig, ax = plt.subplots(num='my_fig')
ax.imshow(data)
plt.show()
```

Origing lies on the top-left edge of the figure

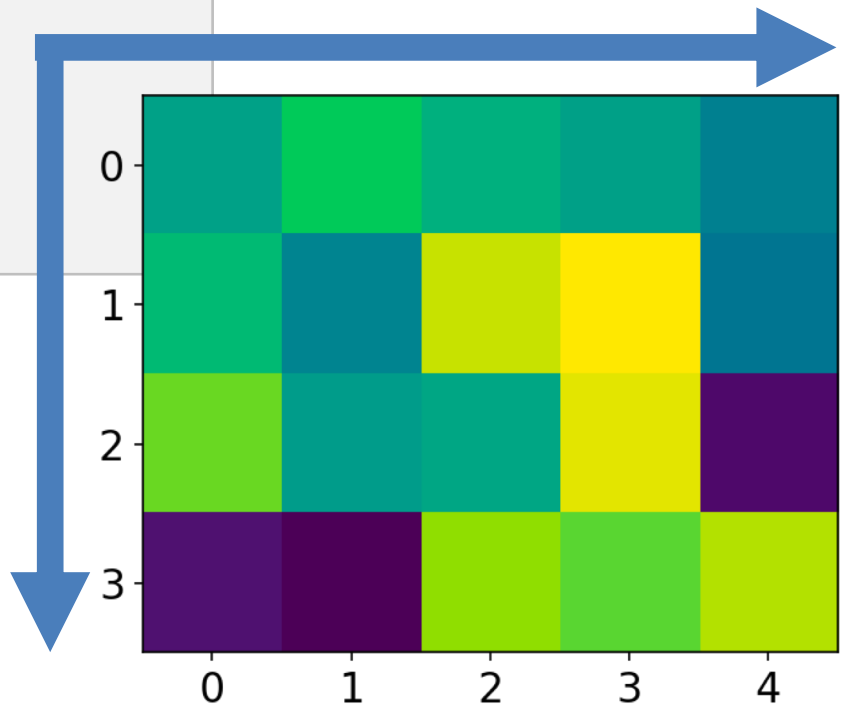


Image Display

Origin

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

np.random.seed(0)
data = np.random.uniform(size=(4, 5))

fig, ax = plt.subplots(num='lower origin')
ax.imshow(data, origin="lower")
plt.show()
```

Bring the origin to lower-left edge of the figure

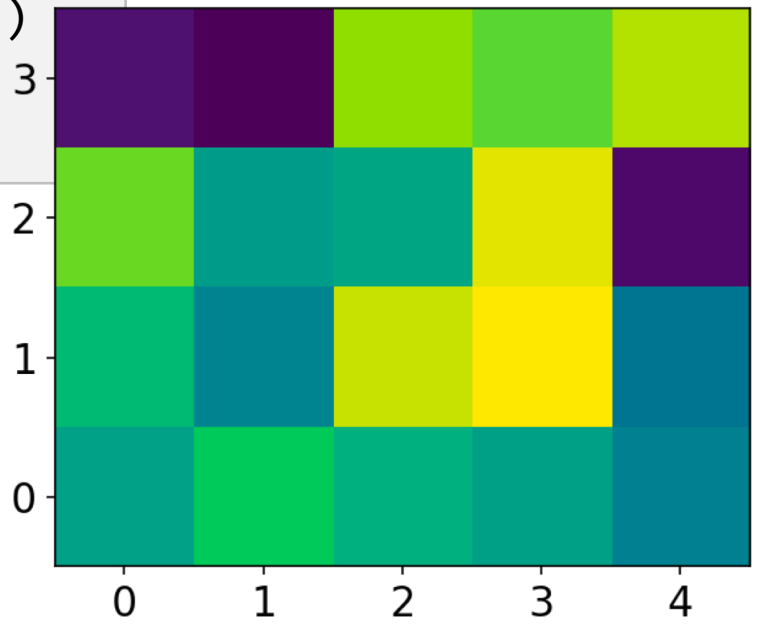


Image Display

Origin

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

np.random.seed(0)
data = np.random.uniform(size=(4, 5))

fig, ax = plt.subplots(num='lower origin')
ax.imshow(data, origin="lower")
plt.show()
```

Bring the origin to lower-left edge of the figure

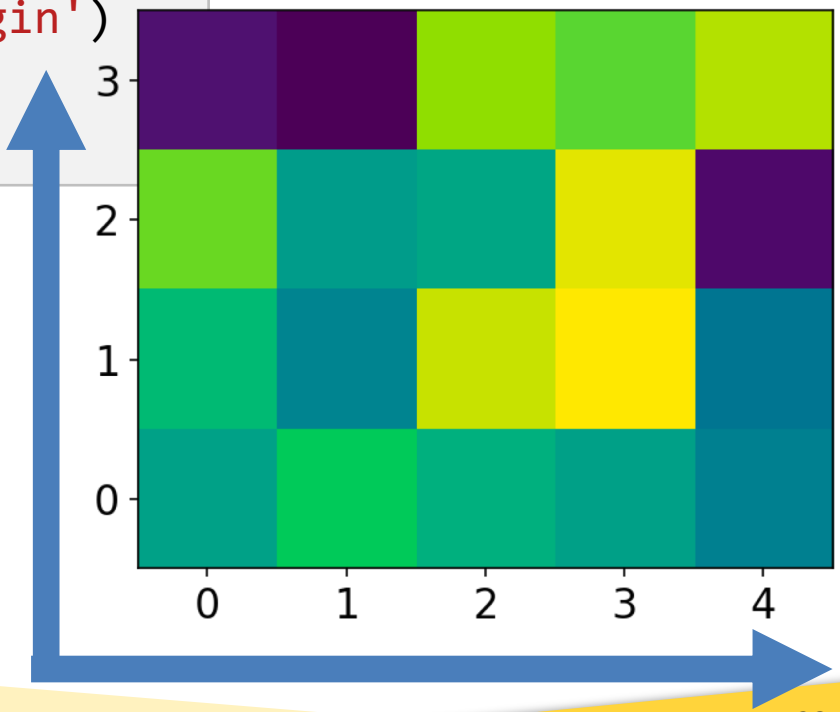


Image Display

Colormaps

List all colormaps available

```
plt.colormaps()
```

```
['Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r',
'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Greys', 'Greys_r',
'OrRd', 'OrRd_r', 'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1',
'Pastel1_r', 'Pastel2', 'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr',
'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu',
'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2',
'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu',
'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn',
'autumn_r', 'binary', 'binary_r', 'bone', 'bone_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis',
'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copper_r', 'cubehelix',
'cubehelix_r', 'flag', 'flag_r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r',
'gist_heat', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r',
'gist_stern', 'gist_stern_r', 'gist_yarg', 'gist_yarg_r', 'gnuplot', 'gnuplot2', 'gnuplot2_r',
'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r', 'hsv', 'hsv_r', 'inferno', 'inferno_r', 'jet', 'jet_r',
'magma', 'magma_r', 'nipy_spectral', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r',
'plasma', 'plasma_r', 'prism', 'prism_r', 'rainbow', 'rainbow_r', 'seismic', 'seismic_r', 'spring',
'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r', 'tab20', 'tab20_r', 'tab20b', 'tab20b_r',
'tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'twilight', 'twilight_r', 'twilight_shifted',
'twilight_shifted_r', 'viridis', 'viridis_r', 'winter', 'winter_r']
```

Image Display

Colormaps

Create array to be displayed

```
n, m = 300, 1000
y, x = np.mgrid[0:n, 0:m]

x = x / x.max()
y = y / y.max()

z = x + y ** 3 * 0.5 * np.cos(2 * np.pi * 2 ** 5 * x)
```

Image Display

Colormaps

Display image with default colormap (Viridis)

```
n, m = 300, 1000
y, x = np.mgrid[0:n, 0:m]

x = x / x.max()
y = y / y.max()

z = x + y ** 3 * 0.5 * np.cos(2 * np.pi * 2 ** 5 * x)
```

```
fig, ax = plt.subplots(num='default_colormap')
ax.imshow(z)
plt.show()
```

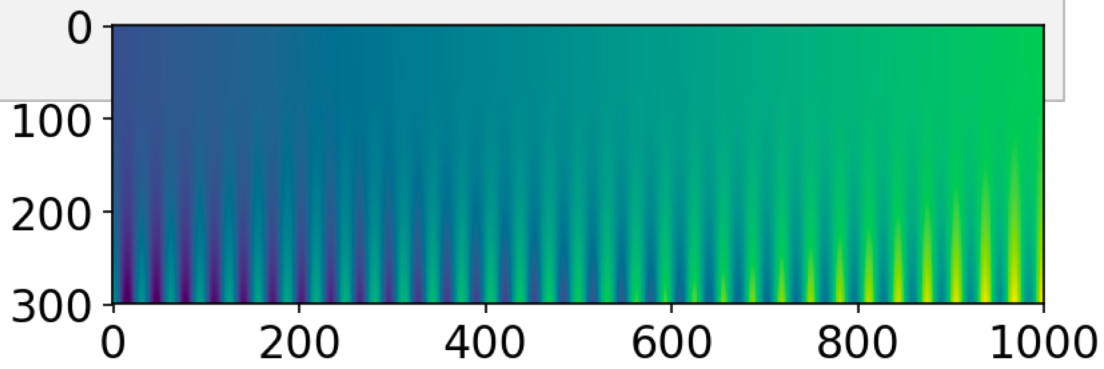


Image Display

Colormaps

Display image with grayscale colormap

```
n, m = 300, 1000
y, x = np.mgrid[0:n, 0:m]

x = x / x.max()
y = y / y.max()

z = x + y ** 3 * 0.5 * np.cos(2 * np.pi * 2 ** 5 * x)
```

```
fig, ax = plt.subplots(num='gray_colormap')
ax.imshow(z, cmap="gray")
plt.show()
```

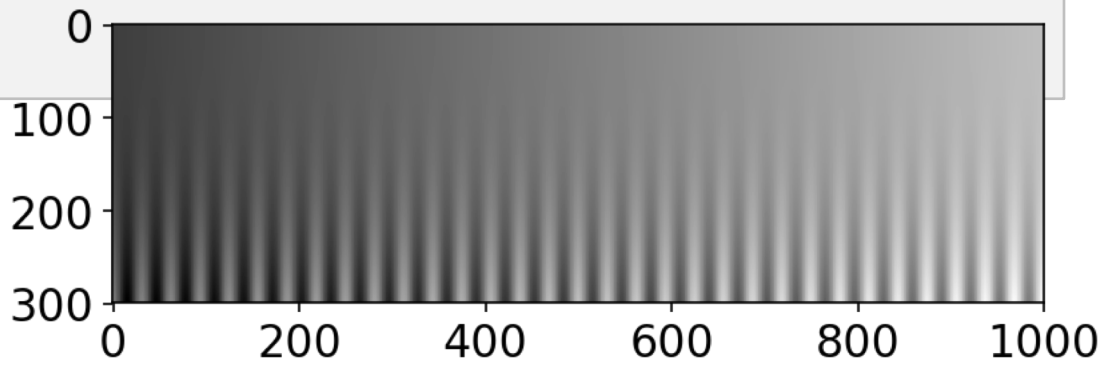


Image Display

Colormaps

Display image with jet (rainbow) colormap

```
n, m = 300, 1000
y, x = np.mgrid[0:n, 0:m]

x = x / x.max()
y = y / y.max()

z = x + y ** 3 * 0.5 * np.cos(2 * np.pi * 2 ** 5 * x)
```

```
fig, ax = plt.subplots(num='jet_colormap')
ax.imshow(z, cmap="jet")
plt.show()
```

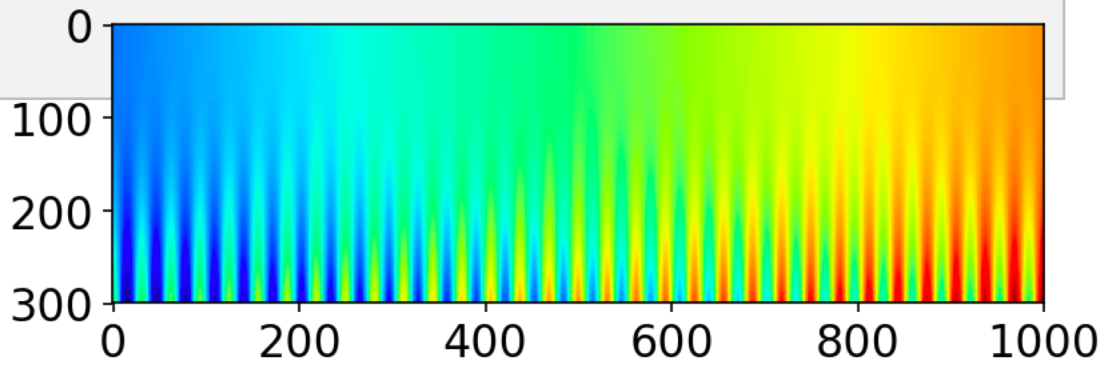


Image Display

Colormaps

This colormap hides features. Avoid using it!

```
n, m = 300, 1000
y, x = np.mgrid[0:n, 0:m]

x = x / x.max()
y = y / y.max()

z = x + y ** 3 * 0.5 * np.cos(20 * pi * x ** 5 * x)
```

```
fig, ax = plt.subplots(num='Bad colormap')
ax.imshow(z, cmap="jet")
plt.show()
```

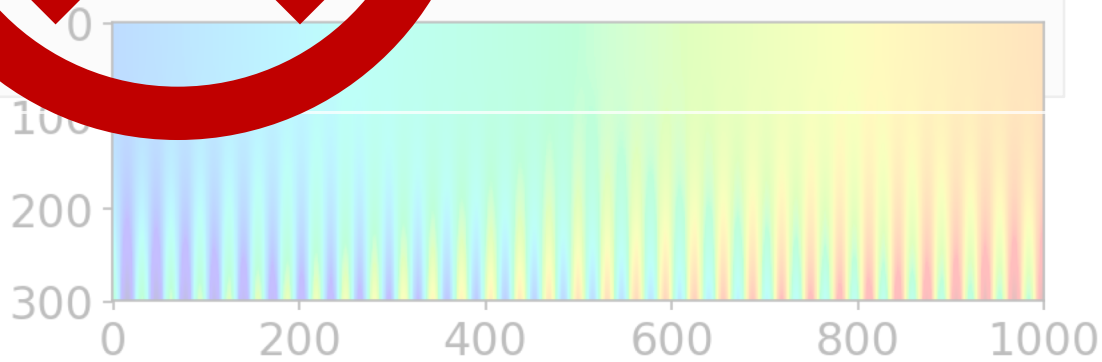


Image Display

Colormaps

Use the proper rainbow colormap instead

```
n, m = 300, 1000
y, x = np.mgrid[0:n, 0:m]

x = x / x.max()
y = y / y.max()

z = x + y ** 3 * 0.5 * np.cos(2 * np.pi * 2 ** 5 * x)
```

```
fig, ax = plt.subplots(num='rainbow_colormap')
ax.imshow(z, cmap="rainbow")
plt.show()
```

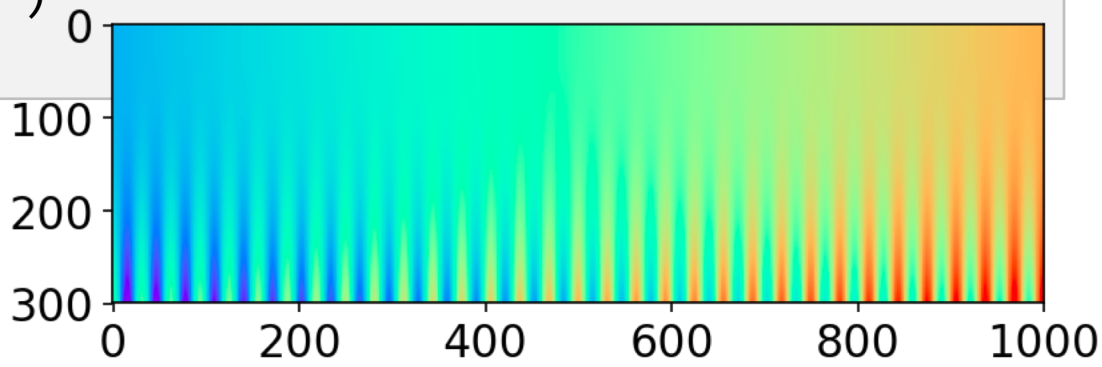


Image Display

Colormaps

Or the one called Spectral

```
n, m = 300, 1000
y, x = np.mgrid[0:n, 0:m]

x = x / x.max()
y = y / y.max()

z = x + y ** 3 * 0.5 * np.cos(2 * np.pi * 2 ** 5 * x)
```

```
fig, ax = plt.subplots(num='Spectral_colormap')
ax.imshow(z, cmap="Spectral")
plt.show()
```

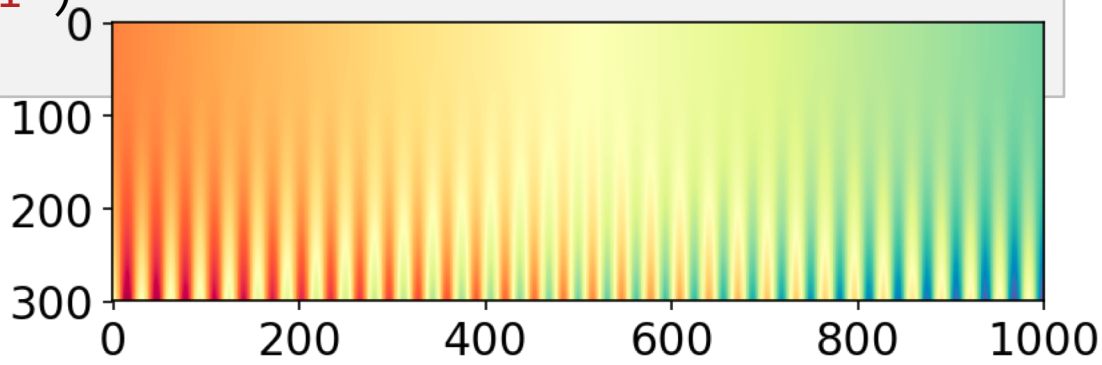


Image Display

Colormaps

Or maybe a qualitative colormap

```
n, m = 300, 1000
y, x = np.mgrid[0:n, 0:m]

x = x / x.max()
y = y / y.max()

z = x + y ** 3 * 0.5 * np.cos(2 * np.pi * 2 ** 5 * x)
```

```
fig, ax = plt.subplots(num='qualitative_colormap')
ax.imshow(z, cmap="tab20c")
plt.show()
```

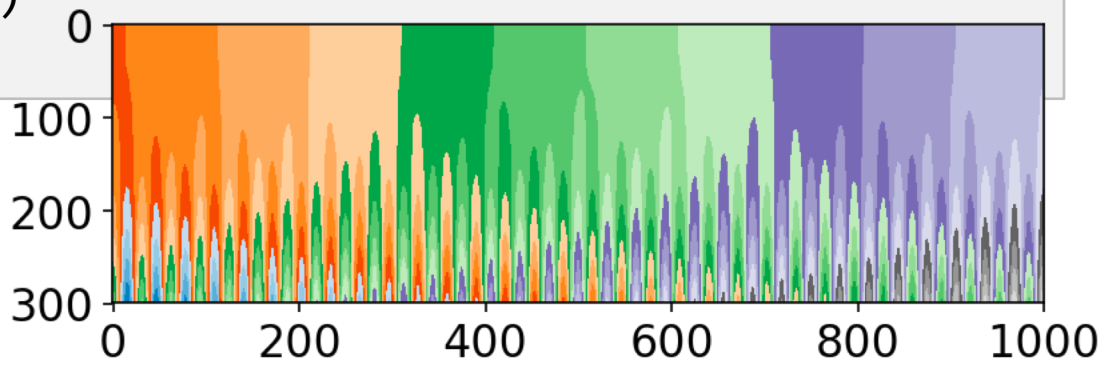


Image Display

Colormaps

Some Perceptually Uniform Colormaps

```
cmaps = ['viridis', 'plasma', 'inferno', 'magma', 'cividis']

fig, axes = plt.subplots(
    num='several_cmaps', nrows=5, sharex=True)

for i, colormap in enumerate(cmaps):
    axes[i].imshow(z, cmap=colormap)
    axes[i].set_yticks([])
    axes[i].set_xticks([])
    axes[i].set_ylabel(colormap)

plt.show()
```

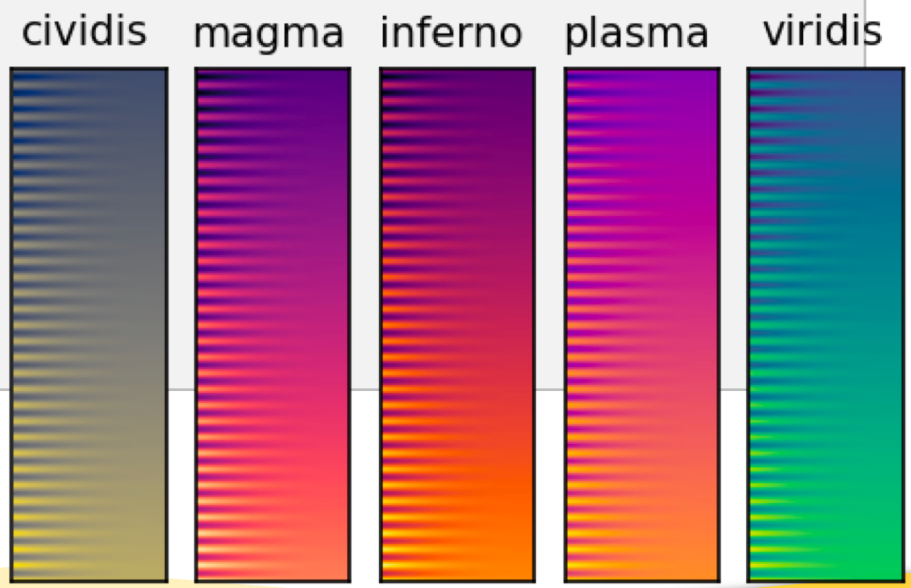


Image Display

Colorbar

```
fig, ax = plt.subplots(num='ex:colorbar_1')

im = ax.imshow(z)

colorbar = fig.colorbar(im)
colorbar.set_label('signal')

fig.tight_layout()
plt.show()
```

Image Display

Colorbar

```
fig, ax = plt.subplots(num='ex:colorbar_1')

im = ax.imshow(z)

colorbar = fig.colorbar(im)
colorbar.set_label('signal')

fig.tight_layout()
plt.show()
```

Image Display

Colorbar

```
fig, ax = plt.subplots(num='ex:colorbar_1')

im = ax.imshow(z)

colorbar = fig.colorbar(im)
colorbar.set_label('signal')

fig.tight_layout()
plt.show()
```

Image Display

Colorbar

```
fig, ax = plt.subplots(num='ex:colorbar_1')

im = ax.imshow(z)

colorbar = fig.colorbar(im)
colorbar.set_label('signal')

fig.tight_layout()
plt.show()
```

Image Display

Colorbar

```
fig, ax = plt.subplots(num='ex:colorbar_1')

im = ax.imshow(z)

colorbar = fig.colorbar(im)
colorbar.set_label('signal')

fig.tight_layout()
plt.show()
```

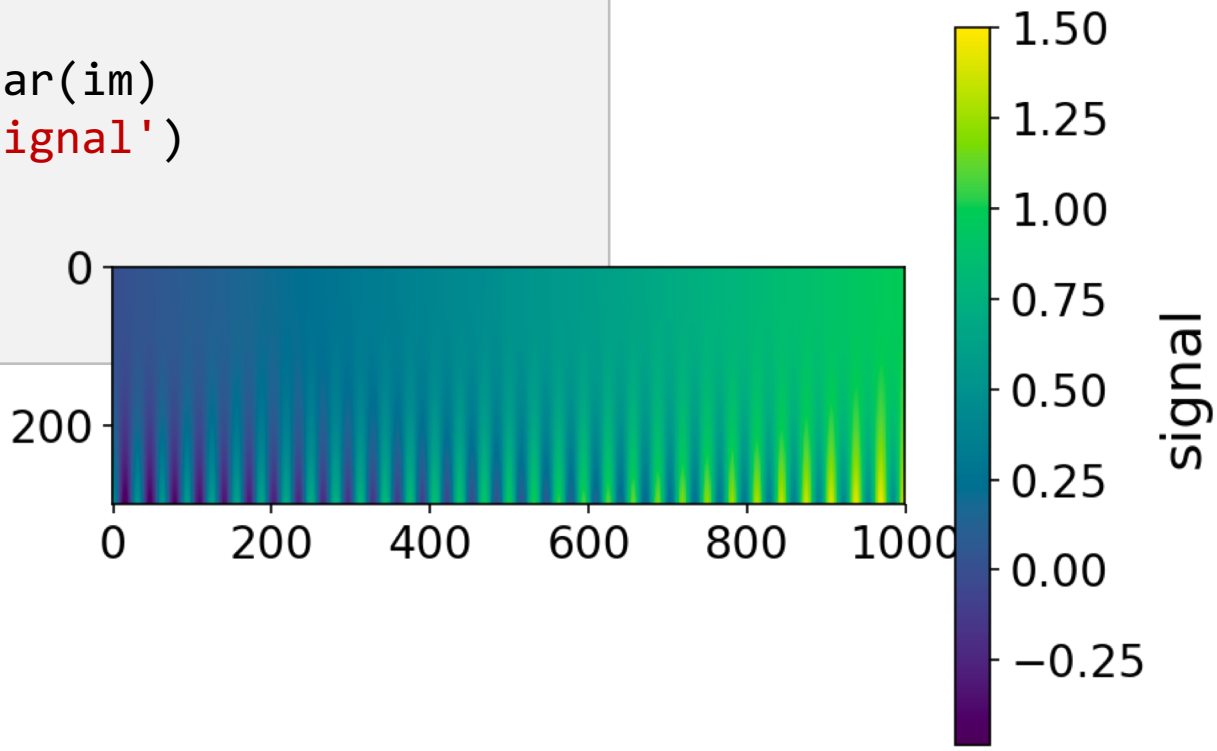


Image Display

Colorbar

```
from mpl_toolkits.axes_grid1 import make_axes_locatable

my_fig, ax = plt.subplots(num='ex:colorbar_2')

im = ax.imshow(z)

divider = make_axes_locatable(ax)

colorbar_ax = divider.append_axes("right", "5%", pad='0.5%')

colorbar = my_fig.colorbar(im, cax=colorbar_ax)
colorbar.set_label('Signal')

plt.show()
```

Image Display

Colorbar

```
from mpl_toolkits.axes_grid1 import make_axes_locatable

my_fig, ax = plt.subplots(num='ex:colorbar_2')

im = ax.imshow(z)

divider = make_axes_locatable(ax)

colorbar_ax = divider.append_axes("right", "5%", pad='0.5%')

colorbar = my_fig.colorbar(im, cax=colorbar_ax)
colorbar.set_label('Signal')

plt.show()
```

Image Display

Colorbar

```
from mpl_toolkits.axes_grid1 import make_axes_locatable

my_fig, ax = plt.subplots(num='ex:colorbar_2')

im = ax.imshow(z)

divider = make_axes_locatable(ax)

colorbar_ax = divider.append_axes("right", "5%", pad='0.5%')

colorbar = my_fig.colorbar(im, cax=colorbar_ax)
colorbar.set_label('Signal')

plt.show()
```

Image Display

Colorbar

```
from mpl_toolkits.axes_grid1 import make_axes_locatable

my_fig, ax = plt.subplots(num='ex:colorbar_2')

im = ax.imshow(z)

divider = make_axes_locatable(ax)

colorbar_ax = divider.append_axes("right", "5%", pad='0.5%')

colorbar = my_fig.colorbar(im, cax=colorbar_ax)
colorbar.set_label('Signal')

plt.show()
```

Image Display

Colorbar

```
from mpl_toolkits.axes_grid1 import make_axes_locatable

my_fig, ax = plt.subplots(num='ex:colorbar_2')

im = ax.imshow(z)

divider = make_axes_locatable(ax)

colorbar_ax = divider.append_axes("right", "5%", pad='0.5%')

colorbar = my_fig.colorbar(im, cax=colorbar_ax)
colorbar.set_label('Signal')

plt.show()
```

Image Display

Colorbar

```
from mpl_toolkits.axes_grid1 import make_axes_locatable

my_fig, ax = plt.subplots(num='ex:colorbar_2')

im = ax.imshow(z)

divider = make_axes_locatable(ax)

colorbar_ax = divider.append_axes("right", "5%", pad='0.5%')

colorbar = my_fig
colorbar.set_labe

plt.show()
```

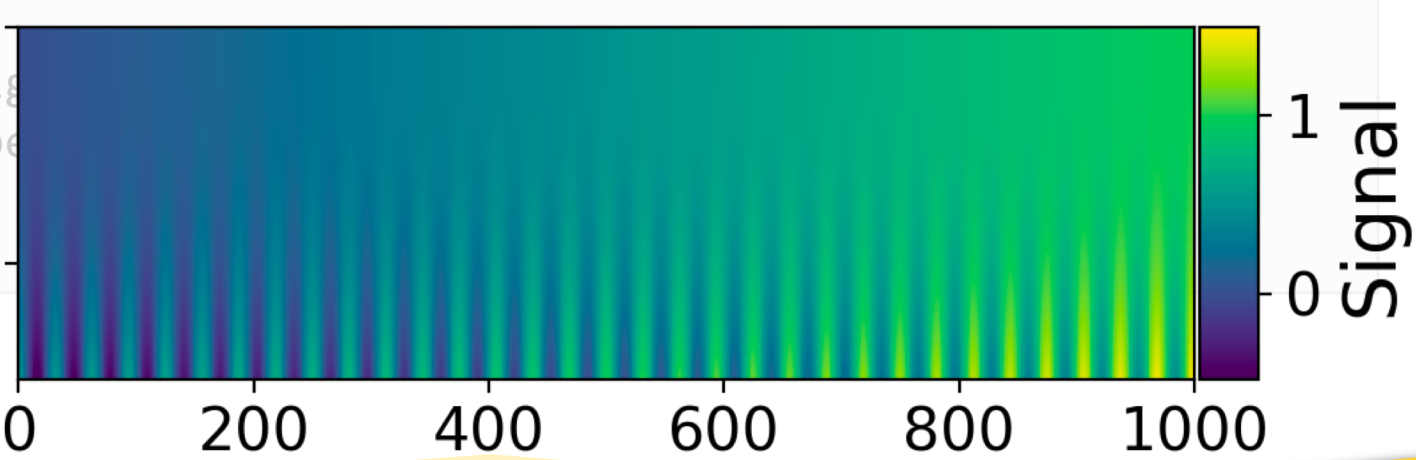


Image Manipulation

Reading PNG files

```
my_image = plt.imread('fig/shapes.png')

fig, ax = plt.subplots(num='shapes_rgb')
ax.imshow(my_image)
plt.show()
```

Image Manipulation

Reading PNG files

```
my_image = plt.imread('fig/shapes.png')

fig, ax = plt.subplots(num='shapes_rgb')
ax.imshow(my_image)
plt.show()
```

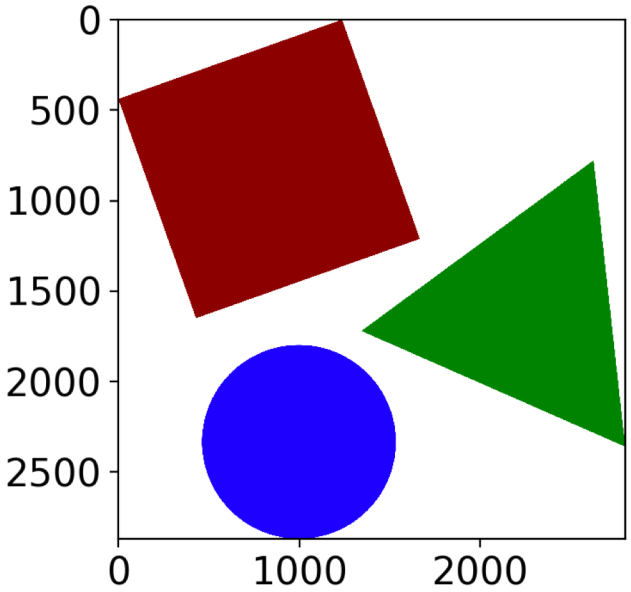


Image Manipulation

Reading PNG files

```
my_image = plt.imread('fig/shapes.png')

fig, ax = plt.subplots(num='shapes_rgb')
ax.imshow(my_image)
plt.show()
```

```
print(type(my_image))
```

```
<class 'numpy.ndarray'>
```

```
print(my_image.shape)
```

```
(2870, 2802, 4)
```

```
print(my_image.dtype)
```

```
float32
```

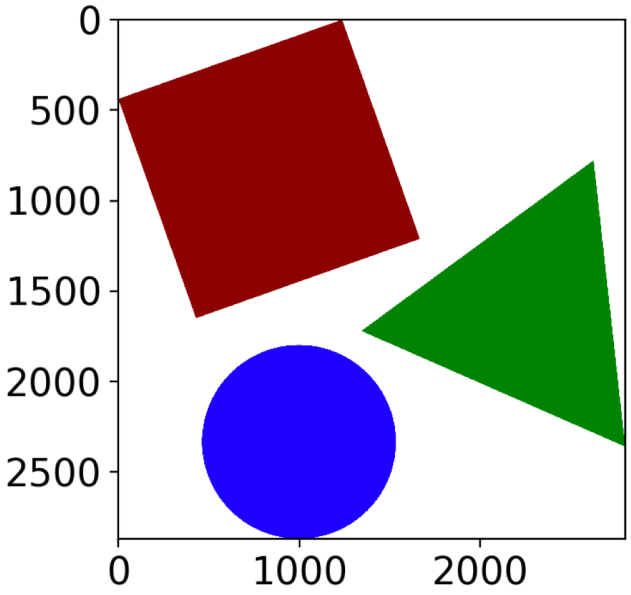


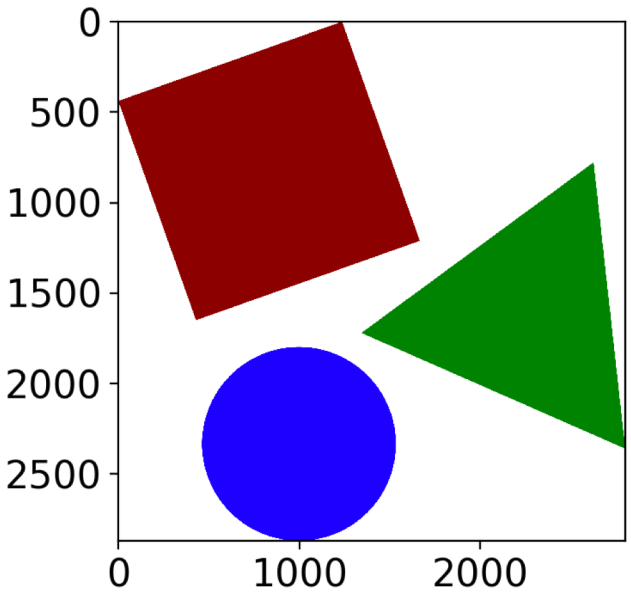
Image Manipulation

Reading PNG files

```
my_image = plt.imread('fig/shapes.png')

fig, ax = plt.subplots(num='shapes_rgb')
ax.imshow(my_image)
plt.show()
```

```
(R, G, B, A)
[0.,  1.]
```




```
print(type(my_image))
```

```
<class 'numpy.ndarray'>
```

```
print(my_image.shape)
```

```
(2870, 2802, 4)
```

```
print(my_image.dtype)
```

```
float32
```

Image Manipulation

Getting one channel

```
red_channel = my_image[:, :, 0]

fig, ax = plt.subplots(num='shapes_red')
ax.imshow(red_channel, cmap="gray_r")
plt.show()
```

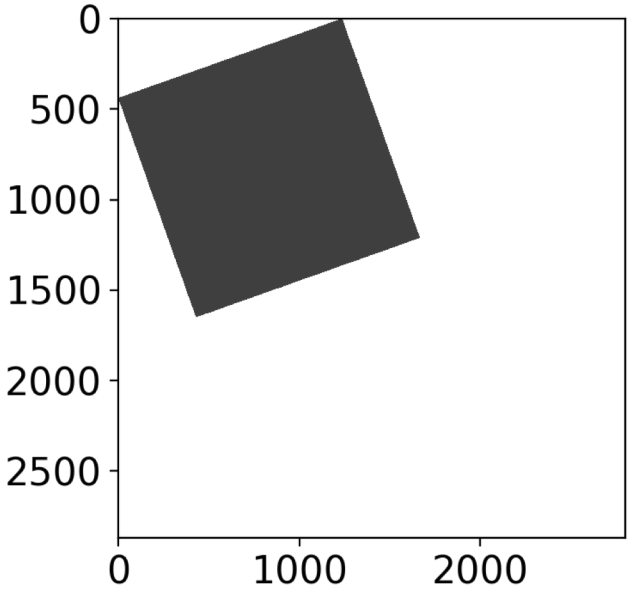


Image Manipulation

From color to grayscale

```
gray_image = my_image[:, :, :3].mean(axis=2)
gray_image = gray_image / gray_image.max()

fig, ax = plt.subplots(num='shapes: gray')
ax.imshow(gray_image, cmap="gray")
plt.show()
```

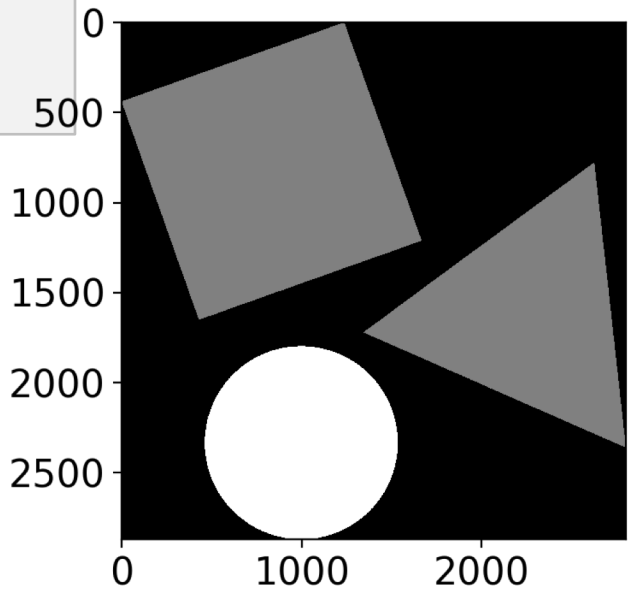


Image Manipulation

Let’s add some noise

```
noise_random = np.random.uniform(
    low=-0.10, high=0.10, size=gray_image.shape)
```

Image Manipulation

Let’s add some noise

```
noise_random = np.random.uniform(
    low=-0.10, high=0.10, size=gray_image.shape)
```

```
noise_poisson = 0.10 * np.random.poisson(gray_image)
```

Image Manipulation

Let's add some noise

```
noise_random = np.random.uniform(
    low=-0.10, high=0.10, size=gray_image.shape)
```

```
noise_poisson = 0.10 * np.random.poisson(gray_image)
```

```
noisy_image = gray_image + noise_random + noise_poisson
```

Image Manipulation

Let's add some noise

```
noise_random = np.random.uniform(
    low=-0.10, high=0.10, size=gray_image.shape)
```

```
noise_poisson = 0.10 * np.random.poisson(gray_image)
```

```
noisy_image = gray_image + noise_random + noise_poisson
```

```
fig, ax = plt.subplots(num='shapes: noisy')
ax.imshow(noisy_image)
plt.show()
```

Image Manipulation

Let's add some noise

```
noise_random = np.random.uniform(
    low=0.50, high=0.60, size=gray_image.shape)
```

```
noise_poisson = 0.10 * np.random.poisson(gray_image)
```

```
noisy_image = gray_image + noise_random + noise_poisson
```

```
fig, ax = plt.subplots(num='shapes: noisy')
ax.imshow(noisy_image)
plt.show()
```

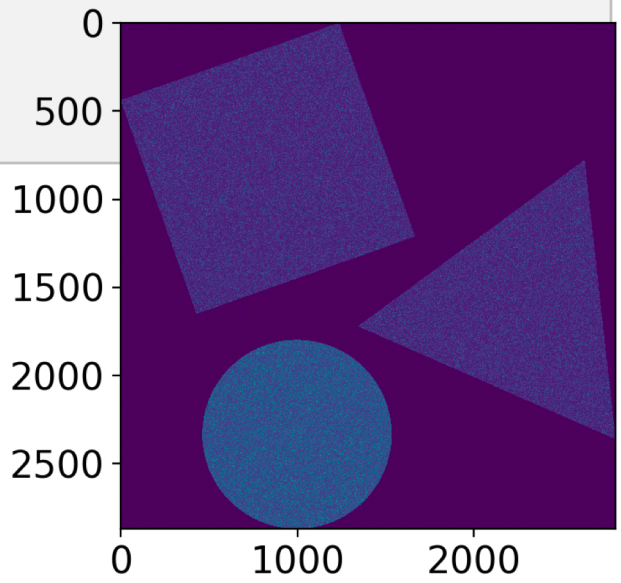


Image Manipulation

Get some statistics with NumPy

```
print(noisy_image.min())
```

```
0.5000000206751437
```

```
print(noisy_image.max())
```

```
6.507237267277877
```

```
print(noisy_image.mean())
```

```
0.9799843028624239
```

```
print(noisy_image.std())
```

```
0.6612658772094668
```

Image Manipulation

Get some statistics with NumPy

```
print(noisy_image.min())
```

```
0.5000000206751437
```

```
print(noisy_image.mean())
```

```
0.979984302862423 9
```

```
print(noisy_image.max())
```

```
6.507237267277877
```

```
print(noisy_image.std())
```

```
0.6612658772094668
```

```
print(np.median(noisy_image))
```

```
0.5925160734130586
```

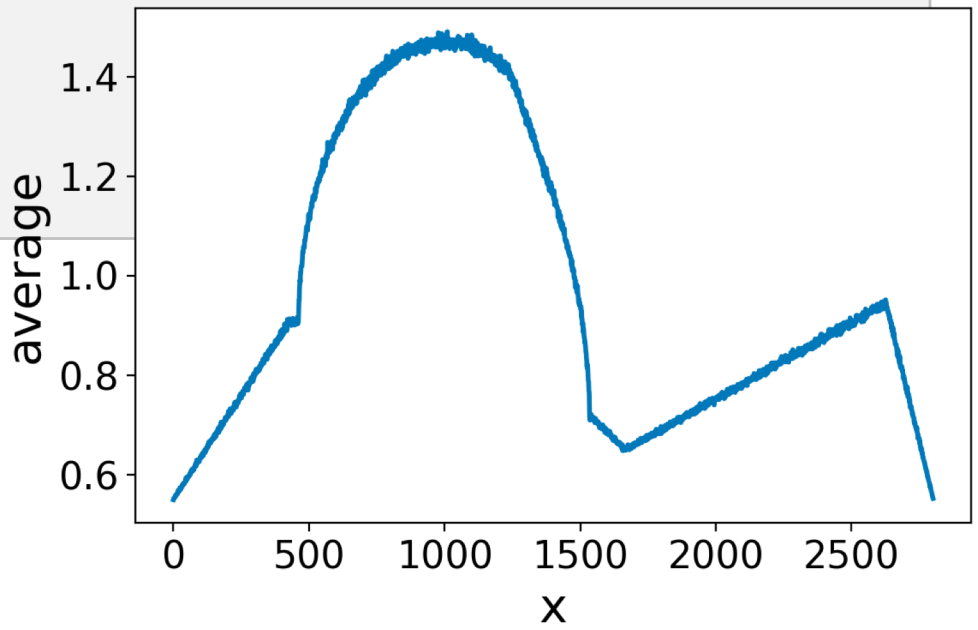
Image Manipulation

Get some statistics with NumPy

```
avg_per_col = noisy_image.mean(axis=0)

fig, ax = plt.subplots(num="avs_cols")
ax.plot(avg_per_col)
ax.set_xlabel('x')
ax.set_ylabel('average')

fig.tight_layout()
plt.show()
```



SciPy

- cluster
- constants
- fftpack
- integrate
- interpolate
- io
- odr
- linalg

- optimize
- ndimage
- signal
- sparse
- spatial
- special
- stats

SciPy

Get some statistics

```
from scipy import stats

results = stats.mode(gray_image)
```

SciPy

Get some statistics

```
from scipy import stats

results = stats.mode(gray_image)
```

```
print(results.__class__)
```

```
<class 'scipy.stats.stats.ModeResult'>
```

SciPy

Get some statistics

```
from scipy import stats

results = stats.mode(gray_image)
```

```
print(results.__class__)
```

```
<class 'scipy.stats.stats.ModeResult'>
```

```
print(results.count)
```

```
[[2866 2863 2859 ... 2843 2852 2861]]
```

```
print(results.mode)
```

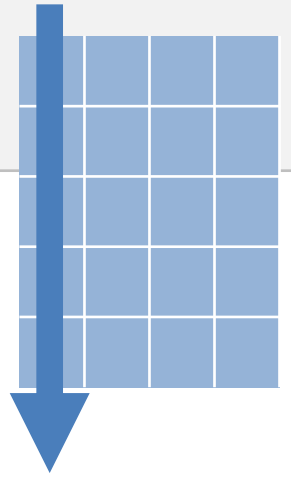
```
[[0. 0. 0. ... 0. 0. 0.]]
```

SciPy

Get some statistics

```
from scipy import stats

results = stats.mode(gray_image)
```



```
print(results.__class__)
```

```
<class 'scipy.stats.stats.ModeResult'>
```

```
print(results.count)
```

```
[[2866 2863 2859 ... 2843 2852 2861]]
```

```
print(gray_image.shape)
print(results.count.shape)
```

```
(2870, 2802)
(1, 2802)
```

```
print(results.mode)
```

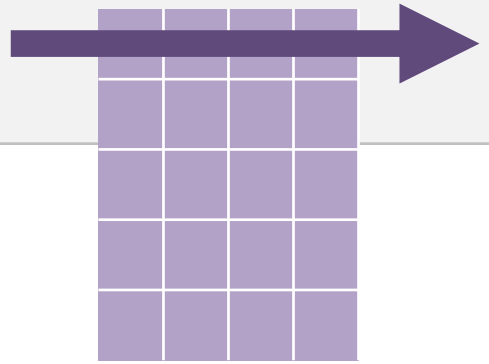
```
[[0. 0. 0. ... 0. 0. 0.]]
```

SciPy

Get some statistics

```
from scipy import stats

results = stats.mode(gray_image, axis=1)
```



```
print(results.__class__)
```

```
<class 'scipy.stats.stats.ModeResult'>
```

```
print(results.count)
```

```
[[2866 2863 2859 ... 2843 2852 2861]]
```

```
print(gray_image.shape)
print(results.count.shape)
```

```
(2870, 2802)
(2870, 1)
```

```
print(results.mode)
```

```
[[0. 0. 0. ... 0. 0. 0.]]
```

Image Manipulation

Get some statistics with SciPy

Flattening 2d into 1d arrays

```
flat_data = gray_image.flatten()
print(flat_slice.shape)
```

```
(8041740,)
```

```
unraveled_data = gray_image.ravel()
print(unraveled_data.shape)
```

```
((8041740,)
```

Image Manipulation

Get some statistics with SciPy

Flattening 2d into 1d arrays

```
flat_data = gray_image.flatten()
print(flat_slice.shape)
```

```
(8041740,)
```

ndarrays only

```
unraveled_data = gray_image.ravel()
print(unraveled_data.shape)
```

```
((8041740,)
```

Image Manipulation

Get some statistics with SciPy

Flattening 2d into 1d arrays

```
flat_data = gray_image.flatten()
print(flat_slice.shape)
```

```
(8041740,)
```

ndarrays only

```
unraveled_data = gray_image.ravel()
print(unraveled_data.shape)
```

```
((8041740,)
```

ndarrays only

Image Manipulation

Get some statistics with SciPy

Flattening 2d into 1d arrays

```
flat_data = gray_image.flatten()
print(flat_slice.shape)
```

```
(8041740,)
```

ndarrays only

```
unraveled_data = gray_image.ravel()
print(unraveled_data.shape)
```

```
((8041740,)
```

ndarrays only

```
results = stats.mode(flat_data)
print(results.count)
print(results.mode)
```

```
[4346287]
[0.]
```

Image Manipulation

Mode on rounded data

```
flat_data = noisy_image.flatten()

rounded_data = np.round(flat_data, decimals=2)

results = stats.mode(rounded_data)

print(results)
```

```
ModeResult(mode=array([0.58]), count=array([436073]))
```

Image Manipulation

Using ndimage

```
data = noisy_image - results.mode[0]

fig, ax = plt.subplots(num="ndimage: raw")
im = ax.imshow(data)

div = make_axes_locatable(ax)
cax = div.append_axes("right", "5%",
                      pad="2.5%")
cbar = fig.colorbar(im, cax=cax)

plt.show()
```

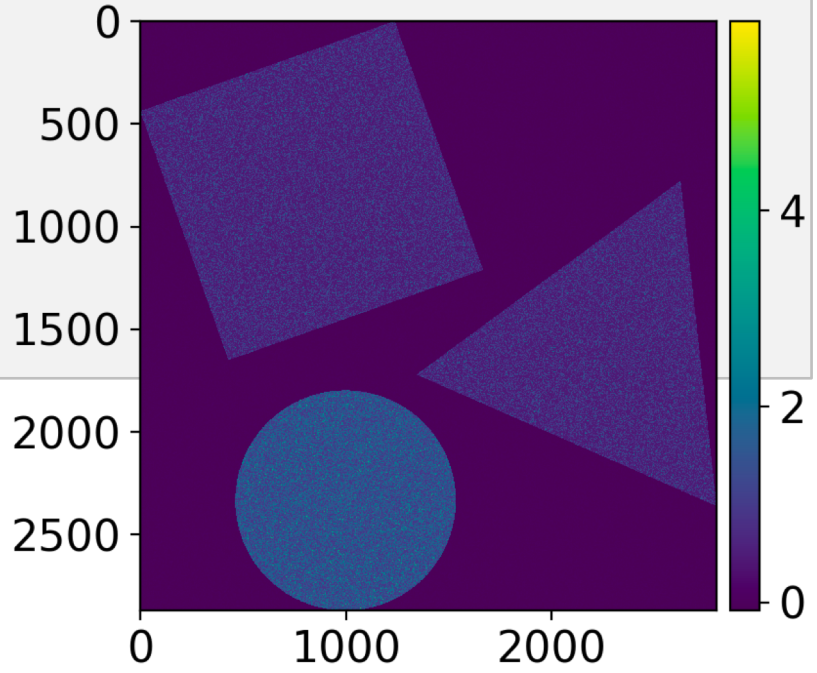


Image Manipulation

Using ndimage

```
from scipy import ndimage

smooth_data = ndimage.gaussian_filter(data, sigma=3)

fig, ax = plt.subplots(num="ndimage: smooth")
im = ax.imshow(smooth_data)

div = make_axes_locatable(ax)
cax = div.append_axes("right", "5%",
                      pad="2.5%")
cbar = fig.colorbar(im, cax=cax)

plt.show()
```

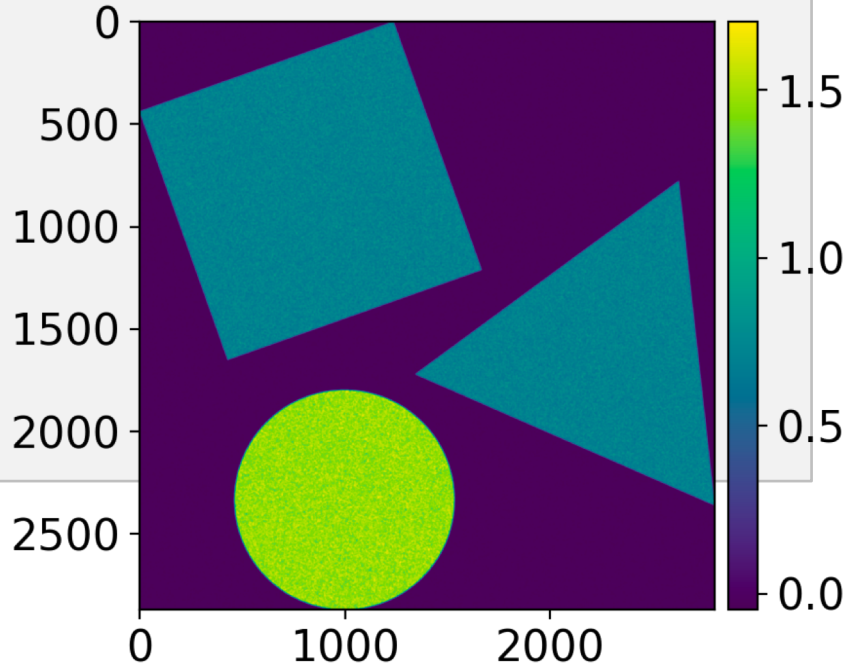


Image Manipulation

Using masked arrays

```
limit = 0.2

masked_data = np.ma.masked_less(smooth_data, limit)
```

```
print(np.ma.mean(masked_data))
```

```
0.8978273280567104
```

```
print(np.mean(data))
```

```
0.3999843028624248
```

Image Manipulation

Using masked arrays

```
from matplotlib import cm

palette = cm.viridis
palette.set_bad('gray', alpha=0.5)

fig, ax = plt.subplots(num="masked array")
ax.imshow(masked_data)

div = make_axes_locatable(ax)
cax = div.append_axes("right", "5%", pad="2.5%")
cbar = fig.colorbar(im, cax=cax)

plt.show()
```

Image Manipulation

Using masked arrays

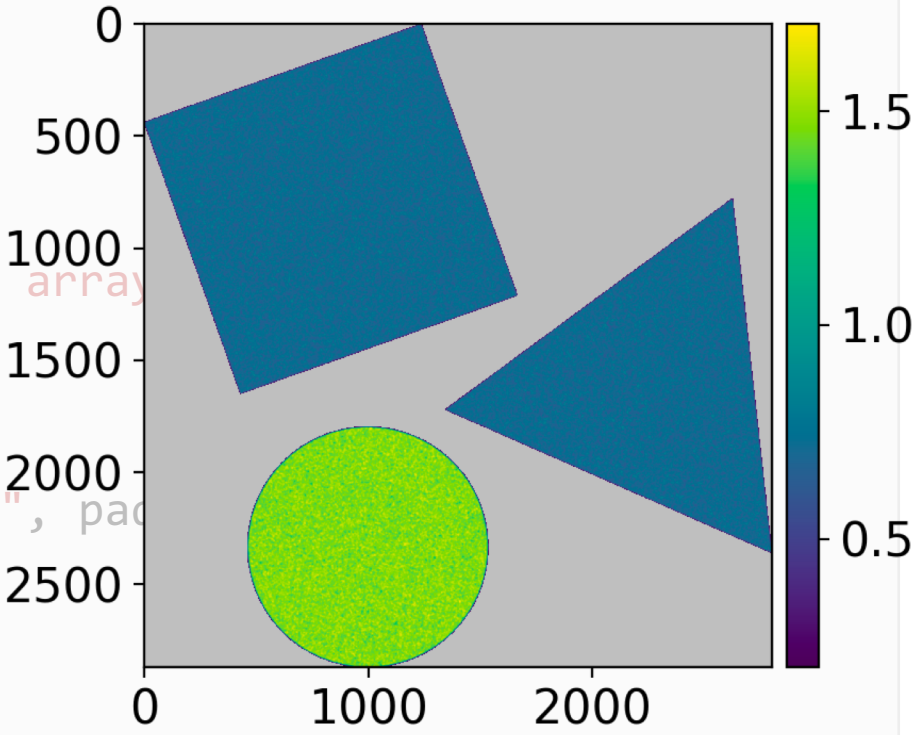
```
from matplotlib import cm

palette = cm.viridis
palette.set_bad('gray', alpha=0.5)

fig, ax = plt.subplots(num="masked array
ax.imshow(masked_data)

div = make_axes_locatable(ax)
cax = div.append_axes("right", "5%", pad
cbar = fig.colorbar(im, cax=cax)

plt.show()
```



Questions?