NumPy 2

Thomas Schwarz, SJ

NumPy Operations

- Numpy allows fast operations on array elements
- We can simply add, subtract, multiply or divide by a scalar

```
>>> vector = np.arange(20).reshape(4,5)
>>> vector
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> vector += 1
>>> vector
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]])
```

NumPy Operations

- Numpy also allows operations between arrays

```
>>> mat = np.random.normal(0,1,(4,5))
>>> mat
array([[ 0.04646031, -1.32970787,  1.16764921, -0.48342653,  0.42295389],
       [ 0.70547825,  1.51980589,  1.46902433, -0.46742839,  1.42472386],
       [ 0.78756679, -0.39975927,  1.24411043, -0.67336526, -0.92416835],
       [ 0.4708628 , -0.29419976, -0.58634161,  0.29038393, -0.78814955]])
>>> vector + mat
array([[ 1.04646031,  0.67029213,  4.16764921,  3.51657347,  5.42295389],
       [ 6.70547825,  8.51980589,  9.46902433,  8.53257161, 11.42472386],
       [11.78756679, 11.60024073, 14.24411043, 13.32663474, 14.07583165],
       [16.4708628 , 16.70580024, 17.41365839, 19.29038393, 19.21185045]])
```

NumPy Operations

- What happens if there is an error?
    - Python would throw an exception, but not so NumPy
        - Example: Create two vectors, one with a zero

```
>>> vector = np.arange(5)
>>> vector2 = np.arange(2,7)
```

- If we divide, we get a warning
- But the result exists, with an inf value for infinity

```
>>> vec = vector2/vector
Warning (from warnings module):
  File "<pyshell#11>", line 1
RuntimeWarning: divide by zero encountered in true_divide
>>> vec
array([       inf, 3.        , 2.        , 1.66666667, 1.5       ])
```

NumPy Operations

- If we divide 0 by 0, we get an nan -- not a value

```
>>> vec=np.arange(4)
>>> vec
array([0, 1, 2, 3])
>>> vec/vec

Warning (from warnings module):
  File "<pyshell#15>", line 1
RuntimeWarning: invalid value encountered in true_divide
array([nan,  1.,  1.,  1.])
```

NumPy Operations

- There are rules for how to define operations with nan and inf, that make intuitive sense
  - IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754)
- We can create inf directly by saying `np.inf`
  - Example: Infinity divided by infinity is not defined

```
>>> np.inf/np.inf
nan
```

NumPy:
Universal Array Functions

- There is a plethora of functions that can be applied to a numpy array.
- These are much faster than the corresponding Python functions
- You can find a list in the numpy u-function manual
  - https://docs.scipy.org/doc/numpy/reference/ufuncs.html

NumPy: Universal Array Functions

- There are universal functions around which the operations are wrapped
  - np.add, np.subtract, np.negative, np.multiply, np.divide, np.floor_divide, np.power, np.mod
- The absolute function is
  - abs
  - np.absolute

NumPy: Universal Array Functions

- Trigonometric functions
  - np.sin, np.cos, np.tan, np.arcsin, np.arccos, np.arctan
- Exponents and logarithms
  - np.log, np.log2 (base 2), np.log10 (base 10)
  - np.expm1 (more exact for small arguments)
  - np.log1p (more exact for small arguments)

NumPy: Universal Array Functions

- Special u-functions:
  - In addition, the submodule scipy.special contains many more specialized functions

NumPy:
Universal Array Functions

- Avoid creating temporary arrays
  - If they are large, too much time spent on moving data
  - Specify the array using the 'out' parameter

```
>>> y = np.empty(10)
>>> x = np.arange(1,11)
>>> np.exp(x, out = y)
array([2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01,
       1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
       8.10308393e+03, 2.20264658e+04])
>>> y
array([2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01,
       1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
       8.10308393e+03, 2.20264658e+04])
```

NumPy:
Universal Array Functions

- Can use np.min, np.max, sum
- Use np.argmin, np.argmax to find the index of the maximum / minimum element
- Can use np.mean, np.std, np.var, np.median, mp.percentile to get statistics
  - Not the only way, see the scipy module

NumPy: Broadcasting

- Operations can be also made between arrays of different sizes
    - Example 1: adding a scalar (zero-dimensional) to a vector

```
>>> x = np.full(5,1)
>>> x+1
array([2, 2, 2, 2, 2])
```

NumPy: Broadcasting

- Adding a vector to a matrix:
  - Create a matrix

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

  - Create a vector

```
>>> x = np.arange(1,6)
>>> x
array([1, 2, 3, 4, 5])
```

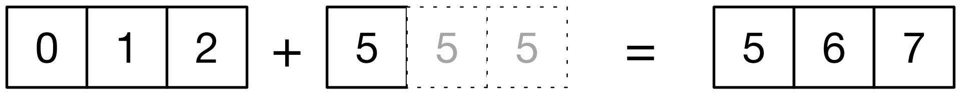
  - Add them together: The vector has been broadcast to a 2 by 5 matrix by doubling the single row

```
>>> matrix+x
array([[ 2,  4,  6,  8, 10],
       [ 7,  9, 11, 13, 15]])
```

NumPy: Broadcasting

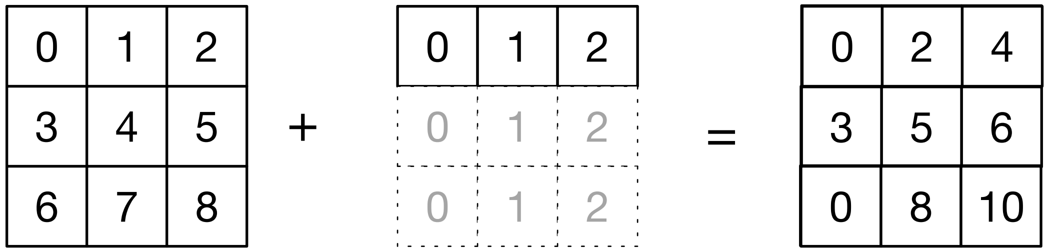
- The broadcast rules: Expand a single coordinate in a dimension in one operand to the value in the other

np.arrange(3) + 5



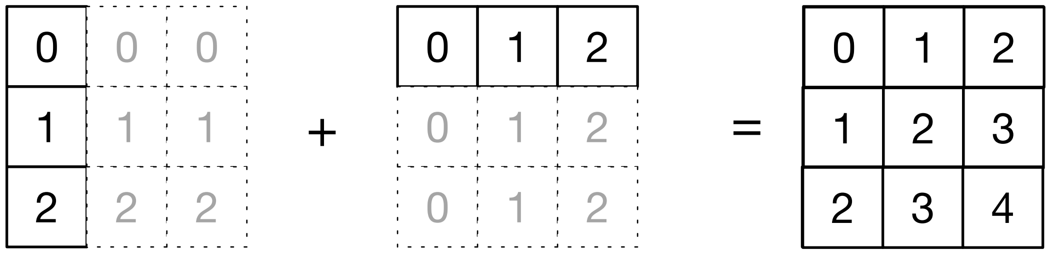
```
0 1 2  +  5 5 5  =  5 6 7
```

np.arrange(9).reshape((3,3)) + np.arrange(3)



```
0 1 2     0 1 2     0 2 4
3 4 5  +  0 1 2  =  3 5 6
6 7 8     0 1 2     0 8 10
```

np.arrange(3).reshape((3,1)) + np.arrange(3)



```
0 0 0     0 1 2     0 1 2
1 1 1  +  0 1 2  =  1 2 3
2 2 2     0 1 2     2 3 4
```

NumPy: Broadcasting

- Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded with ones on its leading site
- Rule 2: If the shape of two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape
- Rule 3: If in any dimensions the sizes disagree and neither is equal to 1, an error is raised

Neat Example

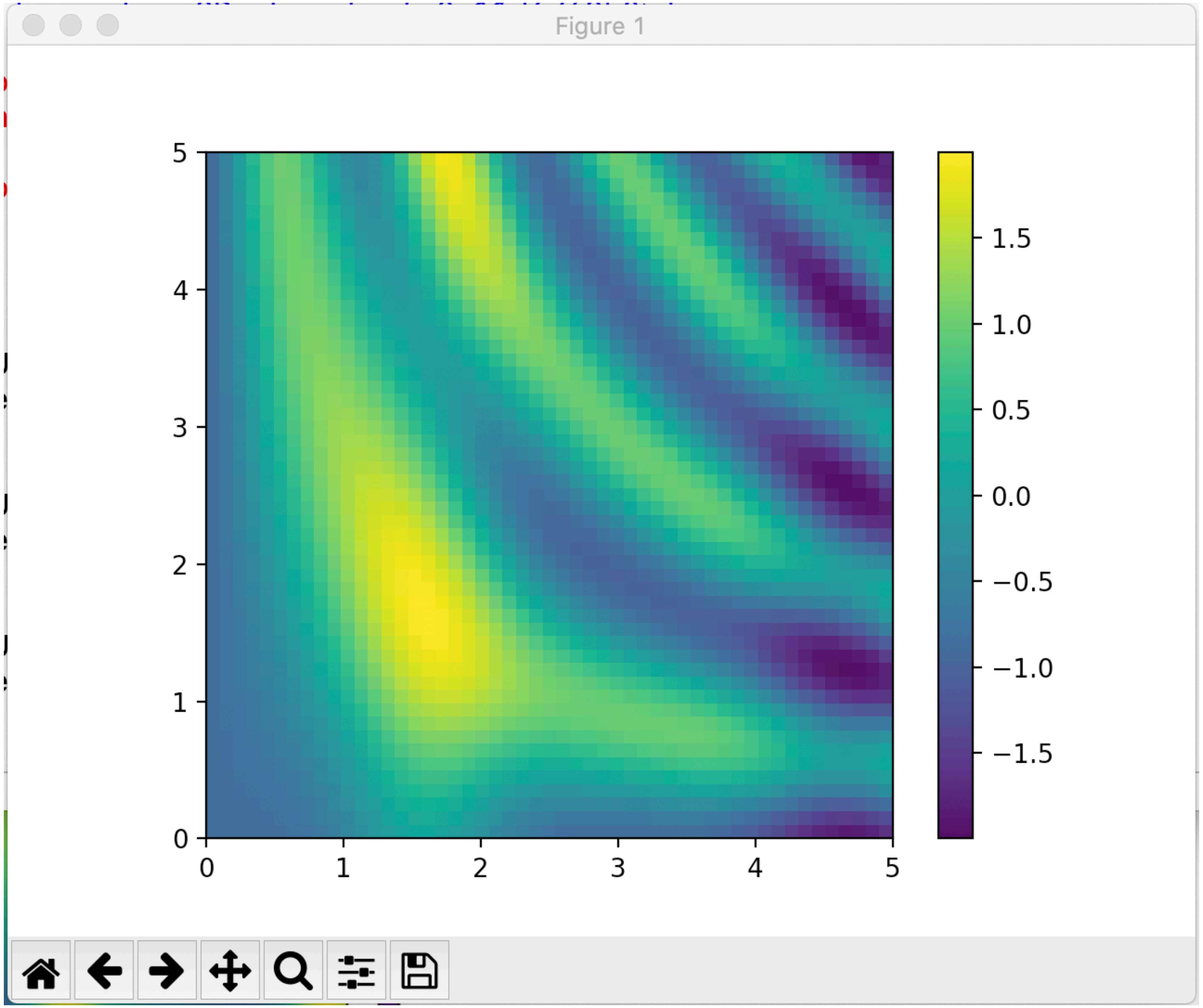
- We combine broadcasting with mathplotlib
  - Using IDLE, we need to call the show function at the end.

NumPy: Broadcasting

- Create a row and a column vector x and y
- Then use broadcasting to combine them for something two-dimensional
- This will get displayed

```
import matplotlib.pyplot as plt
def prob7():
    x = np.linspace(0,5,51)
    y = np.linspace(0,5,51).reshape(51,1)
    z = np.sin(x)**5+np.cos(10+x*y)
    plt.imshow(z, origin='lower', extent=[0, 5, 0, 5],
           cmap='viridis')
    plt.colorbar()
    plt.show()
```

NumPy: Broadcasting



NumPy: Fancy Indexing

- Fancy indexing:
    - Use an array of indices in order to access a number of array elements at once

NumPy: Fancy Indexing

- Example:
  - Create matrix

```
>>> mat = np.random.randint(0,10,(3,5))
>>> mat
array([[3, 2, 3, 3, 0],
       [9, 5, 8, 3, 4],
       [7, 5, 2, 4, 6]])
```

  - Fancy Indexing:

```
>>> mat[(1,2),(2,3)]
array([8, 4])
```

NumPy: Fancy Indexing

- Application:
  - Creating a sample of a number of points
- Create a large random array representing data points

```
>>> mat = np.random.normal(100,20, (200,2))
```

- Select the x and y coordinates by slicing

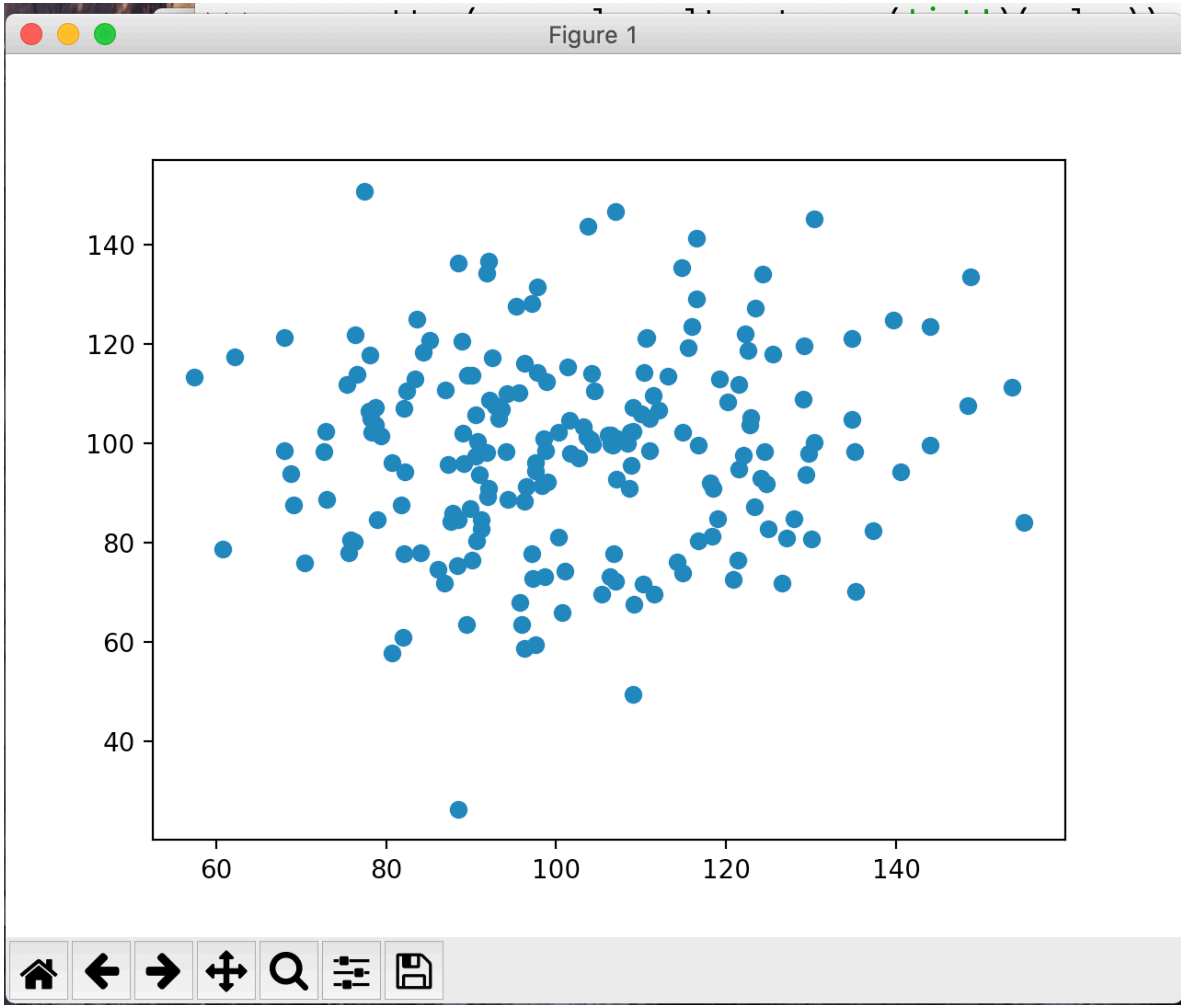
```
>>> x=mat[:,0]
>>> y=mat[:,1]
```

NumPy: Fancy Indexing

- Create a matplotlib figure with a plot inside it

```
>>> fig = plt.figure()
>>> ax = fig.add_subplot(1,1,1)
>>> ax.scatter(x,y)
>>> plt.show()
```

NumPy: Fancy Indexing



NumPy: Fancy Indexing

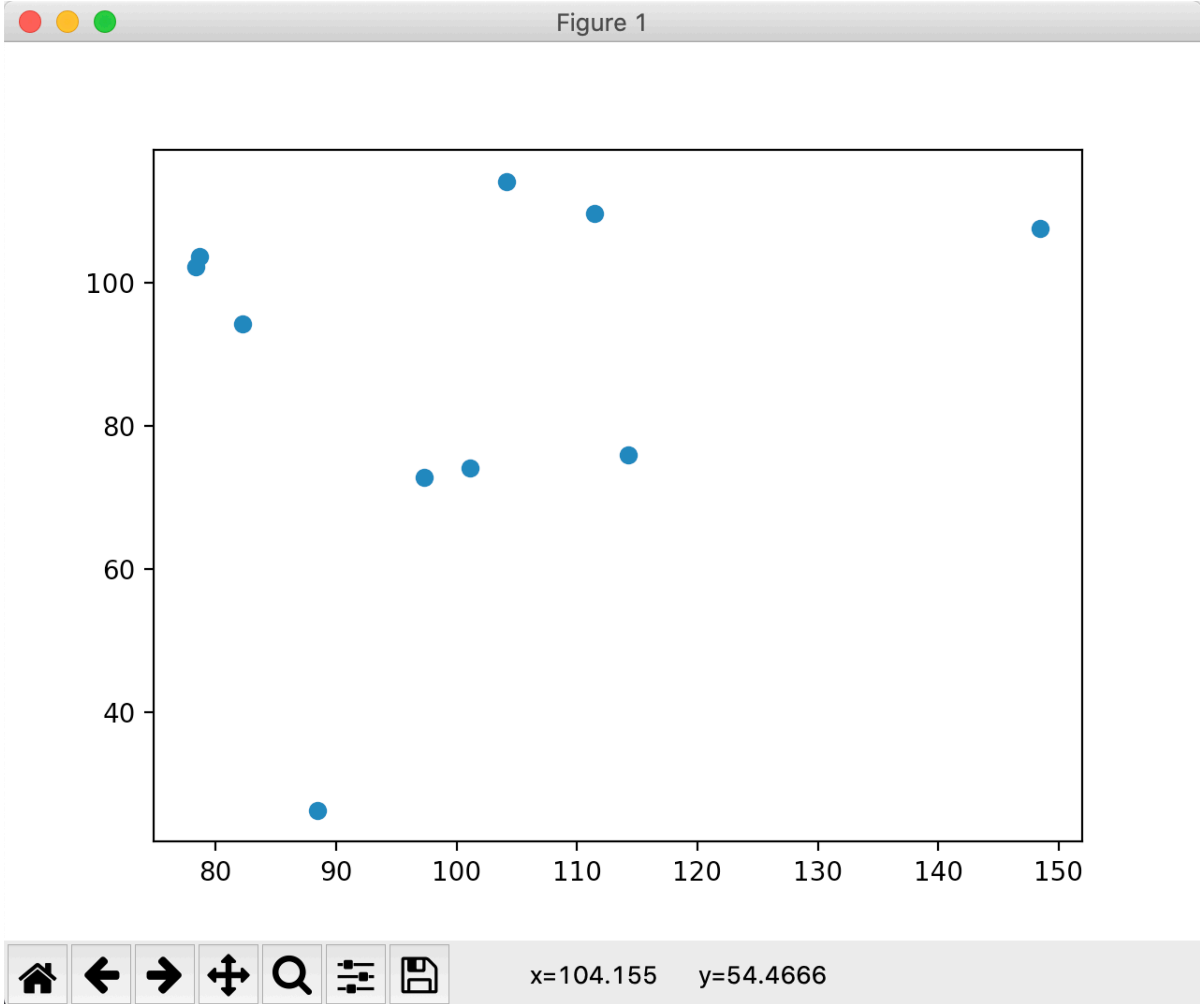
- Create a list of potential indices

```
>>> indices = np.random.choice(np.arange(0,200,1),10)
>>> indices
array([ 32,  93, 172, 134,  90,  66, 109, 158, 188,
30])
```

- Use fancy indexing to create the subset of points

```
>>> subset = mat[indices]
```

NumPy: Fancy Indexing



x=104.155   y=54.4666