Effective Programming Practices for Economists

Scientific Computing

Calculations on arrays

Janoś Gabler and Hans-Martin von Gaudecker

Mathematical functions

```
>>> a = np.array([1, 1.5, 2])
>>> np.exp(a)
array([2.71828183, 4.48168907, 7.3890561 ])

>>> np.log(a)
array([0.        , 0.40546511, 0.69314718])

>>> np.sqrt(a)
array([1.        , 1.22474487, 1.41421356])

>>> np.sin(a)
array([0.84147098, 0.99749499, 0.90929743])
```

- Numpy functions usually apply elementwise
- Faster and more readable than looping
- For more functions see the docs

Reductions

```
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])

>>> a.mean()
2.5

>>> a.std()
1.118033988749895

>>> a.sum()
10

>>> a.sum(axis=1)
array([3, 7])
```

- Reductions take an array of numbers and reduce it to fewer numbers
- Again, faster and more readable than loops
- All reductions support axis arguments

Vectorization

- The above functions are all vectorized

> Vectorization is the process of converting an algorithm from operating on a single value at a time to operating on a set of values (vector) at a time. Hence, we can use these techniques to perform operations on Numpy arrays without using loops.

- The loops are still there, but now in a compiled language
- Faster than Python loops, list comprehensions, ...
- Sometimes vectorization makes code more readable

Linear algebra

```
>>> a = np.array([[1, 0], [0, 4]])
>>> np.linalg.inv(a)

array([[1.  , 0.  ],
       [0.  , 0.25]])

>>> np.linalg.cholesky(a)

array([[1., 0.],
       [0., 2.]])
```

- All matrix decompositions you'll ever need are implemented
- Check out the documentation for an overview