```
>>> import scipy
```

SciPy - a brief introduction

Berian James <berian@berekley.edu>

Astronomy Department, UC Berkeley

Scientific Tools for Python

- SciPy is the scientific toolbox for Python, aimed at mathematics, science and engineering applications.
- It is built on NumPy, i.e., NumPy arrays are the most practical data type; they are generic, efficient and straight-forward to handle.
- SciPy is open-source software, compiled on top of NumPy
- SciPy is also a conference for scientific Python discussion: next meeting is July 16 - 21, in Austin, TX; http://conference.scipy.org/scipy2012/

SciPy resources

- http://scipy-lectures.github.com/
  SciPy lecture notes, fairly complete and usefully formatted

- http://www.tau.ac.il/~kineret/amit/scipy_tutorial/
  Older lecture notes (2004) by Travis Oliphant (Enthought/Continuum); incomplete but very detailed and informative.

- http://scipy-central.org/
  Collection of code snippets and modules, cookbooks, miscellany

- http://docs.scipy.org/doc/scipy/reference/
  SciPy reference guide, tutorial

- http://www.scipy.org/NumPy_for_Matlab_Users

Getting data in and out of SciPy

- See Josh Bloom's lecture tomorrow morning;
  also http://www.scipy.org/Cookbook/InputOutput

- Python provides powerful read/write routines for ascii files and some binary types (C/Fortran)

- Arbitrary input and output
  `np.loadtxt()/savetxt(), np.genfromtxt()/recfromcsv(),`
  `np.save()/load()`

  Certain proprietary (but common) binary formats:
  `scipy.io.matlab, scipy.io.idl`

Special binaries (Matlab, IDL, HDF5): `scipy.io`

- Support for Matlab, IDL, HDF5 (though the PyTables module), as well as Matrix Market and NetCDF.

- Includes support for advanced data structures in these languages

- E.g., Matlab data:

```
>>> from scipy import io
>>> struct = io.loadmat('file.mat', struct_as_record=True)
>>> io.savemat('file.mat', struct)
```

Building and referencing your own arrays quickly

- (Row) vector of numbers: `np/sp.r_` and `np/sp.linspace`

```
>>> np.r_[1.:11.] # N.b. (1,2,...,10)
>>> np.linspace(a,b,n)
```

- n-d grid of coordinates: `np/sp.mgrid`

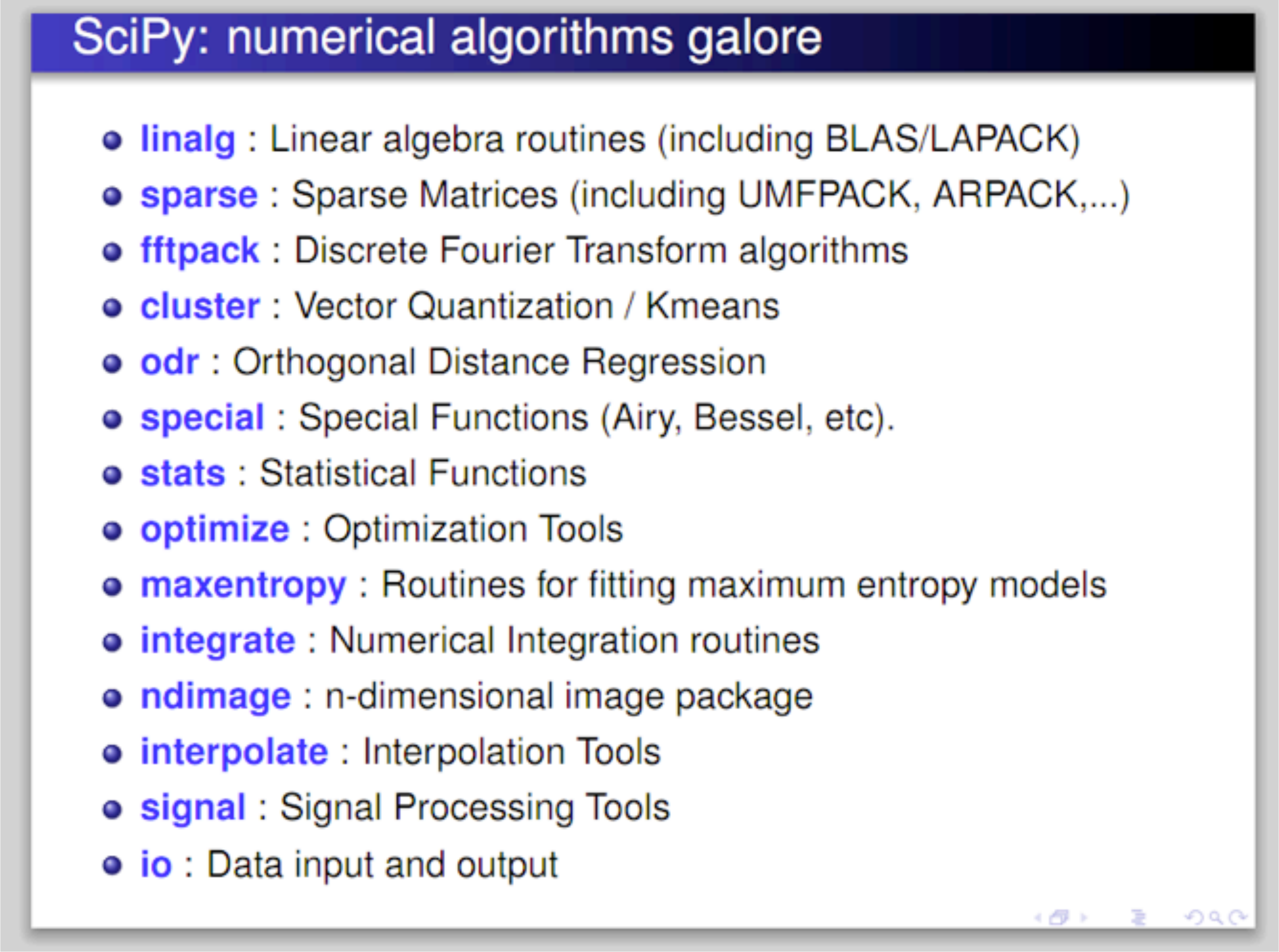
```
>>> x,y = np.mgrid(1:5,1:5) # A 4x4 array
>>> r = np.sqrt(x**2 + y**2)
```

- n-d array: `np/sp.c_` and `np.tile`

```
>>> x = np.linspace(0,10,11);
>>> np.c_[x,x]
```

SciPy packages

- See Sunday lecture on Scientific Programming for fuller discussion of these



SciPy: numerical algorithms galore

- **linalg** : Linear algebra routines (including BLAS/LAPACK)
- **sparse** : Sparse Matrices (including UMFPACK, ARPACK,...)
- **fftpack** : Discrete Fourier Transform algorithms
- **cluster** : Vector Quantization / Kmeans
- **odr** : Orthogonal Distance Regression
- **special** : Special Functions (Airy, Bessel, etc).
- **stats** : Statistical Functions
- **optimize** : Optimization Tools
- **maxentropy** : Routines for fitting maximum entropy models
- **integrate** : Numerical Integration routines
- **ndimage** : n-dimensional image package
- **interpolate** : Interpolation Tools
- **signal** : Signal Processing Tools
- **io** : Data input and output

Symbolic mathematics with Python

- http://sympy.org/
  SymPy home page

- http://docs.sympy.org
  Reference, tutorial

- Think of SymPy as Mathematica for Python, including integration, geometry, linear algebra, statistics, ODE solving and tensor algebra

```
>>> import sympy
```

Interfacing with other languages

- E.g., http://www.scipy.org/PerformancePython
  An interesting and useful comparison of possibilities
- Cython (<- Pyrex)
  The most comprehensive option; requires a lecture of its own
- f2py
  Interface with Fortran, great for number-crunching
- PyPy (<- Psyco)
  Truly amazing, but does not support NumPy :’(
- scipy.weave
  Very cool to use, perhaps becoming less common(?)

Blending languages: f2py

- You will need: Python, a Fortran compiler (e.g. g95, gfortran) and f2py
- This can be a breakout exercise

Blending languages: weave

- See also: http://www.scipy.org/Cookbook/Weave
- You will need: a C/C++ compiler
- This can also be a breakout exercise