SciPy: Scientific Toolkit

SciPy

- SciPy is a collection of mathematical algorithms and convience functions built on Numpy data structures
- Organized into subpackages covering different scientific computing areas
- A data-processing and prototyping environment rivaling MATLAB

SciPy Submodules

- Special functions (`scipy.special`)
- Integration (`scipy.integrate`)
- Optimization (`scipy.optimize`)
- Interpolation (`scipy.interpolate`)
- Fourier Transforms (`scipy.fftpack`)
- Signal Processing (`scipy.signal`)
- Linear Algebra (`scipy.linalg`)
- Sparse Eigenvalue Problems with ARPACK Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
- Statistics (`scipy.stats`)
- Multi-dimensional image processing (`scipy.ndimage`)
- File IO (`scipy.io`)
- Weave (`scipy.weave`)
- And more. . .

Common submodules: `scipy.integrate`

Integrate the function:

$$f(x) = \int_0^4 x^2 \mathrm{d}x$$

```
import scipy.integrate
ans, err = scipy.integrate.quad(lambda x: x ** 2, 0., 4)
ans
```

```
21.333333333333336
```

See also: `dblquad`, `tplquad`, `fixed_quad`, `trapz`, `simps`

Common submodules: `scipy.linalg`

Matrix Inverse

```
import numpy as np
import scipy.linalg
a = np.random.rand(3,3)
scipy.linalg.inv(a)
```

```
array([[ 2.09386567,  0.18794291, -2.33891785],
       [ 4.50278126, -2.39788758, -1.04738682],
       [-6.0432121 ,  2.88320448,  3.46459537]])
```

Eigenvalues

```
scipy.linalg.eigvals(a)
```

```
array([ 2.08083995+0.j,  0.16847753+0.j, -0.30717139+0.j])
```

Common submodules: `scipy.interpolate`

Interpolate a function

```
import numpy as np
from scipy import interpolate, integrate
x = np.arange(-1,11)
y = np.exp(-x/3.0)
f = interpolate.interp1d(x,y); f
```

```
<scipy.interpolate.interpolate.interp1d at 0x1127a2728>
```

Integrate the interpolated function

```
ans, err = integrate.quad(f,0,10); ans
```

```
2.9197153790964223
```

Integrate the data

```
integrate.simps(y[1:-1],x[1:-1])
```

```
2.8550038226912573
```

Why Python/Numpy/SciPy?

- A *free* alternative to MATLAB
- The power of the full Python language
  - Object-oriented
  - Procedural
  - Functional (almost)
- More reasons come:
  - MATLAB-like plotting
  - Call C/C++/Fortran code directly
  - MPI-style parallel programming (take my graduate course!)