Scientific programming in Python

Ask Hjorth Larsen
asklarsen@gmail.com

Nano-bio Spectroscopy Group
and ETSF Scientific Development Centre
Universidad del País Vasco UPV/EHU

May 30, 2017

"The SciPy stack"

- NumPy — basic array library
- SciPy — algorithms of many types
- Matplotlib — plotting
- IPython — interactive console
- Sympy — symbolic algebra
- And more! `https://scipy.org/`

Objectives

- Introduction to array library NumPy
- Introduction to plotting with Matplotlib
- Introduction to SciPy
- Excellent replacement for Matlab (proprietary!) or Octave
- Write your own DFT code!
- Brief discussion of computations with Python and C extensions (talk about parallelism moved till tomorrow!)

Python for Scientific computing

Everyday scripting

- Reading, writing
- String processing / parsing
- Visualization
- Speed not essential

Numerical computations

- Speed is essential (pure-Python numbercrunching is 100–1000 times slower than C/Fortran)
- Parallelism
- C extensions

NumPy arrays

```
>>> import numpy as np
>>> np.array([2, 3, 5])  # int
array([2, 3, 5])
>>> np.array([[2, 3], [4, 5.0]])  # float
array([[ 2.,  3.],
       [ 4.,  5.]])
>>> np.zeros((2, 2))  # float by default
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> x = np.ones(7) * 2
>>> x[3:6]  # Return "view" into array
array([ 2.,  2.,  2.])
>>> x[3:6] += 5.0  # Assign into array
>>> x
array([ 2.,  2.,  2.,  7.,  7.,  7.,  2.])
>>>
```

NumPy arrays

```
>>> import numpy as np
>>> x = np.random.rand(2, 3)
>>> x.shape
(2, 3)
>>> x
array([[ 0.85846037,  0.61976662,  0.29755835],
       [ 0.62705442,  0.57475837,  0.33435306]])
>>> x[0, :]
array([ 0.85846037,  0.61976662,  0.29755835])
>>> x[:, 0]
array([ 0.85846037,  0.62705442])
>>> y = x[:, 1:]
>>> y
array([[ 0.61976662,  0.29755835],
       [ 0.57475837,  0.33435306]])
>>>
```

Arrays in NumPy

- Numbers reside within a contiguous memory buffer
- Array has dimensions and stride
- Last dimension is memory-contiguous (C-style, not Fortran-style)

Matrix operations

Arithmetic

- Elementwise: `A * B`, `A + B`, `A > B`, `np.exp(A)` etc.
- Matrix product `C = np.dot(A, B)`
  From Python3.5: `C = A @ B`
- Eigenvalues `eig, V = np.linalg.eig(A)`
- Sum over axis `A.sum(axis=1)`

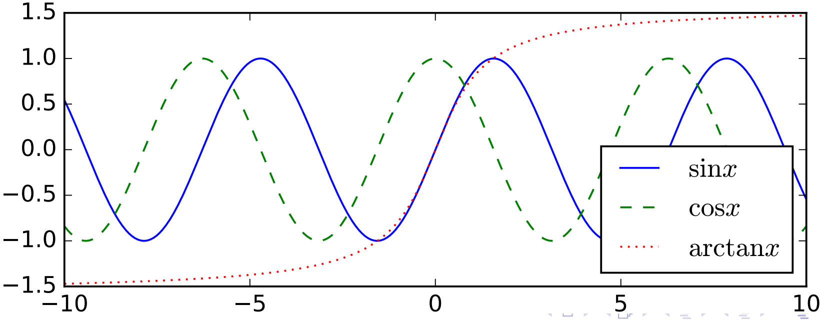
View operations

- Transpose `Atr = A.T`; `A.transpose(0, 2, 1)`
- Slice `A[start1:stop1:step1, start2:stop2:step2, ...]`
- Reshape array `np.zeros(16).reshape(4, 4)`

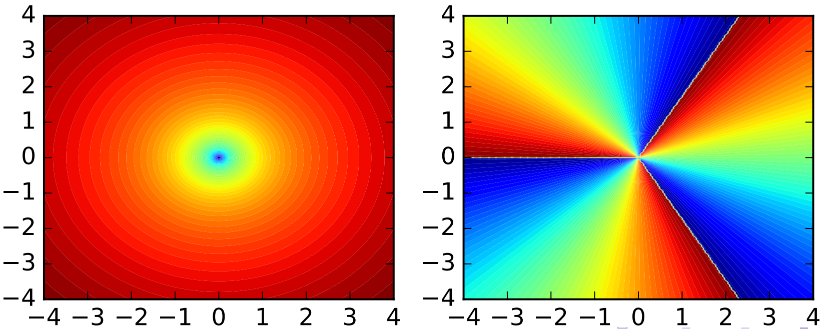
Matplotlib

- High-quality plots
- Extremely large number of examples online
- `https://matplotlib.org/gallery.html`

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-10, 10, 1000)
plt.figure(figsize=(7, 2.5))
plt.plot(x, np.sin(x), 'b-', label=r'$\sin x$')
plt.plot(x, np.cos(x), 'g--', label=r'$\cos x$')
plt.plot(x, np.arctan(x), 'r:', label=r'$\arctan x$')
plt.legend(loc='lower right')
plt.savefig('graphs.pdf')
```



```python
import numpy as np
import matplotlib.pyplot as plt
fig, axes = plt.subplots(ncols=2, figsize=(7, 2.5))
x = np.linspace(-4., 4., 200)
X, Y = np.meshgrid(x, x)
Z = np.log((X + 1j * Y)**3)
axes[0].contourf(X, Y, Z.real, 64)
axes[1].contourf(X, Y, Z.imag, 64)
fig.savefig('2dplot.pdf')
```



SciPy

- Special functions (Bessel, Airy, ...)
- Interpolation / splines
- Optimization algorithms
- Sparse matrices
- `https://docs.scipy.org/doc/scipy/reference/`

Write your own DFT code

1. Harmonic oscillator with non-interacting particles
2. Self-consistency loop including LDA exchange energy (forget about correlation!)
3. “Soft-Coulomb” Poisson solver

Harmonic oscillator with non-interacting particles

- Define a grid on which to solve the problem (`np.linspace`)
- Use simple finite-difference kinetic operator

$$T = -\frac{1}{2}\nabla^2 \rightsquigarrow -\frac{1}{2} \cdot \frac{1}{\mathrm{d}x^2} \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

- Solve for independent particles in box with quadratic potential $v = x^2/2$. Potential is local, $H = T + V$, with V purely diagonal.
- Use (Hermitian) `np.linalg.eigh` to get eigenvalues and vectors. Check energies.
- Plot first few eigenfunctions.

Self-consistency loop with LDA exchange

- Calculate density
- Now include LDA exchange

$$E_x[n] = -\frac{3}{4}\left(\frac{3}{\pi}\right)^{1/3}\int n^{4/3}(x)\mathrm{d}x$$

$$v_x(x) = \frac{\partial E_x[n]}{\partial n(x)} = -\left(\frac{3}{\pi}\right)^{1/3} n^{1/3}(x)$$

- Write selfconsistency loop and terminate at appropriate convergence criterion. (Let's ignore LDA correlation, it is too tedious)

Add “softened” Coulomb interaction

$$E_H[n] = \frac{1}{2}\iint \frac{n(x)n(x')}{\sqrt{1+(x-x')^2}}\mathrm{d}x\mathrm{d}x'$$

$$v_H(x) = \frac{\delta E_H[n]}{\delta n(x)} = \int \frac{n(x')}{\sqrt{1+(x-x')^2}}\mathrm{d}x'$$

Call C from Python

Why

- Tight loops inefficient in Python/NumPy
- Rewrite performance critical parts in C
- Method 1: C extensions
- Method 2: ctypes