Lecture 2: NumPy and SciPy

Hendrik Weimer

Institute for Theoretical Physics, Leibniz University Hannover

Quantum Physics with Python, 18 April 2016

Outline of the course

1. Introduction to Python
2. **SciPy/NumPy packages**
3. Plotting and fitting
4. QuTiP: states and operators
5. Ground state problems
6. Non-equilibrium dynamics: quantum quenches
7. Quantum master equations
8. Generation of squeezed states
9. Quantum computing
10. Grover's algorithm and quantum machine learning
11. Student presentations

Installing NumPy and SciPy

- Using your package manager
  `apt-get install python3-scipy`
- Using pip
  `pip install scipy`
- Build from source code
  `https://www.scipy.org/`

Python lists versus vectors

```
a = [1, 0]
b = [0, 1]
print(a+b)
```

Output:

```
[1, 0, 0, 1]
```

NumPy to the rescue

```
import numpy as np

a = np.array([1, 0])
b = np.array([0, 1])
print(a+b)
```

Output:

```
[1 1]
```

Matrices

Matrices are two-dimensional arrays

```
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
print(A[0,0])
print(A[:,0])
print(A.trace())
```

Output:

```
1
[1 4 7]
15
```

Matrix multiplication

```
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
print(A*A)
print(A.matmul(A)) # Python 3.6: A @ A
```

Output:

```
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]
[[ 30  36  42]
 [ 66  81  96]
 [102 126 150]]
```

Array ranges

```
for x in np.arange(0,1,0.2):
    print(x)

print()
print(np.arange(4,4.4,0.1))
```

Output:

```
0.0
0.2
0.4
0.6
0.8

[ 4.   4.1  4.2  4.3  4.4]
```

Use `numpy.linspace` to have explicit control over endpoints!

Simple statistics

```
from random import random

a = [random() for x in range(4)]
print(a)
print(np.mean(a), np.std(a))
```

Output:

```
[0.6624686402055188, 0.17648996819820462, 0.22978963184239998,
0.9815287045139853]
0.51256923619 0.32988876658
```

Reading data files

```
data = np.loadtxt("data.txt")
```

Better control over header/footer, missing values, etc.: use `numpy.genfromtxt`

Example: GISS Surface Temperature data

- Land-ocean temperature index
  [J. Hansen et al., Rev. Geophys. **48**, RG4004 (2010)]
- Data going back to 1880
- Zero base point: 30 year average from 1951 to 1980

`http://data.giss.nasa.gov/gistemp/graphs_v3/Fig.A2.txt`

```
Global Land-Ocean Temperature Index (C) (Anomaly with Base: 1951-1980)
------------------------------------------------------------------------
 Year  Annual_Mean 5-year_Mean
----------------------------------
 1880     -0.20         *
 1881     -0.11         *
 1882     -0.09     -0.17
 1883     -0.20     -0.19
```

Overview of SciPy

16 sub-packages, including:

- Integration
- Interpolation
- Linear algebra
- Optimization
- Special functions

Integration

Stefan-Boltzmann law:

$$P/A = \sigma \frac{(k_B T)^4}{h^3 c^2}$$

$$\sigma = 2\pi \int\limits_0^\infty dx \, \frac{x^3}{e^x - 1}$$

```
from math import pi, exp
from scipy import integrate

I = integrate.quadrature(lambda x: x^3/(exp(x)-1), 0,
                         float('inf'))
```

Reduce upper limit

- Python can represent only numbers up to $\sim 10^{308}$
- $\exp(710) > 10^{308}$
- However: $x^3/(e^x - 1) \sim 10^{-296}$ for $x = 700$

```
I = integrate.quadrature(lambda x: x^3/(exp(x)-1), 0, 700)
sigma = 2*pi*I[0]
print(I)
print(sigma, sigma-2*pi**5/15)
```

Output:

```
(6.49393940226683, 1.877560045914113e-11)
40.80262463803753 7.105427357601002e-15
```

Solving linear equations

```
from scipy import linalg
A = np.array([[3, 2, 0], [1, -1, 0], [0, 5, 1]])
b = np.array([2, 4, -1])
print(linalg.solve(A, b))
```

Output:

```
[ 2. -2.  9.]
```

Solving nonlinear equations

Eigenvalue equation for the finite potential well (symmetric solutions)

$$\sqrt{V_0^2 - \nu^2} = \nu \tan \nu$$

```
from scipy import optimize
from math import sqrt, tan
V_0 = 5
f = lambda nu: sqrt(V_0**2-nu**2) - nu*tan(nu)
print(optimize.fsolve(f,1))
print(optimize.fsolve(f,4))
```

Output:

```
[ 1.30644001]
[ 3.83746711]
```

Matrix diagonalization

w

```
L_x = 1/sqrt(2)*np.array([[0, 1, 0], [1, 0, 1], [0, 1, 0]])
print(linalg.eigh(Lx))
```

Output:

```
(array([ -1.00000000e+00,   1.11022302e-15,   1.00000000e+00]),
array([[ -5.00000000e-01,   7.07106781e-01,   5.00000000e-01],
       [  7.07106781e-01,   6.66133815e-16,   7.07106781e-01],
       [ -5.00000000e-01,  -7.07106781e-01,   5.00000000e-01]]))
```

Matrix exponentiation

```
sigma_x = np.array([[0, 1], [1, 0]])
print(linalg.expm(1j*pi/4*sigma_x))
```

Output:

```
[[ 0.70710678+0.j          0.00000000+0.70710678j]
 [ 0.00000000+0.70710678j  0.70710678+0.j        ]]
```