Python for scientific research

Number crunching with numpy and scipy

Bram Kuijper

University of Exeter, Penryn Campus, UK

March 6, 2019

UNIVERSITY OF EXETER | DOCTORAL COLLEGE

Researcher Development

What we've done so far

1. Declare variables using built-in data types and execute operations on them
2. Use flow control commands to dictate the order in which commands are run and when
3. Encapsulate programs into reusable functions, modules and packages
4. **Next**: Number crunching using `NumPy/SciPy`

Motivation

- We are typically faced by numerical tasks, such as, computing Pearson's correlation coefficient or generating random numbers

Motivation

- We are typically faced by numerical tasks, such as, computing Pearson's correlation coefficient or generating random numbers
- The `NumPy` (numeric python) package provides a comprehensive set of mathematical data structures (e.g vector, matrix) and functions (e.g trigonometry, random number generator)

Motivation

- We are typically faced by numerical tasks, such as, computing Pearson's correlation coefficient or generating random numbers
- The `NumPy` (numeric python) package provides a comprehensive set of mathematical data structures (e.g vector, matrix) and functions (e.g trigonometry, random number generator)
- The `SciPy` (scientific python) library builds on top of `NumPy` to provide a collection of numerical algorithms for:

Motivation

- We are typically faced by numerical tasks, such as, computing Pearson's correlation coefficient or generating random numbers
- The `NumPy` (numeric python) package provides a comprehensive set of mathematical data structures (e.g vector, matrix) and functions (e.g trigonometry, random number generator)
- The `SciPy` (scientific python) library builds on top of `NumPy` to provide a collection of numerical algorithms for:
  - Statistics (e.g correlation coefficient)

Motivation

- We are typically faced by numerical tasks, such as, computing Pearson's correlation coefficient or generating random numbers
- The `NumPy` (numeric python) package provides a comprehensive set of mathematical data structures (e.g vector, matrix) and functions (e.g trigonometry, random number generator)
- The `SciPy` (scientific python) library builds on top of `NumPy` to provide a collection of numerical algorithms for:
  - Statistics (e.g correlation coefficient)
  - Signal processing (e.g Fourier transform, filtering)

Motivation

- We are typically faced by numerical tasks, such as, computing Pearson's correlation coefficient or generating random numbers
- The `NumPy` (numeric python) package provides a comprehensive set of mathematical data structures (e.g vector, matrix) and functions (e.g trigonometry, random number generator)
- The `SciPy` (scientific python) library builds on top of `NumPy` to provide a collection of numerical algorithms for:
  - Statistics (e.g correlation coefficient)
  - Signal processing (e.g Fourier transform, filtering)
  - Solving differential equations

Motivation

- We are typically faced by numerical tasks, such as, computing Pearson's correlation coefficient or generating random numbers
- The `NumPy` (numeric python) package provides a comprehensive set of mathematical data structures (e.g vector, matrix) and functions (e.g trigonometry, random number generator)
- The `SciPy` (scientific python) library builds on top of `NumPy` to provide a collection of numerical algorithms for:
  - Statistics (e.g correlation coefficient)
  - Signal processing (e.g Fourier transform, filtering)
  - Solving differential equations
  - Optimisation

Motivation

- We are typically faced by numerical tasks, such as, computing Pearson's correlation coefficient or generating random numbers
- The `NumPy` (numeric python) package provides a comprehensive set of mathematical data structures (e.g vector, matrix) and functions (e.g trigonometry, random number generator)
- The `SciPy` (scientific python) library builds on top of `NumPy` to provide a collection of numerical algorithms for:
  - Statistics (e.g correlation coefficient)
  - Signal processing (e.g Fourier transform, filtering)
  - Solving differential equations
  - Optimisation
  - ...

A taste of NumPy

```
import numpy as np

# 1D array/vector
x = np.array([1, 2, 3, 4])
x.min() # return min of array
x.max() # return max of array
x.sum() # sum all elements in array

# 2D array
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
x*x # element-by-element multiplication
x.shape # return dimensions of matrix
np.dot(x, x) # matrix multiplication
np.mat(x)*np.mat(x) # matrix multiplication (cast as matrix)
```

A taste of NumPy

```
# Generate random numbers
np.random.rand() # from a uniform distribution
np.random.randn() # from a normal distribution
np.random.randn(5, 5) # 5 x 5 matrix of random numbers

# Create number sequences
np.arange(0, 1, 0.1) # 0 to 1 in steps of 0.1
np.linspace(0, 1, 100) # 100 values between 0 and 1
np.logspace(0, 1, 10) # 10 values between 10^0 and 10^1
```

A taste of SciPy

```
import scipy.stats as sp

# Create two random arrays
x1 = np.random.randn(30)
x2 = np.random.randn(30)

# Correlation coefficientss
sp.pearsonr(x1, x2) # pearson correlation
sp.spearmanr(x1, x2) # spearman correlation
sp.kendalltau(x1, x2) # kendall correlation

# Statistical tests
sp.ttest_ind(x1, x2) # independent t-test
sp.mannwhitneyu(x1, x2) #  Mann-Whitney rank test
sp.wilcoxon(x1, x2) # Wilcoxon signed-rank test

# Least-squares regression
sp.linregress(x1, x2)
```

Predator prey equations (Lotka Volterra)

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \alpha u - \beta uv$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\gamma v + \delta uv$$

Where:

- u: is the number of prey (e.g rabbits)
- v: is the number of predators (e.g foxes)
- α: prey growth rate in the absence of predators
- β: dying rate of prey due to predation
- γ: dying rate of predators in the absence of prey
- δ: predator growth rate when consuming prey

Predator prey equations in Python

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \alpha u - \beta uv$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\gamma v + \delta uv$$

```python
def predator_prey(x, t):
    """
    Predator prey model (Lotka Volterra)
    """
    # Constants
    alpha = 1
    beta = 0.1
    gamma = 1.5
    delta = 0.075

    # x = [u, v] describes prey and predator populations
    u, v = x

    # Define differential equation (u = x[0], v = x[1])
    du = alpha*u - beta*u*v
    dv = -gamma*v + delta*u*v

    return du, dv
```

Solve differential equations

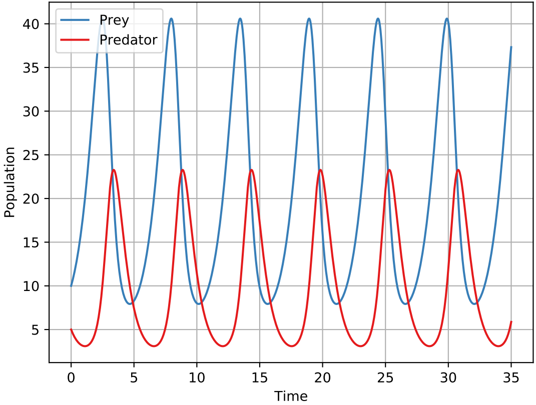
```
from scipy.integrate import odeint

time = np.linspace(0, 35, 1000) # time vector
init = [10, 5] # initial condition: 10 prey, 5 predators
x = odeint(predator_prey, init, time) # solve
```

Solve differential equations

```
from scipy.integrate import odeint

time = np.linspace(0, 35, 1000) # time vector
init = [10, 5] # initial condition: 10 prey, 5 predators
x = odeint(predator_prey, init, time) # solve
```



Fourier transform

```
from scipy.fftpack import fftfreq, fft

# Create frequency vector
N = len(time)
freq = fftfreq(N, np.mean(np.diff(time)))
freq = freq[range(int(N/2))]

# Compute Fast Fourier Transform
y = fft(x[:, 0])/N # compute and normalise fft
y = y[range(int(N/2))] # keep only positive frequencies
```

Fourier transform