Python for scientific research

Number crunching with numpy and scipy

Bram Kuijper

University of Exeter, Penryn Campus, UK

March 20, 2019

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

```
1  import numpy as np
2
3  # 1D array/vector
4  x = np.array([1, 2, 3, 4])
5  x.min() # return min of array
6  x.max() # return max of array
7  x.sum() # sum all elements in array
8
9  # 2D array
10 x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
11 x*x # element-by-element multiplication
12 x.shape # return dimensions of matrix
13 np.dot(x, x) # matrix multiplication
14 np.mat(x)*np.mat(x) # matrix multiplication (cast as matrix)
```

A taste of NumPy

```
1 # Generate random numbers
2 np.random.rand() # from a uniform distribution
3 np.random.randn() # from a normal distribution
4 np.random.randn(5, 5) # 5 x 5 matrix of random numbers
5
6 # Create number sequences
7 np.arange(0, 1, 0.1) # 0 to 1 in steps of 0.1
8 np.linspace(0, 1, 100) # 100 values between 0 and 1
9 np.logspace(0, 1, 10) # 10 values between 10^0 and 10^1
```

A taste of SciPy

```
1  import scipy.stats as sp
2
3  # Create two random arrays
4  x1 = np.random.randn(30)
5  x2 = np.random.randn(30)
6
7  # Correlation coefficientss
8  sp.pearsonr(x1, x2) # pearson correlation
9  sp.spearmanr(x1, x2) # spearman correlation
10 sp.kendalltau(x1, x2) # kendall correlation
11
12 # Statistical tests
13 sp.ttest_ind(x1, x2) # independent t-test
14 sp.mannwhitneyu(x1, x2) #  Mann-Whitney rank test
15 sp.wilcoxon(x1, x2) # Wilcoxon signed-rank test
16
17 # Least-squares regression
18 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

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

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

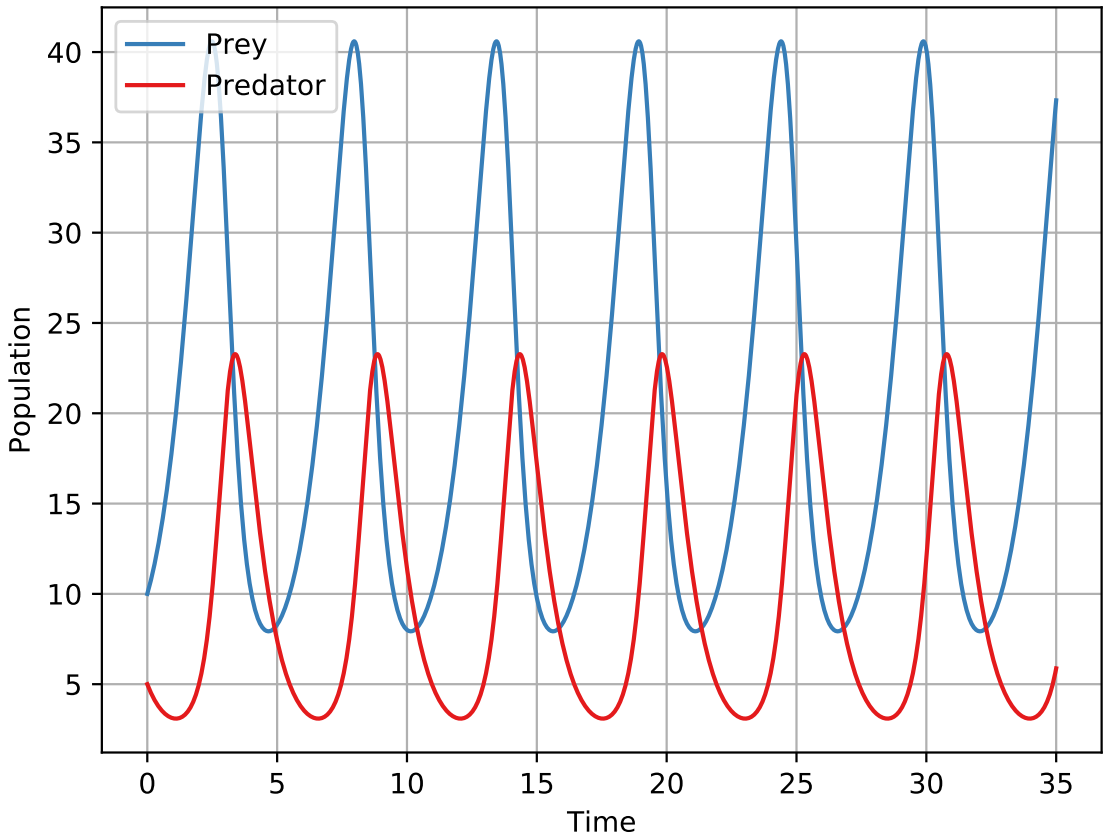
```
1  def predator_prey(x, t):
2      """
3      Predator prey model (Lotka Volterra)
4      """
5      # Constants
6      alpha = 1
7      beta = 0.1
8      gamma = 1.5
9      delta = 0.075
10
11     # x = [u, v] describes prey and predator populations
12     u, v = x
13
14     # Define differential equation (u = x[0], v = x[1])
15     du = alpha*u - beta*u*v
16     dv = -gamma*v + delta*u*v
17
18     return du, dv
```

Solve differential equations

```
1 from scipy.integrate import odeint
2
3 time = np.linspace(0, 35, 1000) # time vector
4 init = [10, 5] # initial condition: 10 prey, 5 predators
5 x = odeint(predator_prey, init, time) # solve
```

Solve differential equations

```
1 from scipy.integrate import odeint
2
3 time = np.linspace(0, 35, 1000) # time vector
4 init = [10, 5] # initial condition: 10 prey, 5 predators
5 x = odeint(predator_prey, init, time) # solve
```



Fourier transform

```
1  from scipy.fftpack import fftfreq, fft
2
3  # Create frequency vector
4  N = len(time)
5  freq = fftfreq(N, np.mean(np.diff(time)))
6  freq = freq[range(int(N/2))]
7
8  # Compute Fast Fourier Transform
9  y = fft(x[:, 0])/N # compute and normalise fft
10 y = y[range(int(N/2))] # keep only positive frequencies
```

Fourier transform