Lab notes: CE 31500, Computational Methods in Civil Engineering

Nir Krakauer

September 13, 2023

1 Introduction to Python

- Commands to conveniently carry out engineering computations, beyond what standard Python offers, are found in *libraries* (with sublibraries) included in add-on *packages*. In this course, we'll concentrate on Math, which provides basic numerical functions, NumPy, which provides an *array* type and array operations, SciPy, which provides a wider variety of numerical tools, and Matplotlib, which provides commands to make figures.

- a library needs to be imported before functions from it can be used in a program, with a few options:

  `import math` to use `math.sqrt(3)`

  `import math as m` to use `m.sqrt(3)`

  `from math import sqrt` (or `from math import *`) to use `sqrt(3)`

- `a = 1`

  Assignment (`=`); Enter to run command

  Variable names: Must start with a letter or _ (underscore); can also include numbers,

  Capitalization matters: after `a = 1`, `A` is still undefined

- Arithmetic operations: `+`, `-`, `*`, `/`, `**` (raising to a power)

- Numpy *arrays* that are vectors:

  `x = numpy.array([1, 2, 3, 4]); z = numpy.array((1, 2, 3, 4, 5));` (1-D arrays)

  `y = numpy.array([[5], [6], [7], [8]])` (a 2-D array, in fact a column (4×1) vector – `y.shape` will give an array's dimensions)

- Transpose: `y2 = numpy.transpose(y)` (a row vector)

- Component-wise arithmetic:

  Scalar-array: `x + 3`, `2*x`, `z ** 2`

  Array-array: `x * y2`, `y2 ** x`; `x + y` (*broadcasting*)

- Elements of arrays:

  Select a single element: `x[0]` (first value in `x`), `z[2]`, , `y[-1]` (last element), `y[-2]` (second to last)

  Select multiple elements: `x[[1, 2]]` or `x[[1:3]]`

- Functions that operate on arrays:

  `sum`, `numpy.mean`, `numpy.median`, `numpy.std`, `min`, `max`, `numpy.sort` ...

- Generating (row) vectors of equally spaced numbers

  Unit increment: `a = numpy.arange(0, 11)` or `a = numpy.linspace(0, 10, 11)`

  Non-unit increment: `b = numpy.arange(0, 10.5, 0.5)` or `b = numpy.linspace(0, 10, 21)`

- Other built-in functions: `numpy.sqrt`, `numpy.exp`, `numpy.log` (base e), `numpy.log10`, `numpy.sin` (argument in radians! – or `numpy.sin(numpy.deg2rad(b))` for `b` in degrees), `numpy.cos`, `numpy.tan`, `numpy.asin`, `numpy.round`, ...

- `help(`*function_name*`)`

- Define your own one-line function ('lambda function'):

  ```
  f = lambda x:  x - numpy.cos(x)
  ```

  ```
  c = 1; f(c)
  ```

  Functions can be applied to an array: `f(z)`

- Plots using Matplotlib

  `u = numpy.linspace(1, 40, 40); v = numpy.log(u); matplotlib.pyplot.plot(u, v)`

  Adjust markers and line: `matplotlib.pyplot.plot(u, v, '*')`, `matplotlib.pyplot.plot(u, v, ':*')`

  Labels: `matplotlib.pyplot.xlabel('u'); matplotlib.pyplot.ylabel('log of v'); matplotlib.pyplot.title('Example')`

  examples of character strings (between two single quotes)

  Change plot limits: `matplotlib.pyplot.xlim(0., 10.); matplotlib.pyplot.ylim(1., 3.)`

  Export plots: `matplotlib.pyplot.savefig('plot_name.png')` (or `.pdf`, etc.)

- Multiple curves in same plot:

  ```
  matplotlib.pyplot.plot(u, v, label="one"); matplotlib.pyplot.plot(u+1, v ** 2, label="two"); matplotlib.pyplot.legend()
  ```

- Other plot types:

  ```
  matplotlib.pyplot.bar(u, v)
  ```

  ```
  matplotlib.pyplot.hist(v)
  ```

- Function files

  First-line syntax:

  `def` *function_name*`(`*input1*`,` *input2*`):`

  The body of a Python function needs to be indented relative to the first line.

  Functions have their own internal workspace (variable names); only interact with the main workspace via inputs and outputs

  A function normally ends with a `return` statement specifying what it outputs

- Explanatory comments (include them!) follow `#`

  Each function or script should have a header comment that explains what it does

  Multiline block comments are preceded and followed by 3 single quotes (apostrophes)

- Script files: also `.py`, but don't start with the word `def`

  Series of Python commands carried out in order when the script is run

  Same effect as typing the commands in sequence (no separate workspace created)