numpy and scipy (1/2)

Computing Methods for Experimental Physics and Data Analysis

L. Baldini

luca.baldini@pi.infn.it

Università and INFN–Pisa

Compiled on October 17, 2019

Introduction

- Python is notorious for coming *with batteries included*...
- ...but as a data scientist numpy and and scipy will be your best friends!
- Among (many) other things, numpy offers:
  - a powerful n-dimensional array object
  - mathematical functions that interoperate natively with arrays
- And scipy provides:
  - integration
  - optimization (a.k.a. fitting)
  - interpolation
  - signal processing

numpy arrays

https://github.com/lucabaldini/cmepda/tree/master/slides/latex/snippets/numpy_arrays.py

```
1  import numpy as np
2
3  # Initialization from a list
4  a1 = np.array([1., 2., 3])
5  print(a1)
6
7  # Zeros, ones, and fixed values
8  a2 = np.zeros(10)
9  a3 = np.ones((2, 2))
10 a4 = np.full(7, 3.)
11 print(a2)
12 print(a3)
13 print(a4)
14
15 # Grids
16 a5 = np.linspace(0., 10., 11)
17 a6 = np.logspace(0., 1., 11)
18 print(a5)
19 print(a6)
20
21 [Output]
22 [1. 2. 3.]
23 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
24 [[1. 1.]
25  [1. 1.]]
26 [3. 3. 3. 3. 3. 3. 3.]
27 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
28 [ 1.          1.25892541  1.58489319  1.99526231  2.51188643  3.16227766
29   3.98107171  5.01187234  6.30957344  7.94328235 10.        ]
```

numpy arrays vs. Python lists

https://github.com/lucabaldini/cmepda/tree/master/slides/latex/snippets/numpy_arrays_vs_lists.py

```
1  import numpy as np
2
3  # arrays and lists seem similar...
4  l = [1., 2., 3.]
5  a = np.array(l)
6  print(l)
7  print(a)
8
9  # ...but they support basic arithmetic in a different fashion
10 print(l + l)
11 print(a + a)
12
13 [Output]
14 [1.0, 2.0, 3.0]
15 [1. 2. 3.]
16 [1.0, 2.0, 3.0, 1.0, 2.0, 3.0]
17 [2. 4. 6.]
```

- arrays and lists are fundamentally different objects
  - different footprint in memory, operate at different speed
  - arrays are homogeneous, lists don't need to
  - arrays offer a much more powerful indexing/slicing
  - arrays interoperate with numpy mathematical functions

Broadcasting

https://github.com/lucabaldini/cmepda/tree/master/slides/latex/snippets/numpy_arrays_broadcasting.py

```
1  import numpy as np
2
3  a1 = np.array([1., 2.])
4  a2 = np.array([[1., 2.], [3., 4.]])
5  c = np.pi
6
7  print(a1)
8  print(a2)
9  print(c)
10 print(a1 * a1)
11 print(a1 * c)
12 print(a1 * a2)
13
14 [Output]
15 [1. 2.]
16 [[1. 2.]
17  [3. 4.]]
18 3.141592653589793
19 [1. 4.]
20 [3.14159265 6.28318531]
21 [[1. 4.]
22  [3. 8.]]
```

- Under certain conditions, numpy can make operations on arrays of different shape
- This is extremely useful when vectorizing problems

Mathematical functions in numpy

https://github.com/lucabaldini/cmepda/tree/master/slides/latex/snippets/numpy_functions.py

```
1  import numpy as np
2  import math
3
4  a = np.array([0.1, 1., 10.])
5
6  print(np.log10(a))
7  print(np.exp(a))
8  print(np.sin(a))
9
10 print(np.log10(0.1))
11
12 print(math.log10(a))
13
14 [Output]
15 [-1.  0.  1.]
16 [1.10517092e+00 2.71828183e+00 2.20264658e+04]
17 [ 0.09983342  0.84147098 -0.54402111]
18 -1.0
19 Traceback (most recent call last):
20   File "snippets/numpy_functions.py", line 12, in <module>
21     print(math.log10(a))
22 TypeError: only size-1 arrays can be converted to Python scalars
```

- numpy mathematical functions interoperate natively with arrays
  - (and work on plain old numbers, too)

Arrays and masks

https://github.com/lucabaldini/cmepda/tree/master/slides/latex/snippets/numpy_masks.py

```
1  import numpy as np
2
3  a = np.linspace(0., 10., 11)
4  mask1 = a >= 2.5
5  mask2 = a < 8.5
6
7  print(a)
8  print(mask1)
9  print(mask2)
10 print(a[mask1])
11 print(a[mask2])
12 print(a[np.logical_and(mask1, mask2)])
13
14 [Output]
15 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
16 [False False False  True  True  True  True  True  True  True  True]
17 [ True  True  True  True  True  True  True  True  True False False]
18 [ 3.  4.  5.  6.  7.  8.  9. 10.]
19 [0. 1. 2. 3. 4. 5. 6. 7. 8.]
20 [3. 4. 5. 6. 7. 8.]
```

- Masks are a powerful tool in numpy
- They can replace conditional expressions in a for loop in vectorization context (more on this later)

Digression: pseudo-random number generators

```
1  import random
2  x = random.random()
```

- Every programming language comes with a Pseudo Random Number Generator (PRNG)
  - Python is no exception: `https://docs.python.org/3/library/random.html`
  - Mersenne-Twister, 53-bit precision, period of $2^{19937} - 1$.
- PRNGs are an interesting (and fun) subject by themselves:
  - Donald E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd Edition
  - M. Matsumoto and T. Nishimura, *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator*, ACM Transactions on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3–30 1998.
- A PRNG produces random floats uniformly in [0.0, 1.0).

Vectorization

a.k.a. avoid explicit for loops in Python whenever you can

https://github.com/lucabaldini/cmepda/tree/master/slides/latex/snippets/vectorization.py

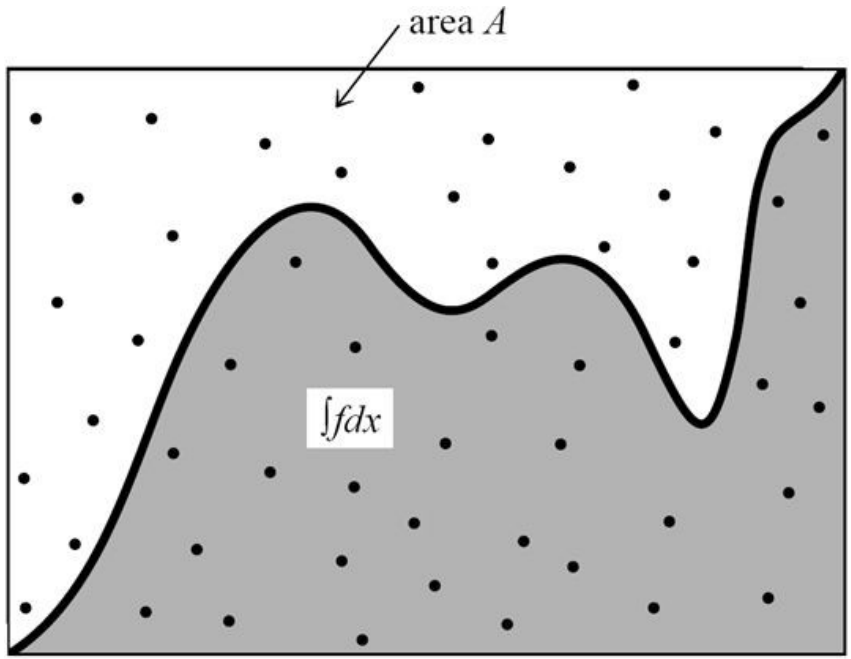
```
1  import random
2  import time
3  import numpy as np
4
5  # How many random numbers (uniformly distributed between 0 and 1) do you
6  # want to throw?
7  n = 1000000
8
9  # The slow way: explicit for loop in Python.
10 t0 = time.time()
11 x = []
12 for i in range(n):
13     x.append(random.random())
14 dt = time.time() - t0
15 print('Elapsed time: %.3f s' % dt)
16
17 # The quick way: vectorizing in numpy
18 t0 = time.time()
19 x = np.random.random(size=n)
20 dt = time.time() - t0
21 print('Elapsed time: %.3f s' % dt)
22
23 [Output]
24 Elapsed time: 0.137 s
25 Elapsed time: 0.015 s
```

How does vectorizaion work?

- Python is know to be slooow
  - This is the price you pay for being so beautiful and flexible
- Does it matter? If depends. . .
  - If you are parsing a text file or fetching a web page probably not
  - If you are performing a CPU-intensive processing on a TB of data probably yes
- What's so magic in using numpy?
- numpy is written in C as a Python extension
  - Routines are highly optimized to cruch numbers
  - When you perform an array operation in Python you are actually executing optimized C code
- Basic message: avoid for loops in pure Python when crunching numbers

How do I throw PRN with arbitrary pdf?

Hit or miss



- Hit or miss, aka acceptance/rejection method:
  - Enclose your pdf in a rectangle
  - Throw a x and a y
  - Accept x if $y \leq f(x)$
- This is horrible—please don't use it!

- Probability density function (pdf)

$$p(x) \quad (\geq 0)$$

- Cumulative function (cf)

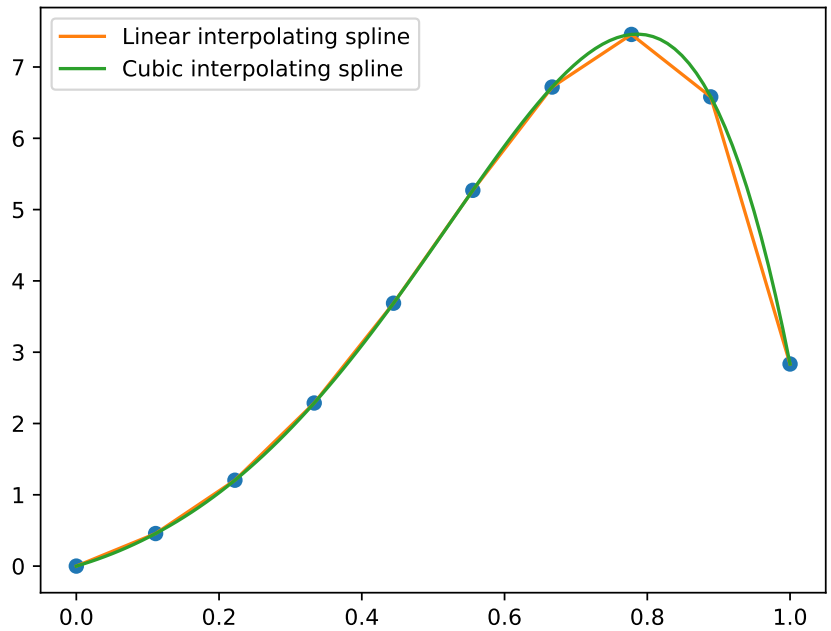
$$F(x) = \int_{-\infty}^{x} p(x')dx'$$

- Percent-point function (ppf)

$$x = F^{-1}(q)$$

- Awesome fact: if q is uniformly distributed in $[0, 1]$, then $x = F^{-1}(q)$ is distributed according to $p(x)$!

An interesting object: splines



- Defined piecewsise by polinomials of degree k ($k = 3$ fairly popular)
  - Interpolating: passing through a set of pre-defined points
  - First $k - 1$ derivatives continuos at the control points
- Superior to polynomial interpolation or curve fitting in many cases

Splines: construction and properties

https://github.com/lucabaldini/cmepda/tree/master/slides/latex/snippets/spline.py

```
1  import numpy as np
2  from scipy.interpolate import InterpolatedUnivariateSpline
3
4  x = np.linspace(0., 1., 10)
5  y = np.exp(3. * x) * np.sin(3. * x)
6
7  s1 = InterpolatedUnivariateSpline(x, y, k=1)
8  s3 = InterpolatedUnivariateSpline(x, y, k=3)
9
10 print(s1(0.234))
11 print(s1.integral(0.2, 0.8))
12
13 [Output]
14 1.3192110648078448
15 2.6659857771053925
```

- Evaluation is fairly inexpensive
  - If the input *x*-array is sorted can do a binary search in O(log(N)) complexity
- Derivatives and integrals are easy
  - Can be calculated *exactly* by means of elementary arithmetic operations

References

- https://numpy.org/
- https://www.scipy.org/
- https://docs.scipy.org/doc/numpy/reference/arrays.indexing.html
- https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html
- https://docs.scipy.org/doc/scipy/reference/interpolate.html