BCH2203 Python - 12. Randomness, structure, and more

Ramses van Zon

10 April 2024

This lecture. . .

We will look at a few additional useful things we can do in Python.

- Randomness
- Protein Structure using Biopython

Randomness

About Probability and Statistics

Probability

The chance of event A happening is its probability P(A). It lies between 0 and 1.

Probabilities of two events A and B can be independent: P(A and B) = P(A)P(B) or else, correlated.

Probabilties can be conditional: the change that A happened if B has happened: P(A|B).

Statistics

The study of collecting, analyzing, interpreting and presenting empirical data.

- Descriptive, determining properties of data and drawing conclusions about parameters.
- Inferential, trying to make preductions, like in machine learning.

Randomness

Statistics deals with uncertainties, whether from measurement errors or inherent randomness.
One often has to assume the randomness to have particular properties, ie. their distribution.

Distributions

- Random variables are numbers of sets that describe possible outcomes of an event.
- Random variables are described by distributions.
- A distribution describes how the probability of an event depends on the (value of the) outcome.

E.g. flipping a coin.

The possible outcomes are “head” or “tail”, the distribution if P(head)=50% and P(tail)=50%.

This is an example of a discrete probabilty distribution.

There are also continuous probability distributions.

E.g. What is the chance that it rains 1cm today?

But the chance that that happens is zero, because there are infinitely many possible outcomes.

Instead, for continuous variable, we need to ask better questions, like, what is the change that it rains between 1 and 1.01 cm? Or, what is the probability that it will rain more than 1cm.

To be able to deal with this, the distributions for continuous variables are probability density functions.

Descriptive Statistics: mean, mode, quantiles...

```
>>> import numpy as np
>>> import pandas as pd
>>> x = np.array([-4,-2,-1,0,1,5,10.])
>>> d = pd.DataFrame(x)
```

sum

```
>>> sum(x) # plain python
9.0
>>> x.sum() # numpy
9.0
>>> d.sum() # pandas
0    9.0
dtype: float64
>>>
```

mean

```
>>> x.mean()
1.2857142857142858
>>> d.mean()
0    1.285714
dtype: float64
```

quantiles

```
>>> sx = np.sort(x)
>>> qf = sx[len(sx)//4]
>>> qm = sx[len(sx)//2]
>>> qt = sx[-len(sx)//4]
>>> qf,qm,qt
(-2.0, 0.0, 5.0)
>>> d.quantile(
      [.25,.5,.75])
         0
0.25 -1.5
0.50  0.0
0.75  3.0
```

mode,hist

```
>>> h = np.histogram(x,8)
>>> counts = h[0]
>>> bnd = h[1]
>>> i = counts.argmax()
>>> mode=(bnd[i]+bnd[i+1])/
>>> mode
-1.375
```

More Statistics with scipy.stat

The SciPy package contains all of the statistical functions that you'll probably ever need.

- The scipy.stats subpackage is based around the idea of a 'random variable' type.
- A whole variety of standard distributions are available:
  - Continuous distributions: Normal, Maxwell, Cauchy, Chi-squared, Gumbel Left-scewed, Gilbrat, Nakagami, ...
  - Discrete distributions: Poisson, Binomial, Geometric, Bernoulli, ...
- The 'random variables' have all of the statistical properties of the distributions built into them already:
  cdf, pdf, mean, variance, moments, ...

Example

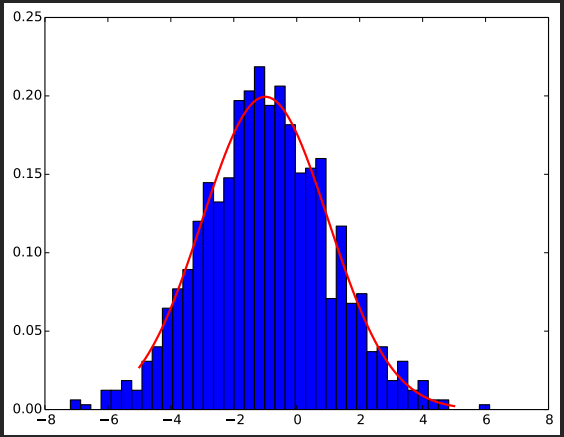
Let us create a normally distributed random variable with a mean of 1.0 and a standard deviation of 0.5.

```
>>> from scipy import stats
>>> nd = stats.norm(1, 0.5)
>>> nd.mean()
1.0
>>> nd.median()
1.0
>>> nd.std()
0.5
>>> nd.var()
0.25
```

Sampling

Sampling from the normal distribution

```
>>> from matplotlib.pyplot import plot,pause,hist
>>> from scipy.stats import norm
>>> from numpy import linspace
>>> # check pdf by sampling
>>> x = linspace(-5, 5, 100)
>>> plot(x, norm.pdf(x, loc = -1, scale = 2), 'r')
>>> samples = norm.rvs( size = 1000,
                        loc = -1, scale = 2)
>>> h = hist(samples,
             bins = 41, density=True)
>>> pause(.1)
>>>
```



- The `.rvs(size=...)` method draws `size` random samples from the distribution.
- `hist` is like `np.histogram` but also plots (`h` contains the histogram values).

Uh, Random Numbers from a Computer?

Computers cannot create truly random numbers because it is a deterministic machine.

- Either get from external source (radioactive decay, external activity)
- Or use **pseudo random number generators**.

  These create a set of numbers, starting from a 'seed'. Sequence is deterministic given the seed.
- These are good if statistical tests on the sequence show little deviation from randomness.
- General good ones are known by now. Even the default in python's `random` module is good (Mersenne-Twister).

Setting the seed

Sometimes you need consistency in your randomness:

- Pseudo-random numbers are generated from an initial 'seed'.
- This seed generates the first number, which is then used as the seed for the second number.
- If you need consistency in your random numbers (for debugging, for example), you can set the seed explicitly so that you get the same random numbers every time.

```
>>> norm.rvs()
1.74481176421648
>>> norm.rvs()
-0.7612069008951028
>>>
>>> numpy.random.seed(1)
>>> norm.rvs()
1.6243453636632417
>>>
>>> import scipy
>>> scipy.random.seed(1)
>>> norm.rvs()
1.6243453636632417
>>>
>>> import random as rd
>>> rd.seed(1)
>>> norm.rvs()
-1.0729686221561705
```

Other use case for random numbers

Machine learning

- We randomly divide data in training and test sets.
  - Some Machine Learning algorithms need random starting points (clustering) or random split points (decision trees)
  - We can also subsample or resample data to repeat an ML algorithm to validate the results.

Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a method to generate samples from general probability distributions.

It's considered one of the most important algorithms of the 20th century.

It's based on Baysian Inference.

Protein Structure using Biopython

So what about structure?

- When usin Biopython, we have focused mostly on sequences.
- However, The 3d structure, or shape, of a protein is essential to its biological function.
- Although the aminoacids in the sequence determine the shape in principle, there is no known easy relation between the sequence and the 3d structure.

  (not to mention that it depends on the surrounding liquid's content and conditions)

Computational approaches

- Molecular dynamics (MD) simulations starting from 'random coil' of protein sequence.

  One would never do MD in Python, performance is too critical, use GROMACS, NAMD, LAMMPS, ... written in C++.

- All-atom MD simultations can't reach time scales of protein folding, but can give 'bits' of the trajectory as input of more coarse grained models.
- Other computational studies start from the (experimentally) known structures.
- We can investigate the known structures in Python using Biopython, in particular, the PDB module.

Experimentally determined protein structures

- Experimental techniques like X-ray crystallography, NMR spectroscopy, and cryo-electron microscopy.
- Results are often submitted to, and can be retrieved from the Protein Databank.

  https://www.wwpdb.org
  https://www.ebi.ac.uk/pdbe
  https://www.rcsb.org
  https://bmrb.io
  https://pdbj.org
- You can download the whole database of ~1TB, weekly updated (but would you?).

Protein structure file formats

There are different file formats for protein structures:

- PDB: Legacy (i.e. old, unmaintained) because of limitations on what it can hold.
- PDBx/mmCif: Macro-Molecular Crystallographic Information File
- PDBML/XML format: An XML format
- MMTF: Highly compressed format
- Bundle: PDB formatted archive for large structures

To download, you'll need to know the PDB code.

This is an entry number is assigned to each structure.

Typically it is a number followed by 3 letters (E.g. 2HVF).

Look up on website.

The same molecule can have multiple entries.

How to download structures

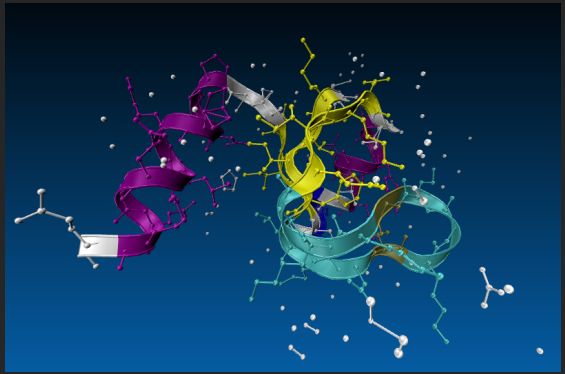
Use `PDBList` from `Bio.PDB` to download these.

```
>>> from Bio.PDB import PDBList

>>> pdbl = PDBList()

>>> filename = pdbl.retrieve_pdb_file("2HVF",
...                file_format="mmCif")
Downloading PDB structure '2HVF'...

>>> print(filename)
'/gpfs/fs1/home/s/scinet/rzon/hv/2hvf.cif'
```



Reading PDB files into Python

We can use PDB to load the file we just downloaded into a convenient data structure in Python:

```
>>> from Bio.PDB.MMCIFParser import MMCIFParser

>>> parser = MMCIFParser()

>>> structure = parser.get_structure("2hvf", filename)
```

There also exist `PDBParser` and `MMTFParser` for PDB Files and MMTF files

There are also functions for writing these formats, so you could e.g. filter out the water molecules and write the result to a file.

Analyzing structures with Bio.PDB

`structure` is structured as a "SMCRA" hierarchy:

- Structures contain models
- Models contain chains
- Chains contain residues
- Residues contain atoms
- An Atom contains a vector (its position) and other properties, but does not have children

```
>>> structure = parser.get_structure("2hvf", filenam
>>> model = structure[0]
>>> chain = model["A"]
>>> residue = chain[1]
>>> atom = residue['CA']
```

More about the structure of `structure`

https://biopython.org/wiki/The_Biopython_Structural_Bioinformatics_FAQ

Th. Hamelryck and B. Manderick,
PDB file parser and structure class implemented in Python, *Bioinformatics* 19, Issue 17, pp. 2308–2310.

Conclusion

This is the end...

We're at the last lecture.

Congrats everyone for getting to this point!

What did we do again?

Python programming

- Data structures
- File I/O
- Regular expressions
- Visualization
- Machine learning
- Randomness

Biochem specific

- Biopython
- Sequences
- Alignment
- Protein Structure

What's next?

There are quite a few topics we did not get into, e.g.

- Object orientied programming
- Generators
- Database interfaces
- Dates/times
- Parallel processing
- GUIs
- Debugging
- Deep learning

There should be a course evaluation coming your way.

Feel free to tell me how the course could be improved.

Although this was a relatively short course, I hope to have given you enough to start you off on your Python Biochem journey.