Randomseq: python command–line random sequence generator

Abstract

Randomly generated hypotheses are important in many sequence analysis studies as they represent null hypotheses. There are several existing tools to generate random sequences but each has its own strengths and weaknesses. Building upon the strengths and weaknesses of existing tools, a command–line random sequence generator, RANDOMSEQ, is presented. Generation of random sequences is versatile: (a) fixed or variable length nucleotide or amino acid sequences can be generated; (b) a variety of frequencies for sequence generation is accepted–source sequence, single or n-length nucleotide / amino acid frequencies; (c) generated sequences can be free of user–defined start or stop codons or both; (d) generated sequences can be flanked with randomly selected start and stop codons; and (e) one or more constant regions can exist within the sequence.

Keywords: fixed/variable length, constant random sequence, sequence generator

Introduction

Randomization is a crucial aspect of any statistical tests as experimental control or to generate null hypotheses. As such, at the core of many simulation experiments, such as Monte Carlo simulations, is a random number or sequence generator. For example, Monte Carlo simulations were used to study mutations and probability of obtaining single cells from serial dilutions. Random nucleotide and amino acid sequences had been used in many studies; thus, demonstrating the importance of random sequence generators in sequence analyses. For example, 500 thousand randomly generated DNA sequences of 50 nucleotides each were examined to test the relationship between DNA sequences and gene expressions, random peptide sequences had been used to test random raising secondary structures, and natural peptides had been shown to have more long–disordered regions than randomly generated peptide sequences. Several random sequence generators had been developed over the years. Many random nucleotide or amino acid generators provided minimal options, such as, fixed length and fixed GC content (http://www.faculty.ucr.edu/~mmaduro/random.htm), random selection from a given sequence (http://www.dave–reed.com/Nifty/randSeq.html), and random peptide generation with defined amino acid composition to output into FASTA format (https://web.expasy.org/randseq/). MacStAn aims to generate random nucleotide sequences for a pre–defined GC content and dinucleotide composition, allowing for maximal base repetitions and user–defined constant regions. Being a desktop application developed for classic Mac OS, MacStAn can be considered obsolete. RANDNA is a Windows desktop application developed in Borland Delphi for the generation of fixed length random nucleotide or amino acid sequences from a given frequency. Random ORF is a web tool to generate a random nucleotide sequence with a single open reading frame. NullSeq is a command–line random sequence generator implemented in Python, which is able to generate random nucleotide or amino acid sequences from a given frequency or source sequence. Being a command–line tool that writes the results into a FASTA file, NullSeq can be easily incorporated into analysis pipelines/tools.

Here, a command–line random sequence generator, RANDOMSEQ, is presented. RANDOMSEQ builds upon the strengths and weaknesses of existing tools to enable versatile generation of random sequences. Firstly, it can generate fixed or variable length nucleotide or amino acid sequences. Secondly, it accepts a variety of frequencies for sequence generation–source sequence, single or n-length nucleotide / amino acid frequencies. Thirdly, generated sequences can be free of user–defined start or stop codons or both. Fourthly, generated sequences can be flanked with randomly selected start and stop codons. Lastly, it can generate sequences with one or more constant regions within the sequence.

Implementation

RANDOMSEQ is implemented as a command–line tool using Python 3 and Python–Fire module (https://github.com/google/python–fire), which aims to simplify the implementation of command–line interface in Python 3. RANDOMSEQ is licensed under GNU General Public License version 3 for academic and non–commercial use, and the source code can be found in Bactome repository (https://github.com/mauriceling/bactome), allowing RANDOMSEQ to be executed in any platforms with Python 3 and Python–Fire installed. A random sequence is generated by concatenating randomly chosen atomic sequences from a bag, up to the required length. Random selection is performed using Python 3 built–in secrets module, which is designed to be a cryptographically strong pseudo–random number generator suitable for security and cryptography. Hence, this process is essentially selection with replacement. There are two methods to provide the bag of atomic sequences to RANDOMSEQ. The first method is to provide a delimited definition string through selection option. For example, A,100;T,200;G,300;C,400 defines 100 “A”s, 200 “T”s, 300 “G”s, and 400 “C”s with the bag of 1,000 atomic sequences. This presents the same interface for definitions of nucleotides (DNA, RNA, or ambiguous nucleotides) or amino acids frequencies. In addition, it does not restrict to mono–nucleotide or amino acids as n–length atomic sequences; such as, dinucleotides, and tripeptides; can be defined. For example, A,200;T,200;G,250;C,250;AT,50;TA,50. The second way is to provide a sequence through source_seq option, which takes precedence over selection option. For example, ATgTCAATggCTTAgCCC will internally generate A,4;T,5;g,4;C,5 as definition string However, the latter means cannot be used to define atomic sequences of more than 1 nucleotide or amino acid. Once the sequence generator is accepted–source sequence, single or n-length nucleotide / amino acid frequencies; (c) generated sequences can be free of user–defined start or stop codons or both; (d) generated sequences can be flanked with randomly selected start and stop codons; and (e) one or more constant regions can exist within the sequence.
atomic sequences had been added into the bag, the bag is shuffled 100 times using a pseudo-random number generator based on Merseenne twister, which produces 53-bit precision floats and has a period of 2¹⁹⁹⁳⁷−¹. For the purpose of generating random nucleotide sequences (DNA or RNA sequences), a list of start and stop codons can be independently defined as comma-delimited codons; such as, TTG, CTG, ATG as start codons. Options to independently allow or disallow the presence of start or stop codons in the generated sequences can be provided using start_codons and stop_codons options respectively. In addition, this allows the generated sequence to be flanked with either a randomly selected start codon at the beginning or stop codon at the end or flanking both ends.

**Generating random sequences**

The general command for generating random sequences is python randomseq.py <method> <options> where <method> is one of the four methods for generating random sequences; namely, shuffle, FLS (fixed length sequence), VLS (variable length sequence), and MS (mixed sequence). Each method will have its own set of options. The first and the simplest method is shuffle, which takes a sequence and returns a shuffled sequence. For example, python randomseq.py shuffle -sequence="GTCCGTAGTCACTAGCTGACTAGTCGATCGA TGATGCTACGATGCTACGATGCTGATGCTACGATCGATC GATGCTAGTGCA" will generate a shuffled sequence, such as CTATTCTAAACTGCTGTTAGAAGATGACGACCAGCAGCGA AGCGATGAAGCTACCGCCGATCGCTAGTTGCCTTTTGTAGT TGATGC. The other 3 methods (FLS, VLS, and MS) have three common options. Firstly, each method can generate one or more random sequences, defined using n option. Secondly, each of the generated sequences can be flanked with start codon at the beginning or stop codon at the end or both or no start/stop codons flanking. This is specified using cap_start and cap_stop options. Lastly, the generated sequences can be flanked with either a randomly selected start codon at the beginning or stop codon at the end or flanking both ends.

For FLS, a length option, which specifies the length of sequence to generate, must be specified. Putting together,

```python
python randomseq.py FLS
  -length=100
  -cap_start=True
  -cap_stop=True
  -fasta=True
  -prefix="Test"
  -allow_start=False
  -allow_stop=False
  -start_codons="TTG,CTG,ATG"
  -stop_codons="TAA,TAG,TGA"
  -selection=A,250;T,250;G,250;C,250
```

will generate 10 random sequence of 100 nucleotides each where each sequence will have uniform nucleotide distribution, and without start and stop codons within the sequence but flanked with a randomly selected start and stop codons from the specified list of start and stop codons. The output will be FASTA format where the description will be “Test_” followed by an incremental number. VLS differs with FLS as VLS takes min_length and max_length options to denote the minimum and maximum length of the sequence, instead of a length option. The length of the generated sequence will be uniformly distributed between the minimum and maximum length. Hence, the following will generate 10 random sequence of 90 to 110 nucleotides with the same specification as FLS above:

```python
python randomseq.py VLS
  -n=10
  -cap_start=True
  -cap_stop=True
  -fasta=True
  -prefix="Test"
  -allow_start=False
  -allow_stop=False
  -min_length=90
  -max_length=110
  -start_codons="TTG,CTG,ATG"
  -stop_codons="TAA,TAG,TGA"
  -selection=A,250;T,250;G,250;C,250
```

Mixed sequence (MS) method can be seen as a combination of FLS and VLS methods with constant regions. As such, it is a versatile method to generate any sequences. The specification of sequence is done through statement option, which consists of three possible definitions; namely, c for constant region, v for variable region, and o for operator region. A constant region is given as a string. For example, c (GAATTC) defines a constant region of GAATTC. Operator region accepts start and stop as parameter; for example, o (start) and o (stop) define a start codon and a stop codon respectively. Variable region definition takes 6 parameters, which corresponds to the 6 options of VLS; namely, min_length, max_length, allow_start, allow_stop, cap_start, and cap_stop. Hence, the following command

```python
python randomseq.py MS
  -n=10
  -cap_start=False
  -cap_stop=False
  -fasta=True
  -prefix="Test"
  -allow_start=False
  -allow_stop=False
  -min_length=90
  -max_length=110
  -start_codons="TTG,CTG,ATG"
  -stop_codons="TAA,TAG,TGA"
  -statement='v(10,15,False,False,False,False);o(start)'
```

will generate 10 random sequence of 100 nucleotides where each sequence will have uniform nucleotide distribution, and without start and stop codons within the sequence but flanked with a randomly selected start and stop codons from the specified list of start and stop codons.
Randomseq: python command-line random sequence generator

Acknowledgements

The author wishes to express his gratitude to A. Khan for his valuable comments on this manuscript.

Conflict of Interest

The author declares no conflict of interest.

References

1. Suresh K. An overview of randomization techniques: An unbiased assessment of outcome in clinical research. J Hum Reprod Sci. 2011;4(1):8–11.

2. Ling MHT, Ban Y, Wen H, et al. Conserved expression of natural antisense transcripts in mammals. BMC Genomics. 2013;14:243.

3. Hooton JWL. Randomization tests: statistics for experimenters. Computer Methods and Programs in Biomedicine. 1991;35(1):43–51.

4. Siderovski DP, Mak TW. RAMHA: a PC–based Monte–Carlo simulation of random saturation mutagenesis. Comput Biol Med. 1993;23(6):463–474.

5. Koyama K, Hokunan H, Hasegawa M, et al. Do bacterial cell numbers follow a theoretical Poisson distribution? Comparison of experimentally obtained numbers of single cells with random number generation via computer simulation. Food Microbiol. 2016;60:49–53.

6. Cuperus JT, Groves B, Kuchina A, et al. Deep learning of the regulatory grammar of yeast 5’ untranslated regions from 500,000 random sequences. Genome Res. 2017;27(12):2015–2024.

7. Tretynenko V, Vymetal J, Bednarová L, et al. Random protein sequences can form defined secondary structures and are well–tolerated in vivo. Sci Rep. 2017;7(1):15449.

8. Yu JF, Cao Z, Yang Y, et al. Natural protein sequences are more intrinsically disordered than random sequences. Cell Mol Life Sci. 2016;73(15):2949–2957.

9. Gast FU. A Macintosh program for the versatile generation of random nucleic acid sequences and their structural analysis. Comput Appl Biosc. 1994;10(1):49–51.

10. Piva F, Principato G. RANDNA: a random DNA sequence generator. In Silico Biol (Gedrukt). 2006;6(3):253–258.

11. Carr SM, Wareham HT, Craig D. A web application for generation of random DNA sequences with a single open reading frame: exemplars for genetics and bioinformatics education. CBE Life Sci Educ. 2014;13(3):373–374.

12. Liu SS, Hockenberry AJ, Lanchinietti A, et al. NullSeq: A Tool for Generating Random Coding Sequences with Desired Amino Acid and GC Contents. PLoS Comput Biol. 2016;12(11):e1005184.

13. Yao AC. Theory and application of trapdoor functions. In: 23rd Annual Symposium on Foundations of Computer Science. 1982;80–91.

14. Matsu mosu T, Nishimura T,bersenne twister: a 623–dimensionally equidistributed uniform pseudo–random number generator. ACM Transactions on Modeling and Computer Simulation. 1998;8(1):3–30.

15. Hoshino T, Inagaki F. Application of Stochastic Labeling with Random–Sequence Barcodes for Simultaneous Quantification and Sequencing of Environmental 16S rRNA Genes. PLoS ONE. 2017;12(1):e0169431.

16. Hoshino T, Hamada Y. Estimation of the influence of sequencing errors and distribution of random–sequence tags on quantitative sequencing. J Biosci Bioeng. 2017;124(3):359–364.

17. Halperin RF, Stafford P, Johnston SA. Exploring antibody recognition of sequence space through random–sequence peptide microarrays. Mol Cell Proteomics. 2011;10(3).

18. Neme R, Aramador C, Yildirim B, et al. Random sequences are an abundant source of bioactive RNAs or peptides. Nat Ecol Evol. 2017;1(6):217.

19. Yona AH, Alm EJ, Gore J. Random sequences rapidly evolve into de novo promoters. Nat Commun. 2018;9(1):1530.