Practical Session 1

Aim of the practical assignment

- Simulate a nucleotide sequence of length 10000 nucleotides so it has in probability:
- 10% of A
- 40% of G
- 30% of T
- 20% of G

Aim of the practical assignment

- To report
- Python code PROPERLY COMMENTED LINE BY LINE (i.e. before the line starts, say what is the next command going to do)
  - More than 20% of uncommented code = 0 points
- Features to be evaluated
  - Code readability
  - Code performance
  - Elegance of solution (usage of functions and classes)

Random multinomial generator

- From the Art of War: **“If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle.”**

- What do we need to generate a random sample with the expected nucleotide proportions?

Random multinomial generator

From Vose (article you have to read!)

A. *Rand*

Our description of *rand* follows that given by Knuth [2]. Let $prob$ and $alias$ be arrays which are initialized by *init*. The body of *rand* is

u =*uniform*(n)
$j = \lfloor u \rfloor$
If $(u - j) \leq prob_j$ then return
j else return $alias_j$.

In Python

- import random
- # integer sampled from uniform between 0 and 100
- print(random.randint(0,100-1))
- # double sampled from uniform between 0 and 1
- print(random.uniform(0,1))

Random multinomial generator

From Vose (article you have to read!)

B. Init

Our version of *init* proceeds in two stages. The first stage divides the indices of the input into two arrays, *small* and *large*, via the rule:

$$p_j > 1/n \;\Rightarrow\; j \in large$$
$$p_j \leq 1/n \;\Rightarrow\; j \in small.$$

The second stage uses the probability distribution p together with *small* and *large* to initialize the arrays *prob* and *alias*. The idea behind this stage is motivated by an analysis of *rand*.

There are two situations in which *rand* returns j:

- If $j = \lfloor u \rfloor$ and $(u - j) \leq prob_j$ then j is returned. This situation occurs with probability

$$\frac{1}{n}\, prob_j$$

In Python

- #Use lists to classify the indexes of p in large and small
- large = [];
- small = [];
- # Identify n
- n = len(p)
- # iterate over the n elements
- for j in range(n):
- # decide to assign j to large or small
- if x > y:
  - # use append()
- else
  - # use append()

Random multinomial generator

From Vose (article you have to read!)

$l = 0 \,;\, s = 0$
For $j = 0$ to $n-1$
if $p_j > \frac{1}{n}$
then $large_l = j \,;\, l = l+1$
else $small_s = j \,;\, s = s+1$
While $s \neq 0$ and $l \neq 0$
$s = s-1 \,;\, j = small_s$
$l = l-1 \,;\, k = large_l$
$prob_j = n * p_j$
$alias_j = k$
$p_k = p_k + (p_j - \frac{1}{n})$
if $p_k > \frac{1}{n}$
then $large_l = k \,;\, l = l+1$
else $small_s = k \,;\, s = s+1$
While $s > 0$ do $s = s-1 \,;\, prob_{small_s} = 1$
While $l > 0$ do $l = l-1 \,;\, prob_{large_l} = 1.$

In Python

- l = len(large)
- s = len(small)
- prob_j = [None]*n
- alias_j = [None]*n

Random multinomial generator

Alias Vose

$l = 0 \; ; \; s = 0$

For $\; j = 0 \;$ to $\; n - 1$

if $\; p_j > \frac{1}{n}$

then $\; large_l = j \; ; \; l = l + 1$

else $\; small_s = j \; ; \; s = s + 1$

While $s \neq 0 \;$ and $\; l \neq 0$

$s = s - 1 \; ; \; j = small_s$

$l = l - 1 \; ; \; k = large_l$

$prob_j = n * p_j$

$alias_j = k$

$p_k = p_k + (p_j - \frac{1}{n})$

if $\; p_k > \frac{1}{n}$

then $\; large_l = k \; ; \; l = l + 1$

else $\; small_s = k \; ; \; s = s + 1$

While $s > 0 \;$ do $\; s = s - 1 \; ; \; prob_{small_s} = 1$

While $l > 0 \;$ do $\; l = l - 1 \; ; \; prob_{large_l} = 1.$

Classical Alias

i	**A**	**C**	**T**	**G**	**Sum**
	0.1	0.3	0.2	0.4	1
	0.4	1.2	0.8	1.6	4
	Esa				
	0.4	1.2	0.8	1.6	

	-1	1	-1	1

Random multinomial generator

Alias Vose

$l = 0 \; ; \; s = 0$
For $j = 0$ to $n - 1$
if $p_j > \frac{1}{n}$
then $large_l = j \; ; \; l = l + 1$
else $small_s = j \; ; \; s = s + 1$
While $s \neq 0$ and $l \neq 0$
$s = s - 1 \; ; \; j = small_s$
$l = l - 1 \; ; \; k = large_l$
$prob_j = n * p_j$
$alias_j = k$
$p_k = p_k + (p_j - \frac{1}{n})$
if $p_k > \frac{1}{n}$
then $large_l = k \; ; \; l = l + 1$
else $small_s = k \; ; \; s = s + 1$
While $s > 0$ do $s = s - 1 \; ; \; prob_{small_s} = 1$
While $l > 0$ do $l = l - 1 \; ; \; prob_{large_l} = 1.$

Random multinomial generator

Python. Work with classes and functions!

```
import random

class RandomMultinomial(object):

    '''
    Constructor
    '''
    def __init__(self, p):
        self.p = p
        self.n = len(self.p)
        self.alias = [None] * self.n
        self.prob_j = [None]* self.n
        self.build_alias()

    '''
    Create the alias for p
    '''

    def build_alias(self):
        large = []
        small = []
        for j in range(self.n):
```

Random multinomial generator

Python. Work with classes and functions!

```
def main():
    p = [0.1,0.2,0.2,0.5]
    alias = RandomMultinomial(p)
    count = [0]*len(p)
    for i in range(100000):
        j = alias.sample()
        count[j] = count[j] + 1.0/100000.0
    print(count)


if __name__ == "__main__":
    main ()
```