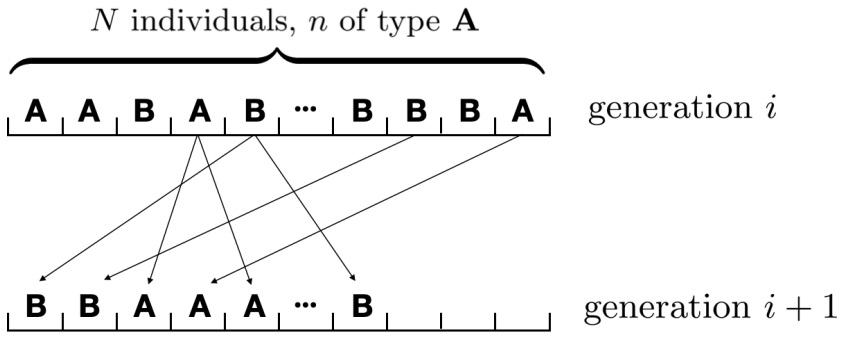
Arnold Sommerfeld Center for Theoretical Physics
Ludwig-Maximilians-Universität München

Prof. Erwin Frey
B. Nettuno, T. Roth, D. Toffenetti

Nonequilibrium Field Theories and Stochastic Dynamics
Sheet 0

Exercise 1 – Binomial Sampling Process (Genetic Drift)

Consider a population ("generation") with N individuals, which are either of type **A** or type **B**. Initially, there are n individuals of type **A** and $(N-n)$ individuals of type **B**. The next generation is obtained as follows: we randomly choose an individual from the present generation and add its copy to the new generation. We repeat the process until there are N individuals in the new generation. This can be viewed as a binomial sampling with replacement, where one picks an **A** individual with probability p_n, and a **B** individual with probability $q_n = 1-p_n$.



a) What is the conditional probability of having n' **A** individuals in the next generation if there are n of them in the present generation? Argue that

$$p(n'|n) = \binom{N}{n'} p_n^{n'} (1-p_n)^{N-n'}, \quad \text{where} \quad p_n = n/N \quad . \tag{1}$$

b) Let n_i be the number of **A** individuals in the ith generation and assume that we know the initial population n_0, i. e.the number of **A** individuals before generating the first generation. Under what condition can we write

$$p(n_{i+1}|n_0) = \sum_{n_i=0}^{N} p(n_{i+1}|n_i) p(n_i|n_0) \quad ? \tag{2}$$

Does the population dynamics considered here fulfill this condition?

c) Show that this sampling process is a *martingale*, i.e. the expected value for the number of **A** individuals in the next generation equals the number of **A** individuals in the present generation: $\mathbb{E}(n_1|n_0) = n_0$. Then show that this expected value is the same for any future generation i, that is $\mathbb{E}(n_i|n_0) = n_0$.
Hint: you may need to use part **b)** *to show it for* $i > 1$.

d) The diversity of a population (in general called *heterozygocity*) is defined as the probability that two randomly picked individuals are different. In our case, the heterozygocity H for a population with n individuals of type **A** is given by

$$H(n) = 2p_n(1-p_n) = 2\frac{n}{N}\left(1-\frac{n}{N}\right) \tag{3}$$

Argue why $H(n)$ is the correct expression.

Let us now denote by $H_i \coloneqq \mathbb{E}(H(n_i)|n_0)$ the expected value of the heterozygocity for the ith generation. Show that for the first generation $H_1 = H(n_0)\left(1-\frac{1}{N}\right)$. Then calculate the expected heterozygocity H_i for any future

generation i.
Hint: as before, you may need to use part **b)** *to find* H_i.

e) Now consider the ith generation of the population. Argue why for large populations ($N \gg 1$) the expected heterozygocity of the ith generation is approximately $H_i \approx H_0 e^{-i/N}$, where $H_0 = H(n_0)$ is the initial heterozygocity. After what time (counted in generations) do you expect the heterozygocity to be completely lost?

f) The states $n = 0$ and $n = N$ are absorbing states, i. e.a system entering into one of these states cannot leave them anymore. Argue why.

g) In the limit $i \to \infty$ (after many generations), the system will almost surely (with probability 1) end up in either of these absorbing states, and the stationary probability distribution will be given by $p^*(n) = (1-\pi)\delta_{n,0} + \pi\delta_{n,N}$, where π is called the fixation probability of species **A** (the probability of **A** eventually taking over the whole population) and $\delta_{i,j}$ is the Kronecker delta.

Calculate π.
Hint: this can be tricky, unless you make use of **c)**.

Exercise 2 – Random numbers*

This exercise is optional, but we encourage you to try it, especially if you have little experience with programming. We have put a short guide on how to use Python and Jupyter on Moodle. That material and the solution to this exercise will be covered in the central tutorial next week (Wednesday, 26.10.2022, 12:00 c.t., room A348). If you run into problems with the numerical parts of the exercise, please ask the tutors or your fellow students for help. Note that part ***d)*** *does not require numerical calculations.*

The most fundamental part of the numerical implementation of stochastic processes is the generation of random numbers. Normally, the realization of a stochastic simulation requires the drawing of a large number of independent random numbers, i. e.one has to come up with a possibility to generate uncorrelated random numbers on a deterministic machine (computer) as fast as possible. In this exercise, we will go through one implementation of a standard random number generator from the 1970's and its flaws.

a) The generation of truly random numbers is impossible on deterministic machines such as computers. One therefore creates 'pseudo-random' numbers by implementing a recurrence relation with pseudo-random properties, which means the sequence of numbers we get from applying this relation many times *looks* random, but it really isn't.
A simple choice of such a relation for generating pseudo-random numbers is

$$x_{n+1} = (a\, x_n + c) \bmod m\ , \tag{4}$$

where a, c, and m are integer parameters and mod denotes the *modulo operation*.[1] Use the pseudo-random number generator defined in Eq. 4 to complete the function `randomInt` in the provided Jupyter notebook. Use your random number generator with the parameters $a = c = x_0 = 7$ and $m = 10$ and generate a sequence of random numbers.
What do you observe?

b) Due to hardware-specific performance reasons, originally the following parameters have been proposed:

$$a = 2^{16} + 3, \quad m = 2^{31}, \quad c = 0$$

Use those parameters for the following tasks:

- Implement the function `randomReal` and generate a sequence of random reals in the interval $[0, 1)$.
- Plot your result in a histogram and calculate the mean and the variance. What are your conclusions? Does the result fulfill your expectations?
- Generate a sequence of 10^6 random numbers using your implementation and the the built-in random number generator of Numpy (`np.random.random`). Measure the evaluation time and compare the performance.

c) In the following, we will reveal some flaws of the naive pseudo-random number generator.

[1] $n \bmod m$ computes the remainder of dividing n by m. E. g.$15 \bmod 4 = 3$.

- Generate a sequence of random reals $(x_0, x_1, x_2, ...)$ and partition the data into sets $\mathbf{y}_k = (x_{3k}, x_{3k+1}, x_{3k+2}) \equiv (x_{x,k}, x_{y,k}, x_{z,k})$. Use the scatter plot function of matplotlib and plot the points $\mathbf{y}_k$. What do you observe and conclude? Dose the data look random?

  Hint: Play around with the viewpoint.

- Show that the pseudo-random numbers generated with the parameter choice given in **b**) obey the relation

$$x_{n+2} = (6\, x_{n+1} - 9\, x_n) \bmod 2^{31} \,. \tag{5}$$

  Hint: First, you might have to prove a few modulo calculation rules. In the following let $a, b, m \in \mathbb{N}$ and $k \in \mathbb{Z}$. Then

$$(a + k\, m) \bmod m = a \bmod m \tag{6}$$
$$((a \bmod m) + (b \bmod m)) \bmod m = (a + b) \bmod m \tag{7}$$
$$((a \bmod m)\, (b \bmod m)) \bmod m = (ab) \bmod m \,. \tag{8}$$

- Using this relation, explain why all points $\mathbf{y}_k$ are located in hyperplanes of the form $h_k = (x_{z,k} - 6\, x_{y,k} + 9\, x_{x,k})$ with $h_k \in \mathbb{Z}$. Use the `plot` function from matplotlib to verify this result.
- How can you use your random number generator to approximate the value of π from the definition of a circle? Check what value you get (10^7 points are sufficient) and compare it to the reference value `numpy.pi`. Again, compare your implementation to Numpy's built-in random number generator.

d) The previously considered random number generators yield uniformly distributed (pseudo-)random numbers. However, in most real-world cases, non-uniform random variables are required. Here, you will learn about a general method of how random numbers with non-uniform distributions can be generated from uniformly distributed ones.

Consider a random variable X and let g be a bijective and monotonous function. Let the random variable Y be defined as $Y := g(X)$. Show that the probability density distribution f_Y is given by

$$f_Y(y)\, dy = f_X\big(g^{-1}(y)\big) \left| \frac{d}{dy} g^{-1}(y) \right| dy \,. \tag{9}$$

Using this result, find the function g that transforms the above uniformly random numbers $X \in [0, 1)$ into exponentially distributed random numbers with $f_Y(y) = \exp(-y)$.

Hints: Start with the cumulative distribution $F_Y(y) \equiv P\{Y \leq y\}$ and express it in terms of F_X. How does $f_X\big(g^{-1}(y)\big)$ look like for a uniform distribution?

Your solutions should be handed in by uploading them to Moodle by **Wednesday, 30th April 2025, 10:00 am**.