Exercise 1

Have a look at the interfaces

`MonteCarloEvaluationsInterface`

and

`MonteCarloEvaluationsWithExactResultInterface`

you can find in the package

`com.niklasweber.montecarlo`

The first interface provides methods to experiment with the Monte-Carlo implementation of a possibly general class of problems, from pricing to the computation of an integral. The second interface extends the first one. For this second interface, we suppose that we already know the exact value of the quantity we approximate by Monte-Carlo, so that we can compute the absolute errors of our approximations.

Write two classes `MonteCarloIntegrationPowerFunction` and `MonteCarloPi` implementing this second interface and computing the integral $\int_0^1 x^\alpha dx$, $\alpha \in (0, \infty)$, in the case of `MonteCarloIntegrationPowerFunction`, and π in the way you have seen in the lecture in the case of `MonteCarloPi`.

Test then your results by running the classes

`com.niklasweber.montecarlo.MonteCarloPiCheck`

and

`com.niklasweber.montecarlo.MonteCarloIntegrationCheck`

in the `test` part of the project. Note that these two classes will give errors until you construct your classes.

Hints:

- Here you have total freedom about the choice of your design. A suggestion is: you can note that the implementation of almost all the methods of the interfaces does not depend on the specific Monte-Carlo computation, as they might use the vectors hosting the computed values. You can think to implement these methods in some abstract class / abstract classes implementing the correspondent interface(s), and then let `MonteCarloIntegrationPowerFunction` and `MonteCarloPi` extend this class / one of these classes. In this way, they would automatically implement the interface(s).
- Consider that the methods of the present interfaces do not take the number of Monte-Carlo computations as argument. So this should be a field of your classes. You can decide which access modifier to assign it according to the design of your solution (i.e., if you have or not the parent abstract classes, for example).
- You can find some methods to perform your experiments (for example, computing the standard deviation, or the average error, of the Monte-Carlo computations) in the class

  `com.niklasweber.usefulmethodsmatricesandvectors.UsefulMethodsMatricesAndVectors`

  in the exercise project (so you don't have to pull the Java project again).

Exercise 2

Write another class `MonteCarloPiFromHypersphere` implementing

`com.niklasweber.montecarlo.MonteCarloEvaluationsWithExactResultInterface`.

This class must provide approximations of π from the Monte-Carlo approximation of the volume of the unit hypersphere

$$\{(x_1, \dots, x_d) \in \mathbb{R}^d | x_1^2 + \dots + x_d^2 \leq 1\}$$

for a given dimension d. The dimension must be a field of the class initialized in the constructor.

Hints: this is basically a generalization for higher dimensions of the Monte-Carlo implementation of π in the exercise above. The volume of the hypersphere is

$$V_d = \int_{-1}^{1} \cdots \int_{-1}^{1} \mathbf{1}_{\{x_1^2 + \dots + x_d^2 \leq 1\}} dx_1 \dots dx_d = 2^d \int_{0}^{1} \cdots \int_{0}^{1} \mathbf{1}_{\{(2(x_1 - 0.5))^2 + \dots + (2(x_d - 0.5))^2 \leq 1\}} dx_1 \dots dx_d.$$

It holds

$$V_{2k} = \frac{\pi^k}{k!},$$
$$V_{2k+1} = \frac{2(4\pi)^k k!}{(2k+1)!}.$$

for $k \geq 1$ natural number.

Test this implementation taking inspiration from the classes for the tests above.

Exercise 3

Write a class `HaltonSequencePiFromHypersphere`, providing an approximation of the value of π via the approximation of the integral

$$V_d = \int_{-1}^{1} \cdots \int_{-1}^{1} \mathbf{1}_{\{x_1^2 + \dots + x_d^2 \leq 1\}} dx_1 \dots dx_d = 2^d \int_{0}^{1} \cdots \int_{0}^{1} \mathbf{1}_{\{(2(x_1 - 0.5))^2 + \dots + (2(x_d - 0.5))^2 \leq 1\}} dx_1 \dots dx_d$$

and the equations

$$V_{2k} = \frac{\pi^k}{k!},$$
$$V_{2k+1} = \frac{2(4\pi)^k k!}{(2k+1)!}.$$

for $k \geq 1$ natural number, where the evaluations points $(x_1^i, \dots, x_d^i)$, $i = 1, \dots, n$, with n number of sample points, are now provided by an Halton sequence with a given d-dimensional base.

You can write a class `HaltonSequence`, with a method providing the sample points, or directly use the one in

`info.quantlab.numericalmethods.lecture.randomnumbers`

in the `numerical-methods-lecture` project.

The class `HaltonSequencePiFromHypersphere` must also provide a `public` method which returns the error in the approximation (note that here, for a given base, only one value of the approximation is produced, so it does not make sense to consider a vector of approximations as for the Monte-Carlo method).

Experiment on the quality of the approximation of the two methods by printing the average error produced by `MonteCarloPiFromHypersphere` of Exercise 2 for 100 computations and the error given by `HaltonSequencePiFromHypersphere`, using in both cases 100000 sample points, for different dimensions.

Regarding the choice of the base of `HaltonSequencePiFromHypersphere`, consider the following cases:

- all the elements of the base are equal to each other (for example, `base = {2,2,2,2}` for dimension 4);
- the elements of the base are different to each other, but share common divisors (for example, `base = {2,4,6,8}` for dimension 4);
- the elements of the base are different to each other, and do not share common divisors (for example, `base = {2,3,5,7}` for dimension 4).

What do you observe regarding the approximation error? How can you explain this behaviour?

Exercise 4

Write a class with two `public` methods, returning the discrepancy and the star discrepancy, respectively, of a set $\{x_1, \ldots, x_n\}$ of one-dimensional points. These points are not supposed to be sorted when given. Test your implementation computing the discrepancy and the star discrepancy of the sets

$$A_1 = \{1/8, 1/4, 1/2, 3/4\}$$

and

$$A_2 = \{1/4, 1/2, 5/8, 3/4\}.$$

You have to get

$$D(A_1) = \frac{3}{8}, \quad D^*(A_1) = \frac{1}{4}, \quad D(A_2) = \frac{1}{2}, \quad D^*(A_2) = \frac{1}{4}.$$

Hint:

The discrepancy may be computed as

$$D(\{x_1, \ldots, x_n\}) = \max_{a \in \{0, x_1, \ldots, x_n\}} \max_{b \in \{x_1, \ldots, x_n, 1\}, b > a} \max \left(b - a - \frac{|x_i \in (a,b)|}{n}, \frac{|x_i \in [a,b]|}{n} - (b-a) \right). \quad (1)$$

One can then use representation (1) by first computing

$$\max_{b \in \{x_1, \ldots, x_n, 1\}, b > a} \max \left(b - a - \frac{|x_i \in (a,b)|}{n}, \frac{|x_i \in [a,b]|}{n} - (b-a) \right) \quad (2)$$

for $a \in \{x_1, \ldots, x_n\}$ fixed, and then computing the discrepancy as the maximum between the star discrepancy, which is

$$\max_{b \in \{x_1, \ldots, x_n, 1\}, b > a} \max \left(b - \frac{|x_i \in (0,b)|}{n}, \frac{|x_i \in [0,b]|}{n} - b \right),$$

and the maximum of the values of (2).