Applied Mathematics for Computer Science IGS

Exercise Sheet 11 Hand in deadline: June 06, 2022 at 23:59

General Information: This assignment contains written and/or programming tasks. Combine all the answers to the written tasks in a single PDF document, named `{lastname}-written.pdf`. You can also scan or take pictures of (readable) handwritten papers. JPEG/PNG image files are accepted in this case and they should be named `{exercisenumber}-{lastname}-written.{jpeg/png}`. Make sure that we can follow the manual calculations. Do not combine too many small steps into one. The programming tasks have to be solved in *Julia* and the source code files have to be submitted using the following naming scheme: `{exercisenumber}-{lastname}.jl`.

(1) (2.5 points) Generate random numbers using your own random number generator implementations. Use the `rng.jl` file for this exercise. The final result should look similar to Figure 1 for each of the implemented random number generators.

a) (0.25 points) Use the `rand` function to create a uniform distribution.

Use the `uniform(N::Int)::Vector{Float64}` signature.

b) (0.75 points) Implement the Middle-square method as described in the lecture slides.

Use the `mid_square(N::Int, seed::Int=34345669)::Vector{Float64}` signature.

Hint: You might want to make use of the `digits` function.

c) (1.25 point) Implement a generator function for the Halton sequence as described in the lecture slides.

Use the `halton(N::Int, base::Int=3)::Vector{Float64}` signature.

Hint: You might want to make use of the `digits` function.

d) (0.25 points) Implement the two-dimensional version for each of the random number generators. Use the respective templates given in the template.

In addition to the three methods above, there is an included `urand.jl` file which contains random numbers that were generated using the operating system's kernel random number generator[1]. You can access these random numbers by calling `urand(N::Int)::Vector{Float64}`. Also implement the 2D version for `urand` by re-using the 1D version.

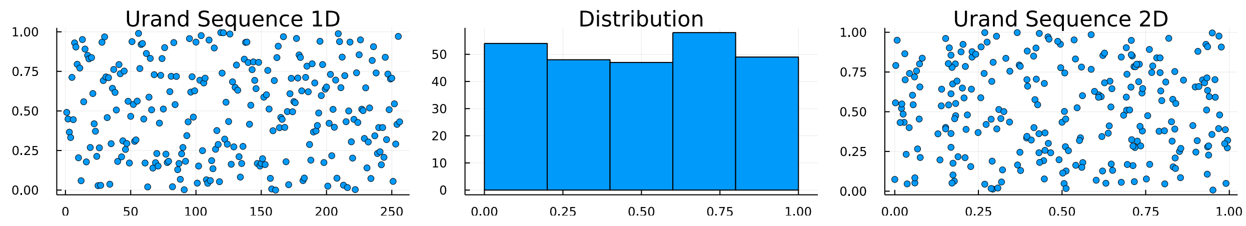

Figure 1: A sequence of random numbers.

(2) (3 points) Approximate a difficult to solve integral using numerical approaches. Use the `integration.jl` file for this exercise. Your solution should look similar to Figure 2. Additionally, you are provided multiple datasets of randomly generated numbers.

a) (1 point) Given a function $f(x) = e^{-x^2}$, compute the following integral by hand:

$$\int_0^1 f(x)\mathrm{d}x.$$

[1] https://linux.die.net/man/4/urandom

You might find it hard to solve this analytically. Instead, you can approximate it using a power series:

$$f(x) \approx \sum_{k=0}^{N} \frac{(-x^2)^k}{k!},$$

with $N \in \mathbb{N}$. Choose whatever amount of terms you deem appropriate and compute the approximation by hand. Try to find a generic solution (for arbitrary N and x) and implement it in `power_series(a::Float64, b::Float64)::Float64`.

b) (1 point) Approximate $\int_0^1 f(x)\mathrm{d}x$, using Monte-carlo integration:

$$\int_a^b f(x)\mathrm{d}x \approx (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i),$$

where x_i denotes a point out of N randomly chosen points $\in f(x)$. Implement this in `mc_integration(a::Float64, b::Float64, N::Int)::Float64`.

c) (1 point) Approximate $\int_0^1 f(x)\mathrm{d}x$, using a different form of Monte-carlo integration (also called integration by darts):

i. Generate random points $(x, y) \in P$ with:

$$0 \leq x \leq 1 \quad \text{and} \quad 0 \leq y \leq \max(f(x)).$$

ii. Count $p \in P$ for which $f(p.x) \geq p.y$ holds.

iii. Divide the count by amount of generated random points.

Implement this in

`mc_integration_by_darts(a::Float64, b::Float64, N::Int)::Float64`

Note: The process is different when the function can take negative values.

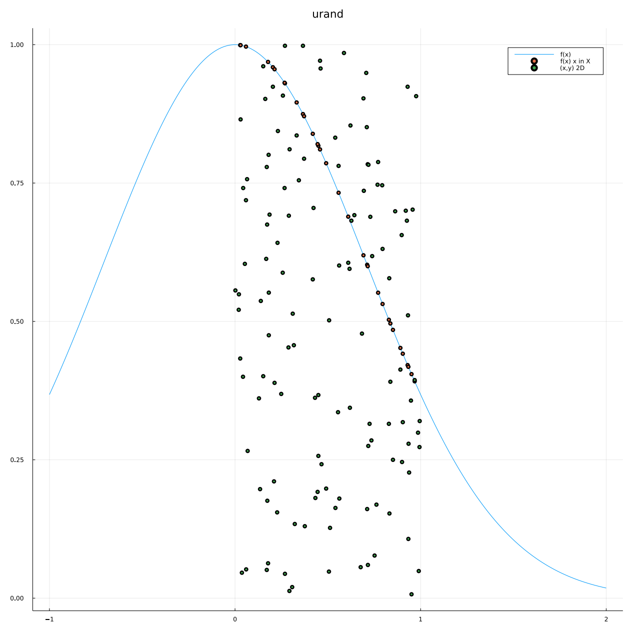

Figure 2: Monte-carlo Integration.

(3) (1.5 points) Use the `its.jl` file for this exercise. Sometimes it is desired to pull samples from a non-uniformly distributed sequence. A popular method for creating such a sequence is the Inverse Transform Sampling. A uniform Distribution U on $[0, 1]$ and a Probability Density Function (PDF) f_{PDF} are given. The PDF's cumulative distribution function (CDF) f_{CDF} has to be computed. We search for a function t that transforms the values from U into a distribution that follows f_{CDF}, we have:

$$f_{\text{CDF}}(x) = \mathbb{P}(t(U) \leq x) = \mathbb{P}(U \leq t^{-1}(x)).$$

It follows, that $f_{\text{CDF}}(x) = t^{-1}(x)$ and thus, $f_{\text{CDF}}^{-1} = t(x)$. This means that the desired transformation function is just the inverse of f_{CDF}. To achieve this, perform the following steps:

a) Given a Probability Density Function f_{PDF}, integrate f_{PDF} to find the corresponding f_{CDF}.
b) Find f_{CDF}^{-1}.
c) Use the `rand` function to construct a uniformly distributed sequence U in $[0, 1]$ of size N. Transform U according to the Inverse Transform Sampling process. Can you obtain the uniform distribution from the resulting distribution? If so, how?

Perform inverse transform sampling using following functions with $X \in [0, 1]$:

a) (0.5 points) $f_{\text{PDF}}(x) = \sin(x)$
b) (0.5 points) $f_{\text{PDF}}(x) = 3x^2$
c) (0.5 points) $f_{\text{PDF}}(x) = e^x$

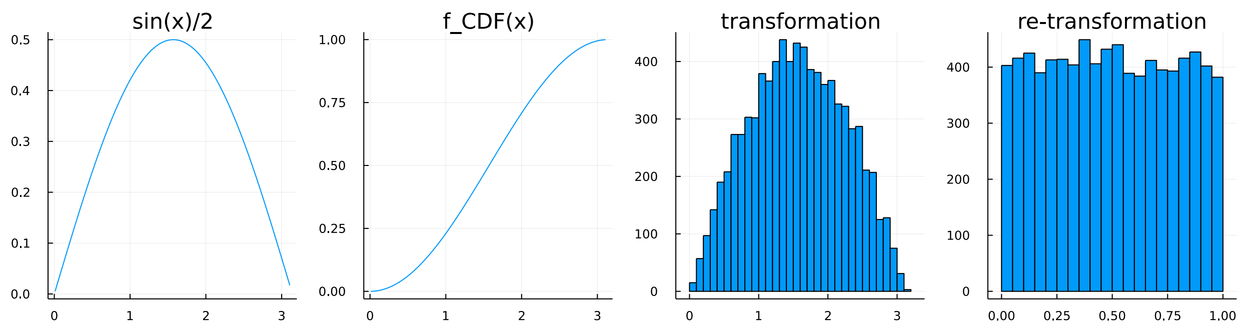

Figure 3: Inverse transform sampling example (see lecture).