Allgemeine Informationen: Dieses Aufgabenblatt enthält schriftliche und/oder Programmieraufgaben. Bitte kombinieren Sie alle Lösungen zu den schriftlichen Aufgaben zu einem einzelnen PDF Dokument, welches Sie nach folgendem Schema benennen:`{lastname}-written.pdf`. Sie können Ihre Lösungen auch scannen oder fotografieren. Achten Sie in diesem Fall auf die Lesbarkeit. Es werden JPEG/PNG Bilddateien akzeptiert welche wie folgt benannt werden müssen:`{exercisenumber}-{lastname}-written.{jpeg/png}`. Stellen Sie sicher, dass alle Rechenschritte nachvollziehbar sind und kombinieren Sie nicht zu viele kleine Schritte zu einem einzelnen. Die Programmieraufgaben müssen in *Julia* gelöst sein und Ihr Quellcode sollte nach folgendem Schema benannt sein: `{exercisenumber}-{lastname}.jl`.

(1) (2.5 Punkte) Generieren Sie Zufallszahlen mit Ihrer eigenen Implementierung von Zufallszahlengeneratoren. Nutzen Sie die `rng.jl` Datei für diese Aufgabe. Das Endergebnis sollte Abbildung 1 für jeden Generator ähneln.

a) (0.25 Punkte) Nutzen Sie die `rand` Funktion um eine Gleichverteilung zu generieren.

Nutzen Sie folgende Funktionssignatur `uniform(N::Int)::Vector{Float64}`.

b) (0.75 Punkte) Implementieren Sie die Mittquadratmethode, we sie in der Vorlesung besprochen wurde.

Nutzen Sie die folgende Funktionssignatur `mid_square(N::Int, seed::Int=34345669)::Vector{Fl`

Hinweis: Sie möchten vielleicht von der `digits` Funktion Gebrauch machen.

c) (1.25 Punkte) Implementieren Sie einen Generator für die Halton Sequenz wie Sie in der Vorlesung beschrieben wurde.

Nutzen Sie die folgende Funktionssignatur `halton(N::Int, base::Int=3)::Vector{Float64}`.

Hinweis: Sie möchten vielleicht von der `digits` Funktion Gebrauch machen.

d) (0.25 Punkte) Implementieren Sie die 2D Version für jede Ihrer Zufallszahlengeneratoren. Nutzen Sie die respektiven Vorlagen in der Datei.

Zusätzlich zu den drei Methoden ist eine `urand.jl` Datei beigefügt, welche Zufallszahlen beinhaltet, welche mit dem Kernel Zufallszahlengenerator[1] des Betriebssystems generiert wurden. Sie können auf diese mit dem Funktionsaufruf `urand(N::Int)::Vector{Float64}` zugreifen. Implementieren Sie auch hier die 2D Version.

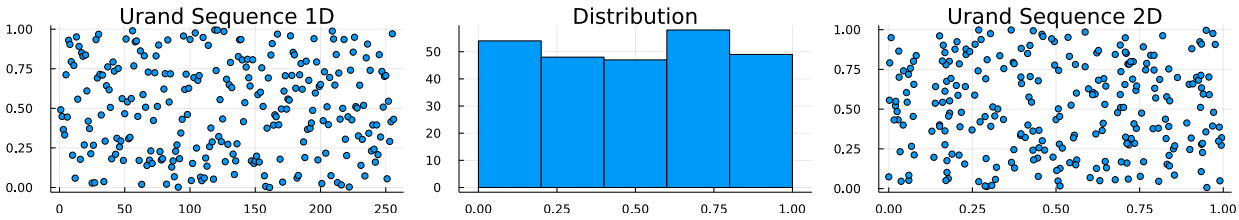

Abbildung 1: Eine Folge von Zufallszahlen.

(2) (3 Punkte) Approximieren Sie ein schwer zu lösendes Integral mittels numerischer Ansätze. Nutzen Sie die `integration.jl` Datei für diese Aufgabe. Ihre Lösung sollte Abbildung 2 ähneln. Zusätzlich, werden Ihnen mehrere Datensätze von Zufallszahlen zur Verfügung gestellt.

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

a) (1 Punkte) Gegeben ist eine Funktion $f(x) = e^{-x^2}$, berechnen Sie folgendes Integral von Hand:

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

Es kann sein, dass es Ihnen schwerfällt einen Lösungsansatz zu finden. Stattdessen können Sie das Integral auch approximieren indem Sie folgende Potenzreihe verwenden:

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

mit $N \in \mathbb{N}$. Wählen Sie eine Anzahl an Termen, welche Sie als angemessen empfinden und berechnen Sie die Annäherung von Hand. Versuchen Sie eine generische Lösung (willkürliche N und x) und implementieren Sie diese in `power_series(a::Float64, b::Float64)::Float64`.

b) (1 Punkt) Approximieren Sie $\int_0^1 f(x)\mathrm{d}x$, mittels Monte-carlo Integration:

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

wobei x_i einen Punkt aus N zufällig gewählten Punkten $\in f(x)$ repräsentiert. Implementieren Sie dazu `mc_integration(a::Float64, b::Float64, N::Int)::Float64`.

c) (1 Punkt) Approximieren Sie $\int_0^1 f(x)\mathrm{d}x$, mit einer anderen Form von Monte-carlo Integration (auch integration by darts):

i. Generieren Sie zufällige Punkte $(x, y) \in P$ mit:

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

ii. Zählen Sie $p \in P$ für welche gilt $f(p.x) \geq p.y$.

iii. Teilen Sie die Anzahl durch die Anzahl der generierten zufälligen Punkte.

Implementieren Sie dazu

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

Notiz: Der Prozess muss abgewandelt werden, damit er für Funktionen funktioniert, welche negative Werte annehmen können.

(3) (2 Punkte) Benutzen Sie das `its.jl` Template für Ihre Implementierung. Manchmal kann es notwendig sein Werte aus einer Verteilung zu beziehen, welche nicht der Gleichverteilung folgt. Populär hierzu, ist die Umformung einer Gleichverteilung durch die Inversionsmethode. Eine Gleichverteilung U über $[0, 1]$ und eine Wahrscheinlichkeitsdichtefunktion (PDF) f_{PDF} sind gegeben. Die kumulative Verteilungsfunktion (CDF) f_{CDF} muss aus der PDF berechnet werden. Gesucht ist eine Funktion t, welche die U in eine Verteilung umwandelt, welche f_{CDF} folgt:

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

Daraus folgt, dass $f_{\text{CDF}}(x) = t^{-1}(x)$ und daraus, $f^{-1}_{\text{CDF}}(x) = t(x)$ was heißt, dass die gesuchte Funktion die Inverse von f_{CDF} ist. Folgen Sie diesen Schritten:

a) Gwegeben sei eine Wahrscheinlichkeitsdichtefunktion f_{PDF}, integrieren Sie f_{PDF} um die dazugehörige f_{CDF} zu finden.

b) Finden Sie f^{-1}_{CDF}.

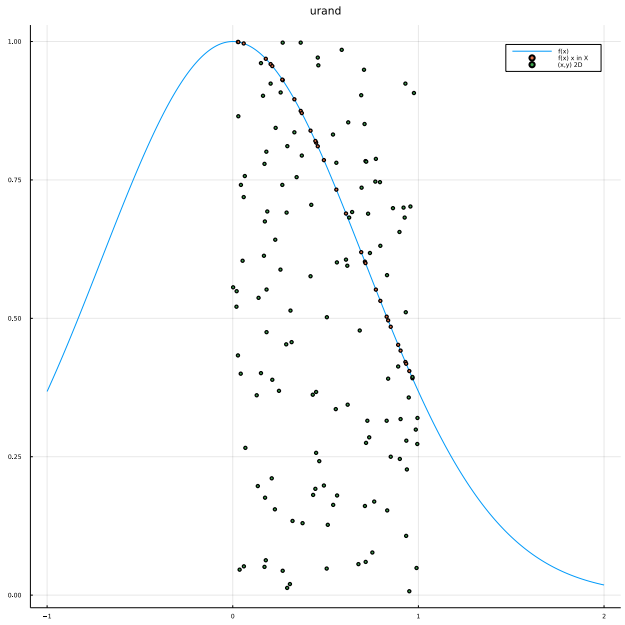

Abbildung 2: Monte-Carlo Integration.

c) Nutzen Sie die `rand` Funktion um eine Gleichverteilte Folge U in $[0, 1]$ von größe N zu generieren. Transformieren Sie U mittels Inversionsmethode. Können Sie die Gleichverteilung von der resultierenden Verteilung zurückerhalten? Wenn ja, wie?

Wenden Sie die Inversionsmethode auf folgende Funktionen mit $X \in [0, 1]$ an:

a) (0.5 Punkte) $f_{\text{PDF}}(x) = \sin(x)$

b) (0.5 Punkte) $f_{\text{PDF}}(x) = 3x^2$

c) (0.5 Punkte) $f_{\text{PDF}}(x) = e^x$

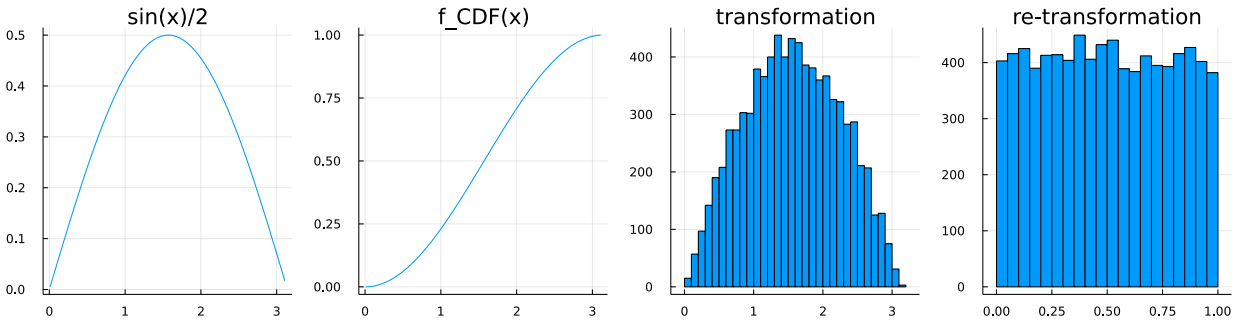

Abbildung 3: Inversionsmethode nach dem Beispiel aus der Vorlesung.