(1) (1 Punkt) Verwenden Sie für diese Aufgabe das `julia` Template `lcg.jl`.

a) (0.25 Punkte) Implementieren Sie einen linearen Kongruenzgenerator (Funktion `lcg`), der eine Folge ganzer Zahlen $z_1, z_2, ... \in [0, m-1]$ generiert:

$$z_{i+1} = (a \cdot z_i + c) \mod m, \quad \text{mit}$$

$$i = 0, 1, 2, ..., \quad m = 2, 3, 4, ..., \quad z_0 \in \{0, 1, ..., m-1\}, \quad a, c \in \{1, 2, ..., m-1\}.$$

b) (0.25 Punkte) Demonstrieren Sie den Satz von Marsaglia graphisch indem Sie sich Zufallszahlen von ihrem linearen Kongruenzgenerator generieren lassen und entsprechend mithilfe der `marsaglia` Funktion plotten.

c) (0.5 Punkte) Demonstrieren Sie den Satz von Knuth anhand eines Beispieles. Der Satz von Knuth besagt, dass ein wie oben beschriebener linearer Kongruenzgenerator genau dann die Periode m hat wenn:

- c und m teilerfremd sind.
- jeder Primteiler p von m auch $a - 1$ teilt.
- wenn m durch 4 teilbar ist, so auch $a - 1$.

Implementieren Sie hierzu die Funktion `knuth`, welche automatisch oben genannte Bedingungen einer Eingabe prüft und die tatsächliche Periodenlänge mit m vergleicht.

(2) (2.5 Punkte) Implementieren Sie Ihre eigenen Zufallszahlengeneratoren und generieren Sie damit Zufallszahlen. Nutzen Sie die `rng.jl` Datei für diese Aufgabe. Die Plots werden automatisch erstellt und das Endergebnis sollte Abbildung 1 für jeden Generator ähneln.

3456

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, wie sie in der Vorlesung besprochen wurde.

Nutzen Sie die folgende Funktionssignatur:

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

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}.
```

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

Zusätzlich zu den drei Methoden ist eine `urand.jl` Datei beigefügt, welche Zufallszahlen beinhaltet, die 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.

Hinweis: Benutzen Sie gegebenenfalls die `digits` Funktion.

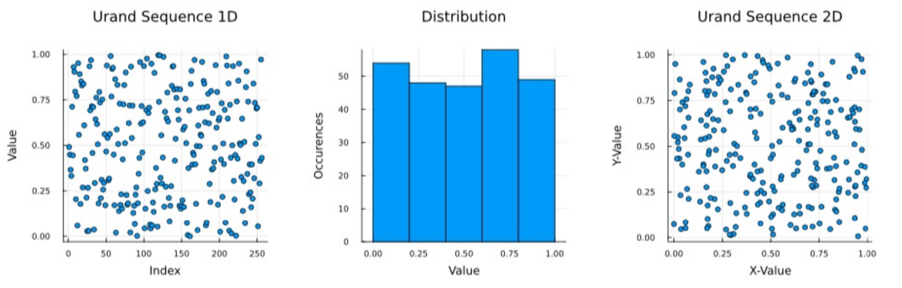

Abbildung 1: Ein Ausschnitt aus der Musterlösung. Dargestellt werden eine Folge von Zufallszahlen per Streudiagramm und Verteilung, sowie ein Streudiagramm generierter Werte für 2D.

(4) (1.25 Punkte) Manchmal kann es notwendig sein Werte aus einer Verteilung zu beziehen, die nicht gleichverteilt sind. Dazu kann die Inversionsmethode eingesetzt werden. 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 die Funktion t, die U in die Verteilung der f_{PDF} umwandelt:

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

Daraus folgt, dass $f_{\text{CDF}}(x) = t^{-1}(x)$ und daraus, $f_{\text{CDF}}^{-1}(x) = t(x)$. Die gesuchte Funktion ist die Inverse von f_{CDF}. Folgen Sie dazu den folgenden Schritten:

(S1) Gegeben sei eine Wahrscheinlichkeitsdichtefunktion f_{PDF}. Integrieren Sie f_{PDF} um die dazugehörige f_{CDF} zu finden $\int_0^X f_{\text{PDF}}(x)\mathrm{d}x$.

(S2) Berechnen Sie f_{CDF}^{-1}.

(S3) Nutzen Sie die `rand` Funktion, um eine gleichverteilte Folge U in $[0, 1]$ von Größe N zu generieren. Transformieren Sie U mittels der Inversionsmethode.

Benutzen Sie das `its.jl` Template für Ihre Implementierung. Wenden Sie die Inversionsmethode auf folgende Funktionen mit $X \in [0, 1]$ an:

a) (0.5 Punkte) $f_{\text{PDF}}(x) = \frac{1}{3x^{\frac{2}{3}}}$

b) (0.5 Punkte) $f_{\text{PDF}}(x) = \frac{\pi e^{-x}}{4} + \frac{\pi e^{x}}{4}$

c) (0.25 Punkte) Können Sie die Gleichverteilung von der resultierenden Verteilung zurückerhalten? Wenn ja, wie? Demonstrieren Sie dies anhand des `julia` templates.

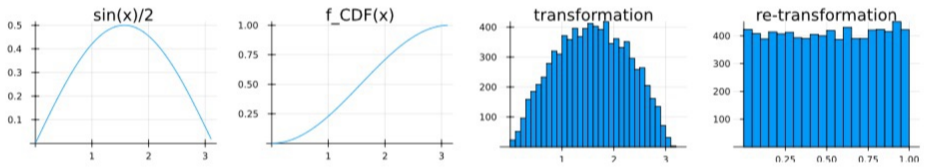

Abbildung 3: Inversionsmethode nach dem Beispiel aus der Vorlesung.

a) $\overset{f_{CDF}}{\int} \frac{1}{3 \cdot x^{\frac{2}{3}}} = \left[\frac{1}{3} \cdot \frac{1}{\frac{1}{3}} \cdot x^{\frac{1}{3}}\right]_0^X = X^{\frac{1}{3}}$

$f_{CDF}^{-1}(x) = t(x)$

INVERTIEREN f_{CDF}^{-1}

$y = x^{\frac{1}{3}}$

$x = y^{\frac{1}{3}}$

$y = x^3$

b) $\int \frac{\pi \cdot e^{-x}}{4} + \frac{\pi \cdot e^{x}}{4} = \left[-\frac{\pi}{4} \cdot e^{-x} + \frac{\pi}{4} \cdot e^{x}\right]_0^X$

$= \frac{\pi}{4} \cdot \left[e^{x} - e^{-x}\right]_0^X$

$= \frac{\pi}{4} \cdot (e^{x} - e^{-x})$

INVERTIEREN:

$$y = \frac{\pi}{4} \cdot (e^{x} - e^{-x})$$

$$x = \frac{\pi}{4} \cdot (e^{y} - e^{-y})$$

$$\frac{4x}{\pi} = e^{y} - e^{-y} \quad \Big| \cdot e^{y} \qquad e^{-y} \cdot e^{y} = 1$$

$$\frac{4x}{\pi} \cdot e^{y} = e^{2y} - 1$$

SUB: $e^{y} = t$

$$\frac{4}{\pi} x \cdot t = t^{2} - 1$$

$$\pi t^{2} - 4xt - \pi = 0 \Rightarrow t = \frac{-(-4x) \pm \sqrt{(-4x)^{2} - 4 \cdot \pi(-\pi)}}{2\pi}$$

RESUB

$$y = \ln\left(4x \pm \sqrt{16x^{2} + 4\pi^{2}}\right.$$

(3) (2.25 Punkte) Für diese Aufgabe können Sie Ihre eigenen Werte aus Aufgabe 2 oder die im `template` Ordner bereitgestellten Werte benutzen. In dieser Aufgabe approximieren Sie ein Integral mittels numerischer Ansätze. Nutzen Sie die `integration.jl` Datei für diese Aufgabe. Zusätzlich werden Ihnen mehrere Datensätze von Zufallszahlen zur Verfügung gestellt.

a) (0.75 Punkte) Gegeben ist die Funktion e^{-x^2}. 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 $\in [a,b]$ repräsentiert. Implementieren Sie dazu `mc_integration(a::Float64, b::Float64, N::Int)::Float64`.

b) (1 Punkt) Approximieren Sie $\int_0^1 f(x)\mathrm{d}x$, mit einer anderen Form von Monte-Carlo Integration (Rejection Sampling):

i. Generieren Sie zufällige Punkte $p = (x, y)$ mit:

$$0 \le x \le 1 \quad \text{und} \quad 0 \le y \le \max(f(x)).$$

ii. Teilen Sie die Anzahl der Punkte für die gilt $f(p.x) \ge p.y$ durch die Anzahl der generierten zufälligen Punkte.

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

Implementieren Sie dazu

`mc_integration_rs(a::Float64, b::Float64, N::Int)::Float64`.

c) (0.5 Punkte) Erweitern Sie die Plots so, dass sie wie in Abbildung 2 zu einer intuitiven Erklärung der beiden Verfahren beitragen.

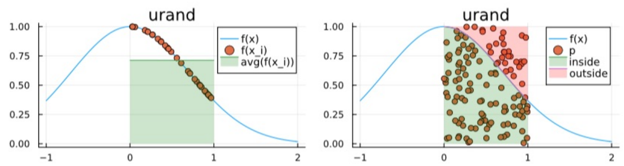

Abbildung 2: Funktionsweisen der Monte-Carlo Integration links für a) und rechts für b) (Rejection Sampling).