Programmierkurs Julia
Johannes Nägele
Lehrstuhl Prof. Dr. Schlather
FSS 2024

Übungsblatt 8

Aufgabe 1 (6 Punkte)

In dieser Aufgabe wollen wir uns mit der bekannten Collatz-Folge beschäftigen. Für einen Startwert $x_0 \in \mathbb{N}$ ist diese gegeben durch $\forall n \in \mathbb{N}$

$$x_n = \begin{cases} x_{n-1}/2 & x_{n-1} \text{ gerade} \\ 3x_{n-1} + 1 & x_{n-1} \text{ ungerade.} \end{cases}$$

Das Interessante ist, dass die Folge scheinbar stets in dem Muster $4, 2, 1, ...$ mündet. Dies konnte bisher aber nicht bewiesen werden.

(a) Schreibt eine Funktion `next(x)`, welche das nächste Folgenglied berechnet.

(b) Schreibt eine Funktion `collatz_length(x)`, welche berechnet, wieviele Schritte wir (für einen Startwert) brauchen, bis wir bei 1 ankommen.

(c) Erstellt ein Histogramm dieser Schrittanzahlen für $x_0 = 1, ..., 10^7$ (Tipp: Entfernt die Umrisse der Säulen via `linealpha` und füllt dafür das Histogramm wieder via `fill` aus. Zum Ausprobieren kann man zum Beispiel auch $1, ..., 10^6$ nehmen).

Aufgabe 2 (6 Punkte)

Zufallszahlen.

(a) Setze einen Seed auf 42.

(b) Generiere die Vektoren

- `v1` mit 1000 gleichverteilten Zahlen auf `[10,20]` und
- `v2` mit 1000 normalverteilten Zufallszahlen mit `mean = 15` und `sd = 2` sowie
- `v3` mit 1000 Gumbel-verteilten Zufallszahlen mit `mean = 15` und `sd = 2`.

(c) Erstelle aus diesen Vektoren einen DataFrame mit Spaltennamen `:Spalte1`, `:Spalte2` und `:Spalte3`.

(d) Was ist die kleinste Zahl in `:Spalte1`, bei der `:Spalte1` > `:Spalte2` ist?

Aufgabe 3 (6 Punkte)

Approximation von π. Dies ist ein einfacher Fall einer Monte-Carlo-Methode.

(a) Generiert einen Array mit 10^5 uniform verteilten Ziehungen aus $[-1,1] \times [-1,1]$ (das ist das Quadrat um den Einheitskreis).

(b) Checkt, ob die Punkte im Einheitskreis liegen (Tipp: Verwendet die euklidische Norm `norm` aus dem Modul `LinearAlgebra`).

(c) Approximiert π. (Tipp: Die Fläche des umgebenden Quadrats ist 4 und somit $\pi \approx 4 \frac{\#\text{Punkte innerhalb des Kreises}}{\#\text{Alle Punkte}}$).

(d) Erstelle einen Scatterplot unserer Approximation. Dabei soll der Bereich rechts oben ($[0,1] \times [0,1]$) zu sehen sein; markiert die Punkte inner- bzw. außerhalb des Kreises in unterschiedlichen Farben.

Aufgabe 4 (3 Punkte)

Performanceorientierte Programmierung (row vs. column major).

(a) Berechne, wieviele Elemente der Matrix `mat = randn(1000, 1000)` strikt größer als 1 sind (Tipp: `sum` oder `count`).

(b) Messt Zeiten mit `@btime` oder `@benchmark` für zwei Varianten unseres Ausgangsproblems:

  (i) Summiere zuerst über die Spalten und dann über den entstehenden Zeilenvektor.

  (ii) Gehe umgekehrt vor.

  Warum ist Variante (i) schneller?

Aufgabe 5 (Zusatzaufgabe, 9 Punkte)

Performanceorientierte Programmierung: Memoization.

(a) Implementiere die rekursive Fibonacci-Aufgabe von Blatt 3 mit caching (das nennt man allgemeiner auch *memoization pattern*). Dazu gehen wir in mehreren Schritten vor:

  - Erstelle ein `const` Dictionary, in welchem wir dann später Zwischenergebnisse speichern wollen.
  - Schreibe die Funktion `_fibo_rec(n)`. Im Unterschied zu unserer ursprünglichen rekursiven Implementierung sollen nun die Funktionswerte f_{n-1} und f_{n-2} nicht mehr durch sich selbst, sondern mit einer weiteren Funktion `fibo_smart(n)` berechnet werden.
  - Schreibe eine Funktion `fibo_smart(n)`.
  - Unterscheide darin zwei Fälle: Falls `f_n` noch nicht berechnet wurde, berechne es mittels `_fibo_rec` füge das Ergebnis zum Dict hinzu. Falls das Ergebnis schon bekannt ist, muss es nur noch aus dem Dict ausgelesen werden.

(b) Schreibe allgemeiner eine Funktion `memoize(f)`, welche für die (inperformante) rekursive Funktion `f` eine anonyme Funktion zurückgibt, die zusätzlich caching betreibt (Tipp: Für `fibo_rec` soll quasi `fibo_smart` zurückgegeben werden; das Pattern **`begin ... end`** könnte hilfreich sein).

Der Trick an der ganzen Geschichte ist quasi, dass man die zurückgegebene Funktion gleich benennt, wie die, die man hineinsteckt (also ist quasi `fibo_rec` dasselbe wie `fibo_smart`). Sprich, wir schreiben in unserem Fall:

```
fibo_rec = memoize(fibo_rec)
```

Zusätzlich gibt es aber noch den Haken, dass wir damit ja die hineingesteckte Funktion überschreiben (was nicht geht, weil Funktionen konstant sind). Deshalb definieren wir `fibo_rec` als anonyme Funktion (`fibo_rec = begin ... end`), denn damit haben wir eine veränderbare Variable.

(c) Teste deine Funktion `memoize` für eine rekursive Implementierung von (zum Beispiel) Pell numbers P_n:

$$P_n = \begin{cases} 0 & \text{für } n = 0, \\ 1 & \text{für } n = 1, \\ 2P_{n-1} + P_{n-2} & \text{sonst.} \end{cases}$$

Viel Erfolg!