Dr. Frank Schöpfer

Einführung in Matlab
Übung 2

Aufgabe 1: Die Folge $x_1, x_2, x_3, \ldots$ sei definiert durch $x_1 := 1$ und

$$x_k := 1 + \frac{1}{x_{k-1}} \quad \text{für} \quad k \geq 2$$

Dann entspricht x_k dem Wert des Kettenbruchs

$$1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1+\ldots}}}$$

und nähert sich für größer werdende k immer mehr der goldenen Schnittzahl $x := \frac{1+\sqrt{5}}{2}$ an (d.h. $x_k \approx x$ und x ist positive Lösung der Gleichung $x = 1 + \frac{1}{x}$).

(a) Schreiben Sie zunächst eine Funktion `x=kettenbruch_test(n)`, welche die ersten n Folgenglieder $x_1, x_2, \ldots, x_n$ mit Hilfe einer for-Schleife berechnet und zusammen als Array `x` zurückgibt. Plotten Sie dann die ersten $n = 10$ Folgenglieder (als Kreise) zusammen mit dem konstanten Wert $\frac{1+\sqrt{5}}{2}$ in ein Bild (analog zum Beispiel babylonisches Wurzelziehen der Vorlesung).

(b) Schreiben Sie nun die Funktion `x=kettenbruch(n)` so, dass nur das letzte Folgenglied x_n zurückgegeben wird (und kein Array berechnet wird).

(c) Schreiben Sie die Funktion aus (b) als rekursive Funktion `x=kettenbruch_rekursiv(n)`.

(d) Schreiben Sie nun die Funktion `[x,iter]=kettenbruch_while(epsilon)` so, dass statt einer vorgegebenen Anzahl an Iterationen nur solange iteriert wird, bis eine gewünschte Genauigkeit erreicht wird, z.B.

$$|x - 1 - \tfrac{1}{x}| < \epsilon$$

und die Anzahl der benötigten Iterationen `iter` mit zurückgegeben wird.

Aufgabe 2: Die Binomialkoeffizienten $\binom{n}{k} := \frac{n!}{k! \cdot (n-k)!}$ erfüllen für $k \leq n \in \mathbb{N}$ die Rekursion

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

mit

$$\binom{n}{0} = \binom{n}{n} = 1 \quad \text{und} \quad \binom{n}{1} = \binom{n}{n-1} = n$$

Schreiben Sie zur Berechnung dazu eine rekursive Funktion `b=binomial(n,k)`. Testen Sie mit relativ kleinen n, da die Zahlen schnell wachsen. Vergleichen können Sie mit dem entsprechenden Matlab-Befehl `nchoosek`.

Aufgabe 3: In der Vorlesung haben wir den Selection Sort-Algorithmus rekursiv programmiert.

(a) Schreiben Sie den Selection Sort-Algorithmus ohne rekursiven Aufruf, stattdessen mit einer geeigneten `for`-Schleife, und vergleichen Sie die Laufzeiten der beiden Versionen mit `tic,toc`. (Im allgemeinen kann man nicht definitiv sagen, ob eine rekursive oder nicht-rekursive Version schneller ist.)

(b) Die entsprechende Matlab-Funktion zum Sortieren `[xs,indices]=sort(x)` gibt zusätzlich noch den zur Sortierung passenden Index-Vektor `indices` zurück, so dass `x(indices)=xs` gilt.

```
>> x=[8 5 7 6];
>> [xs,indices]=sort(x), x(indices)
xs =
     5     6     7     8
indices =
     2     4     3     1
ans =
     5     6     7     8
```

Erweitern Sie daher ihren Selection Sort-Algorithmus so, dass er das gleiche macht.