Schleifensteuerung, Darstellung von Funktionen und Polynome

1. Durch die endliche Rechengenauigkeit bei Floating Point (im Gegensatz zu ganzen) Zahlen kann es schon bei den einfachsten Operationen zu unerwarteten Ergebnissen kommen: Untersuche, wie oft die folgende Schleife in `F`

   ```
   real::x
   ...
   do x=1.0,2.0,0.1
     ...
   end do
   ```

   bzw. in `C`

   ```
   float x;
   ...
   for(x=1.0;x<=2.0;x=x+0.1) {
     ...
   }
   ```

   tatsächlich ausgeführt wird. Hängt das Verhalten von Sprache und Compiler ab?

   Dies ist einer der Gründe, warum die Verwendung von nicht-ganzzahligen Schleifenzählern, obwohl in `F9x` geduldet, in `F` verpönt ist. Ein rigoroser `F`-Compiler sollte in diesem Fall auch einen Fehler melden.

2. In der Statistischen Physik wird häufig die für große n gültige Näherung

   $$\log n! \approx n \log n - n$$

   verwendet. Schreibe ein Programm, das für ein vorgegebenes n diese sowie die verbesserte Stirling'sche Approximation

   $$\log n! \approx n \log n - n + \log \sqrt{2\pi n}$$

   mit dem exakten Wert vergleicht, und berechne den relativen Fehler beider Näherungen.

   Da man bei naiver Berechnung von $n!$ sehr bald den standardmäßig am Computer darstellbaren Bereich ganzer Zahlen überschreitet, empfiehlt es sich, $\log n!$ als

   $$\log n! = \log 1 + \log 2 + \log 3 + \ldots + \log n$$

   zu berechnen. Die Zahl π kann man z.B. mit Hilfe der Funktion `atan` aus der Beziehung $\tan^{-1}(1) = \pi/4$ bekommen.

3. Wie viele andere analytische Funktionen auch, läßt sich die Exponentialfunktion im Prinzip durch eine Reihenentwicklung

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

darstellen. (Dies ist jedoch *nicht* die Art und Weise, wie in Programmiersprachen "eingebaute Funktionen" üblicherweise implementiert werden!) Schreibe ein kurzes Programm, das ein Argument x und einen Index n einliest, die Exponentialfunktion durch Summation aller Terme der Reihenentwicklung bis einschließlich $k = n$ approximiert und mit dem "exakten" Wert der eingebauten Funktion `exp` vergleicht.
Hinweis: Da sowohl x^k als auch $k!$ sehr schnell mit k wachsen können, empfiehlt es sich, nicht für jeden Summanden Zähler und Nenner separat zu berechnen und dann durch einander zu dividieren, sondern jeweils den k-ten Term durch Multiplikation mit x/k aus dem $(k-1)$-ten zu bilden.

4. Wird eine analytische Funktion durch eine endliche Reihenentwicklung approximiert, so bedeutet dies, daß sie eigentlich durch ein Polynom (von eventuell sehr hohem Grad) dargestellt wird, z.B. für den Logarithmus

$$\ln(1+x) \approx \sum_{k=1}^{n} (-1)^{(k+1)} \frac{x^k}{k} = x - \frac{x^2}{2} + \frac{x^3}{3} + \ldots + (-1)^{(n+1)} \frac{x^n}{n}$$

Da Polynome oft numerisch ungünstige Eigenschaften haben, ist es besser, sie nicht wie oben angeschrieben, sondern nach dem *Hornerschema* "von innen nach außen" auszuwerten, d.h. in diesem Fall

$$\ln(1+x) \approx x \left[1 - x \left[\frac{1}{2} - x \left[\frac{1}{3} - x \left[\ldots - x \left[\frac{1}{n-1} - \frac{x}{n}\right] \ldots\right]\right]\right]\right]$$

wobei man mit dem Term in den innersten Klammern beginnt und sich nach außen durcharbeitet. Das hat den Vorteil, daß in jedem Schritt x nur linear vorkommt und nicht hohe Potenzen von x direkt miteinander kombiniert werden müssen.

Schreibe ein Programm, das nach Eingabe von x und n das obige Approximationspolynom sowohl nach dem naiven als auch nach dem Hornerschema berechnet. Um die Unterschiede besser sichtbar zu machen, sollten die Rechnungen in "einfacher Präzision" (Typ `real` in `F` bzw. `float` in `C`) durchgeführt werden.
Hinweis: Untersuche insbesondere das Verhalten bei $|x| \approx 1$.

Da die logarithmische Reihe sehr langsam konvergiert, wird allerdings der Wert des Polynoms i.a. nicht mit dem exakten Ergebnis $\ln(1+x)$ der eingebauten Funktion `log` übereinstimmen: Während für kleine n nur der Grenzwert der Reihe noch nicht erreicht ist, können für hohe Polynomgrade selbst beim Hornerschema Rundungsfehler das Resultat verfälschen.

5. Ein (unendlicher) Kettenbruch ist ein Ausdruck der Form

$$a_0 + \frac{b_1}{a_1+} \, \frac{b_2}{a_2+} \, \frac{b_3}{a_3+} \ldots$$

Dies ist eine vereinfachte Schreibweise für

$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \ldots}}}$$

Eine naive (aber nicht immer numerisch gutartige) Methode, einen Kettenbruch auszuwerten, besteht darin, ab einer gewissen Stufe die durch Punkte angedeuteten Beiträge zu vernachlässigen und, z.B. mit dem Quotienten (b_n/a_n) beginnend, den Bruch von "unten her" durch wiederholtes Dividieren zu berechnen. Kettenbrüche spielen eine wichtige Rolle als Alternativen zu Reihenentwicklungen von mathematischen Funktionen, weil sie oft schneller konvergieren oder überhaupt einen größeren Konvergenzbereich besitzen.

Schreibe ein Programm, das die Logarithmusfunktion des vorigen Beispiels mit Hilfe der Kettenbruchentwicklung

$$\log(1+x) = \cfrac{x}{1+}\ \cfrac{x}{2+}\ \cfrac{x}{3+}\ \cfrac{4x}{4+}\ \cfrac{4x}{5+}\ \cfrac{9x}{6+}\ \cfrac{9x}{7+}\ \ldots$$

approximiert (Abbruch nach dem n-ten Term) und vergleiche die Konvergenzgeschwindigkeit dieser Entwicklung mit der der Taylorreihe.

Im Unterschied zur Taylorreihe, die nur für $|x| < 1$ konvergiert, sollte die Kettenbruchentwicklung für alle $-1 < x < \infty$ gültig sein. Untersuche daher auch, ob bzw. in welchem Bereich die erhaltenen Werte mit jenen der Funktion `log` übereinstimmen.