Computermathematik (für Informatik)

5. Übungsblatt 16. 12. 2009

Die heutigen Übungen sollen mit dem Computeralgebrasystem **Sage** gelöst werden.
Die Lösung der Beispiele soll auf möglichst kompakte Weise erfolgen. Wenn zum Beispiel eine Funktion für mehrere Werte berechnet werden soll, soll das mittels einer geeigneten Schleifen oder Listen Operation erfolgen, und **nicht** alle Werte einzeln eingetippt werden. Zwischenergebnisse welche in einem weiteren Berechnungsschritt benötigt werden, sollen in eine Variable gespeichert und weiterverwendet werden (**nicht** neu eintippen).

Übung 21. Bestimmen Sie ein Polynom $p(x)$ vom Grad 3, mit folgenden Eigenschaften:

$$p(5) = 0, \quad p(2) = -9, \quad p'(1) = -8, \quad p''(2) = 2$$

Übung 22. Die Newton-Iteration ist ein numerisches Verfahren zur näherungsweisen Berechnung von Nullstellen einer Funktion f. Sie ist definiert durch die rekursive Folge:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

und einem Startwert x_0.
Schreiben Sie einen Generator `newton(f, x0)`, der für eine gegebene Funktion f und den Startwert $x0$ die Newton Iteration durchführt.
Schreiben Sie auch eine geeignete Abbruchbedingung für die Iteration (Die Abbruchbedingung soll sich nicht im Generator befinden).

(a) Verwenden Sie Ihren Generator um eine Nullstelle von

$$f(x) = x^3 - 3x^2 - 9x - 1$$

zu bestimmen. Verwenden Sie als Startwert $x_0 = 0$.

(b) Verwenden Sie nun die folgenden Startwerte: $-2, -1, 0, 1, 2, 3$

Versuchen Sie anhand des Funktionsgraphen von f das Verhalten des Newton-Verfahrens für die verschiedenen Startwerte zu erklären.

Achtung: Die Newton-Iteration funktioniert vielleicht nicht für alle Startwerte.

Hinweis: Beispiele für Generatoren finden Sie in der Musterlösung zu Beispiel 19.

Übung 23. Vergleichen Sie die folgenden zwei Versuche einen Generator für Primzahlen zu implementieren.

```
def natural_numbers(start = 0):
    i = start
    while True:
        yield i
        i += 1
```

```
def prime_generator_1():
    primes = natural_numbers(2)

    while True:
        current_prime = primes.next()
        yield current_prime

        primes = (q for q in primes if not current_prime.divides(q))
```

```
def prime_generator_2():
    def sieve_p(primes, current_prime):
        return (q for q in primes if not current_prime.divides(q))

    primes = natural_numbers(2)

    while True:
        current_prime = primes.next()
        yield current_prime

        primes = sieve_p(primes, current_prime)
```

Finden Sie heraus welcher der beiden Generatoren das korrekte Ergebnis liefert, und erklären Sie das unterschiedliche Verhalten der beiden Funktionen.