Beispiele 21-23

January 12, 2010

1 Computermathematik (für Informatik)

5. Übungsblatt (Musterlösung)
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).

1.1 Beispiel 21

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

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

Sage code

```
var('a,b,c,d,x')
```

```
(a, b, c, d, x)
```

Sage code

```
p(x) = a * x^3 + b * x^2 + c * x + d
show(p)
```

$$x \mapsto ax^3 + bx^2 + cx + d$$

Sage code

```
p1(x) = p.derivative(x)
show(p1)
```

$$x \mapsto 3\,ax^2 + 2\,bx + c$$

Sage code

```
p2(x) = p1.derivative(x)
show(p2)
```

$$x \mapsto 6\,ax + 2\,b$$

Sage code

```
eqns = [p(5) == 0, p(2) == -9, p1(-1) == 12, p2(2) == 2]
show(eqns)
```

$$[125\,a + 25\,b + 5\,c + d = 0, 8\,a + 4\,b + 2\,c + d = (-9)\,, 3\,a - 2\,b + c = 12, 12\,a + 2\,b = 2]$$

Sage code

```
s = solve(eqns, [a,b,c,d], solution_dict = True)
show(s)
```

$$[\{c:\ -1, b:\ -5, a:\ 1, d:\ 5\}]$$

Sage code

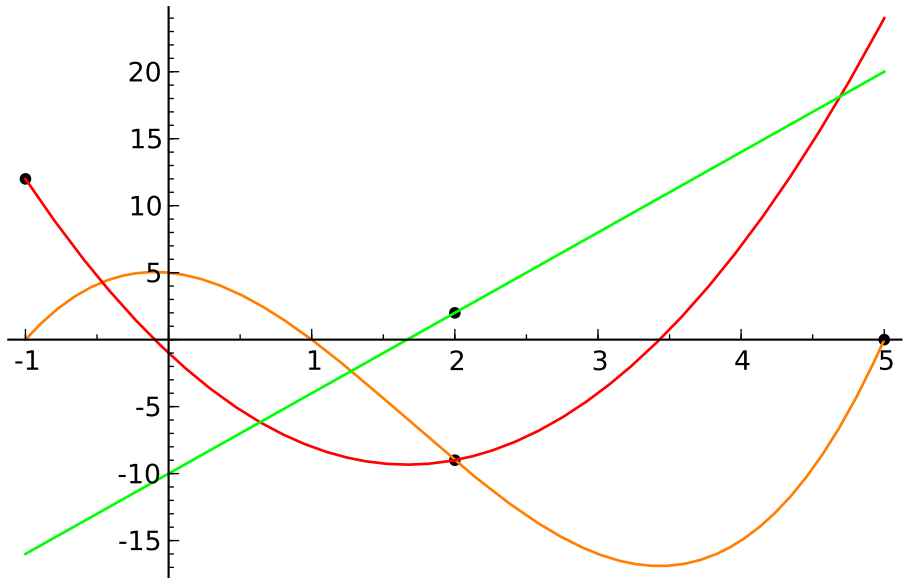
```
f(x) = p(x).subs(s[0])
show(f)
```

$$x \mapsto x^3 - 5\,x^2 - x + 5$$

Sage code

```
g = Graphics()
g += plot(f, (x, -1, 5), rgbcolor = "orange")
g += plot(f.diff(x), (x, -1, 5), rgbcolor = 'red')
g += plot(f.diff(x, 2), (x, -1, 5), rgbcolor = 'green')

g += list_plot([(5,0), (2,-9), (-1,12), (2,2)], rgbcolor = 'black', pointsize = 15)
g.show()
```



1.2 Beispiel 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 dem Startwert $x0$, die Newton Iteration durchführt.
Schreiben Sie auch eine geeignete Abbruchbedingung für die Iteration.

1. Verwenden Sie Ihren Generator um eine Nullstelle von

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

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

2. Verwenden Sie nun die folgenden Startwerte: $-2, -1, 0, 1, 2, 3$

   Versuchen Sie anhand des Funktionsgraphen von f , das Verhalten des Newtonverfahrens 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.

Sage code

```
def newton(f, x0):
    f1 = f.derivative()

    xn = RR(x0)
    yield xn, f(xn)
    while f(xn) != 0:
        xn = xn - f(xn) / f1(xn)
        yield xn, f(xn)
```

Sage code

```
def take(g, n):
    l = []
    for i in xrange(n):
        try:
            l.append(g.next())
        except StopIteration:
            break
    return l
```

Sage code

```
f(x) = x^3 - 3*x^2 - 9*x - 1
show(f)
```

$$x \mapsto x^3 - 3\,x^2 - 9\,x - 1$$

Sage code

```
result = take(newton(f, 0), 5)
result = [['$x_n$', '$f(x_n)$']] + result
html.table(result, header = True)
```

x_n	$f(x_n)$
0.000000000000000	−1.00000000000000
−0.111111111111111	−0.0384087791495198
−0.115740740740741	−0.0000715441307219677
−0.115749396632707	$-2.50789389255601 \times 10^{-10}$
−0.115749396663049	0.000000000000000

Sage code

```
def newton_terminate(g, epsilon):
    x, fx = g.next()
    while abs(fx) > epsilon:
        x, fx = g.next()
    return x, fx
```

Sage code

```
for x0 in [-2..4]:
    try:
        xn, fxn = newton_terminate(newton(f, x0), 10^(-10))
        show((x0, xn, fxn))
    except ZeroDivisionError:
        print "fuer x0 = %s Division durch 0" % x0
```

$$(-2, -1.76873430527639, -1.20081722343457 \times 10^{-12})$$

```
fuer x0 = -1 Division durch 0
```

$$(-2, -1.76873430527639, -1.20081722343457 \times 10^{-12})$$
$$(0, -0.115749396663049, 0.000000000000000)$$

$$(-2, -1.76873430527639, -1.20081722343457 \times 10^{-12})$$
$$(0, -0.115749396663049, 0.000000000000000)$$
$$(1, -0.115749396663049, 0.000000000000000)$$

$$(-2, -1.76873430527639, -1.20081722343457 \times 10^{-12})$$
$$(0, -0.115749396663049, 0.000000000000000)$$
$$(1, -0.115749396663049, 0.000000000000000)$$
$$(2, -0.115749396663049, 0.000000000000000)$$

```
fuer x0 = 3 Division durch 0
```

$$(-2, -1.76873430527639, -1.20081722343457 \times 10^{-12})$$
$$(0, -0.115749396663049, 0.000000000000000)$$
$$(1, -0.115749396663049, 0.000000000000000)$$
$$(2, -0.115749396663049, 0.000000000000000)$$
$$(4, 4.88448370193979, 1.51487711264053 \times 10^{-11})$$

Die Nullstellen von f :

Sage code

```
f.polynomial(QQ).roots(ring=RR)
```

```
[(-1.76873430527628, 1), (-0.115749396663049, 1), (4.88448370193933, 1)]
```

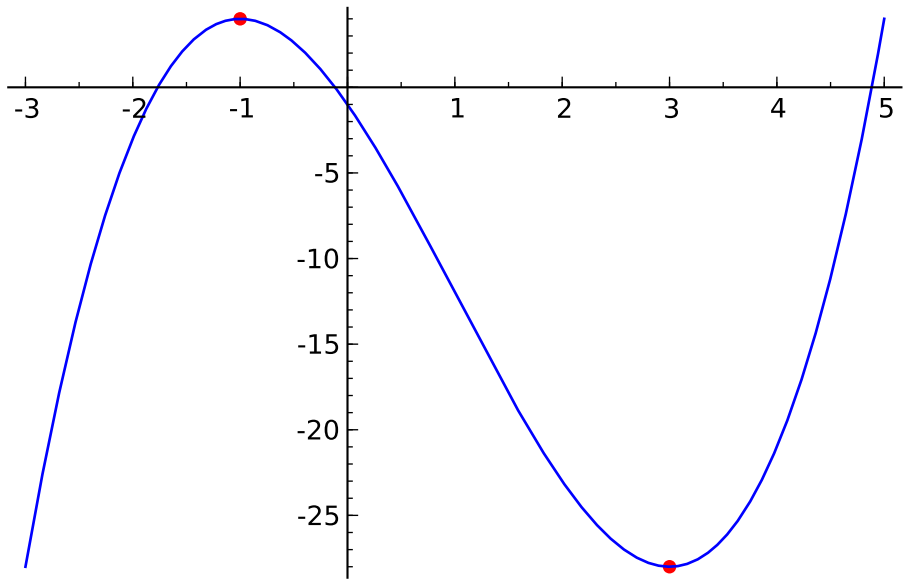
Sage code

```
roots = f.derivative().roots()
roots
```

```
[(3, 1), (-1, 1)]
```

Sage code

```
p = plot(f, -3, 5)
for x, mult in roots:
    p += point((x, f(x)), rgbcolor = 'red', pointsize = 20)
p.show()
```



Newton-Iteration liefert nicht immer eine Nullstelle:

Sage code

```
g(x) = x^3-2*x+2
show(g)
```

$$x \mapsto x^3 - 2\,x + 2$$

Sage code

```
gn = newton(g, 0)
```

Sage code

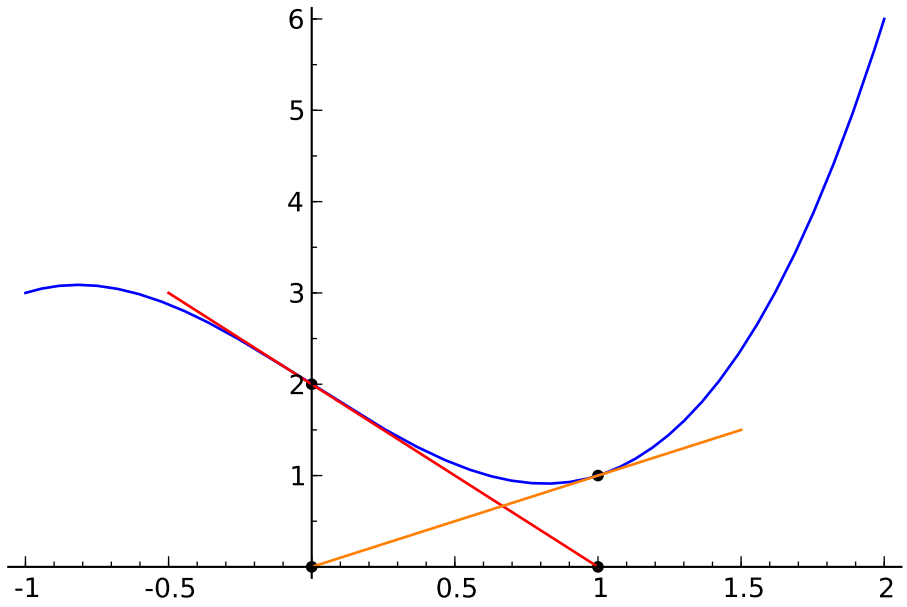
```
html.table([['$x_n$', '$f(x_n)$']] + take(gn, 10), header = True)
```

x_n	$f(x_n)$
0.000000000000000	2.00000000000000
1.00000000000000	1.00000000000000
0.000000000000000	2.00000000000000
1.00000000000000	1.00000000000000
0.000000000000000	2.00000000000000
1.00000000000000	1.00000000000000
0.000000000000000	2.00000000000000
1.00000000000000	1.00000000000000
0.000000000000000	2.00000000000000
1.00000000000000	1.00000000000000

Sage code

```
def tangente(g, x0):
    d = g(x0)
    k = g.diff(x)(x0)
    return k*(x-x0)+d

pl = plot(g, (x, -1, 2))
pl += plot(tangente(g, 0), (x, -0.5, 1), color = 'red')
pl += plot(tangente(g, 1), (x, 0, 1.5), color = 'orange')
pl += point((0,g(0)), rgbcolor = 'black', pointsize = 15)
pl += point((1,g(1)), rgbcolor = 'black', pointsize = 15)
pl += point((0,0), rgbcolor = 'black', pointsize = 15)
pl += point((1,0), rgbcolor = 'black', pointsize = 15)
pl.show()
```



1.3 Übung 23

Vergleichen Sie die folgenden zwei Versuche einen Generator für Primzahlen zu implementieren. Finden Sie heraus welcher der beiden Generatoren das korrekte Ergebnis liefert, und erklären Sie das unterschiedliche Verhalten der beiden Funktionen.

Sage code

```
def take(generator, n = 10):
    liste = []
    for i in xrange(n):
        liste.append(generator.next())
    return liste
```

Sage code

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

Sage code

```
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))
```

Sage code

```
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)
```

Sage code

```
take(prime_generator_1(), 200)
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, \
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 4\
6, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67\
, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,\
 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, \
108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, \
125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, \
142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, \
159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, \
176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, \
193, 194, 195, 196, 197, 198, 199, 200, 201]
```

Sage code

```
take(prime_generator_2(), 200)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, \
83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173\
, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269\
, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373\
, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467\
, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593\
, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691\
, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821\
, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937\
, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 10\
39, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 112\
9, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223]
```