Beispiel 19

December 4, 2009

Computermathematik (für Informatik)

4. Übungsblatt (Musterlösung)
2. 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).

Beispiel 19

$$\pi = 3 + 3\sum_{n=1}^{\infty} \frac{\binom{2n}{n}}{(2n+1)16^n}$$

1. Schreiben Sie ein Funktion **pi_approx(k,d)** die die ersten k Summanden der obigen Reihe berechnet um eine Approximation von π zu erhalten. Die Rechnungen sollen auf d Stellen genau erfolgen.

2. Wieviele Summanden k werden benötigt um π auf $d = 20, 40, \ldots, 200$ Nachkommastellen genau zu berechnen? Erzeugen Sie eine Graphik in der die Wertepaare (d, k) eingezeichnet sind.

Sage code

```
def pi_approx(k, digits = 100):
    R = parent(1.n(digits = digits))
    return 3+3*sum(R(binomial(2*n, n)  / ((2*n+1)*16^n)) for n in [1..k])
```

Sage code

```
pi_approx(6, 500)
```

```
3.14158942531912159292530386280386280386280386280386280386280386280386280386280386280\
3862803862803862803862803862803862803862803862803862803862803862803862803862803862803\
8628038628038628038628038628038628038628038628038628038628038628038628038628038628038\
6280386280386280386280386280386280386280386280386280386280386280386280386280386280386\
2803862803862803862803862803862803862803862803862803862803862803862803862803862803862\
8038628038628038628038628038628038628038628038628038628038628038628038628039
```

Sage code

```
def approximate_pi(pi_approx, digits):
    k = 1
    terminate = False
    calc_digits = digits + 20
    pi_digits = pi.n(digits = calc_digits)
    epsilon = (10^-digits).n(digits = calc_digits)
    while not terminate:
        approx = pi_approx(k, digits = calc_digits)
```

```
        terminate = abs(approx - pi_digits) < epsilon
        k += 1

    return (k, approx)
```

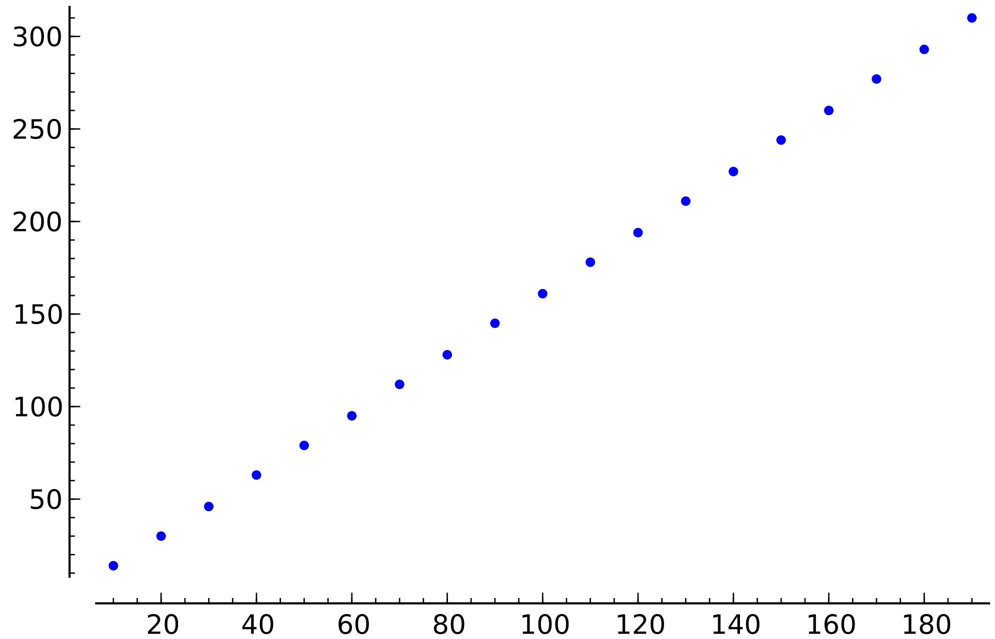
Sage code

```
%time
liste = [(k, approximate_pi(pi_approx, k)[0]) for k in range(10, 200, 10)]
```

```
CPU time: 12.80 s,  Wall time: 12.81 s
```

Sage code

```
list_plot(liste)
```



0.0.1 Lösung mit Generatoren

Ein Generator ist ein Python Konstrukt, dass eine Folge von Werten erzeugt.
Man erzeugt einen Generator, wie eine normale Python Funktion, allerdings wird das Schlüsselwort **yield** anstatt **return** verwendet um Werte zurückzugeben.

Sage code

```
def gen():
    yield 1
    yield 4
    yield 7
```

gen() erzeugt jetzt ein Generator Objekt.Aufrufen der Funktion

Sage code

```
g = gen()
type(g)
```

```
<type 'generator'>
```

.next() berechnet man jeweils den nächsten Wert des Generators:Mit der Methode

Sage code

```
g.next(); g.next(); g.next()
```

```
1
4
7
```

StopIteration.Wenn der Generator kein neues Element mehr erzeugen kann, bekommt man die Fehlermeldung

Sage code
```
g.next()
```

```
StopIteration
```

Man kann mit einer **for**-Schleife über einen Generator iterieren:

Sage code
```
for i in gen():
    print i
```

```
1
4
7
```

Wir definieren uns einen Generator für die natürlichen Zahlen:

Sage code
```
def count():
    n = 0
    while True:
        yield n
        n += 1
```

Aufrufen der Funktion **count()** erzeugt jetzt ein Generator Objekt.

Sage code
```
nat = count()
```

Mit der Methode **.next()** berechnet man jeweils einen Wert des Generators:

Sage code
```
nat.next(); nat.next(); nat.next()
```

```
0
1
2
```

Generatoren können natürlich auch Parameter haben, zum Beispiel auch andere Generatoren:

Sage code
```
def sum_generator(g):
    value = 0
    while True:
        value += g.next()
        yield value
```

sum_generator(g) erzeugt einen generator der die Summe der Werte des Generators **g** erzeugt.

Sage code
```
g = sum_generator(count())
```

Sage code
```
for i in g:
    print i

    if (i > 50):
        break
```

```
0
1
3
6
10
15
21
28
36
45
55
```

Eine **Generator Comprehension** ist das äquivalent einer List Comprehension nur eben für Generatoren.
Hier erzeugen wir uns aus dem Generator für die natürlichen Zahlen einen neuen Generator für die ungeraden Quadratzahlen.

Sage code
```
odd_squares = (x^2 for x in count() if x % 2 == 1)
```

Sage code
```
for i in range(4):
    print odd_squares.next()
```

```
1
9
25
49
```

Die Funktion **take** nimmt einen Generator **g** und eine nate Zahl **n**, und gibt eine Liste der nächsten n Elemente des Generators zurück.

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

Sage code
```
take(odd_squares, 5)
```

```
[81, 121, 169, 225, 289]
```

Zurück zum Beispiel:

Sage code
```
def pi_generator(digits = 100):
    R = parent(1.n(digits = digits)+20)
    summe = 3
    n = 1
    while True:
        summe += R(3*binomial(2*n, n) / ((2*n+1)*16^n))
        n += 1
        yield summe
```

Sage code
```
g = pi_generator(digits = 100)
print g.next()
print g.next()
print g.next()
```

```
3.12500000000000000000000000000000000000000000000000000000000000000000000000000000000\
0000000000000000
3.13906250000000000000000000000000000000000000000000000000000000000000000000000000000\
0000000000000000
3.14115513392857142857142857142857142857142857142857142857142857142857142857142857142\
8571428571428571
```

Sage code
```
def approximate_pi(digits):
    k = 1
    terminate = False
    calc_digits = digits + 20

    pi_gen = pi_generator(digits = calc_digits)

    pi_digits = pi.n(digits = calc_digits)
    epsilon = (10^(-digits)).n(digits = calc_digits)

    while not terminate:
        pi_approx = pi_gen.next()
        terminate = abs(pi_approx - pi_digits) < epsilon
        k += 1

    return (k, pi_approx)
```

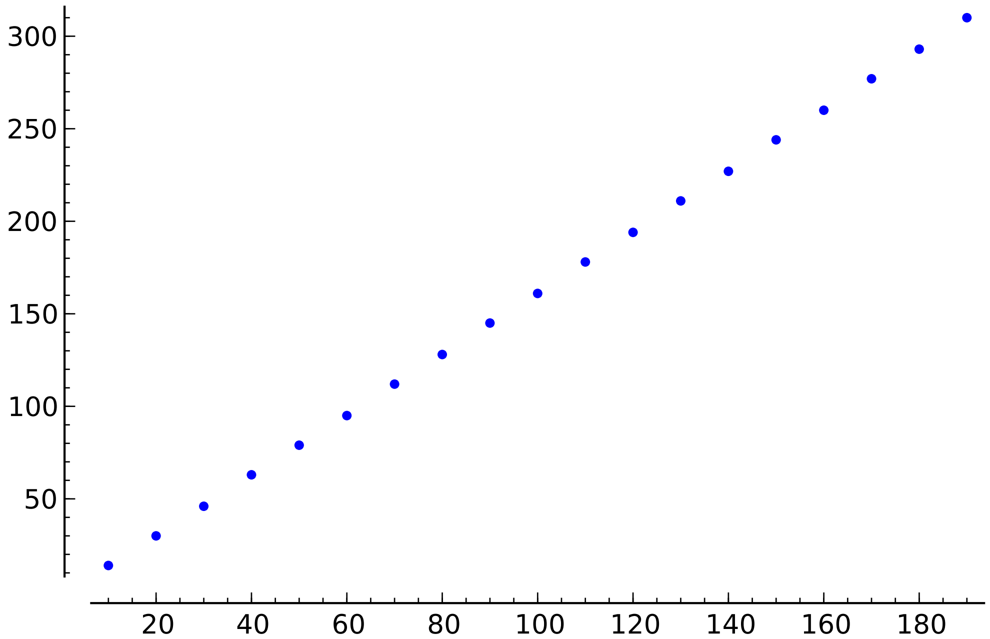
Sage code
```
approximate_pi(200)
```

```
(327, 3.14159265358979323846264338327950288419716939937510582097494459230781640628620\
8998628034825342117067982148086513282306647093844609550582231725359408128481117450284\
102701938521105559644622948954930381961884752011948290212)
```

Sage code
```
liste2 = [(k, approximate_pi(k)[0]) for k in range(10, 200, 10)]
```

```
CPU time: 0.14 s,  Wall time: 0.15 s
```

Sage code
```
list_plot(liste2)
```



300
250
200
150
100
50
20
40
60
80
100
120
140
160
180