Beispiel 20-27

Computermathematik (für Informatik)

3. Übungsblatt (Musterlösung)
19. 11. 2008

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 20

Bestimmen Sie den ganzzahligen Anteil und den Rest bei der Polynomdivision von:

$$\frac{x^8 + 2x^7 - 3x^5 + 2x^4 - x^3 + x - 3}{x^3 - 2x^2 - x - 1}$$

```
P.<x> = QQ[]
```

```
p1 = x^8 + 2*x^7 - 3*x^5 + 2*x^4 - x^3 + x - 3
p2 = x^3 - 2*x^2 - x - 1
show(p1/p2)
```

$$\frac{x^8 + 2x^7 - 3x^5 + 2x^4 - x^3 + x - 3}{x^3 - 2x^2 - x - 1}$$

```
p1 // p2
```

```
x^5 + 4*x^4 + 9*x^3 + 20*x^2 + 55*x + 138
```

```
p1 % p2
```

```
351*x^2 + 194*x + 135
```

```
g, r = p1.quo_rem(p2)
show(g)
show(r)
```

$$x^5 + 4x^4 + 9x^3 + 20x^2 + 55x + 138$$

$$351x^2 + 194x + 135$$

```
show(g * p2 + r)
```

$$x^8 + 2x^7 - 3x^5 + 2x^4 - x^3 + x - 3$$

Beispiel 21

Bestimmen Sie die Partialbruchdarstellung der folgenden rationalen Funktion:

$$\frac{2x^5 + 11x^4 + 3x^3 - 19x^2 - 16x - 12}{x^6 + 2x^5 - 5x^4 - 12x^3 + 3x^2 + 18x + 9}$$

```
var('x')
```

$$x$$

```
p3 = 2*x^5 + 11*x^4 + 3*x^3 - 19*x^2 - 16*x - 12
p4 = x^6 + 2*x^5 - 5*x^4 - 12*x^3 + 3*x^2 + 18*x + 9
show(p3 / p4)
```

$$\frac{2x^5 + 11x^4 + 3x^3 - 19x^2 - 16x - 12}{x^6 + 2x^5 - 5x^4 - 12x^3 + 3x^2 + 18x + 9}$$

```
show((p3 / p4).partial_fraction())
```

$$\frac{-(7x - 52)}{4(x^2 - 3)} - \frac{8x - 27}{2(x^2 - 3)^2} + \frac{15}{4(x + 1)} - \frac{9}{4(x + 1)^2}$$

Beispiel 22

Sei $s(a)$ die jeweils größte reelle Lösung der Gleichung

$$x^5 + x^4 - x^3 + x^2 - x - a = 0.$$

Schreiben Sie eine Funktion die $s(a)$ numerisch berechnet. Und zeichnen sie den Graphen der Funktion im Interval $a \in [0, 100]$. **Hinweis:** Verwenden Sie als Datentyp Polynome mit Koeffizienten in den rellen Zahlen.

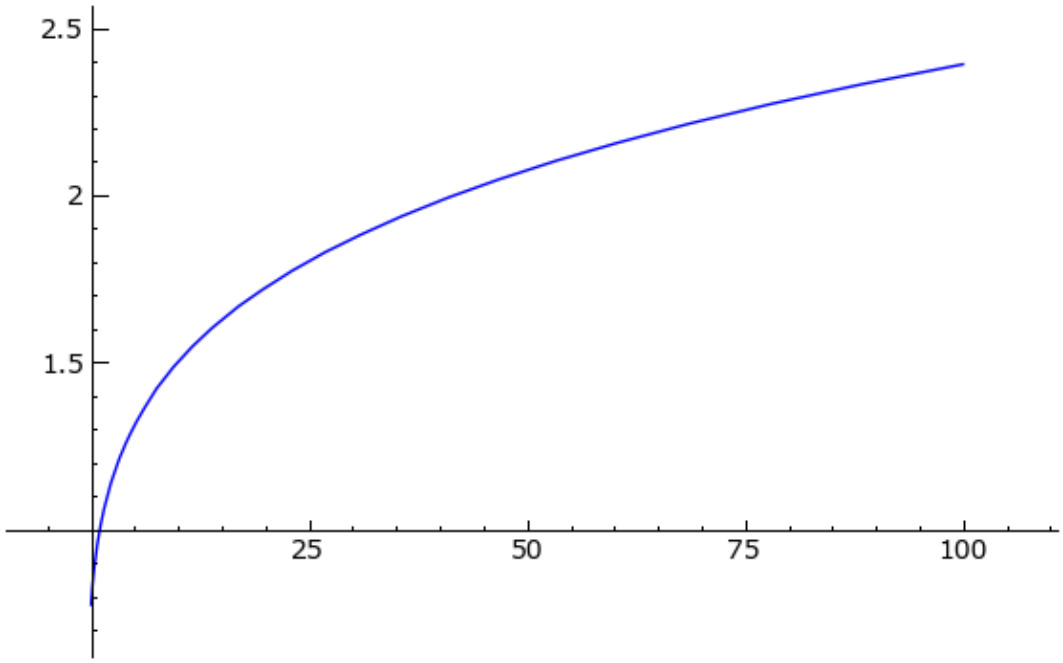
```
P.<x> = RR[]
```

```
polynom = x^5 + x^4 - x^3 + x^2 - x
show(polynom)
```

$$1.00000000000000x^5 + 1.00000000000000x^4 - 1.00000000000000x^3 + 1.0000000$$

```
def finde_groesste_nullstelle(a):
 p = polynom - a
 return max(p.roots(multiplicities = False))
```

```
plot(finde_groesste_nullstelle, (0, 100))
```



Beispiel 23

Bestimmen Sie symbolisch den Wert der unendlichen Reihe

$$\sum_{n=1}^{\infty} \frac{2^n (n!)^2}{(2n)!},$$

und überprüfen sie numerisch ob die Lösung korrekt sein kann.

Hinweis: Verwenden Sie das Maxima Interface von Sage, und den Maxima Befehl simplify_sum aus dem Maxima Paket simplify_sum.

```
maxima.load("simplify_sum")
```

```
    "/local/data/huss/software/sage-3.2.1/local/share/maxima/5.16.3/sha
    e/contrib/solve_rec/simplify_sum.mac"
```

```
%maxima
s: sum((2^n*(n!)^2)/(2*n)!, n, 1, inf)
```

```
    'sum(2^n*n!^2/(2*n)!,n,1,inf)
```

```
show(maxima('s'))
```

$$\sum_{n=1}^{\infty} \frac{2^n\, n!^2}{(2\, n)!}$$

```
summe = maxima.simplify_sum('s').sage().expand()
show(summe)
```

$$\frac{\pi}{2} + 1$$

```
summe.n(digits = 100)
```

```
2.5707963267948966192313216916397514420985846996875529104874722961539082031431044993140174126710585339
9
```

```
def summe_numerisch(k):
 return sum([2^n*(factorial(n))^2/factorial(2*n) for n in [1..k]])
```

```
summe_numerisch(1000).n(digits = 100)
```

```
2.5707963267948966192313216916397514420985846996875529104874722961539082031431044993140174126710585345
34
```

Beispiel 24

Finden Sie eine geschlossene Formel für die Summe

$$\sum_{k=1}^{n} 3k^2 + 3k.$$

Beweisen Sie Ihre Formel durch vollständige Induktion. Die algebraischen Umformungen für diesen Beweis sollen natürlich auch mit Sage vorgenommen werden.

Hinweis: Verwenden Sie das Maxima Interface von Sage, und den Maxima Befehl simplify_sum aus dem Maxima Paket simplify_sum.

```
maxima.load("simplify_sum")
```

```
"/local/data/huss/software/sage-3.2.1/local/share/maxima/5.16.3/share/contrib/solve_rec/simplify_sum.mac"
```

```
var('k, n')
a = 3*k^2 + 3*k
```

```
maxima("a: %s" % repr(a))
```

```
3*k^2+3*k
```

```
%maxima
s1: sum(a, k,1,n)
```

```
'sum(3*k^2+3*k,k,1,n)
```

```
show(maxima('s1'))
```

$$\sum_{k=1}^{n} 3\,k^2 + 3\,k$$

```
sumexpr(n) = maxima.simplify_sum('s1').sage().factor()
show(sumexpr(n))
```

$$n\,(n+1)\,(n+2)$$

Induktionsbasis:

```
bool(sumexpr(1) == a(1))
```

```
True
```

Induktionsschritt:

```
bool(sumexpr(n+1).expand() == (sumexpr(n) + a(n+1)).expand())
```

```
True
```

Beispiel 25

Eine Partition einer Menge A ist eine Zerlegung $A = B_1 \cup B_2 \cup \cdots \cup B_k$ in paarweise disjunkte, nichtleere Teilmengen $B_j \subseteq A$. Die Anzahl der verschiedenen Partitionen der Menge $A = \{1, 2, \ldots, n\}$ in k nichtleere Teilmengen wird mit $\left\{ {n \atop k} \right\}$ bezeichnet. Diese Zahlen heißen Stirlingsche Zahlen der 2. Art und sie erfüllen die Rekursion

$$\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k \left\{ {n-1 \atop k} \right\}.$$

mit

1. Schreiben Sie eine Funktion num_of_partitions(n, k) die $\left\{ {n \atop k} \right\}$ berechnet. Und erstellen Sie eine Tabelle mit allen Werten der Stirling Zahlen 2. Art für $n, k \leq 10$.
2. Berechnen Sie mit Ihrer Funktion $\left\{ {100 \atop 55} \right\}$ und vergleichen Sie Ihr Ergebnis mit der eingebauten Funktion **stirling_number2**

Direkte rekursive Implementation (**Achtung:** Hat exponentielle Laufzeit):

```
def num_of_partitions(n, k):
   if n == 0 and k == 0:
       return 1
   if n == 0 or k == 0:
       return 0
   if n > 0 and k > 0:
       return num_of_partitions(n-1, k-1) + k *
num_of_partitions(n-1, k)

   raise ValueError, "The number of Partitions is not defined for
negative parameters"
```

```
def print_table(f):
```

```
    for n in [0..10]:
        if n == 0:
            string = "n\k| "
            line   = "-----"
            for k in [0..10]:
                string += "%5d " % k
                line += 6 * "-"
            print string
            print line

        string = "%3d| " % n
        for k in [0..10]:
            string += "%5d " % f(n, k)
        print string
```

```
print_table(num_of_partitions)
```

n\k	0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0	0
5	0	1	15	25	10	1	0	0	0	0
6	0	1	31	90	65	15	1	0	0	0
7	0	1	63	301	350	140	21	1	0	0
8	0	1	127	966	1701	1050	266	28	1	0
9	0	1	255	3025	7770	6951	2646	462	36	1
10	0	1	511	9330	34105	42525	22827	5880	750	45

```
print_table(stirling_number2)
```

n\k	0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0	0
5	0	1	15	25	10	1	0	0	0	0
6	0	1	31	90	65	15	1	0	0	0
7	0	1	63	301	350	140	21	1	0	0
8	0	1	127	966	1701	1050	266	28	1	0
9	0	1	255	3025	7770	6951	2646	462	36	1
10	0	1	511	9330	34105	42525	22827	5880	750	45

Rekursive Implementation mit Memoization (lineare Laufzeit, quadratischer Speicherbedarf):

```
partitions_cache = {}

def num_of_partitions(n, k):

    def num_of_partitions_local(n, k):
        if (n, k) not in partitions_cache:
            partitions_cache[(n, k)] = num_of_partitions(n, k)
        return partitions_cache[(n, k)]

    if n == 0 and k == 0:
        return 1
```

```
    if n == 0 or k == 0:
        return 0
    if n > 0 and k > 0:
        return num_of_partitions_local(n-1, k-1) + k *
num_of_partitions_local(n-1, k)

    raise ValueError, "The number of Partitions is not defined for
negative parameters"
```

```
time num_of_partitions(100, 55)
```

```
    447545635354868980237604061478458871381002267537240372484602855877307269241579203801277886440000
    CPU time: 0.02 s,  Wall time: 0.03 s
```

```
time stirling_number2(100, 55)
```

```
    447545635354868980237604061478458871381002267537240372484602855877307269241579203801277886440000
    CPU time: 0.00 s,  Wall time: 0.00 s
```

```
sys.setrecursionlimit(2000)
```

```
time num_of_partitions(500, 250)
```

```
    129681409386815659149956123071140497183128165459666873211756902642656739703856363800950644188543874136543268023589088351758963601321894362751412254041252960253057496423704017978882817887519450342825106886173512274297037628923235394981438292636445378612615192322752959891396386576594180741573746673369922420878438081108723641817748775601525713566470840674828345653331572263002548660795067959227701661835563922950971577389835753890941640717044293921187618577832716284672252407390253102769381762295879807451335169622139964709297551222075785074756607643907225902769917862322488513311324203510704050803467408214494767044333594064937814998126268277270519586969869754960202015591132000
    CPU time: 0.78 s,  Wall time: 0.79 s
```

```
time stirling_number2(500, 250)
```

```
    129681409386815659149956123071140497183128165459666873211756902642656739703856363800950644188543874136543268023589088351758963601321894362751412254041252960253057496423704017978882817887519450342825106886173512274297037628923235394981438292636445378612615192322752959891396386576594180741573746673369922420878438081108723641817748775601525713566470840674828345653331572263002548660795067959227701661835563922950971577389835753890941640717044293921187618577832716284672252407390253102769381762295879807451335169622139964709297551222075785074756607643907225902769917862322488513311324203510704050803467408214494767044333594064937814998126268277270519586969869754960202015591132000
    CPU time: 0.00 s,  Wall time: 0.01 s
```

Memoization Decorator:

```
def remember(f, cache = {}):
   def g(*args):
       key = (f, tuple(args))
       if key not in cache:
           cache[key] = f(*args)
       return cache[key]
   return g
```

```
@remember
def num_of_partitions(n, k):
   if n == 0 and k == 0:
       return 1
   if n == 0 or k == 0:
       return 0
   if n > 0 and k > 0:
       return num_of_partitions(n-1, k-1) + k *
num_of_partitions(n-1, k)

   raise ValueError, "The number of Partitions is not defined for
negative parameters"
```

```
time num_of_partitions(500, 250)
```

```
129681409386815659149956123071140497183128165459666873211756902642
567397038563638009506441885438741365432680235890883517589636013218
436275141225404125296025305749642370401797888281788751945034282510
886173512274297037628923235394981438292636445378612615192322752959
913963865765941807415737466733699224208784380811087236418177487756
152571356647084067482834565333157226300254866079506795922770166183
563922950971577389835753890941640717044293921187618577832716284672
524073902531027693817622958798074513351696221399647092975512220757
507475660764390722590276991786232248851331132420351070405080346740
214494767044333594064937814998126268277270519586969869754960202015
91132000
CPU time: 1.34 s,  Wall time: 1.36 s
```

Beispiel 26

Die Chebyshev-Polynome sind definiert durch die Rekursionsformel:
$T_0(x) = 1 \quad T_1(x) = x$ und für $n > 1$:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).$$

Schreiben Sie eine Funktion **chebyshev(n)** die das n-te Chebyshev-Polynom berechnet.

1. Berechnen Sie die ersten 10 Chebyshev-Polynome.
2. Schreiben Sie eine Tabelle der ersten 10 Chebyshev-Polynome in die Datei **tabelle.tex**. Schreiben Sie ein LaTeX-Dokument chebyshev.tex welches die Datei tabelle.tex inkludiert und die Tabelle ausgibt. Das Ergebnis soll ungefähr folgendermassen aussehen:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ &\vdots \end{aligned}$$

3. Erzeugen Sie den Graphen der Chebyshev-Polynome $T_0(x), \cdots, T_5(x)$ (alle Polynome sollen gemeinsam in ein Bild gezeichnet werden).
4. Finden Sie heraus wie man Sage Grafiken in eine PDF-Datei exportieren kann. Exportieren Sie den Graphen aus Punkt 3, und binden Sie in das Dokument chebyshev.tex ein.

```
P.<x> = QQ[]
```

Mit Memoization:

```
@remember
def chebyshev(n):
 if n == 0:
   return P(1)
 if n == 1:
   return x
 else:
   return 2 * x * chebyshev(n - 1) - chebyshev(n - 2)
```

Iterativ:

```
def chebyshev(n):
 if n == 0:
   return P(1)

 cold, cnew = P(1), x

 for i in xrange(n - 1):
   cold, cnew = cnew, 2 * x * cnew - cold
 return cnew
```

```
for i in [0..10]:
 view(chebyshev(i))
```

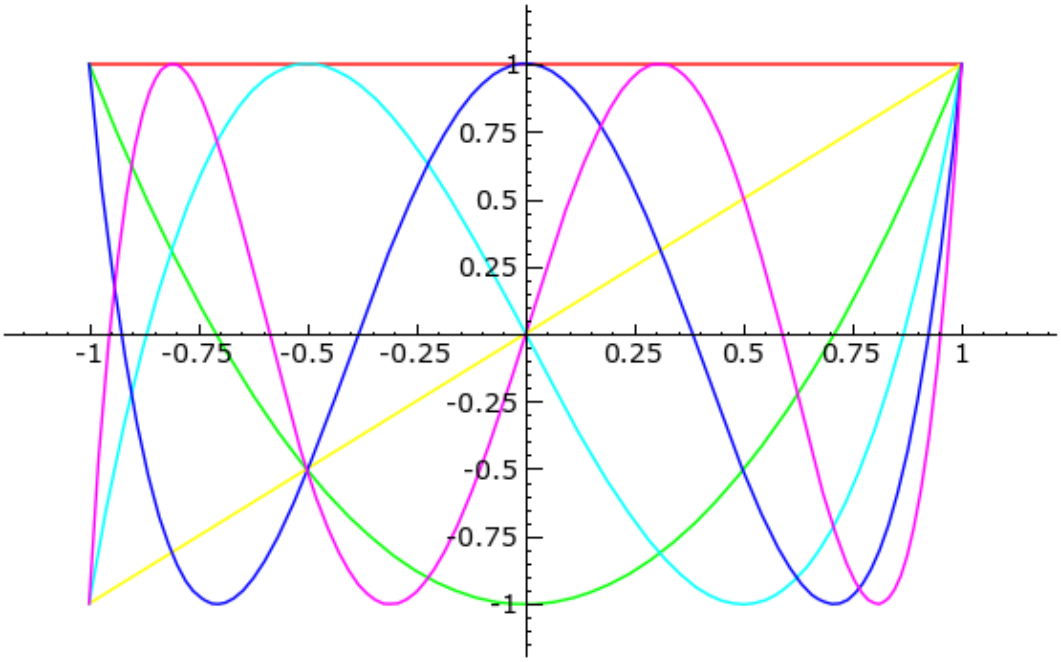
$$1$$
$$x$$
$$2x^2 - 1$$
$$4x^3 - 3x$$
$$8x^4 - 8x^2 + 1$$
$$16x^5 - 20x^3 + 5x$$
$$32x^6 - 48x^4 + 18x^2 - 1$$
$$64x^7 - 112x^5 + 56x^3 - 7x$$
$$128x^8 - 256x^6 + 160x^4 - 32x^2 + 1$$
$$256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x$$
$$512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1$$

```
datei = open("tabelle.tex", 'w')
datei.write("\\begin{align*}\n")
for i in [0..10]:
 datei.write("T_{%d}(x) &= %s \\\\\n" % (i, latex(chebyshev(i))))
datei.write("\\end{align*}\n")
datei.close()
```

tabelle.tex

```
g = Graphics()
for i in [0..5]:
 g = g + chebyshev(i).plot(-1, 1, hue = i/6)

g.show()
```




```
g.save("chebyshev.pdf")
```

chebyshev.pdf

Beispiel 27

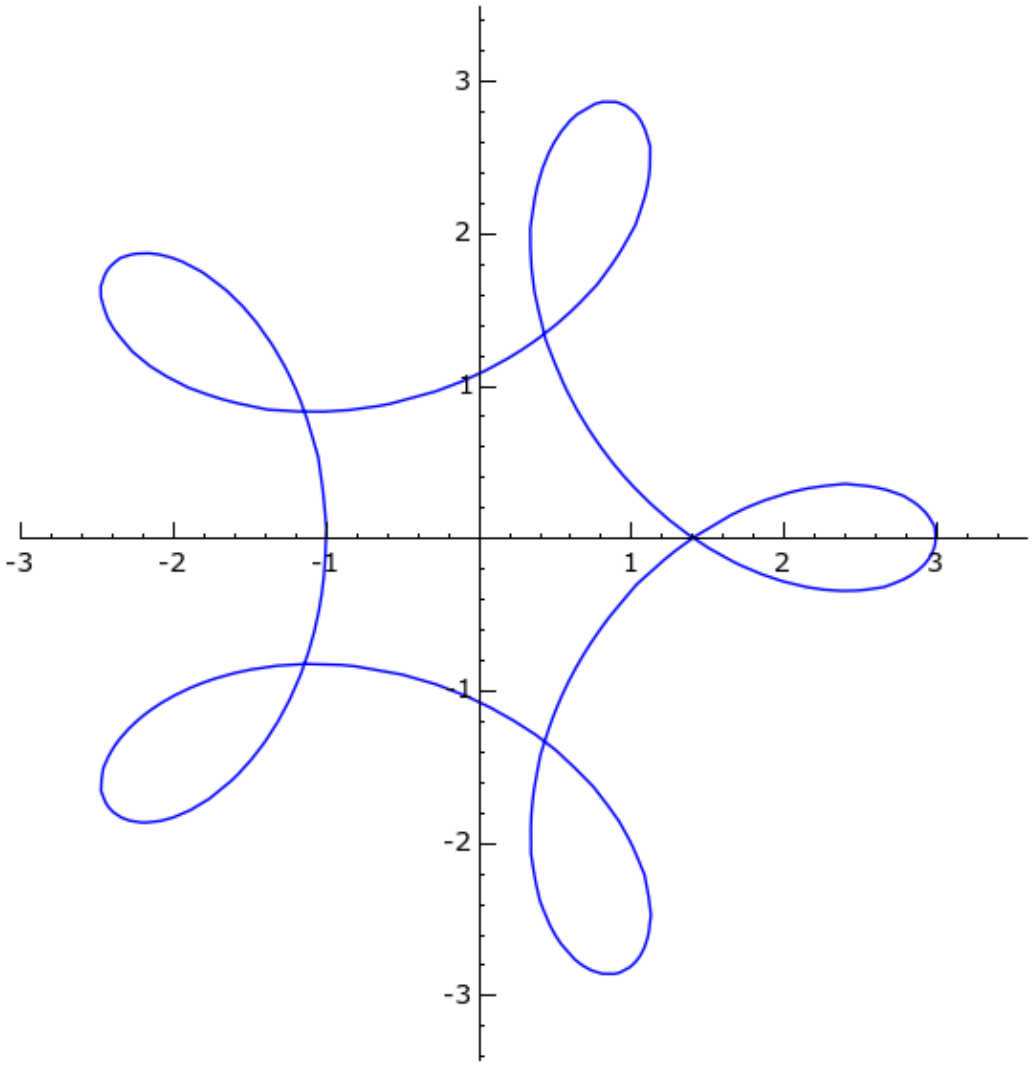
Eine Hypotrochoide ist eine in Parameterform definierte Kurve:

$$x(\phi) = (R-r)\cos(\phi) + d\cos\left(\frac{R-r}{r}\phi\right)$$
$$y(\phi) = (R-r)\sin(\phi) - d\sin\left(\frac{R-r}{r}\phi\right)$$

Zeichnen Sie die Hypotrochoide für die Parameter ($R=5$, $r=4$, $d=2$) und $0 \leq \phi \leq 8\pi$.

```
def hypotrochoid(R, r, d):
    xcoord = lambda phi: (R - r) * cos(RDF(phi)) + d * cos((R - r)/r
* RDF(phi))
    ycoord = lambda phi: (R - r) * sin(RDF(phi)) - d * sin((R - r)/r
* RDF(phi))
    return (xcoord, ycoord)
```

```
parametric_plot(hypotrochoid(5, 4, 2), 0, 8*pi).show(aspect_ratio =
1)
```


3
2
1
-1
-2
-3
-3
-2
-1
1
2
3