UNIKASSEL
VERSITÄT

BACHELORARBEIT

ZUM THEMA

Formale Potenzreihen in Sage

Autor:
Alicenap TAVSANLI
Weinbergstraße 16
34117 Kassel

Betreuer:
Prof. Dr. Wolfram KOEPF
Institut für Mathematik
Heinrich-Plett-Str. 40
34132 Kassel

Kassel, den 2. Juni 2014

Inhaltsverzeichnis

1 Abstrakt

Viele elementare Funktionen aus Analysis lassen sich durch hypergeometische Funktionen ausdrücken. Im Rahmen dieser Bachelorarbeit wird ein Algorithmus vorgestellt, der für Funktionen vom hypergeometrischen Typ eine Potenzreihendarstellung in geschlossener Form liefert.

Formale Potenzreihen spielen in diesem Zusammenhang eine große Rolle. Wir werden sehen, welche Möglichkeiten die freie Open-Source-Software `Sage` zur Verfügung stellt, um mit formalen Potenzreihen zu rechnen.

Die nötige Theorie wird schrittweise anhand von Beispielen sowohl in `Sage` als auch -falls notwendig- in `Mathematica` aufgebaut. Dabei werden uns bestimmte elementare Funktionen durch die ganzen Beispiele begleiten.

An vielen Stellen wird der Vergleich zu der kommerziellen Software `Mathematica` gezogen. Insbesondere wird `Mathematica` dann unterstützend zu der Theorie eingesetzt, wenn `Sage` hierfür keine Mittel zur Verfügung stellt.

Diese Bachelorarbeit behandelt hauptsächlich einen Auschnitt aus dem Kapitel 10 des Buches Computeralgebra - Eine algorithmisch orientierte Einführung- von Herrn Prof. Dr. Wolfram Koepf.

2 Die mathematische Open-Source-Software Sage

2.1 Über Sage

Sage ist eine freie Open-Source-Software, die Forschung und Lehre in Algebra, Geometrie, Zahlentheorie, Kryptographie, numerischen Berechnungen und verwandten Gebieten unterstützt.

Sage benutzt hochoptimierte ausgereifte Software wie GMP, PARI, GAP und NTL, und ist somit bei vielen Aufgaben sehr schnell. Sage stellt robuste Schnittstellen zu vielen anderen Computeralgebrasystemen, einschliesslich PARI, GAP, Singular, Maxima, KASH, Magma, Maple und Mathematica zur Verfügung. Sage ist dazu gedacht, bestehende Mathematik-Software zu vereinheitlichen und zu erweitern. Als bindendes Glied hierzu und für das Arbeiten in Sage wurde die leicht erlernbare und zunehmend beliebte Skriptsprache Python eingesetzt.

Sowohl das Entwicklungsmodell von Sage als auch die Technologie in Sage zeichnen sich durch eine extrem starke Betonung von Offenheit, Gemeinschaft, Kooperation und Zusammenarbeit aus. Das Ziel von Sage ist es, eine aktiv gepflegte und freie Open-Source-Alternative zu Magma, Maple, Mathematica und Matlab zu entwickeln.

Falls Sie Sage auf Ihrem Computer nicht installiert haben und nur ein paar Befehle ausführen möchten, können Sie es online unter http://www.sagenb.org benutzen.

2.2 Installation von Sage

Sage steht unter der Creative Commons Attribution-Share Alike 3.0 License und ist für die Betriebssysteme Linux und OS X als binäre Installationsdateien oder als Source-Code zum Kompilieren erhältlich. Für das Betriebssystem Microsoft Windows geht der beste Weg über eine Installation von VirtualBox for Windows im Zusammenhang mit der zugehörigen binären Distribution von Sage.

Zur Installation von Sage benötigt man eine freie Festplattenkapazität von mindestens 2 Gigabyte. Für den reibungslosen Betrieb empfiehlt sich eine vorinstallierte LaTeX-Umgebung auf dem Rechner.

Nähere Informationen und Anleitungen zur Installation von Sage findet man unter:
http://www.sagemath.org/doc/installation/index.html

2.3 Die Anwendungsmöglichkeiten von Sage

Für das Arbeiten mit Sage gibt es verschiedene Möglichkeiten, die im Folgenden kurz erläutert werden.

2.3.1 Die interaktive Kommandozeile

Sie rufen dazu einfach aus einem Terminal-Fenster den Befehl

```
sage
```

auf, womit Sage nach einer kurzen Meldung über die zugehörige Versionsnummer und Erscheinungsdatum in einem interaktiven Modus zur Verfügung steht. In der Kommandozeile wird nun als Indikator `sage:` vorangestellt, dass Sage bereit ist, ihre Befehle entgegenzunehmen. Um Sage zu beenden drücken Sie Strg-D oder geben Sie

```
sage: quit
```

oder gleichbedeutend

```
sage: exit
```

ein. Sofern man es nicht beendet, ist zusätzlich zu vermerken, dass das graphische Notebook-Interface direkt über die interaktive Kommandozeile gestartet werden kann.

2.3.2 Das graphische Notebook-Interface

Nach der Installation von Sage wird eine Schnittstelle zum Standard-Browser Ihres Betriebssystems zur Verfügung gestellt. Diese ist sicherlich aufgrund der graphischen Oberfläche die angenehmste Variante für den Umgang mit Sage. Man muss dazu in einer aktiven Sage-Sitzung den Befehl

```
sage: notebook()
```

aufrufen. Der Befehl startet das Sage Notebook und ebenso Ihren Standardbrowser. Die Serverstatus-Dateien liegen unter `$HOME/.sage/sage\_notebook`. Die andere Optionen enthalten z.B.

```
sage: notebook("Verzeichnis")
```

was einen neuen Notebook Server mit den Dateien aus dem angegebenen Verzeichnis startet (anstelle des Standardverzeichnises `$HOME/.sage/sage_notebook`). Das kann

hilfreich sein, wenn man einige Worksheets für ein Projekt oder verschiedene gleichzeitig laufende Notebook Server von einander trennen will.

2.3.3 Programme

Man kann in Sage interpretierte oder kompilierte Programme schreiben und diese Sage-Dateien laden und anhängen. Erstellt man eine separate Datei `beispiel.sage` mit dem folgenden Inhalt:

```
print "Hello World"
print 2^3
```

so kann man sie einlesen und ausführen, indem man den `load`-Befehl verwendet.

```
sage: load "beispiel.sage"
Hello World
8
```

Man kann auch eine Sage-Datei an eine laufende Sitzung anhängen, indem man den `attach`-Befehl verwendet:

```
sage: attach "beispiel.sage"
Hello World
8
```

Wenn man nun `beispiel.sage` verändert und eine Leerzeile in Sage eingibt (d.h. `return` drückt) wird der Inhalt von `beispiel.sage` automatisch in Sage neu geladen. Insbesondere lädt der `attach`-Befehl eine Datei jedesmal, wenn diese verändert wird automatisch neu, was beim Debuggen von Code nützlich sein kann. Der `load`-Befehl hingegen lädt eine Datei nur einmal.

Wenn Sage die Datei `beispiel.sage` lädt, wird sie zum Python-Code konvertiert, welcher dann vom Python-Interpreter ausgeführt wird. Diese Konvertierung ist geringfügig; sie besteht hautsächlich daraus Integer-Literale mit `Integer()` und Fließkomma-Literale mit `RealNumber()` zu versehen, `^` durch ** zu ersetzen und z.B. `R.2` durch `R.gen(2)` auszutauschen. Die konvertierte Version von `beispiel.sage` befindet sich im gleichen Verzeichnis wie `beispiel.sage` und ist `beispiel.sage.py` genannt. Diese Datei enthält den folgenden Code:

```
print "Hello World"
print Integer(2)**Integer(3)
```

Integer-Literale wurden mit `Integer()` versehen und das ^ wurde durch ein ** ersetzt. (In Python bedeutet ^ "exklusives oder" und ** bedeutet "Exponentiation".)

2.3.4 Skripte

Man kann eigenständige Pythonskripte schreiben, welche die Sage-Bibliotheken einbinden. Das folgende eigenständige Sageskript faktorisiert ganze Zahlen und Polynome:

```
#!/usr/bin/env sage -python

import sys
from sage.all import *

if len(sys.argv) != 2:
    print "Usage: %s <n>"%sys.argv[0]
    print "Outputs the prime factorization of n."
    sys.exit(1)

print factor(sage_eval(sys.argv[1]))
```

Um dieses Skript benutzen zu können muss `SAGE\_ROOT` in ihrer `PATH`-Umgebungsvariable enthalten sein. Falls das obige Skript factor genannt wurde, ist hier ein beispielhafter Aufruf:

```
bash $ ./factor 2006
2 * 17 * 59
bash $ ./factor "32*x^5-1"
(2*x - 1) * (16*x^4 + 8*x^3 + 4*x^2 + 2*x + 1)
```

Geschwindigkeit ist bei mathematischen Berechnungen äusserst wichtig. Python ist zwar eine komfortable Programmiersprache mit sehr hohem Abstraktionsniveau, jedoch können bestimmte Berechnungen mehrerer Größenordnungen schneller als in Python sein. Insbesondere wenn die Berechnungen in einer kompilierten Sprache mit statischen Datentypen implementiert wurden. In solch einem Fall würden Teile von `Sage` zu langsam agieren, wenn diese komplett in Python geschrieben wären. Um dies zu berücksichtigen unterstützt Sage eine kompilierte "Version" von Python, welche Cython ([Cyt] und [Pyr]) genannt wird. Cython ist gleichzeitig sowohl zu Python, als auch zu C ähnlich. Die meisten von Pythons Konstruktionen, einschliesslich "list comprehensions", bedingte Ausdrücke und Code wie += sind erlaubt; Sie können auch Code importieren, den Sie in anderen Python-Modulen geschrieben haben. Darüberhinaus können Sie beliebige C Variablen definieren und C-Bibliothekaufrufe direkt ausführen. Der daraus entstehende Code wird nach C konvertiert und mithilfe eines C-Compilers kompiliert.

3 Formale Potenzreihen

3.1 Definition: Formale Potenzreihe

Ein Ausdruck in der Form

$$a(x) := a_0\, x^0 + a_1\, x^1 + \cdots = \sum_{k=0}^{\infty} a_k\, x^k \quad \text{mit} \quad a_k \in R$$

nennen wir eine formale Potenzreihe, wobei R ein Integritätsbreich ist und die Folge $(a_k)_{k\in\mathbb{N}_0}$ als die definierende Folge mit $a_k \in R$ für $a(x)$ bezeichnet wird. Der Koeffizient a_0 wird als Absolutglied der formalen Potenzreihe gennant. Die Potenzreihe heißt Nullreihe, wenn alle $a_k = 0$ sind und sie wird konstante Potenzreihe genannt, wenn nur $a_0 \neq 0$ ist. Die Ordnung der Potenzreihe ist der kleinste Index k mit $a_k \neq 0$ und wird mit $ord(a(x), x)$ bezeichnet. Die Nullreihe hat die Ordnung ∞.

Die Menge der formalen Potenzreihen wird mit $R[[x]]$ bezeichnet. Falls R ein Integritätsbereich ist, so ist es auch $R[x]$. Und wir werden sehen, dass auch $R[[x]]$ einen Integritätsbereich darstellt, also einen nullteilerfreien, kommutativen Ring mit 1- Element. Offenbar gilt dann, dass $R[x]$ einen Unterring von $R[[x]]$ darstellt, und zwar mit endlich vielen von Null verschiedenen Koeffizienten $a_k \in R$.

Für $a(x), b(x), c(x) \in R[[x]]$ erfolgt die Addition $c(x) = a(x) + b(x)$ gliedweise mit den Koeffizienten, $c_k = a_k + b_k$ für alle $k \in \mathbb{N}_0$. Die Multiplikation $c(x) = a(x) \cdot b(x)$ geschieht mit Hilfe des Cauchyprodukts $c_k = \sum_{j=0}^{k} a_j\, b_{k-j}$ für alle $k \in \mathbb{N}_0$, woraus auch die Nullteilerfreiheit folgt. Das Nullelement von $R[[x]]$ ist gegeben durch die Nullreihe und das 1- Element von $R[[x]]$ entspricht die Konstante 1.

Beispielsweise gilt auf der Seite der erzeugenden Funktionen $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$ und $e^{2x} = \sum_{k=0}^{\infty} \frac{2^k x^k}{k!}$. Ganz anders sieht die Rechnung aber auf der Seite der formalen Potenzreihen aus. Mit $e(x) := \sum_{k=0}^{\infty} \frac{1}{k!} x^k$ ergibt sich $c_k = \sum_{j=0}^{k} \frac{1}{j!} \cdot \frac{1}{(k-j)!} = \frac{1}{k!} \sum_{j=0}^{k} \frac{k!}{j!\,(k-j)!} = \frac{1}{k!} \sum_{j=0}^{k} \binom{k}{j} = \frac{2^k}{k!}$, also $e(x) \cdot e(x) = \sum_{k=0}^{\infty} \frac{2^k}{k!} x^k$.

Beachten Sie bitte, dass zum Beispiel die Potenzreihe $\sum_{k=0}^{\infty} k!\; x^k$ nirgends in $\mathbb{C}$ konvergiert und daher keine Taylorreihe der Analysis bzw. Funktionentheorie darstellt. Im Ring der formalen Potenzreihen spielt die Konvergenz keine Rolle. Wir dürfen für x keine Werte einsetzen, denn dafür wäre Konvergenz erforderlich.

Betrachten wir nun die Einheiten von $\mathbb{Z}[[x]]$. Es gilt $\mathbb{Z}^\times = \{-1, 1\} = \mathbb{Z}[x]^\times$ und $a(x) = (1 - x) \in \mathbb{Z}[x]$ hat kein Inverses bzgl. Multiplikation. Aber aufgefaßt als Element von

$\mathbb{Z}[[x]]$ hat es sehr wohl, denn $(1-x)\cdot(1+x+x^2+\cdots) = 1+(x-x)+(x^2-x^2)+\cdots = 1$. $\mathbb{Z}[[x]]$ hat also mehr Einheiten als nur -1 und 1. Beachten Sie nun, dass $a_0 \in \mathbb{Z}^\times$ ist. Das ist nämlich kein Zufall und der folgende Satz erklärt diesen Zusammenhang.

3.2 Satz: Zusammenhang zwischen $R^\times$ und $R[[x]]^\times$

Behauptung

$a(x) \in R[[x]]^\times \Leftrightarrow a_0 \in R^\times$

Beweis

"$\Rightarrow$":

$a(x) \in R[[x]]^\times \Rightarrow \exists b(x) \in R[[x]] : a(x)\cdot b(x) = 1$
$\Rightarrow a_0 \cdot b_0 = 1$ und $c_k = 0$ für alle $k > 0$
$\Rightarrow a_0 \in R^\times$

"$\Leftarrow$":

$a_0 \in R^\times \Rightarrow \exists b_0 \in R : a_0 \cdot b_0 = 1$
$c_0 := a_0 \cdot b_0 = 1$

Wir setzen nun $c_k = 0$ für alle $k > 0$

$c_k = \sum_{j=0}^{k} a_j \cdot b_{k-j} = 0 \Leftrightarrow a_0 \cdot b_k + \sum_{j=1}^{k} a_j \cdot b_{k-j} = 0 \Leftrightarrow b_k = -a_0^{-1} \sum_{j=1}^{k} a_j \cdot b_{k-j}$

Wir können somit iterativ die definierende Folge $(b_k)_{k\in\mathbb{N}_0}$ konstruieren und das inverse Element $b(x)$ bestimmen.

$\Rightarrow a(x) \in R[[x]]^\times$

□

Bemerkung

Falls R ein Körper ist, so ist jedes $a_0 \neq 0$ eine Einheit in R und jedes $a(x) \in R[[x]]$ mit $ord(a(x), x) = 0$ und $a_0 \neq 0$ eine Einheit.

3.3 Definition: Komposition von formalen Potenzreihen

Unter Komposition von formalen Potenzreihen verstehen wir den Ausdruck

$$(a \circ b)(x) = a(b(x)) := \sum_{k=0}^{\infty} a_k \cdot \left(\sum_{j=0}^{\infty} b_j \cdot x^j \right)^k$$

Dieser kann durch ausmultiplizieren wieder in die Gestalt einer formalen Potenzreihe überführt werden. Wir setzen aber $ord(b(x), x) = 1$ voraus.

3.4 Satz: Die inverse Abbildung einer formalen Potenzreihe

Behauptung

Sei $a(x) \in R[[x]]$ mit $ord(a(x), x) = 1$ und $a_1 \in R^\times$
$\Rightarrow \exists i(x) \in R[[x]] : i(a(x)) = x$

Beweis

Gesucht ist $i(a(x)) = x$.
Durch Einsetzen $i(a(x)) = i_0 + a_1 i_1 x + \left(a_2 i_1 + a_1^2 i_2\right) x^2 + \left(a_3 i_1 + a_1 a_2 i_2 + a_1^3 i_3\right) x^3 + ... = x$ und Koeffizientenvergleich erhalten wir $i_0 = 0$, $i_1 = a_1^{-1}$ und iterative alle i_k, weil wir die Ausdrücke in Klammern gleich Null setzen und somit Wegen der Linearität des zu bestimmenden Termes eindeutig nach i_k auflösen können, da a_1 eine Einheit ist. Dies liefert $i(x)$.

□

4 Formale Potenzreihen in Sage

Man kann in `Sage` mit formalen Potenzreihen rechnen. Diese existieren aber nicht als unendliche Summen. Sie werden als abgebrochene Reihen dargestellt, weswegen bei den Ergebnissen eine Abbruchsordnung in Groß-O-Notation angeben wird. Dies hat zur Folge, dass die Berechnungen ungenau sind.

Beispiele

Wir definieren zwei formale Potenzreihen $s_1, s_2 \in \mathbb{Z}[[x]]$ mit Hilfe von Listen und mit der Abbruchsordnung 11:

```
R.<x> = PowerSeriesRing(ZZ)

s1 = R([1]*11,11)
s2 = R([factorial(n) for n in range(0,11)],11)

s1
   1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8
   + x^9 + x^10 + O(x^11)

s2
   1 + x + 2*x^2 + 6*x^3 + 24*x^4 + 120*x^5 + 720*x^6
   + 5040*x^7 + 40320*x^8 + 362880*x^9 + 3628800*x^10
   + O(x^11)
```

Die gesammte Arithmetik für formale Potenzreihen steht zur Verfügung:

```
s1+s2
   2 + 2*x + 3*x^2 + 7*x^3 + 25*x^4 + 121*x^5 + 721*x^6
   + 5041*x^7 + 40321*x^8 + 362881*x^9 + 3628801*x^10
   + O(x^11)

s1*s2
   1 + 2*x + 4*x^2 + 10*x^3 + 34*x^4 + 154*x^5 + 874*x^6
   + 5914*x^7 + 46234*x^8 + 409114*x^9 + 4037914*x^10
   + O(x^11)

s1/s2
   1 - x^2 - 4*x^3 - 17*x^4 - 88*x^5 - 549*x^6 - 3996*x^7
   - 33089*x^8 - 306432*x^9 - 3135757*x^10 + O(x^11)
```

Man kann ebenfalls ableiten. Dies verringert allerdings die Abbruchsordnung.

```
s1.derivative()
#Alternativ: derivative(s1)
   1 + 2*x + 3*x^2 + 4*x^3 + 5*x^4 + 6*x^5 + 7*x^6 + 8*x^7
   + 9*x^8 + 10*x^9 + O(x^10)
```

Integrieren funktioniert auch, aber man beachte:

```
s1.integral()
   Traceback (click to the left of this block for traceback)
   ...
   TypeError: no conversion of this rational to integer
```

Diese Operation muss in $\mathbb{Q}[[x]]$ ausgeführt werden.

```
s1.change_ring(QQ).integral()
#Alternativ: integral(s1.change_ring(QQ))
    x + 1/2*x^2 + 1/3*x^3 + 1/4*x^4 + 1/5*x^5 + 1/6*x^6
    + 1/7*x^7 + 1/8*x^8 + 1/9*x^9 + 1/10*x^10 + 1/11*x^11
    + O(x^12)
```

Dabei wird die Integrationskonstante immer 0 gewählt. Beim Integrieren wurde die Abbruchsordnung erhöht, die Division mit x^3 verringert sie jedoch wieder.

```
s1/x^3
    x^-3 + x^-2 + x^-1 + 1 + x + x^2 + x^3 + x^4 + x^5
    + x^6 + x^7 + O(x^8)
```

Das Ergebnis ist eine formale Laurentreihe

$a(x) := \sum_{k=k_0}^{\infty} a_k x^k \quad (k_0 \in \mathbb{Z})$

, bei welcher endlich viele negative Potenzen erlaubt sind. Ist $R = \mathbb{K}$ ein Körper, so bilden die formalen Laurentreihen ebenfalls einen Körper $\mathbb{K}((x))$, den Quotientenkörper von $\mathbb{K}[[x]]$.

Dividieren wir s_1 durch $(1-x^2)$, so wird $\frac{1}{(1-x^2)}$ in eine Potenzreihe umgewandelt und das entsprechende Produkt gebildet.

```
s3 = s1/(1-x^2)
s3
    1 + x + 2*x^2 + 2*x^3 + 3*x^4 + 3*x^5 + 4*x^6 + 4*x^7
    + 5*x^8 + 5*x^9 + 6*x^10 + O(x^11)
```

Dabei ist noch $s_1 \in \mathbb{R}[[x]]$.

```
s3.base_ring()
    Integer Ring
```

Wir können auch die Quadratwurzel von s_1 bilden.

```
s4 = s1.sqrt()
#Alternativ: s4 = sqrt(s1)
s4
    1 + 1/2*x + 3/8*x^2 + 5/16*x^3 + 35/128*x^4 + 63/256*x^5
    + 231/1024*x^6 + 429/2048*x^7 + 6435/32768*x^8
    + 12155/65536*x^9 + 46189/262144*x^10 + O(x^11)
```

Nun ist aber $s_4 \notin \mathbb{R}[[x]]$ sondern $s_4 \in \mathbb{Q}[[x]]$.

```
s4.base_ring()
   Rational Field
```

Eine andere Methode, um Quadratwurzel zu bilden, ist `square_root()`, welche nur dann ein Ergebnis liefert, wenn der Quadratwurzel in dem selben Ring liegt.

```
s1.square_root()
   Traceback (click to the left of this block for traceback)
   ...
   ValueError: Square root does not live in this ring.
```

Die formale Potenzreihen s_1 und s_2 hatten wir mit Hilfe von Listen erzeugt. Wir können eine formale Potenzreihe auch auf die folgende Art und Weise erstellen:

```
s5 = (1+x)^22
s5
   1 + 22*x + 231*x^2 + 1540*x^3 + 7315*x^4 + 26334*x^5
   + 74613*x^6 + 170544*x^7 + 319770*x^8 + 497420*x^9
   + 646646*x^10 + 705432*x^11 + 646646*x^12 + 497420*x^13
   + 319770*x^14 + 170544*x^15 + 74613*x^16 + 26334*x^17
   + 7315*x^18 + 1540*x^19 + 231*x^20 + 22*x^21 + x^22

s5.square_root()
   1 + 10*x + 45*x^2 + 120*x^3 + 210*x^4 + 252*x^5
   + 210*x^6 + 120*x^7 + 45*x^8 + 10*x^9 + x^10
   + O(x^20)
```

Bei der letzten Berechnung erscheint 20 als Abbruchsordnung. Dies ist der Standardwert, wenn man bei der Deklaration eines univariaten Rings, wie in unserem Fall, keine Abbruchsordnung angegeben hat. Bei multivariaten Ringen ist der Standardwert 12.

```
R.default_prec()
   20
```

Als wir s_5 erzeugt haben, haben wir keine Abbruchsordnung angegeben. Sie ist standardmäßig ∞, wie der Befehl `prec()` uns auch zeigt.

```
s5.prec()
   +Infinity
```

Wir hätten am Anfang für s_5 auch eine Abbruchsordnung angeben können.

```
s5 = (1+x)^22 + O(x^5)
```

```
s5
    1 + 22*x + 231*x^2 + 1540*x^3 + 7315*x^4 + O(x^5)

s5.prec()
    5
```

Folgende Berechnung führt aus $\mathbb{Q}[[x]]$ heraus.

```
sqrt(s1-1)
   Traceback (click to the left of this block for traceback)
   ...
   ValueError: power series does not have a square root
   since it has odd valuation.
```

Bemerkung

`Mathematica` liefert für die obige Berechnung $\sqrt{s_1 - 1}$ das Ergebnis:

$$\sqrt{x} + \frac{x^{3/2}}{2} + \frac{3x^{5/2}}{8} + \frac{5x^{7/2}}{16} + \frac{35x^{9/2}}{128} + \frac{63x^{11/2}}{256} + \frac{231x^{13/2}}{1024} + \frac{429x^{15/2}}{2048} + \frac{6435x^{17/2}}{32768} + \frac{12155x^{19/2}}{65536} + \frac{46189x^{21/2}}{262144} + \frac{88179x^{23/2}}{524288} + \frac{676039x^{25/2}}{4194304} + \frac{1300075x^{27/2}}{8388608} + \frac{5014575x^{29/2}}{33554432} + \frac{9694845x^{31/2}}{67108864} + \frac{300540195x^{33/2}}{2147483648} + \frac{583401555x^{35/2}}{4294967296} + \frac{2268783825x^{37/2}}{17179869184} + O[x]^{39/2}$$

Eine Reihe der Form

$$a(x) := \sum_{k=k_0}^{\infty} a_k x^{\frac{k}{q}} \quad (k_0 \in \mathbb{Z},\ q \in \mathbb{N})$$

heisst formale Puiseuxreihe. Hier ist aber zu beachten, dass in einer Puiseuxreihe nicht beliebige rationale Exponenten zugelassen sind, sondern dass die Exponenten einen gemeinsamen Hauptnenner besitzen müssen. Dadurch ensteht aus der Menge der Puiseuxreihen wieder ein Ring. `Mathematica` kann mit Puiseuxreihen rechnen. Das Ergebnis von $\sqrt{\frac{s_1-1}{x}}$ ist wieder eine Puiseuxreihe.

$$1 + \frac{x}{2} + \frac{3x^2}{8} + \frac{5x^3}{16} + \frac{35x^4}{128} + \frac{63x^5}{256} + \frac{231x^6}{1024} + \frac{429x^7}{2048} + \frac{6435x^8}{32768} + \frac{12155x^9}{65536} + \frac{46189x^{10}}{262144} + \frac{88179x^{11}}{524288} + \frac{676039x^{12}}{4194304} + \frac{1300075x^{13}}{8388608} + \frac{5014575x^{14}}{33554432} + \frac{9694845x^{15}}{67108864} + \frac{300540195x^{16}}{2147483648} + \frac{583401555x^{17}}{4294967296} + \frac{2268783825x^{18}}{17179869184} + O[x]^{19}$$

Fortsetzung der Beispiele

Die Komposition von formalen Potenzreihen kann in `Sage` einfach durch Einsetzen berechnet werden.

```
s2(s1-1)
    1 + x + 3*x^2 + 11*x^3 + 49*x^4 + 261*x^5 + 1631*x^6
    + 11743*x^7 + 95901*x^8 + 876809*x^9 + 8877691*x^10
    + O(x^11)
```

Die inverse Abbildung einer formalen Potenzreihe kann mit dem Befehl `reversion()` berechnet werden. Aber man beachte:

```
s2.reversion()
   Traceback (click to the left of this block for traceback)
   ...
   ValueError: Series must have valuation one for reversion.
```

Dies liegt daran, dass s_2 die Voraussetzungen von Satz 2.4 nicht erfüllt. Es gilt nämlich $ord(s_2(x), x) = 0$. Die Ordnung können wir mit dem Befehl `valuation()` abfragen, der sich in Übereinstimmung mit unserer Definition von Ordnung verhält.

```
s2.valuation()
    0

R(0).valuation()
    +Infinity

(-x^8 + O(x^11)).valuation()
    8
```

Man beachte hier ausserdem:

```
O(x^7).valuation()
    7
```

Die formale Potenzreihe $s_2 - 1$ leistet aber für unseren Zweck das Gewünschte.

```
(s2-1).valuation()
    1
```

Deshalb können wir sie invertieren.

```
inv=(s2-1).reversion()
inv
    x - 2*x^2 + 2*x^3 - 4*x^4 - 4*x^5 - 48*x^6 - 336*x^7
    - 2928*x^8 - 28144*x^9 - 298528*x^10 + O(x^11)

inv(s2-1)
    x + O(x^11)
```

Wenn für eine formale Potenzreihe keine Abbruchsordnung angegeben wurde, kann man den Befehl `reversion()` mit dem Parameter `precision` aufrufen. Ansonsten wird die Einstellung für die Abbruchsordnung von dem Ring, in dem die Operation ausgeführt wird, gültig sein.

```
x.prec()
   +Infinity

x.reversion(precision=5)
   x + O(x^5)

x.reversion()
   x + O(x^20)
```

Das Inverse einer formalen Potenzreihe erhält man wie Folgt:

```
s6 = 1/s1
#Alternativ: s6 = s1^-1
s6
   1 - x + O(x^11)

s1*s6
   1 + O(x^11)

s7 = 3+x^2
s7
   3 + x^2

s7^-1
   Traceback (click to the left of this block for traceback)
   ...
   TypeError: no conversion of this rational to integer

s7.is_unit()
   False

s7.base_ring()
   Integer Ring
```

Für $s_7 \in \mathbb{Z}[[x]]$ konnte kein Inverses bestimmt werden. Die Methode `is_unit()` liefert das Ergebnis, ob die formale Potenzreihe invertierbar ist. Dies ist nach Satz 2.2 genau dann der Fall, wenn das absolute Glied eine Einheit in $\mathbb{Z}$ ist, weswegen für $s_7(x)$ mit $s_{7_1} = 3 \notin \mathbb{Z}^\times$ kein Inverses existiert.

Die Methode `coefficients()` liefert alle Koeffizienten einer formalen Potenzreihe, die nicht Null sind. Im Gegensatz dazu liefert `list()` auch die Null-Koeffizienten.

```
s8=R([1,2,0,4])
s8
   1 + 2*x + 4*x^3

s8.coefficients()
   [1, 2, 4]

s8.list()
   [1, 2, 0, 4]
```

Ähnlich kann man mit der Methode `exponents()` die Exponenten von einer formalen Potenzreihe erhalten.

```
s2.exponents()
   [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Ein nützlicher Befehl ist auch `dict()`, womit man ein Dictionary-Objekt über die Koeffizienten von dem zugrundeliegenden Polynom einer formalen Potenzreihe erhält.

```
s2.dict()
   {0: 1, 1: 1, 2: 2, 3: 6, 4: 24, 5: 120, 6: 720,
   7: 5040, 8: 40320, 9: 362880, 10: 3628800}
```

Die Variable kann man mit dem Befehl `variable()` ausgeben lassen.

```
s2.variable()
   'x'
```

Mit der Methode `degree()` kann man den Grad einer formalen Potenzreihe abfragen, der nach Definition in `Sage` der Grad von dem zugrundelegenden Polyom ist, welches man mit dem Befehl `polynomial()` ausgeben kann. Man beachte aber, dass für das Null-Polynom -1 zurückgegeben wird.

```
s2.degree()
   10

s2.polynomial()
   3628800*x^10 + 362880*x^9 + 40320*x^8 + 5040*x^7
   + 720*x^6 + 120*x^5 + 24*x^4 + 6*x^3 + 2*x^2 + x + 1
```

```
R(0).degree()
    -1
```

Die Methode `is_square()` liefert für eine formale Potenzreihe x `True` zurück, wenn es in dem gegebenen Ring `self.parent()` ein Element y existiert, so dass $y^2 = x$ gilt.

```
K.<t> = PowerSeriesRing(QQ)

(1+t).parent()
    Power Series Ring in t over Rational Field

(1+t).is_square()
    True

(2+t).is_square()
    False

(2+t.change_ring(RR)).is_square()
    True
```

5 Berechnung formaler Potenzreihen

5.1 Einführung

Wir kennen unendliche Summen bereits aus der Analysis. Einige sind z.B:

$\sum_{k=0}^{\infty} \frac{x^k}{k!}$, $\sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} x^{2k+1}$, $\sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} x^{2k}$

Sage kann viele Ausdrücke wie Integrale, Gleichungen oder Summen vereinfachen. Sage ist auch in der Lage, die obigen unendlichen Reihen zu vereinfachen.

```
var('k','x')

sum(x^k/factorial(k), k, 0,infinity)
    e^x
```

Bei den nächsten zwei Ausdrücken ist es aber nicht mehr ersichtlich, um welche Funktionen es sich handelt.

```
sum(((-1)^(k))*(x^(2*k+1))/factorial(2*k+1),k,0,infinity)
    1/2*sqrt(2)*sqrt(pi)*sqrt(x)*bessel_J(1/2, x)

sum(((-1)^(k))*(x^(2*k))/factorial(2*k),k,0,infinity)
```

```
1/2*sqrt(2)*sqrt(pi)*sqrt(x)*bessel_J(-1/2, x)
```

Mathematica erkennt diese Reihen durch sein funktionales und regelbasiertes Programmierkonzept sofort.

$\sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} x^{2k+1}$

```
Sin[x]
```

$\sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} x^{2k}$

```
Cos[x]
```

Wir wollen jedoch den umgekehrten Weg gehen und uns im Folgenden ein Verfahren anschauen, welches uns unter bestimmten Vorraussetzungen erlaubt, für eine gegebene Funktion eine unendliche Reihe in geschlossener Form anzugeben. Bei diesem Verfahren geht es um die Bestimmung des k-ten Koeffizienten a_k einer Reihe. Allerdings umfaßt diese Prozedur nicht die Erzeugung einer Reihe wie zum Beispiel

$\tan x = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{2^{2k}(2^{2k}-1)}{(2k)!} B_{2k} x^{2k-1}$

zu deren Darstellung die Bernoullischen Zahlen B nötig sind. Wir wollen hingegen Reihen dann bestimmen, wenn ihre Koeffizienten sich durch Fakultäten oder Binomialkoeffizienten darstellen lassen. Vorher sehen wir uns aber die Reihendarstellung von $\tan x$ bis zum Grad 20 in Sage an.

```
taylor(tan(x),x,0,20)
    443861162/1856156927625*x^19 + 6404582/10854718875*x^17
    + 929569/638512875*x^15 + 21844/6081075*x^13
    + 1382/155925*x^11 + 62/2835*x^9 + 17/315*x^7 + 2/15*x^5
    + 1/3*x^3 + x
```

Mit Hilfe der Funktion `bernoulli()` können wir uns auch die Reihendarstellung von $\tan x$ anschauen und beide Ergebnisse vergleichen. Man beachte aber, dass der gewöhnliche Syntax für symbolische Summation hier fehlschlägt. Der Grund liegt darin, dass die Funktion `bernoulli()` symbolische Ausdrücke nicht verarbeiten kann und ihr Parameter im Vorfeld evaluiert werden muss.

```
sum((-1)**(k+1)*2**(2*k)*(2**(2*k)-1)/factorial(2*k)
*bernoulli(2*k)*x**(2*k-1),k,1,10)
    ...
```

```
    TypeError: unable to convert x (=2*k) to an integer
```

Daher empfiehlt es sich in solchen Situationen mit der folgenden Syntax zu verfahren:

```
sum((-1)**(k+1)*2**(2*k)*(2**(2*k)-1)/factorial(2*k)
*bernoulli(2*k)*x**(2*k-1) for k in [1..10])
    x + 1/3*x^3 + 2/15*x^5 + 17/315*x^7 + 62/2835*x^9
    + 1382/155925*x^11 + 21844/6081075*x^13
    + 929569/638512875*x^15 + 6404582/10854718875*x^17
    + 443861162/1856156927625*x^19
```

Dabei werden die Bernoullischen Zahlen von der Funktion

$\frac{x}{e^x-1} = \sum_{k=0}^{\infty} \frac{B_k}{k!} x^k$

erzeugt. Diese schauen wir uns auch in Sage an.

```
taylor(x/(exp(x)-1),x,0,20)
    -174611/802857662698291200000*x^20
    + 43867/5109094217170944000*x^18
    - 3617/106706228428800000*x^16
    + 1/74724249600*x^14 - 691/1307674368000*x^12
    + 1/47900160*x^10 - 1/1209600*x^8 + 1/30240*x^6
    - 1/720*x^4 + 1/12*x^2 - 1/2*x + 1

sum(bernoulli(k)*x**k/factorial(k) for k in [0..20])
    -174611/802857662698291200000*x^20
    + 43867/5109094217170944000*x^18
    - 3617/106706228428800000*x^16
    + 1/74724249600*x^14 - 691/1307674368000*x^12
    + 1/47900160*x^10 - 1/1209600*x^8 + 1/30240*x^6
    - 1/720*x^4 + 1/12*x^2 - 1/2*x + 1
```

5.2 Maxima

Das in Lisp implementierte Maxima ist ein Teil von Sage. Im Gegensatz zu Maxima wird das gnuplot-Paket (welches Maxima standardmäßig zum plotten nutzt) als optionales Sage-Paket angeboten. Neben anderen Dingen rechnet Maxima mit Symbolen. Maxima integriert und differenziert Funktionen symbolisch, löst gewöhnliche Differentialgleichungen ersten Grades sowie viele lineare Differentialgleichungen zweiten Grades und besitzt eine Methode zur Laplace Transformation linearer Differentialgleichungen von beliebigem Grad. Maxima kennt eine grosse Zahl spezieller Funktionen, plottet mit-

tels gnuplot und hat Methoden, um Polynomgleichungen oder Matrizen zu lösen oder zu verändern (z.B. Zeilenelimination oder Eigenwerte und Eigenvektoren berechnen). Nähere Informationen dazu findet man unter:

http://modular.math.washington.edu/home/wstein/www/home/bleveque/sage/devel/sage/doc/output/html/en/reference/sage/interfaces/maxima.html.

Anhand der Sage's Schnittstelle zur Maxima ermitteln wir nun für die Funktionen e^x, $\sin x$ und $\cos x$ die entsprechenden unendlichen Reihen.

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

```
e=exp(x)
maxima(e).powerseries('x',infinity)
    'sum(x^i1/factorial(i1),i1,0,inf)
```

In Sage kann man mit dem Befehl latex() die LaTeX-Repräsentation der Ausgabe erhalten und mit dem Befehl view() die Ausgabe in LaTeXverarbeiten.

```
view(maxima(e).powerseries('x',infinity))
```

$$\sum_{i_2=0}^{\infty} \frac{x^{i_2}}{i_2!}$$

Dabei sind i_k Laufvariablen, die von Maxima intern bei jeder Berechnung neu vergeben werden.

```
f=sin(x)
view(maxima(f).powerseries('x',infinity))
```

$$\sum_{i_3=0}^{\infty} \frac{(-1)^{i_3}\, x^{2\,i_3+1}}{(2\,i_3+1)!}$$

```
g=cos(x)
view(maxima(g).powerseries('x',infinity))
```

$$\sum_{i_4=0}^{\infty} \frac{(-1)^{i_4}\, x^{2\,i_4}}{(2\,i_4)!} \quad)$$

Maxima liefert auch für die Funktion $tanx$ die vorher betrachtete unendliche Reihe, für welche die Bernoullischen Zahlen nötig sind.

```
h=tan(x)
view(maxima(h).powerseries('x',infinity))
```

$$\sum_{i_5=0}^{\infty} \frac{(-1)^{i_5-1}\left(2^{2\,i_5}-1\right) 2^{2\,i_5}\, bern(2\,i_5)\, x^{2\,i_5-1}}{(2\,i_5)!}$$

Maxima's Implementierung für das Ermitteln von unendlichen Reihen in geschlossener

Form für gegebene Funktionen ist in unserem Fall nicht geeignet. Das Verfahren, das wir uns anschauen werden, ist am besten mit einem Paket namens `SpecialFunctions` zu verstehen, welches für `Mathematica` entwickelt wurde. Im Folgenden werden wir uns in `Mathematica` einen Überblick über das Algorithmus verschaffen, das wir anschließend mit weiterführender Theorie vervollständigen.

5.3 Überblick des Algorithmus

Für unsere Zwecke ist es erforderlich zunächst das Paket zu laden, weil `Mathematica` diese Funktionalität von Haus aus nicht unterstütz. Die Datei kann man unter http://www.mathematik.uni-kassel.de/ koepf/Publikationen/SpecialFunctions.m herunterladen.

Um das Package SpecialFunctions zu installieren verschiebt man die Datei `SpecialFunctions.m` in den Ordner `AddOns/ExtraPackages` , der sich in dem Verzeichnis befindet, wo man Mathematica installiert hat. Alternativ kann man das Verzeichnis, in dem `SpecialFunctions.m` liegt, mit dem `Mathematica`-Befehl `SetDirectory` angeben.

```
Needs["SpecialFunctions‘"]
```

```
SpecialFunctions, (C) Wolfram Koepf, version 2.04, 2012
Fast Zeilberger, (C) Peter Paule and Markus Schorn (V 2.2) loaded
```

Hier ist auch zu beachten, dass die Ausgabe von `FPS` (Für Eng.: Formal power series) nicht die Funktion `Sum` nutzt, weil sie die Ausgabe sofort wieder umwandeln würde. Stattdessen kommt der Funktionsname `sum` zum Einsatz.

`FPS` $[e^x, x]$

$$\texttt{sum}\left[\frac{x^k}{k!}, \{k, 0, \infty\}\right]$$

`FPS[Sin[`x`], `x`]`

$$\texttt{sum}\left[\frac{(-1)^k x^{1+2k}}{(1+2k)!}, \{k, 0, \infty\}\right]$$

`FPS[Cos[`x`], `x`]`

$$\texttt{sum}\left[\frac{(-1)^k x^{2k}}{(2k)!}, \{k, 0, \infty\}\right]$$

Beachten Sie nun das interessante Beweis

`FPS` $\left[e^{\sqrt{x}}, x\right]$

$$\texttt{sum}\left[\frac{x^k}{(2k)!}, \{k, 0, \infty\}\right] + \texttt{sum}\left[\frac{x^{\frac{1}{2}+k}}{(1+2k)!}, \{k, 0, \infty\}\right]$$

An diesem Beispiel kann man erkennen, dass diese Reihen nicht aus einer Tabelle entnommen werden, sondern, dass sie algorithmisch bestimmt werden. Durch Mustererkennung könnte man $e^{\sqrt{x}} = \sum_{k=0}^{\infty} \frac{1}{k!} x^{\frac{k}{2}}$ erwarten aber der Algorithmus hier ist ein völlig Anderer und führt auch zu völlig verschiedenen Zwischenergebnissen. Als Vergleich liefert hier Sage:

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

```
latex(maxima(exp(sqrt(x))).powerseries('x',infinity))
```

$$\sum_{i_6=0}^{\infty} \frac{1}{i_6! \left(\frac{1}{x}\right)^{\frac{i_6}{2}}}$$

Wir können uns in Mathematica mit dem Befehl:

```
specfunprint
```

dem Algorithmus herantasten und schauen uns die letzten Beispiele noch einmal an.

FPS $[e^x, x]$

```
SpecialFunctions, (C) Wolfram Koepf, version 2.04, 2012
specfun-info: DE:
   -f[x]+f'[x] == 0
specfun-info: RE for all k >= 0:
a[1+k] = a[k]/(1 + k)
specfun-info: function of hypergeometric type
specfun-info: for all k <= -1: a[k]=0
specfun-info: a[0] = 1
```

$$\texttt{sum}\left[\frac{x^k}{k!}, \{k, 0, \infty\}\right]$$

Man kann aus diesen Ausgaben erkennen, dass im Hintergrund zur Konversion von $f(x)$

- zuerst eine Differentialgleichung für $f(x)$ hergeleitet wird,
- dann diese Differentialgleichung in eine Rekursionsgleichung für die Koeffizienten $a_k$umgewandelt wird
- und letzlich diese Rekursion gelöst wird.

Das andere Beispiel liefert ähnlich:

FPS $\left[e^{\sqrt{x}}, x\right]$

```
SpecialFunctions, (C) Wolfram Koepf, version 2.04, 2012
```

```
specfun-info: DE:
    -f[x]+2 f'[x]+4 x f"[x] == 0
specfun-info: RE for all k >= 1/2:
     a[1+k] = a[k]/(2*(1 + k)*(1 + 2*k))
specfun-info: RE modified to (k -> k/2)
specfun-info: RE for all k >= 0:
     a[2+k] = a[k]/((1 + k)*(2 + k))
specfun-info: function of hypergeometric type
specfun-info: a[0] = 1
specfun-info: a[1] = 1
specfun-info: PS divided into 2 parts
              (non-symmetric 2-fold function)
```

$$\mathtt{sum}\left[\frac{x^k}{(2k)!},\{k,0,\infty\}\right]+\mathtt{sum}\left[\frac{x^{\frac{1}{2}+k}}{(1+2k)!},\{k,0,\infty\}\right]$$

In dem letzten Beispiel sieht man, dass für $e^{\sqrt{x}}$ keine holonome Differentialgleichung erster Ordnung bestimmt werden konnte und die Prozedur eine gleichwertige Reihe, welche aber aus zwei Summanden, nähmlich einer regulären Potenzreihe und einer (reinen) Puiseuxreihe, besteht, generiert wurde. Diese Summanden stellen den geraden bzw. den ungeraden Anteil von $e^{\sqrt{x}}$ dar und bekanntlich kann man jede Funktion $f(x)$ in einen geraden Anteil $g(x) := \frac{1}{2}(f(x)+f(-x))$ sowie einen ungeraden Anteil $u(x) := \frac{1}{2}(f(x)-f(-x))$ aufspalten. Dieses Vefahren werden wir nun ausführlicher betrachten. Mi dem Befehl:

```
specfunprintoff
```

können wir wieder den Verbosemodus des Pakets ausschalten.

6 Holonome Differentialgleichungen

In diesem Abschnitt befassen wir uns mit dem ersten Teil des Algorithmus und fokussieren uns auf die Suche nach einer gewöhnlichen Differentialgleichung, die von einer gegebenen Funktion $f(x)$ gelöst wird. Genauer gehts es hier um die Bestimmung einer homogenen linearen Differentialgleichung, welche Polynomkoeffizienten in $\mathbb{K}[x]$ hat, wobei $\mathbb{K}[x]$ ein Körper ist.

Meistens ist bei uns $\mathbb{K} = \mathbb{Q}$ und wenn das der Fall ist, nennen wir sowohl eine derartige Differentialgleichung als auch eine Funktion, die diese Gleichung erfüllt, holonom.

Die kleinste Ordnung einer für eine holonome Funktion $f(x)$ gültigen holonomen Diffe-

rentialgleichung nennen wir den holonomen Grad von $f(x)$ und bezeichnen diesen mit $holgrad(f(x), x)$.

6.1 Einige Beispiele für holonome Funktionen:

$f(x) := e^x$ ist holonom, da sie der holonomen Differentialgleichung $f'(x) - f(x) = 0$ genügt. Sie hat den holonomen Grad 1.

$f(x) := \sin x$ ist holonom, weil sie die holonome Differentialgleichung $f''(x) + f(x) = 0$ erfüllt. Sie hat aber $holgrad(\sin x, x) = 2$, weil sie wegen $\frac{f'(x)}{f(x)} = \cot x \notin \mathbb{Q}(x)$ keine holonome Differentialgleichung erster Ordnung besitzt. Beachten Sie, dass rationale Funkionen - im Gegensatz zu $\cot x$ - endlich viele Nullstellen in $\mathbb{C}$ besitzen.

Als nächstes Beispiel suchen wir in Sage eine holonome Differentialgleichung für die Exponentialfunktion e^{x^2}:

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

```
f0=exp(x**2)
view(f0)
```

$$e^{(x^2)}$$

Wir berechnen die Ableitung:

```
f1=f0.derivative()
view(f1)
```

$$2\,xe^{(x^2)}$$

Eine holonome Differentialgleichung erster Ordnung liegt offenbar genau dann vor, wenn der Quotient $\frac{f_1}{f_0} \in \mathbb{Q}(x)$ ist. Dies lässt sich leicht bestätigen:

```
f1/f0
    2*x
```

```
f1/f0 in QQ[x]
    True
```

```
QQ[x]
    Univariate Polynomial Ring in x over Rational Field
```

Somit ist die gesuchte Differentialgleichung $F'(x) - 2xF(x) = 0$, die von e^{x^2} gelöst wird

und es gilt $holgrad\left(e^{x^2}, x\right) = 1$.

Nun betrachten wir das etwas kompliziertere Beispiel $\arcsin x$:

```
f0=arcsin(x)
view(f0)
```
$$\arcsin(x)$$

```
f1=f0.derivative()
view(f1)
```
$$\frac{1}{\sqrt{-x^2+1}}$$

Hier kann man bereits erkennen, dass keine holonome Differentialgleichung erster Ordnung vorliegen kann, da $\frac{f_1}{f_0} \notin \mathbb{Q}(x)$ bwz. $\arcsin x$ und $\frac{1}{\sqrt{1-x^2}}$ linear unabhängig über $\mathbb{Q}(x)$ sind. Also gilt $holgrad(\arcsin x, x) \neq 1$. Wir setzen unsere Suche für eine holonome Differentialgleichung zweiter Ordnung fort:

```
f2=f1.derivative()
view(f2)
```
$$\frac{x}{\left(-x^2+1\right)^{\frac{3}{2}}}$$

Die Gleichung, womit wir zu tun haben, sieht folgendermaßen aus:

$$A_2 f^{(2)} + A_1 f^{(1)} + A_0 f^{(0)} = 0, \quad A_k \in \mathbb{Q}(x), \quad k \in \{0, 1, 2\}$$

Die A_k's wollen wir natürlich noch bestimmen. Aber wir wissen ja bereits, dass keine holonome Differentialgleichung erster Ordnung existiert. Daher können wir schon $A_2 = 1$ setzen und machen den folgenden Ansatz:

$$\frac{x}{\left(1-x^2\right)^{\frac{3}{2}}} + \mathrm{ArcSin}[x]A_0 + \frac{A_1}{\sqrt{1-x^2}}$$

Wir suchen nun linear abhängige Terme über $\mathbb{Q}(x)$, welche wir gegebenfalls zusammenfassen können. Einzige Kandidaten hierfür sind die Summanden $\frac{x}{\left(1-x^2\right)^{\frac{3}{2}}}$ und $\frac{A_1}{\sqrt{1-x^2}}$.

Ein Test ergibt:

```
var('A1')
```

```
view((f2/(A1*f1)))
```
$$\frac{x}{-\left(x^2-1\right)A_1}$$

Also sind die Summanden tatsächlich linear abhängig über $\mathbb{Q}(x)$ und wir können sie

zusammenfassen. Damit der Ansatz aber Null sein kann, müssen die Koeffizienten der über $\mathbb{Q}(x)$ linear unabhängigen Summanden verschwinden. Dazu lösen wir die Gleichungen:

$\frac{x}{(1-x^2)^{3/2}} + \frac{A_1}{\sqrt{1-x^2}} = 0$ und $\arcsin(x)\, A_0 = 0$.

```
var('A0')
```

```
view(solve(A0*f0==0,A0))
```

$[A_0 = 0]$

```
view(solve(f2+A1*f1==0,A1))
```

$\left[A_1 = \frac{x}{x^2-1}\right]$

Somit können wir die linke Seite der Gleichung bestimmen durch:

$\frac{x}{x^2-1} f'(x) + 0 + f''(x)$

Um die holonome Differentialgleichung zu erhalten, müssen wir diese mit dem Hauptnenner multiplizieren:

$x\, f'(x) + \left(x^2 - 1\right) f''(x) = 0$

Also hat $\arcsin x$ einen holonomen Grad von 2.

Mit Hilfe des Pakets `SpecialFunctions` in `Mathematica` berechnen wir ein komplizierteres Beispiel:

$$\texttt{HolonomicDE}\left[\left(\frac{\texttt{ArcSin}[\sqrt{x}]}{\sqrt{x}}\right)^2, F[x]\right]$$

$$2F[x] + 2(-3+7x)F'[x] + 3x(-3+4x)F''[x] + 2(-1+x)x^2F^{(3)}[x] == 0$$

6.2 Holonome Funktionen bilden einen Ring

Behauptung

Summe und Produkt holonomer Funktionen sind ebenfalls holonom. Wir erhalten also den Ring der holonomen Funktionen. Sind $f(x)$ und $g(x)$ holonome Funktionen vom Grad m bzw. n, so ist $f(x)+g(x)$ vom Grad $\leq m+n$, und $f(x)\cdot g(x)$ ist vom Grad $\leq m\cdot n$.

Beweis

Gelte für die Funtionen $f(x)$ und $g(x)$ $holgrad(f(x), x) = m$ bzw. $holgrad(g(x), x) = n$. Sei $V(f)$ der Vektorraum über dem Körper der rationalen Funktionen $\mathbb{K}(x)$, der von den Ableitungen von $f(x)$ erzeugt wird. Also $V(f) =< f^{(0)}(x), f^{(1)}(x), f^{(2)}(x), ... >$. Als Basis für diesen Vektorraum können wir dann $\{f^{(0)}, f^{(1)}, ..., f^{(m-1)}\}$ angeben, weil $f(x)$ eine holonome Differentialgleichung der Ordnung m, aber keine der Ordnung $m - 1$ erfüllt und weil durch sukzessives Ableiten auch alle höheren Ableitungen als Linearkombination über $\mathbb{K}(x)$ der Funktionen $f^{(0)}, f^{(1)}, ..., f^{(m-1)}$ dargestellt werden können. Damit gilt $Dim(V(f)) = m$. Analoge Konstruktion liefert für $g(x)$ den Vektorraum $V(g)$ über $\mathbb{K}(x)$ mit $Dim(V(g)) = n$. Die Summe dieser Vektorräume $V(f) + V(g)$ ist wieder ein Vektorraum und hat die Dimension $\leq m + n$.

Da $h^{(0)} := (f + g)^{(0)}, h^{(1)} = (f + g)^{(1)}, ..., h^{(k)} = (f + g)^{(k)} \in V(f) + V(g)$ gilt, sind jeweils $m + n + 1$ dieser Funktionen über $\mathbb{K}(x)$ linear abhängig. Das heisst aber, es existiert eine Differentialgleichung der Ordnung $\leq m + n$ mit Koeffizienten aus $\mathbb{K}(x)$ für h. Die Multiplikation mit dem Hauptnenner erbringt dann die gesuchte holonome Differentialgleichung von h.

Aus diesem algebraischen Beweis ist jedoch nicht direkt ersichtlich wie man die gesuchte Differentialgleichung konstruieren kann. Dazu stellt man zuerst die vorhandenen holonomen Differentialgleichungen in expliziter Form dar:

$$f^{(m)} = \sum_{j=0}^{m-1} p_j f^{(j)} \quad \text{und} \quad g^{(n)} = \sum_{k=0}^{n-1} q_k g^{(k)} \quad \text{mit} \quad p_j, q_k \in \mathbb{K}(x)$$

Durch sukzessives Differenzieren und rekursives Einsetzen dieser explizieten Darstellungen für $f^{(m)}$ und $g^{(n)}$ erhalten wir wieder (mit verschiedenen Koeffizientenfunktionen) die höheren Ableitungen derselben Form:

$$f^{(l)} = \sum_{j=0}^{m-1} p_j^l f^{(j)} \,, (l \geq m) \quad \text{und} \quad g^{(l)} = \sum_{k=0}^{n-1} q_k^l g^{(k)} \,, (l \geq n) \quad \text{mit} \quad p_j^l, q_k^l \in \mathbb{K}(x)$$

Aus der Linearität der Ableitungen folgen die Gleichungen

$$h^{(0)} = f^{(0)} + g^{(0)}$$
$$h^{(1)} = f^{(1)} + g^{(1)}$$
$$h^{(2)} = f^{(2)} + g^{(2)}$$
$$.$$
$$.$$

.
$h^{(m+n)} = f^{(m+n)} + g^{(m+n)}$

Wir setzen nun $J := max\{m, n\}$ und suchen zunächst nach einer holonomen Differentialgleichung der Ordnung J. M.a.W betrachten wir die ersten $max\{m, n\}$ Gleichungen und eliminieren die höheren Ableitungen von f bzw. g bis wir auf der rechten Seite die Gestallt von $f^{(l)} + g^{(l)}$ erreichen und somit $m + n$ Variablen übrig bleiben. Anschließend lösen wir dieses lineare Gleichungssytem nach $h^{(l)}$ $(l = 0, ..., max\{m, n\})$ auf und versuchen dabei, die Variablen $f^{(l)}$ $(l = 0, ..., m - 1)$ und $g^{(l)}$ $(l = 0, ..., n - 1)$ zu eliminieren. Wenn wir erfolgreich sind, haben wir die gesuchte holomone Differentialgleichung erhalten. Ansonsten erhöhen wir J um 1 und setzen die Suche fort. Spätestens bei $J = m+n$ Versuchen liefert unsere Suche die gewüschte holonome Differentialgleichung.

Bei der Konstruktion der Produkt-Differentialgleichung geht man analog vor. Nur werden diesmal die Produkte $f \cdot g$ mit der Leibnizschen Produktregel abgeleitet. Die zu eliminierende Variablen sind dann die $m \cdot n$ Produkte $f^{(j)}g^{(k)}$ $(j = 0, ..., m - 1, \quad k = 0, ..., n - 1)$. Dieser Prozess resultiert dann wieder in einer holonomer Diffrentialgleichung derOrdnung $\leq m \cdot n$.

Dieser Algorithmus terminiert, da wir bei einem homogenen linearen Gleichungssystem mit $m \cdot n + 1$Gleichungen ankommen, bei dem $m \cdot n$ Variablen eliminiert werden müssen.

□

6.3 Der Quotient von holonomen Funktionen

Nachdem wir die Sunme und das Produkt von holonomen Funktionen betrachtet haben, wollen wir die anfangs angebene Tangensfunktion nun genauer unter die Lupe nehmen.

$\tan x = \sum_{k=1}^{\inf} (-1)^{k+1} \frac{2^{2k}(2^{2k}-1)}{(2k)!} B_{2k} x^{2k-1}$

Der Grund, warum $\tan x$ algorithmisch ausser Reichweite liegt ist darin begründet, dass sie nicht holonom ist. Wir wollen dies nun beweisen.

Behauptung

Die Funktion $\tan x$ ist nicht holonom.

Beweis

Sei $f(x) := \tan x = \frac{\sin x}{\cos x}$. Nun gilt:

$$f(x)' = \left(\frac{\sin x}{\cos x}\right)' = \frac{\sin' x \cdot \cos x - \sin x \cdot \cos' x}{\cos^2 x} = \frac{\cos^2 x + \sin^2 x}{\cos^2 x} = 1 + f(x)^2$$

Also erfüllt $\tan x$ die nicht lineare Differentialgleichung $f' = 1 + f^2$ zweiter Ordnung. Ableiten mit dem Kettenregel liefert

$$f'' = \left(1 + f^2\right)' = 2ff' = 2f\left(1 + f^2\right)$$

Durch wiederholtes Ableiten erhalten wir somit für alle $k \in \mathbb{N}$ Darstellungen der Form $f^{(k)} = P_k(f)$ für Polynome $P_k y \in \mathbb{Z}[y]$. Angenommen, f wäre Lösung der holonomen Differentialgleichung

$$\sum_{k=0}^{n} p_k(x) f^{(k)} = 0 \quad , p_k(x) \in \mathbb{Q}[x]$$

so können wir unsere oben gewonnenen Formeln für $f^{(k)}$ hier einsetzen und erhalten eine algebraische Gleichung

$$G(x, f) = \sum_{j=0}^{J} \sum_{k=0}^{K} c_{jk} x^j f^k = 0 \quad , G(x, y) \in \mathbb{Q}[x, y]$$

für die Tangensfunktion. Somt wäre $\tan x$ eine algebraische Funktion. Eine algebraische Funktion kann aber nicht unendlich viele Nullstellen besitzen, denn löst man die obige Gleichung nach f auf, so erhällt man höchstens K verschiedene Lösungszweige, weswegen f dann aber auch höchstens K Nullstellen haben kann. Wir wissen aber, dass $\tan x$ in $\mathbb{R}$ bzw. in $\mathbb{C}$ unendlich viele Nullstellen besitzt. Dies führt uns also zu einem Widerspruch, weswegen es für $\tan x$ keine holonome Differentialgleichung geben kann.

□

Die holonome Differentialgleichung $F''(x) + F(x)$ wird sowohl von $\sin x$ als auch von $\cos x$ gelöst. Der Quotient von diesen holonomen Funktionen, nähmlich $\tan x$, ist nicht holonom und zeigt uns als Beispiel, dass der Quotient holonomer Funktionen nicht wieder holonom zu sein braucht.

Bemerkung

Im Ring der holonomen Funktionen bildet die holonome Differentialgleichung einer holonomen Funktion vom Grad n zusammen mit n geeigneten Anfangswerten eine Nor-

malform. Diese Tatsache wird in der Theorie der gewöhnlichen Differentialgleichungen durch den Existenzsatz von Peano und den Eindeutigkeitssatz von Picard-Lindelöf in ihrer Richtigkeit bestätigt. Das Identifikationsproblem für holonome Funktionen liesse sich somit durch Bestimmung der zugehörigen holonomen Differentialgleichung und geeigneter Anfangswerte lösen.

Verwendet man Algorithmen, die die holonomen Differentialgleichungen immer in niedrigster Ordnung berechnen können, kann man sie sogar in kanonischer Form darstellen. Dies trifft aber auf die bisher von uns betrachteten Algorithmen nicht immer zu.

Wenn wir zwei holonome Funktionen miteinander identifizieren wollen, die durch zwei holonome Differentialgleichungen $D1$, $D2$ verschiedener Ordnung beschrieben werden, müssen wir zeigen, dass die Gültigkeit der holonomen Differentialgleichung niedrigerer Ordnung auch die Gültigkeit der mit der höheren Ordnung geltend macht. Also löse man zuerst die holonome Differentialgleichung mit der niedrigeren Ordnung nach der höchsten Ableitung $f^{(n)}$ auf und setze diese und sowie alle sich daraus ergebenden höheren Ableitungen in die andere holonome Differentialgleichung mit der höheren Ordnung ein. Wenn die rationale Vereinfachung in $0 = 0$ resultiert, sind die beiden holonomen Differentialgleichungen $D1$, $D2$ miteinander kompatibel, andernfallst nicht.

7 Holonome Rekursionsgleichungen

Nun befassen wir uns mit dem zweiten Schritt des Algorithmus zur Umwandlung eines Ausdrucks in die zugehörige Potenzreihe. Hierbei geht es darum aus der holonomen Differentialgleichung für f , die zugehörigen Koeffizienten a_k zur Darstellung der formalen Potenzreihe zu gewinnen. Wir werden dazu die holonome Differentialgleichung in eine Rekursionsgleichung umwandeln. Es wird sich zeigen, dass diese Rekursionsgleichung genau für holonome Differentialgleichungen ebenfalls holonom ist.

Eine Rekursionsgleichung für a_k nennen wir holonom, wenn sie homogen und linear ist und ihre Polynomkoeffizienten aus $\mathbb{K}[x]$ stammen.

Eine Folge, die eine holonome Rekursionsgleichung löst, heißt ebenfalls holonom.

Die Differenz zwischen grösstem und kleinstem j, der in der holonomen Rekursionsgleichung auftretenden Terme a_{k+j}, bezeichnen wir als die Ordnung der holonomen Rekursionsgleichung.

Die kleinste Ordnung einer für eine holonome Folge a_k gültigen holonomen Rekursionsgleichung nennen wir den holonomen Grad von a_k und bezeichnen ihn mit $holgrad(a_k, k)$.

7.1 Einige Beispiele für holonome Folgen:

Die Fakultät $a_k = k!$ ist wegen $a_{k+1} - (k+1)a_k = 0$ holonom vom Grad $holgrad(k!, k) = 1$, denn es gilt: $(k+1)! - (k+1)k = (k+1)k! - (k+1)k = 0$ und $(k+1) - (k) = 1$ für ihren Grad.

Die Folge $F(n,k) = \binom{n}{k}$ ist holonom bzgl. n und k, weil sie die Rekursionsgleichungen

$\frac{F(n,k+1)}{F(n,k)} = \frac{\binom{n}{k+1}}{\binom{n}{k}} = \frac{\frac{n!}{(k+1)!\cdot(n-k-1)!}}{\frac{n!}{k!\cdot(n-k)!}} = \frac{k!\cdot(n-k)!}{(k+1)!\cdot(n-k-1)!} = \frac{(n-k)!}{(k+1)\cdot(n-k-1)!} = \frac{n-k}{k+1}$ sowie

$\frac{F(n+1,k)}{F(n,k)} = \frac{\binom{n+1}{k}}{\binom{n}{k}} = \frac{\frac{(n+1)!}{k!\cdot(n+1-k)!}}{\frac{n!}{k!\cdot(n-k)!}} = \frac{(n-k)!\cdot(n+1)!}{n!\cdot(n+1-k)!} = \frac{(n-k)!\cdot(n+1)}{(n+1-k)!} = \frac{(n-k)!\cdot(n+1)}{(n+1-k)\cdot(n-k)!} = \frac{n+1}{n+1-k}$ bzw.

$(k+1)F(n,k+1) + (k-n)F(n,k) = 0$ und $(n+1-k)F(n+1,k) - (n+1)F(n,k) = 0$

löst. Sie hat die holonomen Grade $holgrad\left(\binom{n}{k}, k\right) = holgrad\left(\binom{n}{k}, n\right) = 1$

7.2 Holonome Folgen bilden einen Ring

Behauptung

Summe und Produkt holonomer Folgen sind wieder holonom. Wir erhalten also den Ring der holonomen Folgen. Sind a_k und b_k holonome Folgen vom Grad m, bzw. n, so ist $a_k + b_k$ vom Grad $\leq m + n$, und $a_k \cdot b_k$ ist vom Grad $\leq m \cdot n$.

Beweis

Der Beweis verläuft analog zu dem Beweis für holonome Funktionen. Allerdings ist hier der Differentialoperator $D_x : f(x) \to f'(x)$ mit der (Vorwärtsoperator) shiftoperator $S_k : a_k \to a_{k+1}$ zu ersetzen.

□

Bemerkung

Auch im Ring der holonomen Folgen bildet die holonome Rekursionsgleichung einer holonomen Folge vom Grad n zusammen mit n geigneten Anfangswerten eine Normalform, womit es sich das Identifikationsproblem für holonome Folgen lösen lässt.

Folglich können wir unter diesen Voraussetzungen diskrete tranzendente Identitäten beweisen.

Beispiel

Mit $(a)_k := a \cdot (a+1) \cdot \cdots \cdot (a+k-1)$ wird das Pochhammer-Symbol bezeichnet. Wir wollen nun die Identität $\left(\frac{1}{2}\right)_k = \frac{(2k)!}{4^k k!}$ zunächst direkt beweisen. Es gilt nach Definition

$$\left(\tfrac{1}{2}\right)_k = \prod_{n=0}^{k-1}\left(\tfrac{1}{2}+n\right) = \prod_{n=0}^{k-1}\tfrac{2n+1}{2} = \prod_{n=1}^{k}\tfrac{2n-1}{2} = 2^{-k}\prod_{n=1}^{k} 2n-1 = \frac{\prod_{n=1}^{k} 2n-1}{2^k}$$

Nun steht auf der rechten Seite über dem Nenner das Produkt der ersten ungeraden Zahlen $1 \cdot 3 \cdot 5 \cdots 2k-1$. Wir können dieses Produkt mit den fehlenden geraden Zahlen zu $(2k)!$ vervollständigen. Allerdings müssen wir die Gleichung auch durch diese geraden Zahlen $\left(2^k k\right)!$ teilen. Somit erhalten wir $\frac{(2k)!}{4^k k!}$, was zu beweisen war.

□

Mit den neu gewonnenen Erkenntnissen wollen wir nun die Tatsache erneut ins Tageslicht bringen. Der Befehl `HolonomicRE` aus dem Paket `SpecialFunctions` in `Mathematica` liefert für beide Ausdrücke die gleiche holonome Rekursionsgleichung.

```
HolonomicRE[Pochhammer[1/2, k], a[k]]
    (-1 - 2k)a[k] + 2a[1 + k] == 0
```

```
HolonomicRE[(2k)!/(4^k k!), a[k]]
    (-1 - 2k)a[k] + 2a[1 + k] == 0
```

Wir können die Berechnungen per Hand überprüfen. Der Grad der holonomen Rekursionsgleichung ist $k+1-k=1$.

$\left(\frac{1}{2}\right)_0 = \prod_{n=0}^{0-1}\left(\frac{1}{2}+n\right) = 1$ (Anfangswert $k=0$)

$(-1-2k)\left(\frac{1}{2}\right)_k + 2\left(\frac{1}{2}\right)_{k+1} = (-1-2k)\left(\frac{1}{2}\right)_k + 2\prod_{n=0}^{k}\left(\frac{1}{2}+n\right) = -(1+2k)\cdot\left(\frac{1}{2}\right)_k + 2\cdot\left(\frac{1}{2}+k\right)\cdot \prod_{n=0}^{k-1}\left(\frac{1}{2}+n\right) = -(1+2k)\cdot\left(\frac{1}{2}\right)_k + (1+2k)\cdot\left(\frac{1}{2}\right)_k = 0$

$\frac{(2\cdot 0)!}{4^0\cdot 0!} = 1$ (Anfangswert $k = 0$)

$(-1-2k)\frac{(2k)!}{4^k k!} + 2\cdot\frac{(2(k+1))!}{4^{k+1}(k+1)!} = (-1-2k)\frac{(2k)!}{4^k k!} + 2\frac{(2k+2)!}{4^{k+1}(k+1)!} = -(1+2k)\frac{(2k)!}{4^k k!} + \frac{(2k+2)(2k+1)(2k)!}{4^k\cdot 2\cdot(k+1)k!} = -(1+2k)\frac{(2k)!}{4^k k!} + (1+2k)\frac{(2k)!}{4^k k!} = 0$

Damit sind die beiden Folgen identisch.

7.3 Bestimmung der holonomen Rekursionsgleichung

Kommen wir nun zurück auf unsere ursprüngliche Frage und beschäftigen uns damit, wie wir aus einer holonomen Differentialgleichung eine holonome Rekursionsgleichung erhalten, damit wir die zugehörigen Reihenkoeffizienten für eine formale Potenzreihe bestimmen können.

Behauptung

Sei $f(x)$ eine Funktion, welche die holonome Differentialgleichung DE löst. Wird nun die Reihenentwicklung $f(x) = \sum_{n=0}^{\infty} a_n x^n$ in DE eingesetz, entsteht eine holonome Rekursionsgleichung für a_n.

Ist DE in expandierter Form, so erhält man eine holonome Rekursionsgleichung für a_n, wenn man die formale Substitution $x^j f^{(k)} \mapsto (n+1-j)_k \cdot a_{n+k-j}$ in DE durchführt.

Beweis

Angenommen, $f(x)$ habe die Darstellung

$f(x) = \sum_{n=0}^{\infty} a_n x^n = \sum_{n=-\infty}^{\infty} a_n x^n$

Wir setzen technisch $a_n = 0$ für $n < 0$, damit wir bei den Indexverschiebungen keine Fallunterscheidungen beachten müssen. Wir nehmen auch an, dass DE in expandierter Form vorliegt.

$\sum_{j=0}^{J}\sum_{k=0}^{K} c_{jk} x^j f^{(k)}(x) = 0 \quad \left(c_{jk} \in \mathbb{K}, \quad J, K \in \mathbb{N}_{\geq 0}\right)$

Mit Induktion sieht man, dass

$f^{(k)}(x) = \sum_{n=-\infty}^{\infty}(n+1-k)_k a_n x^{n-k}$

gilt. Also können wir sie in *DE* einsetzen und erhalten

$0 = \sum_{j=0}^{J}\sum_{k=0}^{K} c_{jk}x^j = \sum_{n=-\infty}^{\infty}(n+1-k)_k a_n x^{n-k} = \sum_{n=-\infty}^{\infty}\sum_{j=0}^{J}\sum_{k=0}^{K} c_{jk}(n+1-k)_k a_n x^{n-k+j} = \sum_{n=-\infty}^{\infty}\sum_{j=0}^{J}\sum_{k=0}^{K} c_{jk}(n+1-k)_k a_{n+k-j} x^n$

wobei wir im letzen Schritt eine Indexverschiebung bzgl. n vorgenommen haben. Koeffizientenvergleich liefert die Behauptung.

Da $(n+1-j)_k$ für festes $k \in \mathbb{N}_{\geq 0}$ ein Polynom in n (vom Grad k) ist, folgt, dass die resultierende Rekursionsgleichung holonom ist und umgekehrt.

□

Der folgende Code in `Sage` ermittelt für eine holonome Differentialgleichung die entsprechende holonome Rekursionsgleichung. Dabei muss *DE* in expandierter Form vorliegen. Leider ist die Implementierung der formalen Substitution von

$x^j f^{(k)} \mapsto (n+1-j)_k \cdot a_{n+k-j}$

in `Sage` nicht so einfach wie bei `Mathematica`. Hierzu muss der Expression-Baum mühsam traversiert und geparst werden, welches in `Mathematica` oft mit einem Einzeiler zu erledigen ist.

Die Funktion `de_to_re` ruft sich rekursive auf, um die Terme einer holonomen Differentialgleichung zu traversieren und die formale Substitution in der Variablen `terms` zu sammeln. Dabei macht sie davon Gebrauch, dass die Operatoren `mul`, `pow` ihre Operanden sortiert zur Verfügung stellen. Dieses Effekt ist jedoch bei der Ausgabe von *RE* nicht erwünscht, wenn man die Terme von *DE* und *RE* untereinander vergleichen möchte. In `Sage` lässt sich aber darauf kein Einfluss nehmen.

```
def de_to_re(expr,a,i=0,verbose=false):
    """
        Function: de_to_re
        ------------------------------------------------
        converts a given holonomic differential equation
        to a corresponding holonomic recursive equation

        Input:
            expr: holonomic diffenertial in expanded form
            a: function('a',nargs=1)
            i: for output indentation in verbose mode
            verbose: verbose mode (default is false)
        Output:
            re: holonomic recursive equation for a(n)
    """
    from sage.symbolic.operators import FDerivativeOperator

    # variables
    indent = i*' ---'
    n=var('n') # symbolic variable for input a
    global terms # for accumulation of computed terms
    global m # factor c_jk in DE
    global j # power of the symbolic variable in DE
    global k # order of differentation
    global re # recursive equation from computed terms

    op = expr.operator()
    if op is operator.eq:
        if verbose:
            print indent,'DE:',expr
            print
        lhs = expr.lhs()
        op = lhs.operator()
        operands = lhs.operands()
    else:
        if verbose:
            print indent,'expr:',expr
        operands = expr.operands()
    if op:
        if verbose:
            print indent,'operator:',op
            print indent,'operands:',operands
        if op is operator.add:
            # set variables
            terms = 0
```

```
            re = 0
            m = 1
            j = 0
            k = 0
        if op is operator.pow:
            j = operands[-1]
        if op is operator.mul:
            if operands[0].is_symbol():
                j = 1
            if operands[-1] in QQ:
                m = operands[-1]
        if type(op) is FDerivativeOperator:
            k = len(op.parameter_set())
        i += 1
        for operand in operands:
            de_to_re(operand,a,i,verbose)
            if op is operator.add:
                term = m * rising_factorial(n+1-j,k)*a(n+k-j)
                terms = terms + term
                if verbose:
                    print indent,'m =',m
                    print indent,'j =',j
                    print indent,'k =',k
                    print indent,'term =',term
                    print indent,'terms =',terms
                # reset variables
                m = 1
                j = 0
                k = 0
    if op is operator.add:
        re = terms == 0
        if verbose:
            print
            print indent,'Returning RE:',re
            print
        return re
```

Beispiel

Wir bestimmen nun die Rekursionsgleichung, die zu der Exponentialfunktion gehört:

```
x = var('x')
f = function('f', x)
a=function('a',nargs=1)
```

```
de1 = diff(f,x)-f == 0
re1 = de_to_re(de1,a,verbose=true)

   DE: -f(x) + D[0](f)(x) == 0

   operator: <built-in function add>
   operands: [-f(x), D[0](f)(x)]
   --- expr: -f(x)
   --- operator: <built-in function mul>
   --- operands: [f(x), -1]
   --- --- expr: f(x)
   --- --- operator: f
   --- --- operands: [x]
   --- --- --- expr: x
   --- --- expr: -1
   m = -1
   j = 0
   k = 0
   term = -a(n)
   terms = -a(n)
   --- expr: D[0](f)(x)
   --- operator: D[0](f)
   --- operands: [x]
   --- --- expr: x
   m = 1
   j = 0
   k = 1
   term = (n + 1)*a(n + 1)
   terms = (n + 1)*a(n + 1) - a(n)

   Returning RE: (n + 1)*a(n + 1) - a(n) == 0

re1
   (n + 1)*a(n + 1) - a(n) == 0
```

Zusätzlich sei bemerkt, $\frac{1}{k!}$ erfüllt diese holonome Rekursionsgleichung und es gilt $\frac{1}{0!} = e^0$ für den Anfangswert 0.

```
re1.substitute_function(a,1/factorial(k).function(k))
.full_simplify()
   0 == 0

exp(0) == 1/factorial(0)
   True
```

Somit kennen wir die Potenzreihenkoeffizienten und können die Potenzreihe von e^x in geschlossener Form angeben.

$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k$

Beispiel

Für arcsin x hatten wir die holonome Differentialgleichung

$x\, f'(x) + \left(x^2 - 1\right) f''(x) = 0$

erhalten. Wir geben sie in expandierter Form in die Funktion **de_to_re** ein:

```
de2 = x*diff(f,x)+x**2*diff(f,x,2)-diff(f,x,2) == 0
re2 = de_to_re(de2,a,verbose=true)

   DE: x^2*D[0, 0](f)(x) + x*D[0](f)(x) - D[0, 0](f)(x)
   == 0

   operator: <built-in function add>
   operands: [x^2*D[0, 0](f)(x), x*D[0](f)(x),
   -D[0, 0](f)(x)]
   --- expr: x^2*D[0, 0](f)(x)
   --- operator: <built-in function mul>
   --- operands: [x^2, D[0, 0](f)(x)]
   --- --- expr: x^2
   --- --- operator: <built-in function pow>
   --- --- operands: [x, 2]
   --- --- --- expr: x
   --- --- --- expr: 2
   --- --- expr: D[0, 0](f)(x)
   --- --- operator: D[0, 0](f)
   --- --- operands: [x]
   --- --- --- expr: x
   m = 1
   j = 2
   k = 2
   term = (n - 1)*n*a(n)
   terms = (n - 1)*n*a(n)
   --- expr: x*D[0](f)(x)
   --- operator: <built-in function mul>
   --- operands: [x, D[0](f)(x)]
   --- --- expr: x
   --- --- expr: D[0](f)(x)
```

```
   --- --- operator: D[0](f)
   --- --- operands: [x]
   --- --- --- expr: x
   m = 1
   j = 1
   k = 1
   term = n*a(n)
   terms = (n - 1)*n*a(n) + n*a(n)
   --- expr: -D[0, 0](f)(x)
   --- operator: <built-in function mul>
   --- operands: [D[0, 0](f)(x), -1]
   --- --- expr: D[0, 0](f)(x)
   --- --- operator: D[0, 0](f)
   --- --- operands: [x]
   --- --- --- expr: x
   --- --- expr: -1
   m = -1
   j = 0
   k = 2
   term = -(n + 2)*(n + 1)*a(n + 2)
   terms = -(n + 2)*(n + 1)*a(n + 2) + (n - 1)*n*a(n)
   + n*a(n)

   Returning RE: -(n + 2)*(n + 1)*a(n + 2) + (n - 1)*n*a(n)
   + n*a(n) == 0

re2
   -(n + 2)*(n + 1)*a(n + 2) + (n - 1)*n*a(n) + n*a(n) == 0

re2.full_simplify()
   n^2*a(n) - (n^2 + 3*n + 2)*a(n + 2) == 0
```

In Mathematica liefert der Befehl DetoRE aus dem Paket SpecialFunctions:

$$\texttt{DEtoRE}\left[\texttt{HolonomicDE}\left[\left(\frac{\texttt{ArcSin}[\sqrt{x}]}{\sqrt{x}}\right)^2, F[x]\right], F[x], a[k]\right]$$
$$2(1+k)^3 a[k] - (1+k)(2+k)(3+2k)a[1+k] == 0$$

Die resultierende holonome Rekursionsgleichungen werden später genauer analysiert und es wird gezeigt wie man explizit die Lösung finden kann.

8 Hypergeometrische Funktionen und Reihen

8.1 Definition

Eine hypergeometrische Funktion in der Abhängigkeit von der Veränderlichen x wird mit $p, q, k \in \mathbb{N}$ folgendermassen definiert:

$$ {}_pF_q = \left(\begin{matrix} \alpha_1, & \alpha_2, & \cdots & ,\alpha_p \\ \beta_1, & \beta_2, & \cdots & ,\beta_q \end{matrix} \middle| x \right) := \sum_{k=0}^{\infty} A_k = \sum_{k=0}^{\infty} \frac{(\alpha_1)_k \cdot (\alpha_2)_k \cdots \left(\alpha_p\right)_k}{(\beta_1)_k \cdot (\beta_2)_k \cdots \left(\beta_q\right)_k} \frac{x^k}{k!} $$

Wobei die sogenannten obere Parameter $\alpha_1, ..., \alpha_p$ und untere Parameter $\beta_1, ..., \beta_q$ aus dem Körper $\mathbb{K}$ sind. Mit $(\alpha_i)_k$ für $i = 1, ...p$ bzw. $(\beta_j)_k$ für $j = 1, ..., q$ ist das vorher eingeführte Pochammersymbol gemeint.

$$ (a)_k = \underbrace{a \cdot (a+1) \cdots (a+k-1)}_{k \; Faktoren} $$

Man beachte, dass ${}_pF_q$ wohldefiniert ist, falls kein unterer Parameter ganzzahlig negativ (oder Null) ist, und die Reihe konvergiert auf Grund des Quotientenkriteriums für $p \leq q$ bzw. für $p = q + 1$ und $|x| < 1$.

Sei A_k der k-te Summand:

$$ A_k = \frac{(\alpha_1)_k \cdot (\alpha_2)_k \cdots \left(\alpha_p\right)_k}{(\beta_1)_k \cdot (\beta_2)_k \cdots \left(\beta_q\right)_k} \frac{x^k}{k!} $$

Betrachten wir den Quotient:

$$ \frac{A_{k+1}}{A_k} = \frac{(\alpha_1)_{k+1} \cdot (\alpha_2)_{k+1} \cdots \left(\alpha_p\right)_{k+1}}{(\beta_1)_{k+1} \cdot (\beta_2)_{k+1} \cdots \left(\beta_q\right)_{k+1}} \frac{(\beta_1)_k \cdot (\beta_2)_k \cdots \left(\beta_q\right)_k}{(\alpha_1)_k \cdot (\alpha_2)_k \cdots \left(\alpha_p\right)_k} \frac{x^{k+1} k!}{x^k (k+1)!} $$

Für das Pochammersymbol gilt:

$$ (a)_{k+1} = (a)_k \cdot (x+k) $$

Wir kürzen damit den obigen Ausdruck und erhalten:

$$ \frac{A_{k+1}}{A_k} = \frac{(k+\alpha_1) \cdot (k+\alpha_2) \cdots (k+\alpha_p) \cdot x}{(k+\beta_1) \cdot (k+\beta_2) \cdots (k+\beta_q) \cdot (k+1)} $$

Dies kann als rationales Polynom in Abhängigkeit von k mit dem konstanten Faktor x aufgefasst werden. Man beachte, dass jede rationale Funktion $g(k) \in \mathbb{C}(k)$ in voll fakto-

risierter Form diese Gestallt aufweist. Wir erhalten somit die Rekursionsleichung erster Ordnung:

$$(k+\beta_1)\cdot(k+\beta_2)\cdots(k+\beta_q)\cdot(k+1)\cdot A_{k+1} - (k+\alpha_1)\cdot(k+\alpha_2)\cdots(k+\alpha_p)\cdot x\cdot A_k = 0$$

Eine Reihe $\sum_{k=0}^{\infty} A_k$ mit der Eigenschaft $\frac{A_{k+1}}{A_k} \in \mathbb{K}(k)$ wird hypergeometrische Reihe und ihre Terme A_k werden hypergeometrischer Term genannt. Für $\mathbb{K} = \mathbb{C}$ weist sie nach vollständiger Faktorisierung die Struktur einer hypergeometrischen Funktion auf und kann damit beschrieben werden. Seine hypergeometrischen Terme entsprechen genau den Lösungen einer holonomen Rekursionsgleichung erster Ordnung.

Beispiel

Sei

$$\frac{A_{k+1}}{A_k} = \frac{k^2+7k+10}{4k^2+2}$$

Nach Umformung erhalten wir:

$$\frac{A_{k+1}}{A_k} = \frac{(k+2)(k+5)}{4\left(k+\frac{i}{2}\right)\left(k-\frac{i}{2}\right)}$$

Der Quotient muss für unsere Betrachtungen im Nenner $(k+1)$ enthalten. Wir erweitern den Quotient zu :

$$\frac{A_{k+1}}{A_k} = \frac{(k+2)(k+5)(k+1)\frac{1}{4}}{\left(k+\frac{i}{2}\right)\left(k-\frac{i}{2}\right)(k+1)}$$

Hieraus lassen sich bereits die Parameter und das Argument für die entsprechende hypergeometrische Funktion ablesen. Allerdings müssen wir beachten, dass jeder konstante Faktor A_0, der in jedem Summanden enthalten ist, bei der Quotientenbildung gekürzt worden ist, und deswegen gesondert betrachtet werden muss. Somit sieht die Reihe, deren Summanden den obigen Quotienten ergeben, wie folgt aus:

$$\sum_{k=0}^{\infty} A_k = A_0 \cdot {}_3F_2 = \left(\begin{array}{ccc|c} 2, & 5, & 1 & \\ & & & \frac{1}{4} \\ \frac{i}{2}, & -\frac{i}{2} & & \end{array}\right)$$

Wir können alleine aus dem Quotienten keine Rückschlüsse darauf ziehen welchen Wert A_0 hatte.

Beispiel

Für die Cosinusfunktion gilt:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots = \sum_{k=0}^{\infty}(-1)^k \cdot \frac{x^{2k}}{(2k)!}$$

Nach der Quotientenbildung stellen wir fest:

$$\frac{A_{k+1}}{A_k} = \frac{(-1)^{k+1}x^{2(k+1)}}{(2(k+1))!} \cdot \frac{(2k)!}{(-1)^k x^{2k}} = \frac{-x^2}{(2k+2)(2k+1)} = \frac{-\frac{1}{4}x^2}{\left(k+\frac{1}{2}\right)(k+1)}$$

Damit können wir die hypergeometrische Funktion von $\cos x$ bestimmen:

$$\cos x = A_0 \cdot {}_0F_1 = \left(\begin{matrix} - \\ \frac{1}{2} \end{matrix} \middle| -\frac{1}{4}x^2 \right)$$

A_0 ist in diesem Fall gleich 1.

Beispiel

Für die Sinusfunktion gilt :

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{k=0}^{\infty}(-1)^k \cdot \frac{x^{2k+1}}{(2k+1)!}$$

Wir bilden wieder den Quotienten und erhalten:

$$\frac{A_{k+1}}{A_k} = \frac{(-1)^{k+1}x^{2(k+1)+1}}{(2(k+1)+1)!} \cdot \frac{(2k+1)!}{(-1)^k x^{2k+1}} = \frac{-x^2}{(2k+3)(2k+2)} = \frac{-\frac{1}{4}x^2}{\left(k+\frac{3}{2}\right)(k+1)}$$

Somit können wir die hypergeometrische Funktion von $\sin x$ angeben, wobei wir $A_0 = x$ berücksichtigen:

$$\sin x = x \cdot {}_0F_1 = \left(\begin{matrix} - \\ \frac{3}{2} \end{matrix} \middle| -\frac{1}{4}x^2 \right)$$

Beispiel

Die Exponentialfunktion ist die einfachste hypergeometrische Funktion, denn es gilt:

$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k$ und

$$\frac{A_{k+1}}{A_k} = \frac{k! x^{k+1}}{(k+1)! x^k} = \frac{1}{k+1} x$$

Somit erhalten wir wegen $A_0 = 1$:

$$e^x = {}_0F_0\left(\begin{matrix} - \\ - \end{matrix} \middle| x\right)$$

8.2 Hypergeometrische Funktionen in Sage

In Sage kann man mit hypergeometrischen Funktionen numerische Berechnungen ausführen. Dazu muss zunächst das Paket `mpmath` importiert werden. Der Befehl `hyper(a_s,b_s,z)` liefert beispielsweise für ${}_0F_0\left(\begin{matrix} - \\ - \end{matrix} \middle| x\right)$ mit $x = \pi$:

```
from mpmath import *

x=pi
hyper([],[],x)
    23.140692632779267

exp(x)
    23.140692632779267
```

Für $\cos x = {}_0F_1\left(\begin{matrix} - \\ \frac{1}{2} \end{matrix} \middle| -\frac{1}{4}x^2\right)$ mit dem vorhandenen Wert von $x = \pi$ erhalten wir:

```
hyper([],[1/2],(-1/4)*x**2)
    -1.0
```

Einige hypergeometrische Funktionen haben einen Namen. Beispielsweise die Funktion ${}_2F_1\left(\begin{matrix} a, & b \\ & c \end{matrix} \middle| x\right)$ heißt die Gausssche hypergeometrische Reihe, ${}_1F_1\left(\begin{matrix} a \\ b \end{matrix} \middle| x\right)$ heißt die

Kummersche Reihe und ${}_3F_2\left(\begin{matrix}a, & b, & c \\ d & e & \end{matrix}\middle| x\right)$ wird als Clausensche Reihe genannt. Die entsprechende Befehle in `Sage` sind `hyp2f1(a, b, c, z)`, `hyp1f1(a, b, z)` und `hyp3f2(a1, a2, a3, b1, b2, z)`.

Im Vergleich zu `Sage` kann `Mathematica` auch mit symbolischen Ausdrücken in hypergeometrischen Funktionen umgehen:

```
HypergeometricPFQ[{}, {}, x]
```
$$e^x$$

```
HypergeometricPFQ[{}, {1/2}, -x^2/4]
```
```
Cos[x]
```

8.3 Bestimmung des Reihenkoeffizienten a_k

Nun wollen wir den umgekehrten Weg gehen und für einige vorher betrachtete Beispiele den Reihenkoeffizienten a_k herleiten.

Beispiel

Für die Exponentialfunktion hatten wir eine entsprechende holonome Differentialgleichung bestimmt:

$$f'(x) - f(x) = 0$$

Und daraus die zugehörige holonome Rekursionsgleichung abgeleitet:

$$(k+1)\, a_{k+1} - a_k = 0$$

Diesen Ausdruck können wir umformen zu:

$$\frac{a_{k+1}}{a_k} = \frac{1}{k+1} \in \mathbb{Q}(x)$$

Somit geht es um eine hypergeometrische Reihe bzw. Funktion. Wir wissen, dass $A_0 = 1$ ist, und konstruieren daraus die hypergeometrische Darstellung:

$$\sum_{k=0}^{\infty} a_k\, x^k = {}_0F_0\left(\begin{matrix} - \\ - \end{matrix}\,\middle|\, x\right) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

Weil es die einfachste hypergeometrische Funktion ist, ist hier direkt aus der Definition ablesbar, dass $a_k = \frac{1}{k!}$ gilt.

Beispiel

Mit Hilfe von `Mathematica` hatten wir für $f(x) = \left(\frac{arcsin(\sqrt{x})}{\sqrt{x}}\right)^2$ die holonome Rekursionsgleichung

$$2\,(1+k)^3 a_k - (1+k)\,(2+k)\,(3+2k)\,a_{k+1} = 0$$

erhalten. In diesem Fall haben wir $a_0 = lim_{x\mapsto 0} f(x) = 1$. Durch Umformen und Anpassen sieht man analog:

$$a_k = \frac{(1)_k\,(1)_k\,(1)_k}{(2)_k\,\left(\frac{3}{2}\right)_k\,k!} = \frac{k!}{(1+k)\,\left(\frac{3}{2}\right)_k} = \frac{4^k\,(k!)^2}{(1+k)\,(1+2k)!}$$

Es ergibt sich insgesamt:

$$\left(\frac{arcsin(\sqrt{x})}{\sqrt{x}}\right)^2 = \sum_{k=0}^{\infty} \frac{4^k\,(k!)^2}{(1+k)\,(1+2k)!} x^k = {}_3F_2\left(\begin{matrix} 1 & 1 & 1 \\ 2 & \frac{3}{2} \end{matrix}\,\middle|\, x\right)$$

Dieses Ergebnis liefert auch der Befehl `FPS`:

```
FPS[(ArcSin[Sqrt[x]]/Sqrt[x])^2, x]

    sum[4^k x^k (k!)^2/((1+k)(1+2k)!), {k, 0, ∞}]
```

Das nächste Beispiel ist komplizierter und man benötigt den folgenden Satz.

8.4 Satz: Berechnung hypergeometrischer Potenzreihendarstellungen

Behauptung

Sei $f(x) = \sum_{k=0}^{\infty} a_k x^k$ vom hypergeometrischen Typ mit Symmetriezahl m. Dann existiert eine Potenzreihendarstellung für $f(x)$ in Form von:

$f(x) = \sum_{j=1}^{m} f_j(x) = \sum_{j=1}^{m} \sum_{k=0}^{\infty} a_{jk} x^{mk+j}$

deren m Teilreihen $f_j(x)$ als m-fach symmetrische Reihen genannt werden.

Beweis

Jede Funktion $f(x)$ vom hypergeometrischem Typ ist holonom, weil bereits die allgemeine hypergeometrische Funktion eine holonome Differentialgleichung löst und $f(x)$ als Summe holonomer Funktionen $f_j(x)$, die Komposition von rationalen Funktionen sind, holonom ist. Deswegen kann man eine holonome Differentialgleichung für $f(x)$ bestimmen. Diese kann wiederum in eine holonome Rekursionsgleichung für a_k umgewandelt werden. Ist diese Rekurssionsgleichung vom hypergeometrischen Typ mit Symmetrieanzahl m, kann man die Reihe in m-fach symmetrische Reihen $f_j(x)$ zerlegen. Für deren Koeffizienten a_{jm} ergibt sich dann jeweils eine holonome Rekursionsgleichung der Ordnung 1. Anschliessend kann man diese m Rekursionen mit m Anfangswerten gemäss der hypergeometrischen Koeffizientenformel bestimmen. In seltenen Fällen ist die Konversion von der holonomen Differentialgleichung zur holonomen Rekursionsgleichung nicht erfolgreich. D.h.: Man erhält eine Rekursionsgleichung, die nicht vom hypergeometrischen Typ ist, obwohl $f(x)$ diese Eigenschaft besitzt. Die hypergeometrischen Term-Lösungen lassen sich in solchen Fällen mit dem Petkovšek-Algorithmus bestimmen.

□

Beispiel

Betrachten wir nun den folgenden Fall für $f(x) = \arcsin x$ und ihre resultierende holonome Rekursionsgleichung zweiter Ordnung:

$(n+2)\,(n+1)\,a_{n+2} - n^2 a_n = 0$

Der Quotient sieht nun folgendermaßen aus:

$\frac{a_{k+m}}{a_k} = r(k) \in \mathbb{K}(k)$

Wenn in solchen Fällen die Rekursionsgleichung m-ter Ordnung ist, wird m die Symmetriezahl der zugehörigen hypergeometrischen Funktion genannt. In unserem Fall ist

$f(x) = \sum_{k=0}^{\infty} a_n x^n$

vom hypergeometrischen Typ und hat die Symmetriezahl 2. Da $\arcsin 0 = 0$ ist, verschwindet der gerade Anteil und wir können

$h(x) = \sum_{k=0}^{\infty} c_k x^k$

so setzen, dass $f(x) = xh(x^2)$ bzw. $c_k = a_{2k+1}$ gilt. Anschliessend substituieren wir $n = 2k + 1$ in die holonome Rekursionsgleichung und erhalten die folgende hypergeometrische Rekursionsgleichung der Ordnung 1:

$c_{k+1} = \frac{\left(k+\frac{1}{2}\right)^2}{\left(k+\frac{3}{2}\right)(k+1)} c^k$

Für den Anfangswert $c_0 = a_1 = \arcsin' 0 = 1$ bestimmen wir anhand der hypergeometrischen Koeffizientenformel

$c_k = \frac{\left(\frac{1}{2}\right)_k \left(\frac{1}{2}\right)_k}{\left(\frac{3}{2}\right)_k k!} = \frac{(2k)!}{(2k+1)\, 4^k\, (k!)^2}$

Damit erhalten wir die Gausssche hypergeometrische Reihe:

$$\arcsin x = \sum_{k=0}^{\infty} \frac{(2k)!}{(2k+1)\, 4^k\, (k!)^2} x^{2k+1} = {}_3F_2\left(\left.\begin{matrix}\frac{1}{2}, & \frac{1}{2} \\ \frac{3}{2} & \end{matrix}\right| x^2\right)$$

Der Algorithmus zur Bestimmung der Potenzreihendarstellung von Funktionen vom hypergeometrischen Typ ist somit vervollständigt. Diese Prozedur liefert allerdings nur für den hypergeometrischen Fall eine geschlossene Darstellung der Potenzreihenkoeffizienten von $f(x)$ und deckt nicht allgemein den holonomen Fall ab.

9 Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorlegende Arbeit selbständig verfasst habe. Ich habe keine anderen als die angegebenen Quellen benutzt. Diese Arbeit war nach meiner Kenntnis in gleicher oder ähnlicher Form noch nicht Bestandteil einer Studien- oder Prüfungsleistung.

Kassel, den 2. Juni 2014

Alicenap Tavsanli

10 Danksagung

Hiermit möchte ich mich bei allen bedanken, die mich während der Anfertigung meiner Bachelorarbeit unterstützt haben.

Insbesondere gilt mein Dank

- Prof. Dr. Wolfram Koepf für die Bereitstellung des Themas und Betreuung
- meinem Korrekturleser Ismail Mustafa Alat
- meiner Frau Sevgi für die immerwährende Unterstützung während meines Studiums
- meinen zwei Söhnen für ihre Motivation

11 Quellenverzeichnis

Wolfram Koepf
Computeralgebra
Eine algorithmisch orientierte Einführung (Kapitel 10 - Potenzreihen)
Springer-Verlag Berlin Heidelberg 2006
ISBN-10 3-540-29894-0

Power Series Rings
http://www.sagemath.org/doc/reference/power_series/sage/rings
/power_series_ring.html (Stand 2. Juni 2014)

Power Series
http://www.sagemath.org/doc/reference/power_series/sage/rings
/power_series_ring_element.html (Stand 2. Juni 2014)

Power Series Methods http://www.sagemath.org/doc/reference/power_series/sage/rings/
power_series_poly.html (Stand 2. Juni 2014)

SAGE tip: Series Expansion http://doxdrum.wordpress.com/2011/02/19/sage-tip-series-
expansion/ (Stand 2. Juni 2014)

Mathematik: Hypergeometrische Funktionen: Ein Überblick
http://matheplanet.com/default3.html?call=article.php?sid=549&ref=http%3A%2F%2F
www.google.de%2Furl%3Fsa%3Dt%26rct%3Dj%26q%3D%26esrc%3Ds%26source
%3Dweb%26cd%3D3%26ved%3D0CEgQFjAC (Stand 2. Juni 2014)

Hypergeometric functions
http://mpmath.googlecode.com/svn/trunk/doc/build /functions/hypergeometric.html
(Stand 2. Juni 2014)