České vysoké učení technické v Praze

Fakulta jaderná a fyzikálně inženýrská

Katedra matematiky

Aplikace počítačových algebraických systémů na kvantové modely se symetriemi

Výzkumný úkol

Autor práce : **Pavel Neškudla**

Školitel : **Prof. Ing. Pavel Šťovíček, DrSc.**

Školní rok : **2007/2008**

Zadání

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze 20.5. 2008 Pavel Neškudla

Tímto bych chtěl poděkovat panu Prof. Ing. Pavlu Šťovíčkovi, DrSc. za podporu, cenné rady a trpělivé vedení této práce.

Název práce:

Aplikace počítačových algebraických systémů na kvantové modely se symetriemi

Autor:

Pavel Neškudla

Obor:

Inženýrská informatika

Abstrakt:

V této práci se zabývám zejména metodami výpočtů Baker-Campbell-Hausdorffovy formule, která nachází uplatnění v kvantové mechanice. Zkoumám různé možnosti, jak s formulí pracovat a uvádím algoritmus, který na běžném PC vypočte formuli řádu 21 v době několika minut. Předkládám úspornější zápis členů formule a zabývám se implementací Dynkinovy substituce. Většina algoritmů je implementovaná v počítačovém algebraickém systému *Mathematica*.

Klíčová slova:

Baker - Campbell - Hausdorffova formule, Dynkinova substituce, Lieova algebra

Title:

Application of computer algebra systems for quantum models with symmetries

Author:

Pavel Neškudla

Abstract:

In this work there are discussed methods of calculation of the Baker-Campbell-Hausdorff series, which appears in quantum mechanics. I examine various possibilities, how to work with formula and mention the algorithm for ordinary PC to compute it up to order 21 in time of several minutes. I also set up an economy form of the formula and implement the Dynkin's substitution. Majority of the algorithms are implemented in computer algebra system *Mathematica*.

Keywords:

Baker-Campbell-Hausdorff series, Dynkin's substitution, Lie algebra

Obsah

Obsah vztahů

Úvod

Touto prací navazuji na svou bakalářskou práci, v které jsem používal počítačový algebraický systém *Mathematica* ke zkoumání základních vztahů v Lieových algebrách a některé z těchto vztahů jsem následně použil k úpravám propagátoru kvantového harmonického oscilátoru. Při úpravě propagátoru jsem použil Baker-Campbell-Hausdorffovu formuli, která je vyjádřením nekonečné řady pro $\log(\exp X \exp Y)$ pro nekomutativní proměnné X a Y. Formule nalézá uplatnění v teorii diferenciálních rovnic, teorii grup a také v kvantové fyzice [4].

Baker-Campbell-Hausdorffově formuli se věnuje celá tato práce. Zabýval jsem se třemi různými metodami vyčíslení formule do určitého řádu. Nejprve jsem se pokoušel zrychlit algoritmus uvedený v mé bakalářské práci. Dále jsem se zabýval metodou uvedenou v nedávno publikovaném článku. A nakonec jsem se nechal inspirovat dalším článkem k vytvoření vlastní metody.

Jmenované metody jsem implementoval v systému *Mathematica,* vzájemně zkontroloval korektnost a porovnal jsem jejich využitelnost a efektivitu.

1. Zavedení pojmů

1.1 Lieova algebra

■ Definice 1 :

Lieova algebra $\mathfrak{g}$ nad tělesem T je uspořádaná dvojice $\mathfrak{g} = (V, [.,.])$, kde V je vektorový prostor nad tělesem T a $[.,.] : V \times V \to V$ je binární operace (nazývaná **Lieova závorka**), která má následující vlastnosti:

1) bilinearita

$(\forall \alpha, \beta \in T, \forall x, y, z \in V)\ ([\alpha x + \beta y, z] = \alpha[x, z] + \beta[y, z])$

$(\forall \alpha, \beta \in T, \forall x, y, z \in V)\ ([z, \alpha x + \beta y] = \alpha[z, x] + \beta[z, y])$

2) antisymetrie

$(\forall x, y \in V)\ ([x, y] = -[y, x])$

3) Jacobiho identita

$(\forall x, y, z \in V)\ ([x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0)$

Poznámka: Důsledkem 2) je vztah $(\forall x \in V)\ ([x, x] = 0)$.

1.2 Baker - Campbell - Hausdorffova (BCH) formule ([1])

BCH formule pro obecnou Lieovu grupu

Buď $\mathfrak{g}$ libovolná Lieova algebra a G příslušná souvislá a jednoduše souvislá Lieova grupa. Buď $\exp : \mathfrak{g} \to G$ exponenciální zobrazení, které je lokálním difeomorfismem v okolí 0, $\exp(0)$ je jednotkový prvek G, inverzní zobrazení se značí $\log$. Definujme $Z := X * Y = \log(\exp X \exp Y)$. Pak **Baker - Campbell - Hausdorffova formule** pro výpočet Z má tvar :

$$Z = \sum_{n>0} \frac{(-1)^{n-1}}{n} \sum_{\substack{r_i+s_i>0 \\ 1\le i\le n}} \frac{(\mathrm{ad}\,X)^{r_1}(\mathrm{ad}\,Y)^{s_1} \ldots (\mathrm{ad}\,X)^{r_n}(\mathrm{ad}\,Y)^{s_n-1} Y}{(\sum_{i=1}^{n}(r_i+s_i))\, r_1!\, s_1! \ldots r_n!\, s_n!} \qquad (1)$$

kde zobrazení $(\mathrm{ad}\,A) : \mathfrak{g} \to \mathfrak{g}$ je definováno vztahem $(\mathrm{ad}\,A)\,B = [A, B]$. A pokud $s_n = 0$, tak se poslední tři členy interpretují jako $(\mathrm{ad}\,X)^{r_n-1} X$. Tato formule platí, pokud suma na pravé straně konverguje. A ta konverguje vždy pro X a Y z jistého okolí počátku (pro X, Y z vektorového prostoru se počátkem myslí okolí nulového vektoru).

Poznámka: Prvních několik členů formule má tvar:

$$Z = X + Y + 1/2[X, Y] + 1/12[X, [X, Y]] - 1/12[Y, [X, Y]] - 1/48[Y, [X, [X, Y]]] - 1/48[X, [Y, [X, Y]]] + \ldots$$

BCH formule pro maticovou Lieovu grupu

V případě, že Lieova grupa z BCH formule je maticovou Lieovou grupou $G \subset GL(n, \mathbb{R})$ (tj. množina regulárních reálných matic typu $n \times n$), pak příslušná Lieova algebra, coby tečný prostor ke G v bodě I (nyní jednotková matice), je rovněž tvořena maticemi $n \times n$ a Lieova závorka je dána komutátorem $[X, Y] = XY - YX$. Exponenciální relace je v tomto případě definována následovně.

■ Definice 2:

Buď $G \subset GL(n, \mathbb{R})$ Lieova grupa a $\mathfrak{g}$ k ní příslušná Lieova algebra. Zobrazení $\exp : \mathfrak{g} \to G$ definované předpisem:

$$(\forall X \in \mathfrak{g}) \qquad \left(\exp X = e^X = I + \sum_{k>0} \frac{1}{k!} X^k \right),$$

nazveme **exponenciálním zobrazením**.

Speciální případ Baker-Campbell-Hausdorffovy formule pro maticovou Lieovu grupu G:

$$Z = \sum_{n>0} \frac{(-1)^{n-1}}{n} \sum_{\substack{r_i+s_i>0 \\ 1\le i\le n}} \frac{X^{r_1} Y^{s_1} \ldots X^{r_n} Y^{s_n}}{r_1 ! \, s_1 ! \ldots r_n ! \, s_n !} . \qquad (2)$$

Poznámka: Výrazy prvního, druhého, třetího a čtvrtého řádu mají následující tvary:

$$Z_1 = X + Y$$
$$Z_2 = \frac{1}{2} (XY - YX)$$
$$Z_3 = \frac{1}{12} \left(X^2 Y + XY^2 - 2\, XYX + Y^2 X + Y X^2 - 2\, YXY \right)$$
$$Z_4 = \frac{1}{24} \left(X^2 Y^2 - 2\, XYXY - Y^2 X^2 + 2\, YXYX \right)$$

Tomuto speciálnímu případu formule budou věnovány následující kapitoly, v nichž budu představovat algoritmy pro vyčíslení této formule. Výraz (1) lze pak obdržet Dynkinovou substitucí, které se věnuje následující kapitola 1.3.

1.3 Dynkinova substituce

Dynkin [2] uvádí způsob, kterým lze převést BCH formuli vyjádřenou pro maticovou Lieovu grupu (2) na BCH formuli pro obecnou Lieovu grupu (1). Definoval lineární zobrazení ϕ volné asociativní algebry k do volné Lieovy algebry g předpisem:

$$\phi (x_1 x_2 \ldots x_n) = \frac{1}{n} [x_1 [x_2 [\ldots [x_{n-1}, x_n] \ldots]]] \qquad (3)$$

Z předpisu je patrné, že tato substituce přiřazuje výrazům končícím dvojicí stejných znaků nulu. Proto své následující algoritmy uvedu i verzi, která nevyčísluje koeficienty u těchto výrazů a tím pádem je doba běhu takového algoritmu kratší.

Implementace Dynkinovy substituce

Dynkinovu substituci jsem implementoval v systému *Mathematica*. Jedná se o prosté dosazení do vzorce (3) s použitím vlastnosti, že [x, y] = -[y, x]. Žádné další úpravy pomocí vlastností Lieových algeber neprovádím.

Následující funkce na vstupu zpracovává BCH formuli vyjádřenou do určitého řádu přesnosti. Omezující podmínka pro parametr VSTUP je, aby byl tvaru součtu jednotlivých členů formule, které budou tvořeny racionálním číslem a řetězcem tvaru "xy...x" a nesmí obsahovat lineární člen x + y. Výstupem funkce je pak zápis BCH formule v podobě Lieových závorek, které jsem pro použití v systému *Mathematica* označil L[., .].

```
Dynkin[VSTUP_] := Module[{k, o, j, l, m, clen},
  str = ""; dyn = 0; clen = 0;
  l = Apply[List, VSTUP];
  For[j = 0, j < Length[l], k = Apply[List, l[[j]]];
   o = Apply[List, Characters[k[[2]]]];
```

```
    If[o[[Length[o]]] == o[[Length[o] - 1]], ,
     If[o[[Length[o]]] == "x", k[[1]] = -k[[1]]];
     clen = L[ToExpression["x"], ToExpression["y"]];
     For[i = Length[o] - 1, i > 1,
      clen = L[ToExpression[o[[i]]], clen];, i--];];
    dyn = dyn + (1 / Length[o]) * k[[1]] * clen; clen = 0;, j++];
   dyn]
```

Vyjádření Lieovy závorky ve tvaru komutátoru `[x, y] = xy - yx`

Následující pravidla slouží k nahrazení Lieových závorek komutátory, tj. přiřazení `L[x, y] = xy - yx`. Tím lze převést výraz z tvaru (1) na tvar (2). Například výraz `L[x, L[x, y]]` lze pomocí vlastností Lieovy algebry a přiřazení `L[x, y] = xy - yx` upravit na tvar `xxy - 2 xyx + yxx`.

Pravidla z definice 1 o vlastnostech Lieovy algebry:

```
pLieAlg = {L[A_, A_] → 0,
   L[A_ + B_, D_] :→ L[A, D] + L[B, D],
   L[A_, B_ + C_] :→ L[A, B] + L[A, C],
   L[m : (_Integer | _Rational | _Real) A_, B_] :→ m L[A, B],
   L[A_, m : (_Integer | _Rational | _Real) B_] :→ m L[A, B],
   L[0, A_] → 0, L[A_, 0] → 0 };
```

Pravidlo pro nahrazení `L[x, y]` za `xy - yx`:

```
pKomRel = {L[X_, Y_] :→ X ** Y - Y ** X};
```

Distribuční a další pravidla pro nekomutativní násobení:

```
pExpandCR = {
   (A_ + B_) ** (C_) :→ (A ** C + B ** C) ,
   (A_) ** (B_ + C_) :→ (A ** B + A ** C),
   (A_ + B_) ** (C_ + D_) :→ (A ** C + B ** C + A ** D + B ** D) ,
   (A_ + B_ + C___)^m_ :→ (A + B + C) ** (A + B + C)^(m-1),
   ((-A_ ** B_) ** C_) :→ -(A ** B ** C),
   (C_ ** (-A_ ** B_)) :→ -(C ** A ** B),
   (C_ (-A_ ** B_)) :→ -C (A ** B)
  };
```

Ukázka použití Dynkinovy substituce

Použití a funkčnost Dynkinovy substituce ukážu na BCH formuli řádu 4:

$$\texttt{BCHrad4} = \texttt{"x"} + \frac{\texttt{"xxy"}}{12} + \frac{\texttt{"xxyy"}}{24} + \frac{\texttt{"xy"}}{2} - \frac{\texttt{"yxx"}}{6} - \frac{\texttt{"xyxy"}}{12} + \frac{\texttt{"xyy"}}{12} + \texttt{"y"} - \frac{\texttt{"yx"}}{2} + \frac{\texttt{"yxx"}}{12} - \frac{\texttt{"yxy"}}{6} + \frac{\texttt{"yxyx"}}{12} + \frac{\texttt{"yyx"}}{12} - \frac{\texttt{"yyxx"}}{24};$$

Dynkinovu substituci lze použít na všechny členy BCH formule kromě lineárního členu:

```
BCHrad4L = Dynkin[BCHrad4 - "x" - "y"]
```

$$\frac{1}{2}\,L[x, y] + \frac{1}{12}\,L[x, L[x, y]] - \frac{1}{48}\,L[x, L[y, L[x, y]]] - \frac{1}{12}\,L[y, L[x, y]] - \frac{1}{48}\,L[y, L[x, L[x, y]]]$$

Nyní opět z Dynkinova zápisu dostanu původní zápis (viz. 1.3.2). Musím přičíst lineární člen, který jsem odečetl před provedením substituce:

```
BCHrad4fromL =
 "x" + "y" + BCHrad4L //. pKomRel //. pExpandCR /. {x → "x", y → "y"} /.
   {NonCommutativeMultiply → StringJoin} // Expand
```

$$x + \frac{xxy}{12} + \frac{xxyy}{24} + \frac{xy}{2} - \frac{xyx}{6} - \frac{xyxy}{12} + \frac{xyy}{12} + y - \frac{yx}{2} + \frac{yxx}{12} - \frac{yxy}{6} + \frac{yxyx}{12} + \frac{yyx}{12} - \frac{yyxx}{24}$$

```
BCHrad4fromL - BCHrad4
```

```
0
```

2. Úpravy mého původního algoritmu z bakalářské práce

2.1 Připomenutí implementované metody

Touto kapitolou navazuji na svoji bakalářskou práci, v jejímž průběhu jsem implementoval metodu vyčíslující Baker - Campbell - Hausdorffovu formuli. Jednalo se o metodu naprogramovanou v jazyce C, která byla pomocí rozhraní Mathlink včleněna do systému *Mathematica*. Metoda byla založena na generování všech možných hodnot indexů s_i a r_i. Po každém vygenerování těchto indexů se do výstupního řetězce přidala další část formule tvořená řetězcem znaků x a y a koeficientem vypočteným podle vzorce (2).

Touto metodou jsem BCH formuli vyčíslil do 13.tého řádu v čase přibližně 5 minut. Výsledek zabíral asi 300 MB a byl celý uložen ve virtuálním adresovém prostoru počítače. Vzhledem k tomu, že paměťové nároky rostly s každým řádem přibližně čtyřnásobně, byl tento řád konečným možným při výpočtu na původní testovací konfiguraci (Sempron 3100+, 768 MB RAM).

2.2 Práce na zlepšení metody

Vzhledem k dnešnímu rapidnímu rozvoji výpočetní techniky ve směru vícejádrových procesorů jsem se rozhodl svůj program upravit tak, aby byl schopen využít právě těchto možností nových procesorů, které se stávají standardem i v běžném domácím použití.

Vícevláknová implemantace metody

Vícejádrové procesory umožňují spouštět několik úloh paralelně, tedy souběžně. Je však nutné program při jeho tvorbě upravit tak, aby tento paralelní přístup mohl využít. Existuje několik metod jak program upravit tak, aby byl automaticky zparalelizován již při kompilaci. Já jsem se ale rozhodl řídit si paralelizaci sám. Moderní multitaskingové operační systémy nabízejí několik možností jak postupovat. Umožňují spouštět několik procesů najednou a v rámci jednoho procesu mohou spustit takzvaná vlákna (anglicky thread, případně fiber), která se v podstatě chovají jako samostatný proces, s tím rozdílem že spolu dohromady s ostatními vlákny procesu sdílejí stejné systémové prostředky. Každý proces je automaticky tvořen alespoň jedním vláknem. Jedná se o hlavní vlákno, v kterém začíná chod programu. Operační systémy pak všem vláknům ve všech spuštěných procesech přidělují procesorový čas v určitých časových intervalech (kolem 10 ms). Tímto střídavým přiřazováním procesorového času lze dosáhnout dojmu paralelního běhu více vláken i na jednojádrových systémech. Na vícejádrových systémech se již nejedná pouze o dojem, ale operační systém přiděluje vláknům procesorový čas více jader, takže pokud by například v systému běžela pouze dvě vlákna, pak každé by mohlo běžet neustále a bez přerušení na vlastním jádru (takovýto model je ovšem, alespoň co se týče systému Windows, naprosto nereálný).

Pokud jsem svůj původní program spustil na počítači s dvoujádrovým procesorem, celý výpočet algoritmu probíhal pouze na jednom jádře a druhé zůstalo nevyužité. Mým cílem tedy bylo rozdělit program na jednotlivá nezávislá vlákna a ta spustit zároveň. Program jsem rozdělil na jednotky, které odpovídají jednotlivým členům vnější sumy BCH formule (2), tyto jednotky jsou na sobě datově naprosto nezávislé. Tedy například při vytváření formule řádu 12 se spustí 12 samostatných vláken, která mohou běžet současně.

Příkladová situace na dvoujádrovém procesoru s nainstalovaným operačním systémem Windows a spuštěném programu vypisující formuli řádu 4 by vypadala takto: Program současně spustí všechna 4 vlákna, z nichž každé vyčísluje jeden určitý sčítanec vnější sumy formule. Systém Windows jim začne přiřazovat procesorový čas. Nejprve předpokládejme, že všechna vlákna mají stejnou prioritu. Z tohoto důvodu se systém Windows bude snažit přiřazovat procesorový čas spravedlivě střídavě, každému vláknu přibližně stejně. Tedy vlákno číslo 1 přiřadí například prvnímu jádru. Vlákno číslo 2 druhému jádru. Po krátkém běhu těchto vláken, systém tato vlákna uspí a místo nich přiřadí procesorové časy čekajícím vláknům číslo 3 a 4. A po krátkém běhu vláken 3 a 4 zase uspí vlákna 3 a 4 a znovu aktivuje vlákna 1 a 2. Takto se situace opakuje, dokud všechna vlákna postupně neskončí svou práci. Ve skutečnosti neprobíhá aktivace a deaktivace vláken takto pravidelně, princip je, ale doufám, zřejmý.

V tomto schématu, kdy vlákna jsou ztotožněna s jednotlivými sčítanci vnější sumy, však nastává celkem nepřívětivá situace. Doba běhu každého vlákna je různá. Je tedy třeba tyto doby proměřit a zjistit, která vlákna zabírají nejvíce procesorového času. Některá vlákna končí svůj běh opravdu rychle. Na druhé straně se však může vyskytnout i 1 vlákno, které poběží déle než všechna ostatní vlákna dohromady. Tato situace naštěstí nenastává. Jednotlivým vláknům tak lze nastavit různé priority a díky tomu pak několik vláken končí svůj běh přibližně ve stejnou dobu a nezůstávají tedy nevyužitá jádra procesoru (alespoň co

se týče testování na dvoujádrovém systému). Výslednou implementací se rychlost programu oproti jednovláknové verzi přibližně zdvojnásobila.

Odstranění chyb předchozí verze a další úpravy programu

V původní implementaci metody jsem k počítání koeficientů používal číselné typy jazyka C a k ukládání těchto čísel do řetězce jsem používal funkci _itoa. Tato funkce převádí číslo typu integer na textový řetězec. Protože vstupem je typ integer, tak již při počítání řádu 13 jsem u dvou výrazů obdržel špatný koeficient. Totiž, bylo potřeba zapsat číslo 13!, které se již nevejde do 4 bytového typu integer. Další problémy nastaly i při použití číselného typu double, případně long integer (8 bytový typ), které pro výpočty vysokých řádů nebyly dostatečné.

S vidinou možnosti dosáhnout vyšších řádů přesnosti jsem nejprve upravil celý styl zapisování výsledného vzorce. Jak jsem už dříve připomenul, výsledek výpočtu formule pro řád 13 zaujímal 300 MB paměti a přitom po úpravách tohoto výrazu systémem *Mathematica* se velikost výsledku zmenšila na 400 KB. Rozhodl jsem se tedy většinu úprav vedoucích k takovémuto zkrácení implementovat přímo do algoritmu. Koeficienty jsem tedy přestal zapisovat přímo do výsledného řetězce a místo toho je spolu sčítám a ukládám do pole a teprve po skončení všech výpočtů výsledek z pole zapíši. Dále jsem se výsledek rozhodl ukládat na pevný disk do souboru, jehož umístění lze určit v balíku BCHfle.m, který obsahuje volání mého programu. Defaultní cesty k souboru jsou C:/BCHfle.txt, resp. C:/-BCHfleD.txt.

Řešení nedostatku velkých datových typů jazyka C jsem nalezl v podobě použití knihovny GMP (GNU Multiple Precision Arithmetic Library). Ta obsahuje datové typy představující čísla s neomezenou přesností a také funkce umožňující aritmetické operace s těmito čísly. Obával jsem se zejména zdržení výpočtů s těmito umělými datovými typy. Po vhodné implementaci však bylo zdržení minimální.

Během těchto úprav programu jsem poupravil také některé části algoritmu, vyčíslujícího formuli. Zejména jsem zrychlil generování sčítacích indexů s[i] a r[i] ve vnitřní sumě tím, že se při generování přeskočí mnoho variant indexů, které, kvůli podmínkám, které jsou na tyto indexy kladeny, nemohou připadat v úvahu. Celkově se tím doba potřebná pro běh programu zkrátila téměř na jednu desetinu původní doby.

2.3 Výsledné verze programu

Výsledkem práce nakonec byly dva programy vyčíslující formuli (viz dodatek). Jeden standardní, který vyčísluje koeficienty u všech možných výrazů vyskytujících se ve vzorci (BCHfle.exe). Druhým je program vhodný pro následné použití Dynkinovy substituce (BCHfleD.exe), který nevyčísluje koeficienty u slov končících dvojicí stejných znaků.

Popis toho, jak programy napsané v jazyce C spojit s *Mathematicou*, je obsažen v mé Bakalářské práci. Nyní zde uvedu jen postup, který je nutno dodržet pro spuštění. Adresář "Propagátor" (viz dodatek) je nutno zkopírovat adresáře s *Mathematicou*, konkrétně do adresáře AddOns/ExtraPackages.

Práce s balíkem pak probíhá následovně importováním funkcí.

```
<< Propagator`BCHfle`
```

```
BCHCfle[4]
```

$$x + \frac{xxy}{12} + \frac{xxyy}{24} + \frac{xy}{2} - \frac{xyx}{6} - \frac{xyxy}{12} + \frac{xyy}{12} + y - \frac{yx}{2} + \frac{yxx}{12} - \frac{yxy}{6} + \frac{yxyx}{12} + \frac{yyx}{12} - \frac{yyxx}{24}$$

```
BCHCfleForDynkinSubst[4]
```

$$x + \frac{xxy}{12} + \frac{xy}{2} - \frac{xyx}{6} - \frac{xyxy}{12} + y - \frac{yx}{2} - \frac{yxy}{6} + \frac{yxyx}{12} + \frac{yyx}{12}$$

3. Reinschova metoda

Jak jsem se již zmínil i ve své bakalářské práci, na Wikipedii jsem narazil na odkaz na článek [3], v němž autor představuje jednoduchou metodu pro spočítání výrazů BCH formule. Jeho metoda je založena na konečném počtu násobení matic. Uvedu pouze postup při vyčíslování určitého řádu formule a v článku uvedenou implementaci jeho metody. Jak ale autor uvádí, je tato implementace uzpůsobená spíše přehlednosti a jednoduchosti než efektivitě a rychlosti. Další informace o této metodě a i důkaz teorému je uveden v odkazované literatuře [3].

3.1 O metodě

K vypočtení koeficientů u všech výrazů z BCH formule (2), konkrétního řádu n, se počítá polynom n proměnných $\sigma_1, \sigma_2, \ldots \sigma_n$ a pak se provede záměna těchto proměnných substitučním operátorem. Celý n-tý řád BCH formule, tedy výraz, v němž se vyskytují všechny členy řádu n označíme z_n. Pak z_n lze vypočíst podle:

$$z_n = T\,(\log [F.G])_{1,n+1}, \qquad (4)$$

kde matice F a G typu $(n+1) \times (n+1)$ jsou je dány:

$$F_{ij} = \frac{1}{(j-i)!}, \qquad F = \begin{pmatrix} 1 & 1 & \frac{1}{2} & \frac{1}{6} & \cdots & \\ & 1 & 1 & \frac{1}{2} & \frac{1}{6} & \cdots \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & & 1 \end{pmatrix},$$

$$G_{ij} = \frac{1}{(j-i)!} \prod_{k=1}^{j-1} \sigma_k, \qquad G = \begin{pmatrix} 1 & \sigma_1 & \frac{1}{2}\sigma_1\sigma_2 & \cdots & \\ & 1 & \sigma_2 & \frac{1}{2}\sigma_2\sigma_3 & \cdots \\ & & \ddots & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix},$$

logaritmus `log [F.G]` je v tomto případě dán jako konečná suma:

$$\log [F.G] = -\sum_{q=1}^{n} \frac{(-1)^q}{q} (F.G - I)^q.$$

Index $_{1,n+1}$ na pravé straně rovnice (2) značí pravý horní prvek matice `log [F.G]`.

Operátor T nahrazuje součin proměnných σ součinem proměnných `x` a `y`. $\sigma_i{}^{\mu_i}$ se nahradí za `x` pokud $\mu_i = 0$, a nebo za `y` pokud $\mu_i = 1$. Tedy například pro `n = 6` platí, že `T (`σ_2`, `σ_4`, `σ_5`) = xyxyyx`.

3.2 Implementace metody

Vytvoření matic `F` a `G` a spočtení logaritmu.

```
p[n_] := Block[{p, F, G, i, j, k, qthpower, σ, FGm1},
  F = Table[1 / (j - i) !, {i, n + 1}, {j, n + 1}];
  G = Table[1 / (j - i) ! Product[σk, {k, i, j - 1}],
    {i, n + 1}, {j, n + 1}];
  qthpower = IdentityMatrix[n + 1];
  FGm1 = F.G - qthpower;
  Expand[-Sum[qthpower = qthpower.FGm1; (-1) ^ q / q qthpower,
      {q, n}][[1, n + 1]]]
 ]
```

Implementace operátoru T.

```
T[n_] := Block[{temp, σ, k, term, i, j},
  (temp = Expand[Product[σk^2, {k, n}] p[n]];
   Sum[term = Apply[List, temp[[i]]]; term[[1]]
     Apply[StringJoin, Take[term, -n] /. {σi_^2 → "x", σi_^3 → "y"}],
    {i, Length[temp]}])
 ]
```

Vyjádření BCH formule do určitého řádu pomocí uvedené metody.

```
BCHReinsch[n_] := Block[{i}, "x" + "y" + Sum[T[i], {i, 2, n}]]
```

3.3 Příklad použití implementované metody

p[3]

$$\frac{\sigma_1}{12} - \frac{\sigma_2}{6} + \frac{\sigma_1\,\sigma_2}{12} + \frac{\sigma_3}{12} - \frac{\sigma_1\,\sigma_3}{6} + \frac{\sigma_2\,\sigma_3}{12}$$

T[3]

$$\frac{xxy}{12} - \frac{xyx}{6} + \frac{xyy}{12} + \frac{yxx}{12} - \frac{yxy}{6} + \frac{yyx}{12}$$

BCHReinsch[3]

$$x + \frac{xxy}{12} + \frac{xy}{2} - \frac{xyx}{6} + \frac{xyy}{12} + y - \frac{yx}{2} + \frac{yxx}{12} - \frac{yxy}{6} + \frac{yyx}{12}$$

4. Metoda rozkladů

4.1 O metodě

Pro připomenutí uvedu ještě jednou speciální tvar BCH formule (2), kterou budu vyčíslovat.

$$\mathbf{Z} = \sum_{\mathbf{n>0}} \frac{(-1)^{n-1}}{n} \sum_{\substack{r_i+s_i>0 \\ 1\le i\le n}} \frac{X^{r_1} Y^{s_1} \ldots X^{r_n} Y^{s_n}}{r_1!\, s_1! \ldots r_n!\, s_n!}\,. \qquad (5)$$

■ Definice 3:

Buď $n \in \mathbb{N}$ libovolné přirozené číslo. Pak libovolnou posloupnost $(k_1, k_2, \ldots, k_m) : m \in \mathbb{N}, k_i \in \mathbb{N}\ \forall\ i \in \{1, \ldots, m\}$ nazveme **kompozicí** čísla n, pokud platí, že $\sum_{i=1}^{m} k_i = n$.

Pokud kompozice $(k_1, k_2, \ldots, k_m)$ splňuje podmínku $k_1 \le k_2 \le \ldots \le k_m$, pak ji nazveme **rozkladem** čísla n.

■ Značení:

Zápisem slova označuji výraz tvaru $X^{r_1} Y^{s_1} \ldots X^{r_n} Y^{s_n}$ pro pevné nezáporné indexy r_i, s_i splňující podmínku $r_i + s_i > 0\ \forall\ i \in \{1, \ldots, n\}$. Tuto podmínku označím jako (**rs) podmínku**. Budu se na ni odkazovat dále v textu.

Každému takovému zápisu slova pak jednoznačně přiřadím posloupnost tvaru $(r_1, s_1, \ldots, r_n, s_n)$, kterou budu označovat jako **mocninový zápis slova** $X^{r_1} Y^{s_1} \ldots X^{r_n} Y^{s_n}$. Je zřejmé, že několik různých mocninových zápisů může být mocninovým zápisem "významově" stejného slova. Například mocninové zápisy $(1, 2, 1, 0)$, resp. $(1, 1, 0, 1, 1, 0)$ odpovídají zápisům slov XY^2X, resp. $XYYX$, která jsou "významově" stejná. Tím se myslí, že odpovídají stejnému prvku ve volné asociativní algebře generované X a Y. Takováto slova budeme ztotožňovat.

Změnou ze znaku X **na znak** Y ve slově A se myslí počet podslov XY ve slově A zapsaném bez mocnin pouze pomocí znaků X a Y. Počet změn z X na Y ve slově označíme jako P_{XY} a počet změn z Y na X jako P_{YX} a dále definujeme $P := P_{XY} + P_{YX}$. Například pro slovo XXYYYXY je $P_{XY} = 2$ a $P_{YX} = 1$ a tedy $P = 3$.

Princip metody

K tomu, abych určil koeficient pro dané slovo (při výše uvedeném ztotožnění), musím určit všechny možné zápisy onoho slova a k nim příslušné mocninové zápisy splňující (rs) podmínku. Koeficient u výchozího slova lze pak vypočítat jako sumu **částečných koeficientů** (výrazů příslušejících každému konkrétnímu mocninovému zápisu):

$$\frac{(-1)^{n-1}}{n} \frac{1}{r_1 ! \, s_1 ! \, \ldots \, r_n ! \, s_n !} \qquad (6)$$

přes všechny možnosti mocninových zápisů slova $(r_1, s_1, \ldots, r_n, s_n)$ splňující (rs) podmínku. (Tato skutečnost vyplývá ze zápisu vzorce (5).)

Pokud bych se tedy například zajímal o koeficient pro slovo X, pak zřejmě existuje pouze jediný možný zápis tohoto slova vyhovující (rs) podmínce, a to $X^1 Y^0$. Mocninový zápis je tedy $(1, 0)$, z čehož dostáváme $n = 1$, $r_1 = 1$ a $s_1 = 0$. Další možnosti zápisu (např. $X^0 Y^0 X^1 Y^0$) jsou zakázány (rs) podmínkou. Tedy koeficient bude podle (6): $\frac{(-1)^{1-1}}{1} \frac{1}{1! \times 0!} = 1$. V případě slova XY je již situace trochu složitější. Slovo lze totiž v souladu s (rs) podmínkou zapsat jako $X^1 Y^1$ a i jako $X^1 Y^0 X^0 Y^1$. Celkový koeficient u slova XY je pak podle (6): $\frac{(-1)^{1-1}}{1} \frac{1}{1! \times 1!} + \frac{(-1)^{2-1}}{2} \frac{1}{1! \times 0! \times 0! \times 1!} = 1 - \frac{1}{2} = \frac{1}{2}$. Uvedu ještě příklad slova YX. Slovo lze (na rozdíl od předešlého) zapsat pouze jedním způsobem (kvůli pořadí prvků X a Y ve vzorci (5)) jako $X^0 Y^1 X^1 Y^0$. Koeficient je tedy $\frac{(-1)^{2-1}}{2} \frac{1}{0! \times 1! \times 1! \times 0!} = -\frac{1}{2}$.

Už tyto dva poslední příklady slov XY a YX naznačují částečnou asymetrie slov, ve kterých se zamění prvky X a Y. Tato asymetrie bude podrobně vysvětlena v kapitole 4.1.3, ale její podstata je následující. Kdykoliv se ve slově mění sekvence znaků z X na Y (tedy v části slova, kde se vyskytuje podslovo XY), tak se v mocninovém zápisu naskýtá možnost zapsat slovo ještě jedním dalším mocninovým zápisem a to takovým, že se do původního mocninového zápisu vloží dvojice 0, 0 mezi členy r_i a s_i, pokud $r_i > 0$ a $s_i > 0$. To znamená, že původní mocninový zápis $(r_1, s_1, \ldots, r_i, s_i, r_{i+1}, \ldots r_n, s_n)$ přejde v mocninový zápis $(r_1, s_1, \ldots, r_i, 0, 0, s_i, r_{i+1}, \ldots r_n, s_n)$. Zatímco při změně Y na X (tedy v místě, kde slovo obsahuje podslovo YX) se tato možnost kvůli (rs) podmínce nenabízí. Například slova YXY a XYX jsou, na rozdíl od výše uvedených slov XY a YX, vzhledem k počtu obsažených podslov XY rovnocenná.

Nyní uvedu postup, který používám pro nalezení úplně všech možností mocninových zápisů daného slova, které splňují (rs) podmínku.

Nejprve určím takzvaný **nejkratší mocninový zápis** slova, který je jednoznačný. Slovo lze jednoznačně zapsat ve tvaru, ve kterém po sobě nenásledují dva stejné znaky. Příslušný nejkratší mocninový zápis je posloupnost celých nezáporných čísel sudé délky, kdy pouze první a poslední člen může být nula (první je nula, pokud slovo začíná na Y, a poslední je nula, pokud slovo končí na X). Tedy například slovo YYYXX má nejkratší zápis $X^0\ Y^3\ X^2\ Y^0$ a k němu přísluší nejkratší mocninový zápis (0, 3, 2, 0).

Dále pro každé číslo nejkratšího mocninového zápisu určím všechny jeho kompozice. Nahrazením jednoho konkrétního čísla mocninového zápisu jeho kompozicí (upravenou přidáním nuly mezi každé dva členy kompozice) získám další mocninový zápis téhož slova. Pokud pak vytvořím všechny *kombinace různých kompozic jednotlivých čísel* výchozího nejkratšího mocninového zápisu, pak tím získám úplně všechny možnosti jak dané slovo zapsat, až na ty možnosti, kde se v mocninovém zápisu vyskytují dvě nuly za sebou.

Ono vytváření všech *kombinací různých kompozic jednotlivých čísel* nejkratšího mocninového zápisu přiblížím na příkladu. Vlastně vytvořím jakousi 2-rozměrnou tabulku, kde první řádek bude obsahovat výchozí číslo (člen nejkratšího mocninového zápisu slova) a další případné řádky budou obsahovat různé kompozice výchozího čísla (ve sloupci pod ním). Takže pro náš konkrétní příklad slova YYYXX s mocninovým zápisem (0, 3, 2, 0) by tabulka vypadala takto:

$$\begin{pmatrix} \{0\} & \{3\} & \{2\} & \{0\} \\ & \{2,\ 1\} & \{1,\ 1\} & \\ & \{1,\ 2\} & & \\ & \{1,\ 1,\ 1\} & & \end{pmatrix}$$

Z této tabulky je vidět, že podslovo YYY lze zapsat čtyřmi způsoby: 1) Y^3, 2) $Y^2\ Y$, 3) YY^2, 4) YYY a XX lze zapsat dvěma způsoby: 1) X^2 a 2) XX. A pokud vytvoříme kombinace těchto zápisů jednotlivých stejno-znakových podslov (které odpovídají kombinacím kompozic čísel nejkratšího mocninového zápisu), pak celkem lze slovo YYYXX podle předchozích informací zapsat $4 \times 2 = 8$ mi způsoby: 1) $Y^3\ X^2$, 2) $Y^2\ YX^2$, 3) $YY^2\ X^2$, 4) $YYYX^2$, 5) Y^3 XX, 6) Y^2 YXX, 7) YY^2 XX, 8) YYYXX. Tyto zápisy odpovídají mocninovým zápisům (0, 3, 2, 0), (0, 2, 0, 1, 2, 0), (0, 1, 0, 2, 2, 0), (0, 1, 0, 1, 0, 1, 2, 0), (0, 3, 1, 0, 1, 0), (0, 2, 0, 1, 1, 0, 1, 0), (0, 1, 0, 2, 1, 0, 1, 0), (0, 1, 0, 1, 0, 1, 1, 0, 1, 0). Stejným způsobem lze postupovat pro libovolné slovo a získáme velkou množinu mocninových zápisů.

Většinou lze tuto množinu rozšířit tím, že na některé pozice již získaných mocninových zápisů vložím dvojici nul (sekvenci (0, 0)). (Tuto situaci jsem již naznačil dříve v této kapitole.) Vložení bude možné provést kdykoliv, když se v zápisu slova přechází znaky z X na Y (tedy v části slova, kde se vyskytuje podslovo XY), což odpovídá tomu, kdy v mocninovém zápisu po sobě následují indexy r_i, s_i takové, že $r_i,\ s_i > 0$. Takový zápis $(r_1,\ s_1,\ \ldots,\ r_i,\ s_i,\ \ldots\ r_n,\ s_n)$ lze upravit na $(r_1,\ s_1,\ \ldots,\ r_i,\ 0,\ 0,\ s_i,\ \ldots\ r_n,\ s_n)$ a ten je pořád mocninovým zápisem původního slova. Je samozřejmě možné, že do jednoho mocninového zápisu půjde vložit

nuly na více pozic (počet těchto pozic označím p_{xy}). Bude tedy existovat více dvojic indexů r_i, s_i splňujících $r_i, s_i > 0$. Nechť postupně vypíši indexy i identifikující tyto různé pozice a označím je jako $i_1, i_2, \ldots, i_{p_{xy}}$. Pak je potřeba vytvořit všech $2^{p_{xy}}$ (včetně výchozího) různých mocninových zápisů z výchozího mocninového zápisu. Tyto zápisy vzniknou z původního mocninového zápisu vždy výběrem, zda na jeho i_j-té místo vložím dvojici nul nebo ne. To jsou dvě možnosti pro každou z m pozic, celkem je tedy $2^{p_{xy}}$ možností jak upravit původní mocninový zápis. V tomto počtu je obsažena i možnost (kdy se na žádnou z m pozic nevloží dvojice nul), která je přímo původním mocninovým zápisem.

Po tomto kroku je již vytvořena množina úplně všech mocninových zápisů pro dané slovo, které splňují (rs) podmínku. Další změnou nebo přidáním nenulového čísla do některého z již vytvořených by se změnil počet některého ze znaků v původním slově, nejednalo by se tedy už o původní slovo. Přidáním nul na jiné místo než určené v předchozím odstavci by se porušila (rs) podmínka.

Když se opět vrátím k příkladu slova YYYXX, pak do tohoto slova nelze přidat nuly nikde. Tedy dříve vypsané možnosti mocninových zápisů jsou již všechny. Výsledný koeficient pak bude:

$$\frac{(-1)^{2-1}}{2}\frac{1}{0!\times 3!\times 2!\times 0!}+\frac{(-1)^{3-1}}{3}\frac{1}{0!\times 2!\times 0!\times 1!\times 2!\times 0!}+\frac{(-1)^{3-1}}{3}\frac{1}{0!\times 1!\times 0!\times 2!\times 2!\times 0!}+\frac{(-1)^{4-1}}{4}\frac{1}{0!\times 1!\times 0!\times 1!\times 0!\times 1!\times 2!\times 0!}+\frac{(-1)^{3-1}}{3}\frac{1}{0!\times 3!\times 1!\times 0!\times 1!\times 0!}+\frac{(-1)^{4-1}}{4}\frac{1}{0!\times 2!\times 0!\times 1!\times 1!\times 0!\times 1!\times 0!}+\frac{(-1)^{4-1}}{4}\frac{1}{0!\times 1!\times 0!\times 2!\times 1!\times 0!\times 1!\times 0!}+\frac{(-1)^{5-1}}{5}\frac{1}{0!\times 1!\times 0!\times 1!\times 0!\times 1!\times 1!\times 0!\times 1!\times 0!}$$

$$\frac{1}{180}$$

Naopak pro slovo XYXY, která má jediný možný mocninový zápis bez dvojice nul $(1, 1, 1, 1)$, existují dvě pozice kam lze dvojici nul přidat ($i_1 = 1$, $i_2 = 2$). Celkem lze vytvořit čtyři mocninové zápisy odpovídající tomuto slovu: 1) $(1, 1, 1, 1)$, 2) $(1, 0, 0, 1, 1, 1)$, 3) $(1, 1, 1, 0, 0, 1)$ a 4) $(1, 0, 0, 1, 1, 0, 0, 1)$ a koeficient je:

$$\frac{(-1)^{2-1}}{2}\,\frac{1}{1!\times1!\times1!\times1!}+\frac{(-1)^{3-1}}{3}\,\frac{1}{1!\times0!\times0!\times1!\times1!\times1!}+\frac{(-1)^{3-1}}{3}\,\frac{1}{1!\times1!\times1!\times0!\times0!\times1!}+\frac{(-1)^{4-1}}{4}\,\frac{1}{1!\times0!\times0!\times1!\times1!\times0!\times0!\times1!}$$

$$-\frac{1}{12}$$

Zrychlení výpočtu koeficientu pro dané slovo

■ Zrychlení počítáním s rozklady místo kompozic

Samotný výpočet koeficientu pro dané slovo lze velmi zefektivnit tím, že místo s kompozicemi čísel počítám jen s rozklady čísel. Těch je podstatně méně než kompozic a poskytují všechny potřebné informace o kompozicích (ty lze totiž vytvořit permutacemi rozkladů).

Například číslo 3 má 3 rozklady `{{1, 1, 1}, {1, 2}, {3}}` a 4 kompozice `{{1, 1, 1}, {1, 2}, {2, 1}, {3}}`. V případě čísla 20 jsou ale tyto počty velmi odlišné, rozkladů je 627 a kompozic 524 288.

```
<< Combinatorica` (*balík obshující funkci Partitions*)
Length[Partitions[20]](*Partitions - česky rozklady*)
Length[Flatten[Cases[Partitions[20], x_ ↦ Permutations[x]], 1]]
```

```
627
```

```
524 288
```

Pokud máme seznam všech rozkladů daného čísla, pak seznam všech kompozic získáme snadno jako seznam všech permutací jednotlivých rozkladů (viz předchozí zápis v systému *Mathematica*).

Možnost využít rozkladů čísel místo kompozic si ukážeme na následujícím příkladu. Částečný koeficient pro zápis XX^2 (odpovídající kompozici `{1, 2}`) slova X^3 je totiž úplně stejný jako koeficient pro zápis $X^2\,X$ (odpovídající kompozici `{2, 1}`) téhož slova, přičemž oba zápisy vycházejí z permutací rozkladu `{1, 2}` čísla 3. Částečný koeficient se totiž podle (6) počítá násobením, které je komutativní a proto získáme pro všechny permutace jednoho rozkladu stejný výsledek.

Stejně tak to platí i částečné koeficienty složitějších slov, kde mocninové zápisy vytváříme pomocí kombinací kompozic. Tyto koeficienty lze obdržet ze zápisů vzniklých pouze kombinacemi rozkladů. Zkombinujeme-li například 2 rozklady čísel, pak koeficient u takto získaného mocninového zápisu bude stejný, jako kdybychom zkombinovali libovolné permutace výchozích rozkladů (opět plyne z (6)).

Problém však nastává, když chceme získat celkový koeficient u nějakého slova. Ten se počítá jako součet částečných koeficientů přes všechny možnosti mocninových zápisů splňujících (rs) podmínku.

A všechny tyto možnosti jsem obdržel kombinacemi kompozic (ne rozkladů) čísel nejkratšího mocninového zápisu. Stačí si ale pro každý rozklad pamatovat, kolik má možných permutací. Pak spočítat jen koeficienty pro kombinace rozkladů (ne kompozic) a tyto koeficienty vynásobit počtem možností kombinací permutací těchto rozkladů. Tyto počty možností kombinací permutací se spočítají prostým násobením počtů možností jednotlivých permutací rozkladů. Vezměme za příklad slovo XXXYYY v mocninovém zápisu (1, 0, 2, 1, 0, 2), který je vyjádřen pomocí rozkladu {1, 2} čísla 3. To odpovídá zápisu $XX^2\ YY^2$. Pak rozklad {1, 2} má 2 permutace {1, 2}, {2, 1} a celkem lze tedy z původního mocninového zápisu vytvořit $2 \times 2 = 4$ kombinace permutací rozkladů. Z toho plyne, že celkem 4 mocninové zápisy původního slova budou mít stejný částečný koeficient (protože n bude pro tyto zápisy stejné a na pořadí indexů s_i a r_i při výpočtu koeficientu nezáleží). Konkrétně to budou zápisy: $XX^2\ YY^2$, $XX^2\ Y^2\ Y$, $X^2\ XYY^2$, $X^2\ XY^2\ Y$.

■ Zrychlení nevytvářením zápisů s přidanými nulami

Popisovanou metodu jsem nejprve implementoval tak, že jsem pro každou z možností kombinací rozkladů vytvořil ještě další možnosti tím, že jsem přidával dvojici nul do mocninových zápisů podle poslední části kapitoly 4.1.1. Tedy například pro slovo XY jsem měl jediný mocninový zápis bez dvojice nul (1, 1). A ještě jsem vytvořil další možnost (1, 0, 0, 1) přidáním dvojice nul. Následně jsem spočítal částečný koeficient pro každý z těchto mocninových zápisů zvlášť.

Při počítání částečného koeficientu podle předpisu (6) je však zřejmé, že člen $\frac{1}{r_1!\ s_1!\ldots r_n!\ s_n!}$ bude jak pro původní mocninový zápis, tak i pro odvozený mocninový zápis, vzniklý přidáním některých dvojic nul, stejný. Jediné, co se v předpisu (6) změní, je parametr n. Ten bude růst společně s počtem přidaných dvojic nul do původního mocninového zápisu.

Nechť do původního zápisu lze přidat dvojici nul na p_{xy} míst, pak je potřeba zjistit kolik je možností přidání právě jedné dvojice nul, dvou dvojic nul, atd. až m dvojic nul. Tento počet je možné vyjádřit kombinačním číslem $\binom{p_{xy}}{i}$ pro přidání právě i dvojic nul. Přidáním i nul se n z původního mocninového zápisu změní na n + i a tak celkový částečný koeficient zahrnující všechny možnosti rozšíření původního mocninového zápisu o dvojice nul je podle (6) dán součtem

$$\sum_{i=1}^{p_{xy}} \binom{p_{xy}}{i} \frac{(-1)^{n-1+i}}{n+i} \frac{1}{r_1!\ s_1!\ldots r_n!\ s_n!}, \text{ kde } \frac{1}{r_1!\ s_1!\ldots r_n!\ s_n!}$$

je pro všechny uvažované mocninové zápisy stejné. Celkový koeficient získaný sečtením tohoto koeficientu a koeficientu u původního mocninového zápisu je roven

$$\frac{1}{r_1!\ s_1!\ldots r_n!\ s_n!} \sum_{i=0}^{p_{xy}} \binom{p_{xy}}{i} \frac{(-1)^{n-1+i}}{n+i}, \qquad (7)$$

kde $(r_1, s_1, \ldots, r_n, s_n)$ je původní mocninový zápis, bez přidaných nul. Tuto sumu lze dokonce sečíst. Výsledek systému *Mathematica*:

$$\left(\sum_{i=0}^{p_{xy}} \text{Binomial}\left[p_{xy},\, i\right] * \frac{(-1)^{(n-1+i)}}{n+i}\right)$$

$$-\frac{(-1)^{n}\,\text{Gamma}[n]\,\text{Gamma}[1+p_{xy}]}{\text{Gamma}[1+n+p_{xy}]}$$

A pro gamma funkci přirozených čísel platí, že $\Gamma(n) = (n-1)!$. Tedy výsledný koeficient pro množinu mocninových zápisů založených na jednom mocninovém zápisu bez přidaných nul je:

$$-\frac{(-1)^{n}\,(n-1)!\,p_{xy}!}{(n+p_{xy})!\,r_1!\,s_1!\,\ldots\,r_n!\,s_n!}. \qquad (8)$$

Tímto postupem velmi klesne počet generovaných kombinací při výpočtu koeficientu a tím pádem se výpočet velice zefektivní. Například pro slovo $(XY)^{20}$ je počet možností, kterými lze přidat dvojice nul roven:

```
Sum[Binomial[20, i], {i, 1, 20}]
```

```
1048575
```

Ale existuje pouze jediný mocninový zápis takového slova bez přidaných nul. Tedy místo výpočtu částečných koeficientů u přibližně 1 milionu možností mocninových zápisů stačí spočíst jen jeden souhrnný koeficient podle (8).

Vliv na zrychlení výpočtu koeficientu má tento postup samozřejmě rozdílný v závislosti na vstupním slově. Pokud je slovo typu $(XY)^{k} : k \in \mathbb{N}$, pak ať je k jakkoliv velké, tak doba výpočtu bude velmi malá (vytvářený mocninový zápis bude jen jeden) ve srovnání s dobou, kterou by to výpočet trval bez tohoto postupu. Naopak pro slovo typu $Y^{k}X^{l} : k, l \in \mathbb{N}$ nepřinese tato metoda vůbec žádné zrychlení (nuly nelze přidat vůbec, takže není prostor pro použití postupu). Při několika měřeních jsem zjistil průměrně asi trojnásobné zrychlení při výpočtu koeficientů u všech slov BCH formule vyjadřované do určitého řádu.

Výpočet koeficientů u všech slov BCH formule určitého řádu

Předcházející metoda pro výpočet jednoho konkrétního koeficientu je velice rychlá a proto jsem se ji rozhodl využít i pro vyjádření koeficientů u všech slov BCH formule určitého řádu. Je nutné pouze získat seznam všech slov, která se v daném řádu vyskytují a pro tato slova spočíst jejich příslušné koeficienty pomocí předchozího. Slova vyskytující se v řádu n mají počet znaků roven n a jejich počet je 2^{n} (na každé z n pozic slova může být buď X nebo Y).

■ Přiřazení jednoho koeficientu dalším slovům

Postup, kdy se počítají koeficienty pro všechny slova zvlášť, lze ještě velmi zefektivnit následujícími postupy.

■ Permutace mocninového zápisu

Nechť je dáno libovolné slovo BCH formule a k němu jeho příslušný nejkratší mocninový zápis. Takový zápis je posloupnost, která nemůže obsahovat nulu jinde než na začátku nebo na konci. Pokud v tomto mocninovém zápisu provedu libovolnou permutaci nenulových čísel, pak dostanu nový nejkratší mocninový zápis příslušející novému slovu, které začíná na stejný znak jako původní slovo a je v něm také stejný počet přechodů ze znaku `X` na `Y` (počet podslov tvaru `XY`) jako v původním slově. Vzhledem k (6) musí mít tato slova stejný koeficient, dojde totiž pouze k prohození členů mocninového zápisu.

Naleznu-li koeficient například pro slovo `XXYYYXXXX` (v nejkratším mocninovém zápisu `(2, 3, 4)`), pak stejný koeficient má i slovo `XXXYYXXXX` (v nejkratším mocninovém zápisu `(3, 2, 4)`), a stejně tak i další slova vzniklá permutacemi mocninového zápisu.

Tohoto faktu lze využít nejen k rychlejšímu vyčíslení koeficientů slov určitého řádu ale i pro kratší formu zápisu BCH formule. Nemusíme vypisovat všechna slova s takto "příbuzným" mocninovým zápisem, ale stačí vybrat jednoho reprezentanta z takové skupiny (přes permutace příbuzných slov) a ostatní ze skupiny nezapisovat. Implementace tohoto způsobu zápisu BCH formule je uvedena pod názvem BCHFleKratce.

■ Částečná symetrie koeficientů slov se zaměněnými znaky

- **Pozorování 1**

Nechť je dáno slovo `A` řádu $r = \sum_{i=1}^{n} r_i + s_i$, které má koeficient C_A. Pak slovo `B`, které vznikne z původního slova `A` nahrazením všech výskytů symbolu `X` za `Y` a naopak `Y` za `X`, má koeficient $C_B = C_A * (-1)^{r-1}$.

Tedy pokud je řád slova lichý, pak slovo `B` má stejný koeficient jako slovo `B`. V případě, že řád je sudý, tak se koeficienty slov `A` a `B` liší jen znaménkem.

Důkaz tohoto tvrzení je uveden v [3] jako vztah (6.1). Rovnost $e^z = e^x e^y$ totiž implikuje $e^{-z} = e^{-y} e^{-x}$ a tedy, že $z = -\log e^{-y} e^{-x}$. Z toho a z předpisu (2) plyne, že $z =$

$$-\log e^{-y} e^{-x} = -\sum_{n>0} \frac{(-1)^{n-1}}{n} \sum_{\substack{r_i+s_i>0 \\ 1\le i\le n}} \frac{(-Y)^{r_1} (-X)^{s_1} \dots (-Y)^{r_n} (-X)^{s_n}}{r_1!\, s_1! \dots r_n!\, s_n!} =$$

$$\sum_{n>0} \frac{(-1)^{n-1}}{n} \sum_{\substack{r_i+s_i>0 \\ 1\le i\le n}} (-1)^{r-1} \frac{Y^{r_1} X^{s_1} \dots Y^{r_n} X^{s_n}}{r_1!\, s_1! \dots r_n!\, s_n!}$$

A po srovnání tohoto výrazu s původním (2), kde chybí člen $(-1)^{r-1}$:

$$z = \sum_{n>0} \frac{(-1)^{n-1}}{n} \sum_{\substack{r_i+s_i>0 \\ 1\le i\le n}} \frac{X^{r_1} Y^{s_1} \dots X^{r_n} Y^{s_n}}{r_1!\, s_1! \dots r_n!\, s_n!},$$

je tvrzení zřejmé.

Pozorování 2

Následující úvahy navazují na značení změn definované zpočátku této kapitoly. Dále označím n_z jako n získané ze základního nejkratšího mocninového zápisu slova $(r_1, s_1, \ldots, r_n, s_n)$. Pak pro sudé P platí, že $P_{XY} = \frac{P}{2}$ a $n_z = \frac{P+2}{2}$. Je to proto, že nejkratší mocninové zápisy takovýchto slov mohou vypadat jedním z následujících dvou způsobů:

a) když slovo začíná znakem X, pak
$$MZ = (r_1, s_1, \ldots, r_n, s_n) = (i_0, i_1, \ldots, i_P, 0).$$

b) když slovo začíná znakem Y, pak
$$MZ = (r_1, s_1, \ldots, r_n, s_n) = (0, i_0, i_1, \ldots, i_P).$$

Kde pro oba případy $i_j > 0 \; \forall \; j \in \{0, 1, \ldots, P\}$. Pokud budeme uvažovat slova A a B z pozorování 1, pak obě tato slova mají stejné P a podle předchozího i stejná P_{XY} a n_z.

- **Důsledek obou pozorování**

Pokud je řád slova sudý a P je také sudé, pak toto slovo má nulový koeficient v BCH formuli.

Důkaz: Souhrnný koeficient spočtený pro jeden konkrétní mocninový zápis slova A označím $C_A(mz)$. Pak podle (7) pro tento koeficient platí:

$$C_A(mz) = \frac{1}{r_1!\, s_1! \ldots r_n!\, s_n!} \sum_{i=0}^{P_{XY}} \binom{P_{XY}}{i} \frac{(-1)^{n-1+i}}{n+i}, \qquad (9)$$

pro daný mocninový zápis $mz = (r_1, s_1, \ldots, r_n, s_n)$ slova A. Dále je pak z pozorování 2 zřejmé, že slova A a B mají stejný počet možností mocninových zápisů. A každá taková možnost mocninového zápisu slova A má svého analogického "dvojníka" v podobě zápisu mocninového zápisu slova B. Pokud má totiž slovo A konkrétní mocninový zápis $mz_A = (i_0, i_1, \ldots, i_P, 0)$, pak slovo B má určitě mocninový zápis $mz_B = (0, i_0, i_1, \ldots, i_P)$, nebo naopak. Z toho dostáváme rovnosti $\left(\frac{1}{r_1!\, s_1! \ldots r_n!\, s_n!}\right)_{mz_A} = \left(\frac{1}{r_1!\, s_1! \ldots r_n!\, s_n!}\right)_{mz_B}$ a $(P_{XY})_{mz_A} = (P_{XY})_{mz_B}$ a $(n)_{mz_A} = (n)_{mz_B}$. Z toho a z (9) vyplývá, že $C_A(mz_A) = C_B(mz_B)$.

Celkem z toho plyne, že celkový koeficient C_A slova A je stejný jako koeficient C_B slova B, protože $C_A = \sum_{mz_A} C_A(mz_A) = \sum_{mz_B} C_B(mz_B) = C_B$. Kde suma je vždy přes všechny možné základní (bez přidaných nul) mocninové zápisy slova. Ale, z pozorování 1 platí $C_B = C_A * (-1)^{r-1}$, a nyní je řád r sudý a tedy $C_B = -C_A$. Nutně tedy musí být $C_A = C_B = 0$.

Slov sudého řádu, u kterých lze využít předcházejícího důsledku, je vždy polovina všech slov tohoto řádu, protože vznik zápisu libovolného slova si lze představit následovně. Nejprve zvolím počáteční znak. Mám tedy zvolen doposud poslední znak slova a po něm

zvolím následující znak buď stejný jako současný nebo zvolím ten druhý. Takto se lze rozhodovat v každém kroku, přičemž každé takovéto rozhodnutí je zároveň rozhodnutím o tom, zda počet změn znaků P bude sudý nebo lichý. A protože pro výpis všech slov daného řádu BCH formule je potřeba provést všechny kombinace těchto rozhodnutí, pak polovina těchto rozhodnutí vede k sudému počtu změn znaků P a takto vzniklá slova splňují podmínky důsledku.

Inspirace z článku [4]

V článku [4] se pojednává o praktickém algoritmu, který je určen k vyčíslení koeficientů u jednotlivých členů formule (1).

Algoritmus vychází z formule (2), z níž vybere takzvané homogenní členy a u nich určí koeficienty. Homogenní členy jsou všechna slova $X^{r_1} Y^{s_1} \ldots X^{r_n} Y^{s_n}$ splňující $\sum_{i=1}^{n} r_i = r$, $\sum_{i=1}^{n} s_i = s$. Dynkin odvodil předpis pro takový homogenní člen pomocí komutátorů:

$$P_{r,s} = \frac{1}{r+s} \sum \frac{(\mathrm{ad}\, X)^{r_1} (\mathrm{ad}\, Y)^{s_1} \ldots (\mathrm{ad}\, X)^{r_n} (\mathrm{ad}\, Y)^{s_n - 1} Y}{r_1!\, s_1! \ldots r_n!\, s_n!}$$

kde suma jde přes všechny možnosti posloupností nezáporných indexů $(r_1, s_1, \ldots r_n, s_n) : \sum_{i=1}^{n} r_i = r, \sum_{i=1}^{n} s_i = s, s_i + r_i > 0$.

Při vytváření své metody rozkladu jsem se přímo inspiroval tímto článkem a využil jsem i několik postupů, které jsou v tomto článku uvedeny. Na rozdíl od článku, kde se získávají koeficienty homogenních členů, já jsem se zaměřil na získání koeficientu pro libovolné slovo. Tyto dvě úlohy jsou i vzhledem k Dynkinově substituci velmi podobné. Článek mě inspiroval zejména k využití rozkladů čísel pro generování možností mocninových zápisů a přidávání dvojic nul pro rozšíření množiny mocninových zápisů.

4.2 Implementace metody

■ Vynulování globálních proměnných

Následující proměnné jsou využity pro urychlení počítání koeficientů. Aby se nemusely počítat stále stejné věci dokola, tak jsou uloženy v těchto pomocných polích.

```
gRozklady = {};
gRozkladySNulama = {};
gPoctyPermutaciRozkladu = {};
```

■ Základní funkce

Zde jsou uvedeny nejjednodušší funkce algoritmu. Tyto funkce se neodkazují na žádné mnou naprogramované funkce.

Rozklady[cislo_]

```
Rozklady[cislo_] := Block[{vysl},
  vysl = Cases[Partitions[cislo], x_ :> Sort[x, Greater]]
 ]
```

■ SNulama[rozklad_]

Tato funkce přidá za každý prvek vstupního seznamu nulu a vrátí nově upravený seznam. Přiřazuje rozkladům čísel jim odpovídající mocninové zápisy.
Příklad: Slovo XX má nejkratší mocninový zápis {2, 0}, číslo 2 má rozklady {{2}, {1, 1}} a rozkladu {1, 1} odpovídá mocninový zápis (1, 0, 1).

```
SNulama[rozklad_] :=
 Block[{i, j}, Table[{Take[Table[{rozklad[[i]][[j]], 0},
        {j, 1, Length[rozklad[[i]]]}] // Flatten,
      Length[rozklad[[i]]] * 2 - 1]} //
    Flatten, {i, 1, Length[rozklad]}]
 ]
```

■ MZnaSlovo[mz_,bZacniX_]

Převod mocninového zápisu bez případné první nuly na odpovídající slovo. V takovém mocninovém zápisu nerozeznávám zda slovo začíná znakem X nebo Y. Pro výstupní slovo je zde tedy možnost volby, zda slovo má začínat na X či Y.

```
MZnaSlovo[mz_, bZacniX_] := Block[{slovo = {}, i, j, znak},
  If[bZacniX == True, znak = "x", znak = "y"];
  Do[slovo = StringJoin[slovo,
     StringJoin @@ Table[znak, {j, 1, mz[[i]]}]];
   If[znak == "x", znak = "y", znak = "x"];, {i, 1, Length[mz]}];
  slovo]
```

■ SlovoNaMZ[slovo_]

Funkce, která převede slovo na mocninový zápis bez případné nuly na začátku (nerozlišuje X a Y).

```
SlovoNaMZ[slovo_] :=
 Block[{i, j, k, vyraz, minz, vs}, vs = Characters[slovo];
  vyraz = {};
  If[vs[[1]] == "x", minz = "x";, minz = "y";];
  vyraz = {1};
  Do[If[minz == vs[[i]], vyraz[[Length[vyraz]]] =
     vyraz[[Length[vyraz]]] + 1, vyraz = Join[vyraz, {1}];];
   minz = vs[[i]];, {i, 2, Length[vs]}];
  vyraz]
```

SlovoNaMZXY[slovo_]

Funkce, která slovo převede na standardní mocninový zápis (rozlišující znaky).

```
SlovoNaMZXY[slovo_] :=
 Block[{minz, vyraz, vs, i, j, k}, vs = Characters[slovo];
  vyraz = {};
  If[vs == {"x"}, vyraz = {1, 0}, If[vs[[1]] == "x",
    vyraz = {1}; minz = "x";, vyraz = {0, 1}; minz = "y";];
   Do[If[minz == vs[[i]], vyraz[[Length[vyraz]]] =
      vyraz[[Length[vyraz]]] + 1, vyraz = Join[vyraz, {1}];];
    minz = vs[[i]];, {i, 2, Length[vs]}]];
  If[OddQ[Length[vyraz]], vyraz = Append[vyraz, 0]];
  vyraz]
```

■ SpoctiKombinaci[k_, poctyPerm_, pocetMoznostiProNuly_]

Spočítá souhrnný koeficient pro příslušné možností kombinací. V každém kroku spočítá jeden částečný koeficient pomocí kombinace rozkladů, počtu jejích permutací a počtu míst, na která je možná přidat do kombinace nuly.

```
SpoctiKombinaci[k_, poctyPerm_, pocetMoznostiProNuly_] :=
 Block[{s, vysl, n, i, j, koef, c, mz, delky, pm, d}, s = 0;
  pm = (pocetMoznostiProNuly)!;
  poctyPerm.
   Cases[k, x_ :> (-((-1)^Length[x]/2 (Length[x] / 2 - 1)! * pm) /
          (pocetMoznostiProNuly + Length[x] / 2)!)) * (1 /
        Times @@ (x!))]
 ]
```

■ Složené funkce

Tyto funkce ke svému běhu potřebují základní funkce.

■ SlovaBezPermutaci[delka_, bKonecDvojici_]

Vygeneruje všechna slova dané délky až na permutace skupin znaků (ty mají totiž stejné koeficienty). Lze určit, zda generovat slova končící na dvojici stejných znaků.

```
SlovaBezPermutaci[delka_] :=
 Block[{sr, slova, i, x}, sr = gRozklady[[delka]];
  slova = Cases[sr, x_ :> MZnaSlovo[Reverse[x], True]];
  slova]
```

■ KombinaceIndexu[slovo_]

Vytvoří základní možnosti kombinací rozkladů čísel nejkratšího mocninového zápisu slova tím, že rozdistribuuje rozklady čísel mocninového zápisu slova. Dále také spočítá, kolik lze z každé výsledné kombinace odvodit dalších kombinací permutací rozkladů. Tuto funkci nelze volat bez inicializovaných globálních polí.

```
KombinaceIndexu[slovo_] :=
 Block[{zaklad, rozklady, startT, poctyPermZaklad, poctyPermDistr,
   vysl}, If[slovo == "x", Return[{{{1, 0}}, {1}}]];
  zaklad = SlovoNaMZXY[slovo]; (*nejkratší mocninový zápis*)
  (*množina množin rozkladů
   všech čísel nejkratšího mocninového zápisu*)
  rozklady = Cases[zaklad,
    x_ :> If[x == 0, {{0}}, gRozkladySNulama[[x]]]];
  (*množina počtů permutací pro každý rozklad*)
  poctyPermZaklad =
   Cases[zaklad,
    x_ :> If[x == 0, {1}, gPoctyPermutaciRozkladu[[x]]]];
  (*rozdistribuováním vytvoříme všechny kombinace rozkladů*)
  rozklady = Distribute[rozklady, List];
  (*pronásobené počty permutací po distribuci,
  pro celkový počet možností,
  jak ze zápisu odvodit ještě další*)
  poctyPermDistr = Cases[Distribute[poctyPermZaklad, List],
    x_ :> Times @@ x];
  vysl = Flatten[rozklady, {{1}, {2, 3}}];
  (*vysl=
    Table[rozklady[[i]]//Flatten,{i,1,Length[rozklady]}];*)
  {vysl, poctyPermDistr}
 ]
```

■ KoefFromGlobalParams[slovo_]

Spočítá koeficient pro vstupní slovo. Funkci nelze volat bez inicializovaných globálních polí kvůli funkci KombinaceIndexu.

```
KoefFromGlobalParams[slovo_] :=
 Block[{znaky = Characters[slovo], indexy, vysl, startT,
   zs, i, j, k, temp, poctyPermDistr, poctyPermNul,
   pocetMoznostiProNuly}, If[slovo == "", Return[0]];
  If[slovo == "x", Return[1]];
  If[slovo == "y", Return[1]];
  pocetMoznostiProNuly = SlovoNaMZXY[slovo];
  (*zda nesplňuje podmínku pro nulový koeficient*)
  If[
   EvenQ[Length[Cases[pocetMoznostiProNuly, x_ /; x ≠ 0]] - 1] &&
    EvenQ[StringLength[slovo]], Return[0]];
  temp = KombinaceIndexu[slovo];
  indexy = temp[[1]];
  poctyPermDistr = temp[[2]];
  pocetMoznostiProNuly =
   Table[{pocetMoznostiProNuly[[2 * i + 1]], pocetMoznostiProNuly[[
```

```
      2 * i + 2]]}, {i, 0, Length[pocetMoznostiProNuly] / 2 - 1}];
  pocetMoznostiProNuly = Plus @@ Cases[pocetMoznostiProNuly,
     {x_, y_} :> 1 /; (x > 0 && y > 0)];
  vysl = SpoctiKombinaci[indexy, poctyPermDistr,
    pocetMoznostiProNuly];
  vysl
 ]
```

■ Konečné funkce pro praktické použití

Následující funkce nejprve inicializují globální pole a poté používají předcházející funkce.

■ BCHKoef[slovo_]

Spočítá koeficient pro libovolné slovo.

```
BCHKoef[slovo_] := Block[{d, rad, cisla, pocetZmen},
  If[slovo == "", Return[0]];
  If[slovo == "x", Return[1]];
  If[slovo == "y", Return[1]];
  pocetZmen = SlovoNaMZXY[slovo];
  (*zda nesplňuje podmínku pro nulový koeficient*)
  If[EvenQ[Length[Cases[pocetZmen, x_ /; x ≠ 0]] - 1] &&
    EvenQ[StringLength[slovo]], Return[0]];
  cisla = Sort[Union[SlovoNaMZ[slovo]]];
  rad = Max[cisla];
  d = Length[gRozklady];
  If[rad > d,
   gRozklady = Flatten[
     Append[{gRozklady}, Table[Rozklady[i], {i, d + 1, rad}]], 1];
   gRozkladySNulama = Flatten[Append[{gRozkladySNulama},
      Table[SNulama[gRozklady[[i]]], {i, d + 1, rad}]], 1];
   gPoctyPermutaciRozkladu = Flatten[Append[
      {gPoctyPermutaciRozkladu}, Table[Cases[gRozklady[[i]],
        x_ :> Length[Permutations[x]]], {i, d + 1, rad}]], 1];];
  KoefFromGlobalParams[slovo]]
```

■ BCHRad[rad_, bKonecDvojici_, bBezPermutaciSlov_]

Spočítá a zobrazí koeficienty všech slov daného řádu v BCH formuli. Lze si zvolit, zda počítat koeficient i u slov končících na dvojici stejných znaků. Dále umožňuje počítat koeficienty jen u základních slov, z nichž lze ostatní slova získat permutacemi mocninových zápisů a záměnou znaků (viz. následující funkce). Při tomto způsobu výpočtu je výsledný zápis velmi krátký a i rychleji spočtený. Jednotlivá základní slova jsou v zápisu uvozena symboly t[ ].

```
BCHRad[rad_, bKonecDvojici_, bBezPermutaciSlov_] :=
 Block[{vysl, sl, perm, koef, i, j,
   slovaperm, slovo, startT, d}, vysl = 0;
```

```
d = Length[gRozklady];
If[rad > d,
 gRozklady = Flatten[
   Append[{gRozklady}, Table[Rozklady[i], {i, d + 1, rad}]], 1];
 gRozkladySNulama = Flatten[Append[{gRozkladySNulama},
    Table[SNulama[gRozklady[[i]]], {i, d + 1, rad}]], 1];
 gPoctyPermutaciRozkladu = Flatten[Append[
    {gPoctyPermutaciRozkladu}, Table[Cases[gRozklady[[i]],
      x_ :> Length[Permutations[x]]], {i, d + 1, rad}]], 1];];
If[rad == 1, Return["x" + "y"]];
If[bKonecDvojici, sl = SlovaBezPermutaci[rad];
 , sl = SlovaBezPermutaci[rad - 1]];
Do[slovo = sl[[i]];
 If[bKonecDvojici == False, slovo = StringJoin[slovo,
     If[Last[Characters[slovo]] == "x", "y", "x"]];];
 koef = KoefFromGlobalParams[slovo];
 If[bBezPermutaciSlov == True,
  vysl = vysl + koef * StringJoin["t[", sl[[i]], "]"],
  If[koef ≠ 0,
    If[bKonecDvojici == False,
     perm = Cases[Permutations[SlovoNaMZ[sl[[i]]]],
        x_ :> Flatten[{x, 1}]];,
     perm = Permutations[SlovoNaMZ[sl[[i]]]];
    ];
    slovaperm = Table[MZnaSlovo[{perm[[j]]} // Flatten, True],
      {j, 1, Length[perm]}]; (*X*)
    vysl = vysl + koef * (Plus @@ slovaperm);
    slovaperm = Table[MZnaSlovo[{perm[[j]]} // Flatten, False],
      {j, 1, Length[perm]}]; (*Y*)
    vysl = vysl + ((-1) ^ (rad + 1)) * koef * (Plus @@ slovaperm);];],
 {i, 1, Length[sl]}];
vysl]
```

■ BCHFleRozkladem[rad_, bKonecDvojici_, bBezPermutaciSlov_]

Vyčíslí BCH formuli do určitého řádu pomocí předchozí funkce, přičemž zachovává její volby.

```
BCHFleRozkladem[rad_, bKonecDvojici_, bBezPermutaciSlov_] :=
  Sum[BCHRad[i, bKonecDvojici, bBezPermutaciSlov],
    {i, 1, rad}] // Expand;
```

■ BCHRad[rad_], BCHFle[rad_], BCHRadProDynkinovuSubstituci[rad_], BCHFleProDynkinovuSubstituci[rad_], BCHRadKratce[rad_]

Následují funkce, které pouze zpřístupňují varianty předchozích funkcí.

```
BCHRad[rad_] := BCHRad[rad, True, False]

BCHFle[rad_] := Sum[BCHRad[i], {i, 1, rad}] // Expand;

BCHRadProDynkinovuSubstituci[rad_] := BCHRad[rad, False, False]

BCHFleProDynkinovuSubstituci[rad_] :=
  Sum[BCHRadProDynkinovuSubstituci[i], {i, 1, rad}] // Expand;

BCHRadKratce[rad_] := BCHRad[rad, True, True]

BCHFleKratce[rad_] := Sum[BCHRadKratce[i], {i, 1, rad}] // Expand;
```

4.3 Ukázky použití metody

■ Výpisy BCH formule do určitého řádu

BCHFle[5]

$$x - \frac{xxxxy}{720} + \frac{xxxyx}{180} + \frac{xxxyy}{180} + \frac{xxy}{12} - \frac{xxyxx}{120} - \frac{xxyxy}{120} + \frac{xxyy}{24} - \frac{xxyyx}{120} + \frac{xxyyy}{180} + \frac{xy}{2} - \frac{xyx}{6} + \frac{xyxxx}{180} - \frac{xyxxy}{120} - \frac{xyxy}{12} + \frac{xyxyx}{30} - \frac{xyxyy}{120} + \frac{xyy}{12} - \frac{xyyxx}{120} - \frac{xyyxy}{120} + \frac{xyyyx}{180} - \frac{xyyyy}{720} + y - \frac{yx}{2} + \frac{yxx}{12} - \frac{yxxxx}{720} + \frac{yxxxy}{180} - \frac{yxxyx}{120} - \frac{yxxyy}{120} - \frac{yxy}{6} + \frac{yxyx}{12} - \frac{yxyxx}{120} + \frac{yxyxy}{30} - \frac{yxyyx}{120} + \frac{yxyyy}{180} + \frac{yyx}{12} - \frac{yyxx}{24} + \frac{yyxxx}{180} - \frac{yyxxy}{120} - \frac{yyxyx}{120} - \frac{yyxyy}{120} + \frac{yyyxx}{180} + \frac{yyyxy}{180} - \frac{yyyyx}{720}$$

BCHFleProDynkinovuSubstituci[5]

$$x - \frac{xxxxy}{720} + \frac{xxxyx}{180} + \frac{xxy}{12} - \frac{xxyxy}{120} - \frac{xxyyx}{120} + \frac{xy}{2} - \frac{xyx}{6} - \frac{xyxxy}{120} - \frac{xyxy}{12} + \frac{xyxyx}{30} - \frac{xyyxy}{120} + \frac{xyyyx}{180} + y - \frac{yx}{2} + \frac{yxxxy}{180} - \frac{yxxyx}{120} - \frac{yxy}{6} + \frac{yxyx}{12} + \frac{yxyxy}{30} - \frac{yxyyx}{120} + \frac{yyx}{12} - \frac{yyxxy}{120} - \frac{yyxyx}{120} + \frac{yyyxy}{180} - \frac{yyyyx}{720}$$

BCHFleKratce[5]

$$\frac{t[xxyy]}{24} + \frac{t[xxyyy]}{180} + \frac{t[xy]}{2} - \frac{t[xyx]}{6} + \frac{t[xyxxx]}{180} - \frac{t[xyxy]}{12} + \frac{t[xyxyx]}{30} - \frac{t[xyxyy]}{120} + \frac{t[xyy]}{12} - \frac{t[xyyxx]}{120} - \frac{t[xyyyy]}{720} + x + y$$

■ Ukázky koeficientů jednotlivých slov

```
startT = SessionTime[];
BCHKoef["xxxxxxxxxxxxxxxxxxxxxxxxx"]
SessionTime[] - startT
```

0

$0. \times 10^{-8}$

```
startT = SessionTime[];
BCHKoef["xyxxyxyxyxyxyxyxyxxyyxyyyyxxxxxxx"]
SessionTime[] - startT
```

$$-\frac{3851}{1\,455\,587\,813\,167\,488\,000}$$

0.1250000

5. Použití metod a jejich vzájemné srovnání

Výsledků měření obsažených v této kapitole jsem dosáhl na počítači s následující konfigurací: Procesor Intel Core 2 Duo @ 3,08 GHz, 2 GB RAM.

5.1 Porovnání výsledků metod

■ Ukázka shodnosti výsledků metod pro 4.tý řád

```
BCHCfle[4]
BCHFle[4]
BCHReinsch[4]
```

$$x+\frac{xxy}{12}+\frac{xxyy}{24}+\frac{xy}{2}-\frac{xyx}{6}-\frac{xyxy}{12}+\frac{xyy}{12}+y-\frac{yx}{2}+\frac{yxx}{12}-\frac{yxy}{6}+\frac{yxyx}{12}+\frac{yyx}{12}-\frac{yyxx}{24}$$

$$x+\frac{xxy}{12}+\frac{xxyy}{24}+\frac{xy}{2}-\frac{xyx}{6}-\frac{xyxy}{12}+\frac{xyy}{12}+y-\frac{yx}{2}+\frac{yxx}{12}-\frac{yxy}{6}+\frac{yxyx}{12}+\frac{yyx}{12}-\frac{yyxx}{24}$$

$$x+\frac{xxy}{12}+\frac{xxyy}{24}+\frac{xy}{2}-\frac{xyx}{6}-\frac{xyxy}{12}+\frac{xyy}{12}+y-\frac{yx}{2}+\frac{yxx}{12}-\frac{yxy}{6}+\frac{yxyx}{12}+\frac{yyx}{12}-\frac{yyxx}{24}$$

■ Srovnání výsledků metod do 18.tého řádu

```
startT = SessionTime[];
rad = 11;
BCHCfle[rad] - BCHFle[rad]
BCHFle[rad] - BCHReinsch[rad]
Print["Doba kontroly[s]:", SessionTime[] - startT]
```

```
0
```

```
0
```

```
Doba kontroly[s]:5.4687500
```

Srovnání formulí tímto způsobem jsem provedl až do řádu 18. Tato kontrola trvala 3000 s.

■ Srovnání formulí určených k Dynkinově substituci

Použití a funkčnost Dynkinovy substituce ukážu na BCH formuli řádu 9 (detaily viz kapitola 1.3.3).

```
startT = SessionTime[];
rad = 9;
BCHrad11 = BCHFleProDynkinovuSubstituci[rad];

BCHrad11L = Dynkin[BCHrad11 - "x" - "y"];

BCHrad11fromL = "x" + "y" + BCHrad11L //. pKomRel;

BCHrad11fromL = % //. pExpandCR /. {x → "x", y → "y"} /.
    {NonCommutativeMultiply → StringJoin} // Expand;

BCHrad11fromL - BCHFle[rad]
```

```
0
```

```
Print["Doba kontroly[s]:", SessionTime[] - startT]
```

```
Doba kontroly[s]:4.0937500
```

Tento způsob kontroly jsem provedl do 12.tého řádu. Pro vyšší řády nestačila systému *Mathematica* paměť na zpětný převod.

Následuje ještě porovnání obou mých metod vypisujících formuli vhodnou pro Dynkinovu substituci.

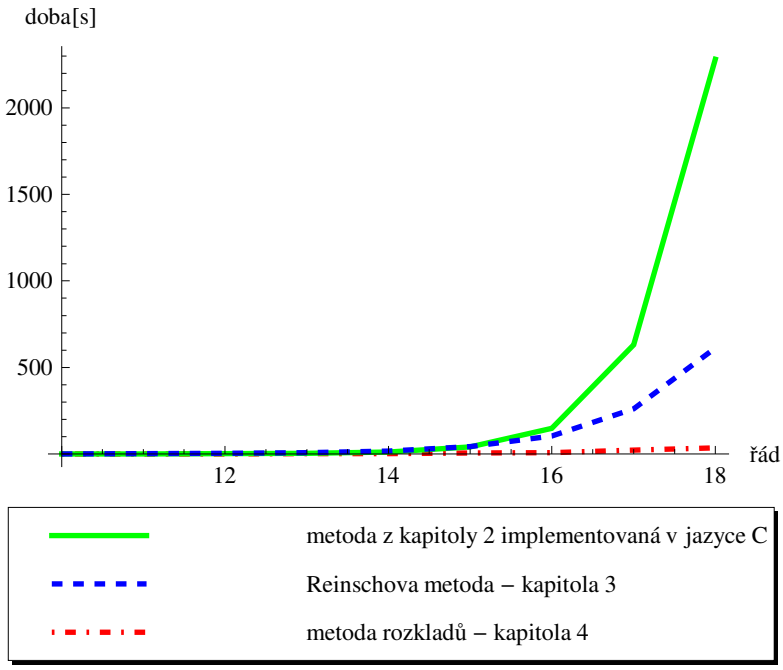
```
startT = SessionTime[];
rad = 13;
BCHCfleForDynkinSubst[rad] - BCHFleProDynkinovuSubstituci[rad]
Print["Doba kontroly[s]:", SessionTime[] - startT]
```

```
0
```

```
Doba kontroly[s]:8.6093750
```

5.2 Srovnání dob výpočtů metod

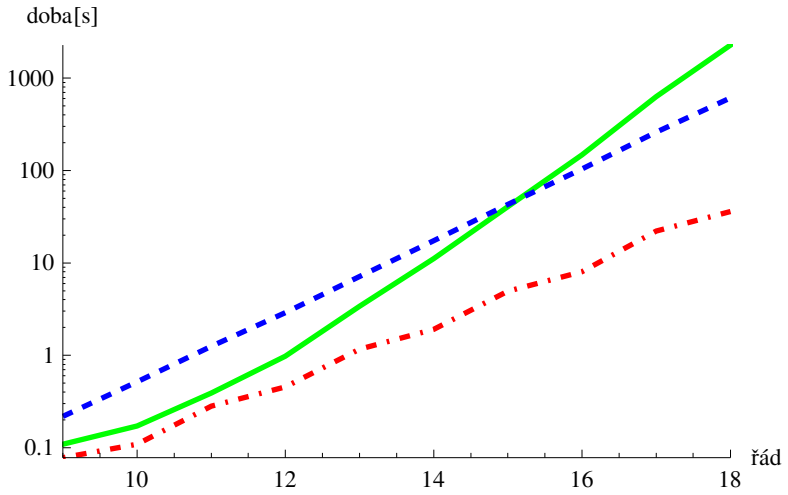
■ Graf 1 - Doba trvání výpočtů uvedených metod

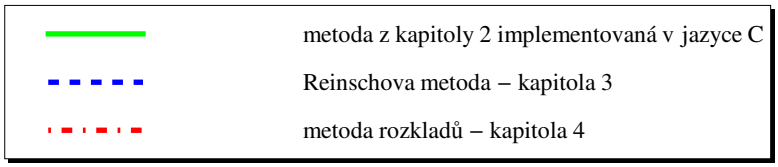


Z prvního grafu je patrné, že s narůstajícími řády je nejrychlejší metodou metoda rozkladu, následuje metoda Reinsche a nejpomalejší je metoda implementovaná v jazyce C.

■ Graf 2 - Doba trvání výpočtu uvedených metod - logaritmické měřítko

Stejný graf jako Graf 1 zobrazený v logaritmickém měřítku.

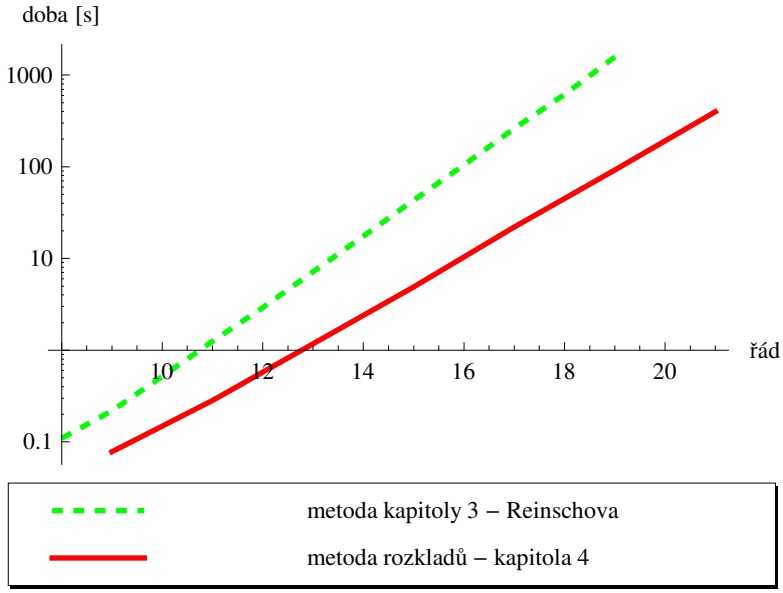




Z tohoto grafu jsou, při uvedených řádech, patrné zejména dvě skutečnosti. Za prvé, metoda rozkladů je přibližně 10x rychlejší než Reinschova metoda. Za druhé, metoda implementovaná v jazyce C je do 14.tého řádu včetně rychlejší než Reinschova metoda.

■ Graf 3 - srovnání Reinschovy metody a metody rozkladu

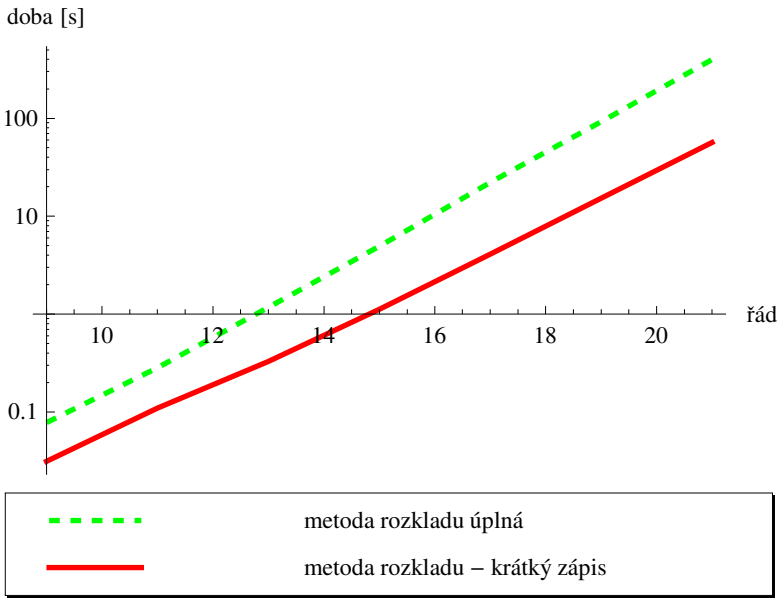
Jedná se o vyobrazení pouze lichých řádů, aby byla lépe vidět tendence grafů.



Z grafu je patrné, že doba potřebná k výpočtu roste v případě metody rozkladů pomaleji.

■ Graf 4 - srovnání zápisů metodou rozkladů - úplného a krátkého zápisu

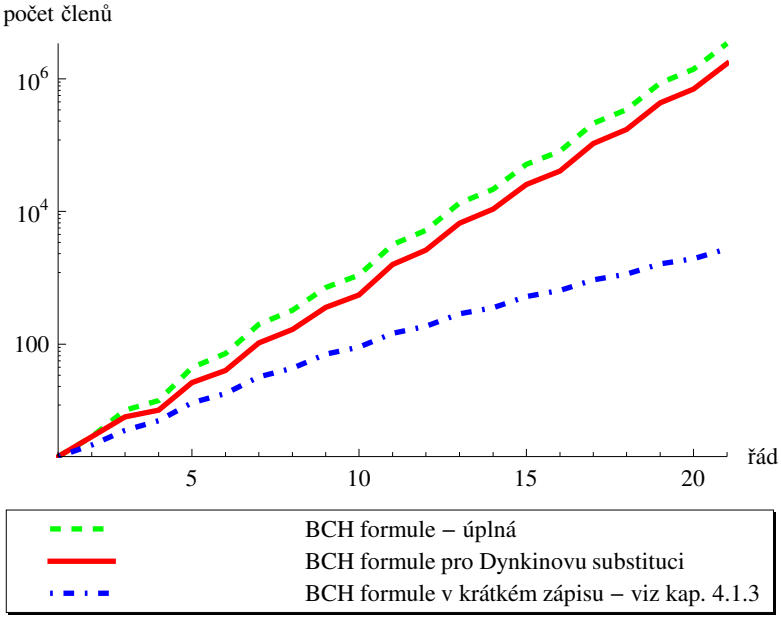
Jedná se opět o vyobrazení pouze lichých řádů, aby byla lépe vidět tendence grafů.



Patrná je úspora času v případě krátkého zápisu, přičemž tato úspora se s rostoucími řády ještě zvyšuje (procentuelně vzhledem k úplnému zápisu).

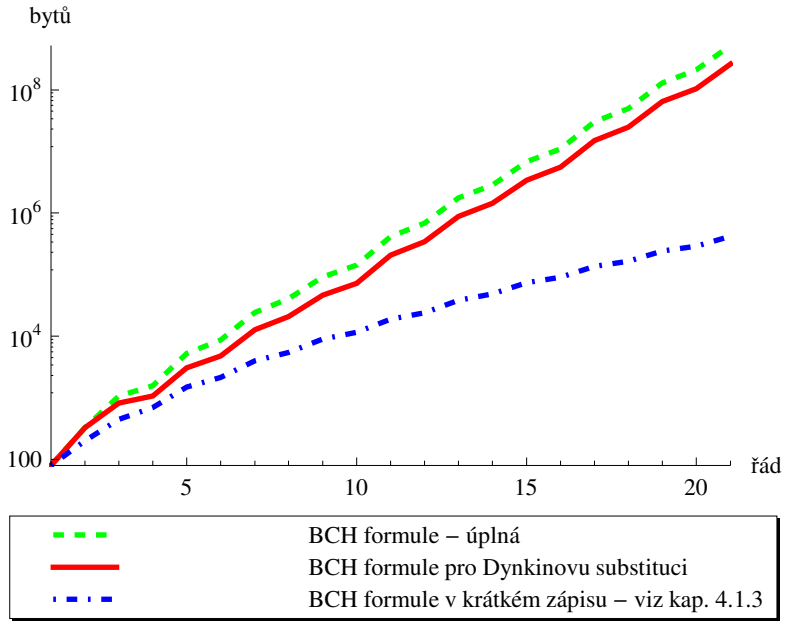
5.3 Srovnání délek různých zápisů formule

■ Graf 1 - Úplný a krátký zápis - počet členů zápisu



Z grafu je patrné, že zápis pro Dynkinovu substituci, umožní při stejném počtu členů vyčíslit o jeden řád více než úplný zápis. Krátký zápis je ještě mnohem úspornější a s rostoucí velikostí řádu je tato jeho vlastnost patrnější (počet členů úplného zápisu je v případě 20.tého řádu roven 1 392 166, zatímco krátkého zápisu pouze 1 934).

Graf 2 - Úplný a krátký zápis - paměťové nároky



Závěry plynoucí z tohoto grafu jsou totožné se závěry předchozího grafu. Zde si lze však konkrétně všimnout, že zatímco úplný zápis 20.tého řádu zabírá přes 100 MB dat, tak krátký zápis nespotřebuje ani 0,5 MB.

Závěr

V této práci jsem se zabýval metodami vyčíslení Baker-Campbell-Hausdorffovy formule. Ta nachází uplatnění v matematice a fyzice, zejména pak v kvantové mechanice. Ve své bakalářské práci jsem pomocí ní odvodil požadovaný tvar propagátoru v případě časově nezávislého kvantového harmonického oscilátoru. Dále jsem naprogramoval program, který formuli vyčísloval do určitého řádu a pomocí tohoto programu jsem mohl ověřit výsledky odvození s určitou přesností.

V tomto výzkumném úkolu jsem původní program zlepšil a v počítačovém algebraickém systému *Mathematica* jsem implementoval metodu získanou z literatury [3] a také svou vlastní metodu pro vyjádření formule. Tato má vlastní metoda vykazuje v provedených testech nejvyšší rychlost výpisu formule. Jejím použitím by bylo možné ověřit odvození z bakalářské práce s vyšší přesností.

Pro dosažení ještě vyšších řádů přesnosti by bylo možné metodu implementovat v nějakém nízkoúrovňovém jazyce a připadně ji paralelizovat.

Seznam použité literatury

[1] Yu.A.Bakhturin, Campbell - Hausdorff formula, SpringerLink Encyclopaedia of Mathematics, 2001, http://eom.springer.de/C/c020090.htm
[2] E. B. Dynkin, Mat. Sb. **25**, 155 (1949); Math. Rev **11**, 80 (1949)
[3] Matthias W.Reinsch : A simple expression for the terms in the Baker - Campbell - Hausdorff series, Department of Physics, University of California, Berkeley, 2006
[4] A. Bose: Dynkin's method of computing terms of the Baker-Campbell-Hausdorff series, J. Math. Phys. **30**, 2035 (1989)

Dodatky

Soubory odkazované v této práci a další materiály jsou umístěné na přiloženém CD.