Řešení vybraných úloh z teorie automatického řízení v prostředí Mathematica

Solution of chosen tasks from control theory in Mathematica

Lukáš Sedlák

Bakalářská práce
2008

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2007/2008

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lukáš SEDLÁK**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Řešení vybraných úloh z teorie automatického řízení v prostředí Mathematica**

Zásady pro vypracování:

1. **Seznamte se s prostředím Mathematica a stručně je popište spolu se základními funkcemi.**
2. **Zpracujte literární rešerši na vybrané úlohy, především řešení diofantických rovnic a spektrální faktorizaci.**
3. **Popište algoritmy, které jsou vhodné pro programovou implementaci daných úloh.**
4. **Vhodné algoritmy naprogramujte v prostředí Mathematica jako samostatné funkce.**
5. **Funkčnost algoritmů ověřte srovnáním s již vyřešenými úlohami.**
6. **Naprogramované funkce implementujte při řešení vhodné komplexní úlohy z oblasti teorie automatického řízení.**

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **DOSTÁL, Petr, GAZDOŠ, František, BOBÁL, Vladimír. Design of Controllers for Processes with Time Delay by Polynomial Method. In Proceedings of the European Control Conference. 2007th edition. Kos (Greece) : [s.n.], 2007. s. 4540-4545. ISBN 978-960-89028.**
2. **KUČERA, Vladimír. Diophantine equations in control - A survey. Automatica. 1993, vol. 29, no. 6, s. 1361-1375.**
3. **PROKOP, Roman, MATUŠů, Radek, PROKOPOVÁ, Zdenka. Teorie automatického řízení - lineární spojité dynamické systémy. Zlín : Univerzita Tomáše Bati ve Zlíně, 2006. 102 s. První vydání. ISBN 80-7318-369-2.**
4. **ŠEBEK, Michael. Algoritmy pro spektrální faktorizaci polynomů. In Využití polynomiálních metod v řízení technologických procesů : Seminární kurz ÚTIA ČSAV. Praha : [s.n.], 1988. s. 118-126.**
5. **Wikipedia : The Free Encyclopedia [online]. c2001 , 28 February 2006 [cit. 2008-01-25]. Text v angličtině. Dostupný z WWW:http://en.wikipedia.org/wiki/Diophantine.**
6. **Wolfram Mathematica Documentation Center [online]. 2008 [cit. 2008-01-25]. Text v angličtině. Dostupný z WWW:http://reference.wolfram.com/mathematica/guide/Mathematica.html**

Vedoucí bakalářské práce: **Ing. Libor Pekař**
Ústav pedagogických věd

Datum zadání bakalářské práce: **20. února 2008**

Termín odevzdání bakalářské práce: **5. května 2008**

Ve Zlíně dne 20. února 2008

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ Fakulta aplikované informatiky L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Tato práce se zabývá vybranými úlohami z teorie automatického řízení a jejich algoritmizací. Je zde popsáno řešení největšího společného dělitele dvou polynomů, řešení diofantických rovnic a spektrální faktorizace polynomu Newtonovou iterační metodou. V praktické části je využita implementovaná funkce faktorizace polynomu pro návrh regulátoru metodou systému se dvěma stupni volnosti (2DOF).

Klíčová slova: největší společný dělitel, diofantická rovnice, spektrální faktorizace, 2DOF teorie automatického řízení

ABSTRACT

This work deals with solution of chosen tasks from control theory and their algorithm development. There is described solution of greatest common divisor of two polynomials, solution of Diophantine equation and spectral factorization of polynomial via Newton iteration method. In the practical part there are used developed algorithms for synthesis of controller by Two-Degree-Of-Freedom (2DOF) system.

Keywords: Greatest common divisors, Diophantine equation, spectral factorization, 2DOF, control theory

Tímto bych chtěl poděkovat Ing. Liboru Pekařovi za jeho pomoc a vedení, které mi poskytoval po celou dobu , během níž jsem vyhotovoval tuto bakalářskou práci.

Prohlašuji, že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně ……

Podpis diplomanta

OBSAH

ÚVOD

Úvod

Teorie automatického řízení obsahuje velké množství komplikovaných úloh, které se dříve obtížně a zdlouhavě řešili pomocí ručních výpočtů. Z nástupem počítačů se řešení většiny těchto úloh začalo hledat právě pomocí výpočetní techniky. Byli to právě počítače, které, tak jako ostatně ve všech vědních oborech, usnadnili většinu složitých výpočtů a umožnili další zkoumání díky své rychlosti a přesnosti, s jakou dokázali úlohy řešit.

V počátcích využití počítačů pro řešení technických problémů neexistovaly komplexní software tak, jak je známe dnes. Byly to většinou jen knihovny, které nám umožňovaly řešit určitou úlohu konkrétního typu. Až postupným vývojem jak v oblasti software, tak v oblasti hardware, se na trhu objevily dokonalejší a rozsáhlejší programy, které uživateli nabízely celou řadu integrovaných funkcí a byly schopny řešit velkou škálu technických problémů v uživatelsky příjemném prostředí. Tyto programy jsou s každou novou verzí rychlejší, obsáhlejší a přesnější ve svých výpočtech. Namátkou je možné uvést programy MathWorks Matlab nebo Wolfram Research Mathematica.

S inženýrskými výpočty z oblasti teorie automatického řízení je ve velké míře spojován software Matlab od firmy MathWorks. V této práci byla právě naopak využita Matematica. Cílem této práce bylo ověřit její využití při řešení vybraných úloh z teorie automatického řízení. Mathematica je velmi silným nástrojem pro symbolické řešení úloh, což se z výhodou v této práci využilo.

Práce je rozdělena na dvě části, z nichž ta první je věnována popisu prostředí Mathematicy, jejich základních funkcí a dále pak teoretickému rozboru řešených úloh . Druhá část práce obsahuje popis praktického řešení těchto vybraných úloh a ověření funkčnosti tohoto řešení na příkladech.

I. TEORETICKÁ ČÁST

1 MATHEMATICA

1.1 Obecný úvod

Mathematica je software vyvíjený firmou Wolfram Research . Jeho první verze byla vydána v roce 1988 a podle mnohých představovala, ne-li začátek, tedy alespoň zlom na poli využití počítačů v inženýrské praxi. Tento software nalezl uplatnění ve většině vědních disciplín. Je také ve velké míře využívám v edukačním procesu.

Tento software dovoluje uživatelům řešit nepřeberné množství úloh. V primitivních případech může Mathematica být použita jako velmi komplexní kalkulátor. Pokročilejší uživatele v ní mohou řešit optimalizační úlohy, nejrůznější druhy rovnic, statistické výpočty, calculus, úlohy polynomiální algebry a jiné. Všechny tyto a ještě mnohé další problémy dokáže Mathematica řešit pomocí implementovaných funkcí. Pro ty, kteří nenajdou žádnou vhodnou funkci pro řešení jejich problému, je zde možnost naprogramovat si své vlastní funkce. Základní prvek Mathematicy tvoři atomy. Atom je v podstatě každý prvek. Ať už jde o číslo, znak ale i o operace, mluvíme o nich jako o atomech. Atomy jsou velice podobné objektům, které se využívají v objektově orientovaných programovacích jazycích. A tak podobně jako v objektově orientovaném programování, kde se vlastnosti jednotlivých objektů mohou dědit nebo používat sami na sebe, se u atomů využívá podobných principů.

Pro velkou část problémů a úloh obsahuje Mathematica implementované funkce, které jsou pro uživatele připraveny pro přímé využití. Podstatnou část problému však nelze řešit přímou aplikací implementovaných funkcí. Mathematica je ale v neposlední řadě velmi výkonným programovacím nástrojem, který dovoluje uživateli naprogramovat si své vlastní funkce, které může využít k řešení praktických úloh.

Jak je uvedeno v [7] Mathematica podporuje v zásadě tři přístupy k programování. Jednotlivé přístupy se liší interpretací jednotlivých příkazů a jejich vyhodnocováním.

Prvním přístupem může být přístup funkcionální. Tento styl můžeme definovat jako interpretaci uživatelských funkcí pomocí matematických funkcí. Tyto funkce mohou pracovat s libovolnými výrazy, popř. s dalšími funkcemi. Tento styl programování je proto tím, čím se Mathematica odlišuje od běžných programovacích jazyků.

Jako reprezentanta tohoto stylu programování můžeme uvést funkci *Map[]*.Tato funkce je hojně využívána a dovoluje uživateli značnou kontrolu nad dalšími použitými funkcemi.

Funkce *Map[]* se používá ve tvaru *Map[f, výraz, dimenze]*. Pomocí Map se aplikuje funkce *f*, která je v definici na prvním místě, na všechny prvky *výrazu*, v definici na druhém místě, na určitou *dimenzi* výrazu, třetí prvek definice.

```
In[1]:= Map[Reverse, {{a, b}, {c, d}, {e, f}}]
Out[1]= {{b, a}, {d, c}, {f, e}}
```

Obr. 1: Použití funkce Map

Při zápisu funkce *Map[]* je možné také použít zkráceného zápisu formou /@, jímž se docílí naprosto totožného výsledku.

```
In[2]:= Reverse /@ {{a, b}, {c, d}, {e, f}}
Out[2]= {{b, a}, {d, c}, {f, e}}
```

Obr. 2: Použití funkce Map – zkrácený zápis

Dalším přístupem k programování je procedurální přístup. Ten je shodný s procedurálním programováním v jiných jazycích, např. jazyk C. Uživatelem napsaný program obsahuje příkazy, podmíněné příkazy a smyčky. Kód programu je psán krok za krokem. Procedurální přístup je stále vhodný při řešení mnoha problémů, proto se využívá i dnes, přestože v dnešní době nastolují nové programovací jazyky jiné metody přístupu k programování.

Pro názornou ukázku procedurálního přístupu lze uvést jednoduchý cyklus *While*, který ve svém těle pouze inkrementuje proměnnou *i* a přičítá ji k proměnné *x* v každém průchodu cyklem. Výsledkem bude součet všech čísel od jedné do 1 do 10.

```
In[3]:= i = 0;
        x = 0;
        While[i < 10,
         i++;
         x += i;]
        Print["Soucet cisel od 1 do 10 je roven ", x]
        Soucet cisel od 1 do 10 je roven 55
```

Obr. 3: Ukázka procedurálního stylu programování

Posledním přístupem k programování je přístup rekurzivní. Mnoho problémů v inženýrské praxi vyžaduje rekurzivní přístup. Funkce je definována rekurzivně, pokud ve své definici volá sama sebe. V Mathematice je rekurze jednoduše definovatelná a snadno využitelná. Dokonce podstatná část implementovaných funkcí využívá rekurze.

Příkladem použití rekurze je možné ukázat na kódu, který počítá nuly v proměnné *L*, která je argumentem uživatelem definované funkce *pocetNul.* Na začátku je funkce *pocetNul* nastavena na hodnotu 0, protože se má vracet počet nul z vektoru *L.* Vše probíhá tak, že se otestuje zda je prvek *L* nulový, pokud ano, přičte se k výsledku jednička a pokračuje se v aplikaci funkce na zbylé prvky proměnné *L.* Pokud není prvek nulový, nepřičte se k výsledku nic a opět se pokračuje v aplikaci na další prvky proměnné *L.*

```
In[22]:= pocetNul[{}] := 0
         pocetNul[L_List] :=
          If[First[L] === 0, 1 + pocetNul[Rest[L]], pocetNul[Rest[L]]]

In[24]:= pocetNul[{0, 0, 1, 0, 5, 6, 7, 5, 0}]

Out[24]= 4
```

Obr. 4: Ukázka rekurzivního stylu programování

V neposlední řadě se v dnešní době Mathematica začíná velmi uplatňovat ve výuce, kde se naplno uplatňují její přednosti v podobě velmi dobré práce s grafickými výstupy řešených úloh, názornost systému využívání notebooků, které dovolují vytvářet interaktivní studijní materiály, přenositelnost mezi různými platformami, vytváření webových aplikací, které mohou sloužit jako e - lerningové pomůcky.

1.2 Funkce v Mathematice a jejich definování

1.2.1 Okamžité a odložené přiřazení funkce

Mathematica dle [7] dovoluje uživateli definovat funkci dvěma způsoby. Prvním způsobem je okamžité přiřazení. Pro toto přiřazení se používá operátor = . Toto definování funkce je možné interpretovat jako okamžité přiřazení výrazu na pravé straně do proměnné na levé straně. Toto je možné demonstrovat na jednoduchém případu

```
In[39]:= C1 = Random[]
Out[39]= 0.648718
In[41]:= Table[C1, {4}]
Out[41]= {0.648718, 0.648718, 0.648718, 0.648718}
```

Obr. 5: Ukázka použití okamžitého přiřazení

Zde je jasně vidět, že pomocí funkce *Random[]* je do proměnné *C1* uloženo náhodné číslo z intervalu <0,1>. Toto náhodné číslo se nemění nezávisle na tom, kolikrát danou proměnnou pomocí funkce *Table[]* umístím do vektoru.

Naproti tomu odložené přiřazení, používá se pro něj operátor := , pouze do speciální proměnné uloží pravidla, podle kterých se bude výraz na pravé straně vyhodnocovat. Toho je možné využít v případech, kdy potřebujeme definovat funkci po částech, nebo když proměnná na levé straně závisí na parametrech na pravé straně. Toto přiřazení se nazývá odložené. Je možné ho demonstrovat na velmi podobném příkladu jako u okamžitého přiřazení.

```
In[43]:= C2 := Random[]
In[44]:= Table[C2, {4}]
Out[44]= {0.774196, 0.449372, 0.752888, 0.88094}
```

Obr. 6: Ukázka použití odloženého přiřazení

Na tomto ilustrativním příkladu je naopak vidět, že čísla která dostaneme po vykonání funkce *Table[]* nejsou stejná jako v předchozím případě. Je to způsobeno tím, že při definici proměnné *C2* bylo použito odložené přiřazení. To znamená, že bylo proměnné *C2* přiřazeno pouze pravidlo, které říká, jak se má tato proměnná reprezentovat. Při použití funkce *Table[]* se tedy při každé iteraci vygenerovalo nové náhodné číslo, protože takovým pravidlem byla proměnná *C2* definována.

1.2.2 Funkce s vícenásobnou definicí

Při definici funkce můžeme použít jedno jméno pro různé definice, které mají různý počet parametrů a systém je přesto schopen je od sebe odlišit. Předpokládá to však použití odloženého přiřazení.

Podtržítko, které se ve funkci používá říká, že argumentem funkce může být cokoliv. *x_* znamená, že jako argument funkce *f* můžeme použít jeden objekt, který není typově specifikován. Může to tedy být celé číslo, reální číslo, výraz nebo třeba proměnná.[7]

```
In[51]:= f[x_] := (z + 0.5 + 1) * x

In[52]:= f[8]

Out[52]= 8 (1.5 + z)
```

Obr. 7: Funkce s vícenásobnou definicí

Tato vlastnost je velmi výhodná pro definování různých funkcí, u kterých bude výsledek záležet na vstupních parametrech. Proto není nutné definovat každou jednu funkci pro náš požadovaný výpočet. Je daleko vhodnější využít funkce s vícenásobnou definicí, kde se nám výsledek bude lišit na základě námi zadaných parametrů, ale programově půjde stále o jedinou konkrétní definici pravidel v paměti, podle kterých se bude výsledek vyhodnocovat.

2 NEJVĚTŠÍ SPOLEČNÝ DĚLITEL DVOU POLYNOMŮ

Největší společný dělitel(*GCD*) dvou polynomů je takový polynom, který dělí dva polynomy rovnoměrně. *GCD* dvou polynomů se definuje podobně jako je tomu u hledání *GCD* dvou celých čísel *A*, *B*. Tj., pokud uvažujeme celá čísla *A* a *B*, největší celé číslo, které dělí *A*,*B* beze zbytku. U polynomů je ovšem situace komplikovanější, protože nejsme schopni rozhodnout který polynom je „největší“. Proto se hledá polynom nejvyššího možného stupně, který dělí oba polynomy rovnoměrně. Někdy je také *GCD* dvou polynomů označován jako největší společný faktor polynomů.

2.1 Definice

Mějme polynomy $a(x), b(x)$; oba nenulové, s koeficienty z množiny *M*. Pak *GCD* polynomu a*(x), b(x)* je normovaný polynom *d(x)* nejvyššího možného stupně, který dělí zároveň polynom *a(x)* i polynom *b(x).* Je možné zapisovat *d(x)=(a(x), b(x))*, nebo *GCD(a(x),b(x)).*

Množina *M* může být podmnožinou oboru reálných, celých nebo komplexních čísel.

Je – li polynom *a(x) = b(x) = 0*, pak každý polynom je společným dělitelem polynomu *a(x), b(x).* Největší společný dělitel ale v tomto případě neexistuje.

Polynomy *a(x), b(x)* musí splňovat nutné podmínky: *M* musí být těleso a polynom *d(x)* musí být normovaný. Pokud jsou tyto podmínky splněny, největší společný dělitel existuje, a je definován jednoznačně.

Pokud by polynom d*(x)* nebyl normovaný a *M* by nebylo těleso, největší společný dělitel polynomů *a(x), b(x)* by nebyl určen jednoznačně, popř. by jej nebylo možné určit, jelikož axiom dělení je definován pouze v tělese.

Pro polynomy *a(x), b(x)* představuje konstanta 1 vždy společného dělitele. Můžeme ji považovat za normovaný polynom nultého řádu. Jestliže *GCD(a(x),b(x)) = 1*, pak je možné říci, že polynomy *a(x) a b(x)* jsou nesoudělné.

2.2 Vlastnosti GCD polynomů *a(x),b(x)*

- Je – li *a(x),b(x)* různé od nuly a zároveň koeficienty náleží tělesu *M*, pak existuje jednoznačně definovatelný polynom *d(x)*, který je největším společným dělitelem polynomů *a(x),b(x)*.
- Jestliže je *c(x)* společným dělitelem *a(x)* a *b(x)*, pak *c(x)* dělí *d(x)*. Toto někdy bývá zaměňováno s formulací definice uvedené výše, kde je vyžadován jako *GCD(a(x),b(x))* polynom nejvyššího stupně. Tyto dvě formulace jsou logicky ekvivalentní.
- $GCD(a(x), b(x)) = GCD(b(x), a(x))$
- $GCD(a(x), b(x)) = GCD(a(x), a(x) + b(x))$
- Pro jakýkoliv nenulový skalár z tělesa *M* platí:

  $GCD(a(x), b(x)) = GCD(a(x), a(x) + k \cdot b(x))$

2.3 Metody určení GCD dvou polynomů

Existuje několik metod, jak získat největší společný dělitel polynomů *a(x), b(x).*

Jedná se o tyto metody:

- Pomocí rozkladu na prvočinitele
- Pomocí Euklidova algoritmu
- Pomocí zobecněného Euklidova algoritmu.

Všechny tyto metody vedou k nalezení největšího společného dělitele. Avšak ne všechny tyto metody jsou stejně dobře použitelné pro algoritmizaci. V této práci byl využit Euklidův algoritmus. Podrobněji o něm dále v kapitole 5.1 .

3 DIOFANTICKÉ ROVNICE

Podle[3] se rovnice ve tvaru:

$$ax + by = c \tag{1}$$

nazývá diofantická a je definovaná v množině, která se nazývá okruhem. V rovnici (1) představují *a,b,c* známé a *x,y* neznámé, hledané prvky z daného okruhu. Diofantické rovnice dostali své jméno podle Diofanta(kolem roku 240), který podstatně zjednodušil řecké zapisování číslic a svými pracemi vybudoval základy algebraického zápisu. Mimo jiné se zajímal o řešení rovnic v okruhu celých čísel ve tvaru:

$4x + 6y = 10$ $\qquad\qquad$ $4x + 6y = 10$

nekonečně mnoho řešení: $\qquad\qquad$ nemá řešení

$x = 1 + 3t$

$y = 1 - 2t$

Diofantická rovnice (1) má řešení tehdy a jen tehdy, jestliže *GCD(a,b)* dělí *c*. V tom případě lze bez újmy na obecnosti uvažovat rovnici (1) s nesoudělnými *a* a *b*. Jak již bylo naznačeno, diofantická rovnice má nekonečně mnoho řešení, která jsou dána:

$$x = x_0 + bt \tag{2}$$

$$y = y_0 - at$$

kde x_0, y_0 tvoří partikulární řešení a t je libovolný prvek z daného okruhu.

Obecně lze partikulární řešení nalézt s využitím zobecněného Euklidova algoritmu.

Pokud se uvažuje řešení rovnice (1) v okruhu polynomů(s předpokladem nesoudělnosti polynomů *a* a *b*), pak při splnění podmínek patřičných stupňů polynomů lze nalézt partikulární řešení např. pomocí metody neurčitých koeficientů.

3.1 Řešení diofantických rovnic metodou neurčitých koeficientů

Tato metoda umožňuje pouze určení partikulárního řešení. Porovnáním koeficientů u příslušných mocnin z^{-i} se sestaví soustava lineárních rovnic. Jejich vyřešením se získají hledané koeficienty diofantické rovnice. Při sestavování diofantické rovnice je nutné dbát

podmínek pro její sestavení. Tj. určení stupňů hledaných koeficientů. Ve většině případů postačuje postupovat podle následujících kritérií:

$$\delta a + \delta b > \delta c \qquad\qquad \delta a + \delta b \le \delta c$$

a) $\delta x = \delta b - 1$ $\qquad\qquad$ b) $\delta x = \delta b - 1$ nebo $\delta x = \delta c - \delta b$

$\delta y = \delta a - 1$ $\qquad\qquad$ $\delta y = \delta c - \delta b$ $\qquad$ $\delta y = \delta a - 1$

Výsledná diofantická rovnice tedy bude vypadat následovně:

$$a(z) \cdot x(z) + b(z) \cdot y(z) = c(z) \tag{3}$$

3.1.1 Příklad

Je dána rovnice $(1 - z^{-1} + z^{-2})x + (1 - z^{-1})y = 1 + z^{-1}$

Určení stupňů polynomů *x,y*

$$\begin{aligned} &\delta a + \delta b > \delta c \\ &\delta x = \delta b - 1 = 1 - 1 = 0 \\ &\delta y = \delta a - 1 = 2 - 1 = 1 \end{aligned}$$

Výsledný tvar řešené diofantické rovnice

$$(1 - z^{-1} + z^{-2})x_0 + (1 - z^{-1}) \cdot (y_0 + y_1 z^{-1}) = 1 + z^{-1}$$

Roznásobením a srovnáním koeficientů u záporných mocnin se získá soustava lineárních rovnic

$$x_0 - x_0 z^{-1} + x_0 z^{-2} + y_0 + y_1 z^{-1} + y_0 z^{-1} - y_1 z^{-2} = 1 + z^{-1}$$

$$\begin{aligned} &z^0 : x_0 + y_0 = 1 && \rightarrow \quad x_0 = 1 - y_0 \quad \rightarrow y_0 = -1 \\ &z^{-1} : -x_0 - y_0 + y_1 = 1 && \rightarrow -1 + y_0 - y_0 + y_1 = 1 \\ & && \qquad y_1 = 2 \\ &z^{-2} : x_0 - y_1 = 0 && \rightarrow x_0 - 2 = 0 \\ & && \qquad x_0 = 2 \end{aligned}$$

Hledané partikulární řešení je jen jediné, protože polynomy *a,b* jsou nesoudělné a má tedy tvar: $x_p = 5$ $\qquad$ $y_p = -1 + 2z^{-1}$

3.2 Řešení diofantických rovnic s využitím největšího společného dělitele dvou polynomů

K polynomům a,b najdeme jejich největší společný dělitel $d = GCD(a,b)$, některou z metod, které jsou popsány výše(např. pomocí zobecněný Euklidův algoritmus). Dále je nutné určit polynomy p, q, r, s takové že platí

$$ap + bq = d \tag{4}$$

$$ar + bs = 0 \tag{5}$$

Pro partikulární řešení má rovnice tvar

$$ax_p + by_p = c \tag{6}$$

Rovnici(6) vynásobíme výrazem $\frac{d}{c}$, rovnici je poté ve tvaru

$$ax_p \frac{d}{c} + by_p \frac{d}{c} = d \tag{7}$$

Porovnáním a úpravou se získá

$$xp = \frac{c}{d} p \tag{8}$$

$$yp = \frac{c}{d} q \tag{9}$$

Obecná zkrácená rovnice bude ve tvaru

$$ax + by = 0 \tag{10}$$

Řešením rovnice(10)

$$\frac{a}{b} = -\frac{y}{x} = \frac{a_0 d}{b_0 d} \Rightarrow y = -a_0, x = b_0 \tag{11}$$

$$ab_0 - ba_0 = 0 \rightarrow (ar + bs = 0)$$

a_0, b_0 jsou nesoudělné polynomy

Opětovným porovnáním a dosazením se získá

$$r = b0 = \frac{b}{(a,b)} \tag{12}$$

$$s = -a0 = -\frac{a}{(a,b)} \tag{13}$$

Konečně obecné řešení rovnice bude ve tvaru

$$x = x_p + \frac{b}{(a,b)} = p\frac{c}{d} + \frac{b}{(a,b)}t \tag{14}$$

$$y = y_p - \frac{a}{(a,b)} = q\frac{c}{d} - \frac{a}{(a,b)}t \tag{15}$$

kde t je libovolný polynom. Diofantická rovnice má řešení pouze tehdy, když c/d bude polynomem, tzn. pokud d\c.

3.2.1 Příklad

Je dána rovnice $(2 - z^{-1} - 2z^{-2} + z^{-3})x + (-2z^{-1} + z^{-2})y = -2 + z^{-1} + 4z^{-3} - 4z^{-4} + z^{-5}$

Pomocí Euklidova algoritmu určíme největší společný dělitel polynomů *a,b* a další parametry *p, q, r, s.*

$$Q = \begin{bmatrix} p & q \\ r & s \end{bmatrix} \qquad L = \begin{bmatrix} 2 - z^{-1} - 2z^{-2} + z^{-3} \\ -2z^{-1} + z^{-2} \end{bmatrix}$$

Po aplikaci Euklidova algoritmu popsaného v 5.1 je možné nalézt toto řešení

$$Q = \begin{bmatrix} 1 & -z^{-1} \\ z^{-1} & 1 - z^{-1} \end{bmatrix} \qquad L = \begin{bmatrix} 2 - z - 1 \\ 0 \end{bmatrix}$$

Je nutné zjistit zda je splněna podmínka řešitelnosti diofantické rovnice, tj.největší společný dělitel polynomů *a* a *b* musí dělit bezezbytku polynom *c*

$$\frac{c}{d} = \frac{-2 + z^{-1} + 4z^{-3} - 4z^{-4} + z^{-5}}{2 - z^{-1}} = -1 + 2z^{-3} - z^{-4} \quad \rightarrow c \setminus d \text{ bezezbytku}$$

Obecné řešení je potom ve tvaru

$$x = p\frac{c}{d} + rt = 1(-1 + 2z^{-3} - z^{-4}) + z^{-1} \cdot t = -1 + 2z^{-3} - z^{-4} + z^{-1} \cdot t$$

$$y = q\frac{c}{d} + st = -z^{-1}(-1 + 2z^{-3} - z^{-4}) + 1 - z^{-1} \cdot t = z^{-1} - 2z^{-4} + z^{-5} + 1 - z^{-4} \cdot t$$

4 SPEKTRÁLNÍ FAKTORIZACE POLYNOMU

4.1 Úvod

Matematická metoda spektrální faktorizace byla objevena Wienerem ve čtyřicátých letech při řešení problémů optimální filtrace. Od té doby nalezla uplatnění v mnoha oborech elektrotechniky. Vedle teorie řízení, kde se pomocí spektrální faktorizace řeší například úlohy optimální rekonstrukce stavu a syntézy LQ – optimálních regulátorů, se tato metoda využívá hlavně v teorii obvodů.[4]

4.2 Formulace

Spojitá verze spektrální faktorizace se formuluje:

K polynomu *B(s)* z R[s], který je

1. Symetrický, tj. $B(s) = B(-s)$ a

2. Pozitivní, tj. $B(s) > 0$ pro $Re\ s = 0$

najdeme stabilní polynom *A(s)* z *R[s]*(tj. *A(s)* je různé od nuly pro Re $\geq$ 0) tak, aby

$$B(s) = A(s)A(-s)$$

Jestliže zavedeme k polynomu A sdružený polynom A_*, dáno vztahem $A_*(s) = A(-s)$, můžeme rovni zapsat jako

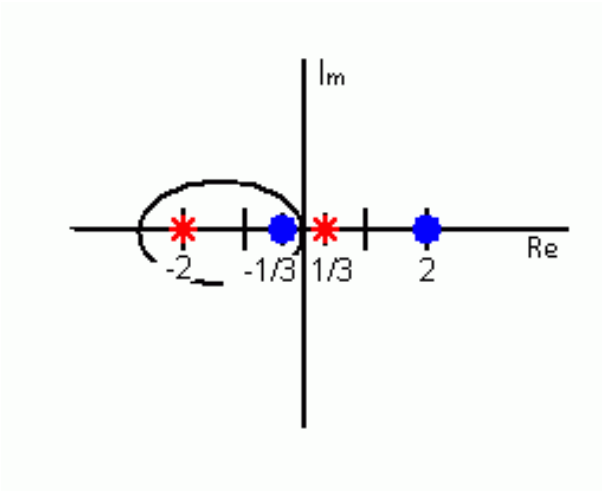
$$B(s) = A \cdot A_*$$

Polynom A se nazývá spektrální faktor. Je určen rovnicí jednoznačně až na znaménko.

4.3 Geometrická interpretace spektrální faktorizace pro spojité systémy

Pro názornou ilustraci geometrické interpretace je možné využít jednoduchého polynomu nízkého stupně, na kterém je přesto zřetelný význam spektrální faktorizace.

Je dán polynom $C(s) = (2+s)(1-3s)$, nuly tohoto polynomu jsou -2 a $\frac{1}{3}$. Tyto nuly jsou vyneseny do komplexní roviny d jako křížky.



Obr. 8: Vynesení nul polynomu $C(s)$ a sdruženého polynomu $C_*(s)$

Dále pak nuly sdruženého polynomu $C_*(s) = (2-s)(1+3s)$, tj. 2 a $-\frac{1}{3}$, jsou na obrázku vyneseny jako kolečka. Nuly symetrického polynomu $B(s)$ jsou tedy všechny nuly na obrázku 8. Proces spektrální faktorizace je založen na tom, že se nuly rozdělí na stabilní a nestabilní. Jelikož u spojitých systémů je stabilní část v levé části komplexní roviny je zřejmé že polynom $C(s)$ má právě tyto nuly: -2 a $-\frac{1}{3}$. Zbylé nuly náleží polynomu $C_*(s)$.

Pak tedy polynom $C(s) = (2+s)(1+3s)$

4.4 Metody spektrální faktorizace

Jak bylo zmíněno výše, spektrální faktorizace hrála dosti zásadní roli v celé řadě vědních oborů. Speciálně v teorii automatického řízení hraje důležitou roli v oblasti syntézy LQ – regulátorů a odhadech robustnosti a řiditelnosti systémů.

Bylo vyvinuto velké množství algoritmů, které se zaměřují na řešení problému spektrální faktorizace. Jako příklad je možné uvést několik známých metod; např. Bauerova metoda, Levinson – Durbin metoda, Shurova metoda a metody využívající Riccatiho rekurze.[8] Další metodou pro nalezení spektrálního faktoru polynomu je Newtonova iterační metoda, která je použita v této prácí.

4.4.1 Bauerova metoda spektrální faktorizace

Tato metoda pro faktorizace diskrétních systémů P(z) je založena na aproximaci koeficientů P(z) Choleskiho trojúhelníkovým rozkladem pozitivně definitní Toeplitzovi matice se vzrůstajícími koeficienty.

4.4.2 Spektrální faktorizace s využitím Riccatiho rekurze

Při aplikaci tohoto postupu se využívá pro získání spektrálního faktoru polynomu řešení asociované diskrétní algebraické Riccatiho rovnice. Tato metoda se ukázalo jako velmi efektivní při řešení problému spektrální faktorizace.

4.4.3 Newtonova iterační metoda spektrální faktorizace

Stejně jako u předešlých metod se jedná o iterační metodu. Jejím základem je řešení kvadratické rovnice v okruhu polynomů. Podrobněji o této metodě bude zmíněno v 5. 3

5 POPIS ALGORITMŮ VYBRANÝCH ÚLOH

5.1 Algoritmus nalezení největšího společného dělitele dvou polynomů

GCD polynomů *A, B* je možné najít pomocí Euklidova algoritmu. Pokud jsou polynomy *A* a *B* ve tvaru kořenových činitelů, je možné GCD určit jako součin všech společných kořenových činitelů. Jestliže polynomy nejsou ve tvaru kořenových činitelů, je možné určit *GCD* pomocí tohoto algoritmu:

1. Definujeme matice *Q* a *L* pomocí polynomů *p, q, r, s*, *a, b*. Matice mají tento tvar:

$$Q = \begin{bmatrix} p & q \\ r & s \end{bmatrix} \qquad L = \begin{bmatrix} a \\ b \end{bmatrix}$$

2. Inicializujeme hodnoty $p=1$, $q=0$, $r=1$, $s=0$. Matice *Q* bude v tomto tvaru:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

3. Určí se nenulový polynom menšího stupně v matici *L*. Jsou – li oba polynomy nulové, algoritmus končí.
4. Je – li polynom nižšího stupně v druhém řádku matice *L*, zaměníme řádky v matici *L* a *Q*.
5. Je – li polynom v druhém řádku matice *L* nulový, algoritmus končí.
6. Určí se číslo λ jako podíl koeficientů u nejvyšších mocnin z^{-i}, a číslo ρ jako rozdíl stupňů polynomů v druhém a prvním řádku matice *L*.
7. Odečteme první řádek matice *L* vynásobený $\lambda \cdot z^{-\rho}$ od druhého řádku matice *L*.
8. Opakuje se postup od bodu 2.

Po ukončení algoritmu se nachází *GCD* polynomů *A, B* na prvním řádku matice *L*. V druhém řádku se nachází nula. Čistě pro určení GCD dvou polynomů není třeba počítat polynomy *p, q, r, s*. Tyto polynomy však hrají roli při výpočtu obecného řešení diofantické rovnice, jak bylo zmíněno výše v kapitole 3. 2

5.2 Algoritmus pro řešení diofantických rovnic metodou neurčitých koeficientů

Při řešení diofantických rovnic metodou neurčitých koeficientů je možné rozdělit postup na dvě části. První část programu vyhodnocuje stupně polynomů *a, b, c* a generuje podle příslušných pravidel tvar neznámých polynomů *x* a *y*. Druhá část programu potom počítá soustavu lineárních rovnic, které vzniknou srovnáním koeficientů u příslušných mocnin z^{-i}.

Algoritmus:

1. Načtení polynomů *a, b , c.*
2. Určení stupňů těchto polynomů.
3. Podle pravidel pro určení tvaru neznámých polynomů *x, y* vygenerování těchto polynomů.
4. Roznásobení diofantické rovnice do tvaru, kdy je možné porovnávat koeficienty u mocnin z^{-i}.
5. Srovnáním podle příslušných mocnin z^{-i} se do vektoru *resitel* uloží jednotlivé lineární rovnice.
6. Pomocí funkce *Solve[]* se vyřeší soustava lineárních rovnic
7. Pomocí dvou cyklů *While* se sestaví hledané partikulární řešení diofantické rovnice
8. Řešení získaná v bodě 7 se uloží do proměnné *xp* a *yp*

Po provedení tohoto algoritmu obsahují proměnné *xp a yp* partikulární řešení diofantické rovnice

5.3 Newtonova iterační metoda pro spektrální faktorizaci polynomu

5.3.1 Obecný popis algoritmu

Protože spektrální faktorizace je možné definovat jako řešení kvadratické rovnice v okruhu polynomů, celý algoritmus se v podstatě zabývám nalezením „nulového bodu“ funkce v okruhu polynomů

$$f(A) = AA_* - B \tag{16}$$

Pro funkci (16) je to tedy

$$dif\ f(A_n) = (A_n + 1 - A_n)A_{n*} + A_n(A_{n+1*} - A_{n*}) \tag{17}$$

Po úpravě má tvar

$$A_{n+1}A_{n*} + A_nA_{n+1*} - A_nA_{n*} = B$$

Substitucí tohoto typu

$$A_{n+1} = \frac{1}{2}(A_n + X_n) \tag{18}$$

Dostaneme konečně symetrickou polynomiální rovnici

$$A_{n*}X_n + A_nX_{n*} = 2B \tag{19}$$

Z toho tedy vyplývá, že jeden iterační krok vypadá takto:

k známému A_n se vypočte řešením rovnice (19) X_n. Toto X_n se poté dosadí do (18), tímto se získá nové A_{n+1}

Tato metoda má tyto vlastnosti:

- Zachovává stabilitu, tj. pro stabilní A_n vychází i A_{n+1} stabilní
- Konverguje monotónně a kvadraticky,

5.3.2 Nastartování výpočtu

Jako počáteční hodnotu pro nastartování algoritmu je možné použít libovolný stabilní polynom A_0. Obvykle se volí tak, aby se shodoval s výsledným A alespoň v jednom koeficientu přesně. Pro spojitou verzi volíme

$$A_0 = (\sqrt[p]{b_0} + \sqrt[p]{b_p})^{\frac{p}{2}}$$

kde $p = stupeň\ B$, b_0 a b_p jsou absolutní a vedoucí koeficienty B

5.3.3 Ukončení vypočtu

Iterační proces se zastaví, jestliže se dvě po sobě jdoucí vypočítané A_n liší méně než o určitou zvolenou hodnotu, neboli jakmile přestane být A_n vlivem numerických chyb stabilní. V některých případech pro využití u adaptivního řízení je možné provést jen předem stanovený počet kroků.

5.3.4 Programová implementace

Pro výpočet spektrální faktorizace je nutné mít funkci, která dokáže určit sdružený polynom k polynomu *A*. Dále je nutné mít funkci pro řešení symetrické polynomiální rovnice a nakonec hlavní smyčku, kde se v cyklu vždy počítá A_{n+1} pomocí (18).

5.3.4.1 Faktorizace polynomu

Tato funkce je naprogramována jako funkce *faktor[p_]*. Na vstup funkce se zadává polynom který má být faktorizován. Faktorizace probíhá pomocí tohoto algoritmu:

1. Načtení polynomu *A*
2. Separování mocnin a koeficientů příslušných mocnin polynomu *A*
3. Vytvoření vektoru *inver,* který se vytvoří tak, že každý sudý člen je 1, každý lichý člen je -1.

4. Vytvoření vektoru *vektorsdruzeny,* který je vytvořen jako součin *i*-tého koeficientu s na i-tou mocninu. Tím se získá vektor který obsahuje členy polynomu. Má stejnou délku jako *inver*
5. Vynásobení vektoru *inver* a *vektorsdruzeny.* Výsledkem je vektor který obsahuje členy již sdruženého polynomu *A*
6. Pomocí cyklu *While* se znovu vystaví polynom *A*, ale již sdružený, tedy A_*

5.3.4.2 Získání X_n pomocí řešení symetrické polynomiální rovnice

Symetrická polynomiální rovnice je řešena pomocí metody neurčitých koeficientů. Tato rovnice vlastně představuje diofantickou rovnici ve tvaru

$$A(s)X(-s) + A(-s)X(s) = 2B$$

kde A(-s) je sdružený polynom k A(s)

X(-s) je sdružený polynom k X(s)

Algoritmus:

1. Polynomy *A(s), A(-s), B* se zadávají jako parametry funkce *resenispr[]*. K získání *A(-s)* se použije funkce *faktor[],* popsaná v kapitole 5.3.4.1
2. Určí se stupeň polynomu *A(s)*
3. Pomocí cyklu While se vygenerují polynomy X(s) a X(-s). Tyto polynomy se generují stejného stupně jako má polynom A(s) a logicky A(-s). Zároveň se generuje vektor neznámých *x*, tj. *neznamex*, a vektor neznámých *y*, tj. neznameyps. Tyto vektory se využívájí jako parametr funkce *Solve[]*, která řeší soustavu lineárních rovnic, které vzniknou roznásobením rovnice (19)
4. Roznásobení rovnice (19) a její uložení do proměnné *rovnice*
5. Pomocí funkce *CoefficientList[],* se vytáhnou koeficienty od příslušných mocnin *s* a uloží se do proměnné *casti.* Jednotlivé prvky této proměnné reprezentují lineární rovnice, které je pro nalezení řešení rovnice (19) nutné vyřešit.
6. V cyklu While se rovnice, které vznikly v bodě 5. převedou do korektního tvaru, tj. aby funkce *Solve[]* byla schopná tyto rovnice řešit. Tzn. každý prvek proměnné

casti se pomocí operátoru = = položí roven 0. Což je korektní forma pro zápis rovnice která se má řešit pomocí funkce *Solve[]*

7. Vyřešení soustavy rovnic pomocí funkce *Solve[]* a uložení výsledků do proměnné výsledek.
8. Protože funkce *Solve[]* obecně vrací výsledky ve tvaru *(x_n -> q),* je nutné pomocí cyklu *While* získat pouze číselné hodnoty, které je možné dále použít pro rekonstrukci polynomu $X_n(s)$. Podmíněným výrazem *If* se vyhodnotí, zda se vyrušily liché části polynomu $X_n(s)$. To se děje tak, že se porovná A(s) s A(-s), jestliže jsou totožné, pak se vyruší liché části polynomu $X_n(s)$. Jestli ano, potom se polynom $X_n(s)$ rekonstruuje tak, že iterátory *j* a *k* se v každém kroku zvětšují o hodnotu 2. To zajišťuje, že se polynom $X_n(s)$ vytvoří pouze se sudými mocninami *s*. Pokud se ale A(s) nerovná A(-s), pak se iterátory *j* a *k* zvyšují pouze o hodnotu 1. Polynom $X_n(s)$ se potom vytvoří i s lichými mocninami *s*. Výsledek celé operace se uloží do proměnné *hledanex*.
9. Funkce *Return[]* vrací hodnotu proměnné *hledanex.*

5.3.4.3 Vnější cyklus Newtonovi iterační metody

Celé řešení jak bylo zmíněno v kapitole 5.3.1 se skládá z nalezení X_n, které se dosadí do rovnice (18) a spočítá se nové A_n. Z toho vyplývá i algoritmus řešení

1. Máme polynom *p*, který chceme spektrálně faktorizovat.
2. Vytvoříme sdružený polynom *psdruz* s polynomu *p* pomocí funkce *faktor[]*
3. Vynásobíme *p*psdruz,* dostaneme polynom *B*.
4. Z polynomu *B* vytvoříme podle vztahu uvedeného v kapitole 5.3.2 startovací polynom *pstart.*
5. Opět faktorizujeme *pstart*, dostaneme *pstartsdruz*. Tím máme všechny potřebné parametry pro řešení X_n, které se řeší pomocí funkce *resenispr[]*.
6. V cyklu While se vyřeší rovnice(19), dosadí se do vztahu pro výpočet A_{n+1}, tj. do rovnice(18). Nové A_n se faktorizuje a tím se nachystají všechny proměnné potřebné pro další otočku cyklu.

II. PRAKTICKÁ ČÁST

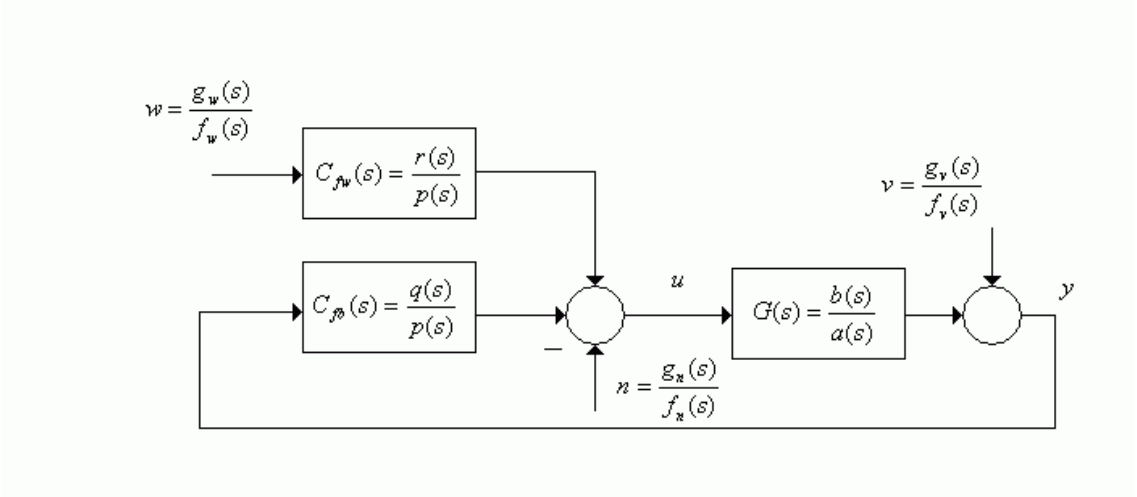
6 ŘEŠENÍ KOMPLEXNÍ ÚLOHY Z TEORIE AUTOMATICKÉHO ŘÍZENÍ

Jako komplexní úlohu řešenou s využitím naprogramovaných funkcí jsem zvolil návrh 2DOF regulátoru pro regulovanou soustavu druhého řádu, do které vstupuje porucha. Pro řešení této úlohy jsou využil funkci pro faktorizaci polynomu pomocí Newtonovi iterační metody, kterou jsem implementoval v prostředí Mathematica. Pro simulaci výsledků regulace jsem využil software Matlab od firmy MathWorks. Speciálně jeho součást Simulink, která dovoluje pomocí jednotlivých bloků poskládat požadovaný regulační obvod a tím simulovat jeho chování a parametry.

6.1 Princip metody návrhu regulátoru 2DOF

Při samotném řízení se používají dvě základní konfigurace uzavřeného regulačního obvodu:

- 1DOF (FB) – systém s jedním stupněm volnosti (pouze zpětnovazební části ¨ regulátoru)
- 2DOF (FBFW) – systém se dvěma stupni volnosti (s přímovazební i zpětnovazební částí)



Obr. 9: Systém řízení se dvěma stupni volnosti (2DOF)

Na Obr. 9 představuje $G(s) = \frac{b(s)}{a(s)}$ regulovanou soustavu, $C_{fw} = \frac{r(s)}{p(s)}$ přímovazební část regulátoru a $C_{fb} = \frac{q(s)}{p(s)}$ zpětnovazební část regulátoru. Signály *w*, *u*, *y* určují žádanou veličinu, akční zásah a výstupní veličinu. Dále *n* značí poruchu na vstupu řízené soustavy . Porucha *v* na výstupu soustavy je zanedbána.

Obecně lze říci, že systém řízení 2DOF poskytuje regulační pochody s redukovanými překmity, neboli, že zlepšuje schopnost asymptotického sledování žádané veličiny. Toto asymptotické sledování je zajištěno pomocí přímovazební části regulačního obvodu.[3]

6.2 Návrh regulátoru metodou 2DOF pro přenos $G(s) = \frac{s+3}{s^2 - s - 2}$

Po určení stupňů polynomů se získá dvojice diofantických rovnic

$$(s^2 + a_1 s + a_0)s(p_2 s + p_1) + (b_1 s + b_0)(q_2 s^2 + q_1 s + q_0) = D \tag{21}$$

a

$$s(t3s^3 + t_2 s^2 + t_1 s + t_0) + (b_1 s + b_0) r_0 = D \tag{22}$$

Pravou stranu D jsem určil pomocí spektrální faktorizace polynomu $s^2 - s - 2$ Newtonovou metodou naprogramovanou v rámci této bakalářské práce. Výsledkem faktorizace bylo polynom $P = (s+1)(s+2) = s^2 + 3s + 2$.

Tento polynom bylo nutné rozšířit o další polynom, aby byl splněn požadavek na stupeň pravé strany diofantické rovnice. Pro rozšíření jsem zvolil polynom $(s + 0.5)^2$ .Vysledný polynom D byl tedy ve tvaru $D = s^4 + 4s^3 + 5{,}25s^2 + 2{,}75s + 0{,}5$ Tímto se stupeň pravé strany zvýšil na 4, čímž byla splněna nutná podmínka.

Po roznásobení měla rovnice (21) tuto podobu

$$s^4(p_2) + s^3(p_1 + a_1 p_2 + b_1 q_2) + s^2(a_0 p_2 + a_1 p_1 + b_1 q_1 + b_0 q_2) + s(a_0 p_1 + b_0 q_1 + b_1 q_0) + \\ + b_0 q_0 = s^4 + 4s^3 + 5{,}25s^2 + 2{,}75s + 0{,}5$$

Porovnáním koeficientů u jednotlivých mocnin jsem získal soustavu rovnic

$$\begin{aligned}
s^4 &: p_2 && && = 1 \\
s^3 &: a_1 p_2 + && p_1 + b_1 q_2 && = 4 \\
s^2 &: a_0 p_2 + a_1 p_1 + && b_0 q_2 + b_1 q_1 && = 5{,}25 \\
s^1 &: && a_0 p_1 + && b_0 q_1 + b_1 q_0 = 2{,}75 \\
s^0 &: && && b_0 q_0 = 0{,}5
\end{aligned} \tag{23}$$

Pro rovnici (22) platí obdobně

$$t_3 s^4 + t_2 s^3 + t_1 s^2 + s(t_0 + b_1 r_0) + b_0 r_0 = s^4 + 4s^3 + 5{,}25s^2 + 2{,}75s + 0{,}5$$

Porovnáním koeficientů u jednotlivých mocnin jsem získal soustavu rovnic

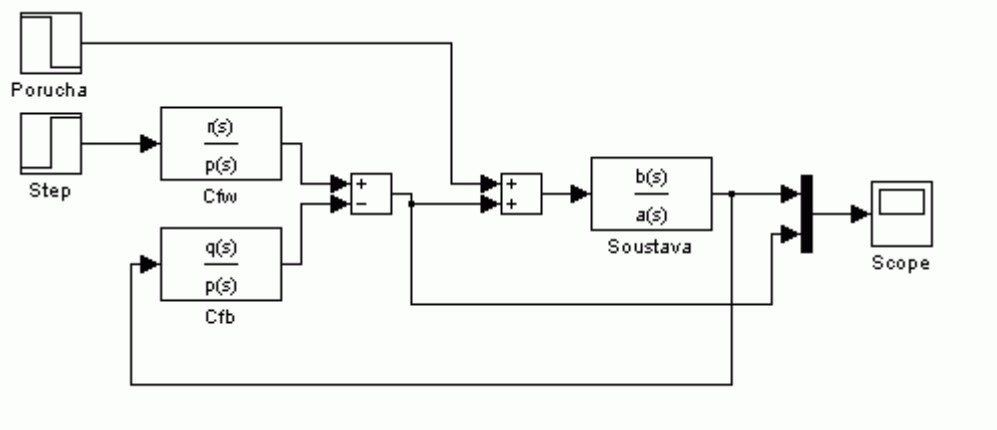
$$\begin{aligned}
s^4 &: t_3 && && = 1 \\
s^3 &: && t_2 && = 4 \\
s^2 &: && && t_1 = 5{,}25 \\
s^1 &: && && t_0 + b_1 r_0 = 2{,}75 \\
s^0 &: && && b_0 r_0 = 0{,}5
\end{aligned} \tag{24}$$

Řešením (23) a (24) jsme získal parametry přímovazebního regulátoru $C_{fw} = \dfrac{r_0}{p_1 s + p_0}$ i zpětnovazebního regulátoru $C_{fb} = \dfrac{q_2 s^2 q_1 s + q_0}{p_1 s + p_0}$

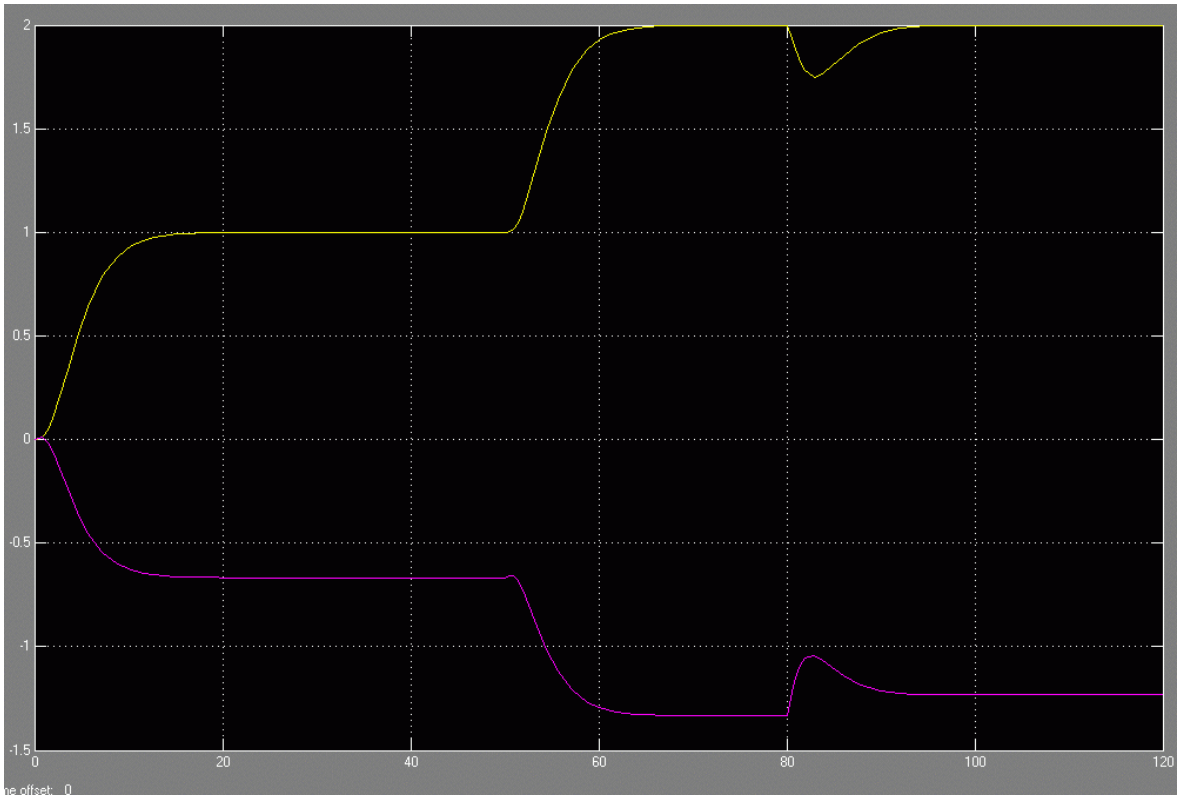
Pro řešení soustav rovnic (23) a (24) jsem využil programovou utilitu napsanou v prostředí Matlab, která je volně přístupná v kurzu Teorie automatického řízení I. Bylo nutné ji mírně upravit, jelikož se v ní neuvažovala porucha vstupující na vstup řízené soustavy. Zdrojový kód aplikace je uveden v příloze P IV.

Po vyřešení soustav rovnic (23) a (24) jsem dostal regulátory v tomto tvaru:

$$C_{fw} = \frac{0{,}16667}{s^2 + 2{,}5833s} \qquad C_{fb} = \frac{2{,}4167s^2 + 2{,}5833s + 0{,}16667}{s^2 + 2{,}5833s}$$



Obr. 10: Simulační schéma v prostředí Simulink



Obr. 11: Průběhy regulace soustavy řízené regulátorem navrženým metodou 2DOF s poruchou na vstupu regulované soustavy

ZÁVĚR

V této bakalářské práci jsem řešil vybrané úlohy z teorie automatického řízení. Práce je rozdělena do dvou částí. První část je teoretická a popisuje prostředí Mathematica a vybrané úlohy z teorie automatického řízení a jejich algoritmy. Další část je praktická a řeší komplexní úlohu z teorie automatického řízení s využitím mnou naprogramovaných funkcí v software Mathematice.

V teoretické části jsem se zabýval nalezením největšího společného dělitele dvou polynomů, řešením Diofantických rovnic a spektrální faktorizací polynomu. V této části práce jsem také zběžně přiblížil software Mathematica. Popsal jsem jeho vlastnosti a možné styly programování, které má uživatel k dispozici. Dále jsem teoreticky popsal vhodné algoritmy pro řešení výše uvedených úloh z teorie automatického řízení, které jsem následně naprogramoval jako funkce v prostředí Mathematica.

Při programování funkce pro nalezení největšího společného dělitele jsou využil Euklidův algoritmus. Funkce pro nalezení největšího společného dělitele dvou polynomů je definovaná pro polynomy s mocninou z^{-1}. Řešení diofantických rovnic je realizováno pomocí metody neurčitých koeficientů. Opět je definováno pro polynomy s mocninou z^{-1}. Další úlohou kterou jsem se zabýval je spektrální faktorizace polynomu.Tuto úlohu jsem řešil pomocí Newtonovi iterační metody. Tato funkce je definována pro spojité systémy. To znamená, že pracuje s polynomy založenými na s^i.

Správnost programového řešení jsem ověřil na již dříve vyřešených příkladech.

Stěžejní části této práce bylo aplikování naprogramovaných funkcí při řešení komplexní úlohy z oblasti teorie řízení. Zabýval jsem se návrhem systému se dvěma stupni volnosti (2DOF). Pro řešení této úlohy jsem využil mnou naprogramovanou funkci spektrální faktorizace polynomu Newtonovou metodou.

V programu Matlab, v jeho části Simulink jsem sestavil schéma regulačního systému a simulací jsem prověřil navřený regulátor. Regulátor který byl touto metodou navržen soustavu řídil bez problémů.

ZÁVĚR V ANGLIČTINĚ

In this bachelor thesis I solved chosen tasks from control theory. This thesis is separated into two parts. The first one is theoretical and there it's described software Mathematica and chosen tasks from control theory and their algorithms. The next part is practical and it solved complex task from control theory using by myself coded functions in software Mathematica.

In theoretical part of this thesis I deal with solution of finding greatest common divisor of two polynomials. For this task I used Euclidean algorithm. Function for finding greatest common divisor is defined for z^{-i} based polynomials. Solution of Diophantine equations is realized via inexplicit coefficient method. This function is as well defined for z^{-I} based polynomials. The next task I deal with is spectral factorization of polynomial. I solved this task via Newton iterative method. This function is implemented for continuous time systems. It is mean the function is working with s^{i} based polynomials.

The correct implementation of hereinbefore functions was tested on earlier solved tasks.

Main part of this thesis was application implemented functions for complex task from control theory. I deal with synthesis Two – Degree – Of – Freedom system (2DOF). For solution of this task I used by myself implemented function for computing spectral factorization of polynomial via Newton iteration method.

In Matlab, actually in its part Simulink, I created control loop and I tested by simulations the correct form of my synthesis of controller. The controller which was found by synthesis 2DOF system was able to control system without serious problems

SEZNAM POUŽITÉ LITERATURY

[1] DOSTÁL, Petr, GAZDOŠ, František, BOBÁL, Vladimír. Design of Controllers for Processes with Time Delay by Polynomial Method. In Proceeding of the European Control Conference. 2007th edition. Kos (Greece) : [s.n.], 2007. s. 4540 – 4545. ISBN 978-960-89028.

[2] KUČERA, Vladimír. Diophantine equations in control – A survey. Automatica.1993, vol. 29, no. 6, s.1361 – 1375.

[3] PROKOP, Roman, MATUŠů, Radek, PROKOPOVÁ, Zdenka. Teorie automatického řízení – lineární spojité dynamické systémy. Zlín : Univerzita Tomáše Bati ve Zlíně, 2006. 102 s. První vydání. ISBN 80-7318-369-2.

[4] ŠEBEK, Michael. Algoritmy pro spektrální faktorizaci polynomů. In Využití polynomiálních metod v řízení technologických procesů : Seminární kurz ÚTIA ČSAV. Praha : [s.n.], 1988. s. 118 – 126.

[5] Wikipedia : The Free Encyklopedia [online]. c2001 , 28 February 2006 [cit. 2008-01-25]. Text v angličtině. Dostupný z WWW:http://en.wikipedia.org/wiki/ /Diophantine.

[6] Wolfram Mathematica Documentation Center [online]. 2008 [cit. 2008-01-25]. Text v angličtině. Dostupný z WWW:http://www.reference wolfram.com/guide/ Mathematica.html

[7] TESAŘOVÁ, Barbora. Základní Elementy Programování V Systému Mathematica. Hradec Králové : Universita Hradec Králové, Fakulta Informatiky a managementu

[8] SAYED A . H, KAILATH, T. A Survey of Spectral Factorization Methods. Numer. Linear Algebra Appl. 2001, vol. 8, pp. 467 – 496.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

GCD	Greatest common divisor; největší společný dělitel.
1DOF	One degree of freedom, systém s jedním stupněm volnosti
2DOF	Two degree of freedom, systém se dvěma stupni volnosti
G(s)	Přenos regulovaného systému
$C(C_{fw},C_{fb})$	Přenosy regulátoru systému.
u(t)	Akční veličina
w(t)	Žádaná veličina
y(t)	Výstupní veličina
v(t)	Porucha regulované veličiny(šum měření)
n(t)	Porucha akční veličiny

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: GCD POLYNOMŮ *A, B*

Zdrojový kód programu Mathematica:

```
nsd[a_, b_] := Module[{l, q = IdentityMatrix[2], sta, sa, stb, sb, koa, kob, λ, ρ, pom1 = a, pom2 = b},
  l = {{pom1}, {pom2}};
  sa = Exponent[l[[1, 1]], z, List];
  sb = Exponent[l[[2, 1]], z, List];
  If[sb[[1]] <= sa[[1]], l = {{pom1}, {pom2}}, l = {{pom2}, {pom1}}; q = {{0, 1}, {1, 0}}];
  pom1 = l[[1, 1]];
  pom2 = l[[2, 1]];
  pom3 = q[[1]];
  pom4 = q[[2]];
  While[pom2 =!= 0,
   sta = Exponent[l[[1, 1]], z, List];
   stb = Exponent[l[[2, 1]], z, List];
   If[stb[[1]] ≤ sta[[1]], l = {{pom1}, {pom2}}; q = {pom3, pom4}, l = {{pom2}, {pom1}}; q = {pom4, pom3}];
   stupennahore = Exponent[l[[1, 1]], z, List];
   stupendole = Exponent[l[[2, 1]], z, List];
   koa = Last[CoefficientList[l[[1, 1]], z^-1]];
   kob = Last[CoefficientList[l[[2, 1]], z^-1]];
   λ = kob / koa;
   ρ = z^-(Abs[stupendole[[1]]] - Abs[stupennahore[[1]]]);
   l[[2, 1]] = l[[2, 1]] - λ * ρ * l[[1, 1]];
   q[[2, 1]] = q[[2, 1]] - λ * ρ * q[[1, 1]];
   q[[2, 2]] = q[[2, 2]] - λ * ρ * q[[1, 2]];
   pom1 = l[[1, 1]];
   pom2 = l[[2, 1]] // Simplify // Expand;
   pom3 = q[[1]];
   pom4 = q[[2]];
  ];
  Return[{l // Simplify // Expand // MatrixForm, q // MatrixForm}]]
```

Nalezení řešení pro *GCD(a,b),* jestliže $a = -2z^{-1} + z^{-2}$ a $b = 2 - z^{-1} - 2z^{-2} + z^{-3}$:

```
nsd[-2 z^-1 + z^-2, 2 - z^-1 - 2 z^-2 + z^-3]
```

$$\left\{\begin{pmatrix} 2 - \frac{1}{z} \\ 0 \end{pmatrix}, \begin{pmatrix} -\frac{1}{z} & 1 \\ 1 - \frac{1}{z^2} & \frac{1}{z} \end{pmatrix}\right\}$$

PŘÍLOHA P II: ŘEŠENÍ DIOFANTICKÝCH ROVNIC METODOU NEURČITÝCH KOEFICIENTŮ

Zdrojový kód programu Mathematica:

```
diof[] := Module[{result = {}},
  a = Input["Zadejte polynom a"];
  b = Input["Zadejte polynom b"];
  c = Input["Zadejte polynom c"];
  sta = Exponent[a, z^-1];
  stb = Exponent[b, z^-1];
  stc = Exponent[c, z^-1];
  Which[sta + stb > stc, stx = stb - 1; sty = sta - 1, sta + stb ≤ stc, stx = stb - 1; sty = stc - stb];
  polynomx = "x0";
  neznamex = {x0};
  polynomyps = "y0";
  neznameyps = {y0};
  If[stx == 0, brzda = 0, brzda = stx];
  koeficienty = Table[i, {i, 1, brzda}];
  j = 1;
  While[brzda != 0,
   pom = koeficienty[[j]] // ToString;
   polynomx = polynomx <> "+" <> "x" <> pom <> "*" <> "z^-1";
   pom1 = "x" <> pom // ToExpression;
   AppendTo[neznamex, pom1];
   j++;
   brzda--;
  ];
  pomoc = Length[neznamex];
  If[sty == 0, brzda = 0, brzda = sty];
  koeficienty = Table[i, {i, 1, brzda}];
  j = 1;
  While[brzda != 0,
   pom = koeficienty[[j]] // ToString;
   polynomyps = polynomyps <> "+" <> "y" <> pom <> "*" <> "z^-1";
   pom2 = "y" <> pom // ToExpression;
AppendTo[neznameyps, pom2];
brzda--;
   j++;
  ];
```

```
nezvektor = AppendTo[neznamex, neznameyps];
 plosteni = Length[nezvektor];
 nezvektor = FlattenAt[nezvektor, plosteni];
 polynomx = polynomx // ToExpression;
 polynomyps = polynomyps // ToExpression;
 rovnice = (a * polynomx) + (b * polynomyps) - c // Expand;
 casti = CoefficientList[rovnice, z^-1];
 delka = Length[casti];
 i = 1;
 resitel = {};
 While[delka ≠ 0,
   AppendTo[resitel, casti[[i]] == 0];
   delka--;
   i++;
  ];
 resitel;
 vysledek = Solve[resitel, nezvektor] // Flatten;
 i = 1;
 delka = Length[vysledek];
 While[delka ≠ 0,
   AppendTo[result, ReplaceAll[nezvektor[[i]], vysledek[[i]]]]
    i++;
   delka--];
 i = 1;
 k = 0;
 partx = 0;
 While[i < pomoc + 1,
   partx = partx + result[[i]] * z^-k;
   i++;
   k++];
 delka = Length[nezvektor];
 i = 1 + pomoc;
 k = 0;
 partyps = 0;
 While[i < delka + 1,
   partyps = partyps + result[[i]] * z^-k;
   i++;
   k++];
 Return[Print["xp=", partx]; Print["yp=", partyps]]]
```

Nalezené partikulární řešení diofantické rovnice $(1-z^{-1}+2z^{-2})x+(1-z^{-1})y=1-2z^{-1}$

diof[]

xp=5

$$yp=-3+\frac{3}{z}$$

PŘÍLOHA P III: FAKTORIZACE POLYNOMU NEWTONONOU ITERAČNÍ METODOU

Zdrojový kód programu Mathematica:

```
faktor[p_] := Module[{poly = p},
  inver = {};
  vektorsdruzeny = {};
  b = 0;
  mocniny = Exponent[poly, s, List];
  cleny = CoefficientList[poly, s];
  brzda = Length[cleny];
  i = 0;
  a = 1;
  While[i < brzda,
   AppendTo[inver, a];
   a = a * -1;
   i++];
  a = 1;
  i = 0;
  sdruzeny = 0;
  While[a < brzda + 1,
   AppendTo[vektorsdruzeny, cleny[[a]] * s^i];
   i++;
   a++];
  pom = inver * vektorsdruzeny;
  brzda = Length[pom];
  a = 1;
  While[a < brzda + 1,
   sdruzeny = sdruzeny + pom[[a]];
   a++
  ];
  Return[sdruzeny]
 ]
```

```
resenispr[m_, n_, o_] := Module[{result = {}, promenna1 = m, promenna2 = n, promenna3 = o},
  sta = Exponent[promenna1, s];
  stx = sta;
  sty = sta;
  polynomx = "x0";
  neznamex = {x0};
  polynomyps = "x0";
  neznameyps = {x0};
  If[stx == 0, brzda = 0, brzda = stx];
  koeficienty = Table[i, {i, 1, brzda}];
  j = 1;
  While[brzda != 0,
   pom = koeficienty[[j]] // ToString;
   polynomx = polynomx <> "+" <> "x" <> pom <> "*" <> "s" <> "^" <> pom;
   pom1 = "x" <> pom // ToExpression;
   AppendTo[neznamex, pom1];
   j++;
   brzda--;
  ];
  neznamex;
  If[sty == 0, brzda = 0, brzda = stx];
  koeficienty = Table[i, {i, 1, brzda}];
  j = 1;
  While[brzda != 0,
   pom = koeficienty[[j]] // ToString;
   polynomyps = polynomyps <> "+" <> "x" <> pom <> "*" <> "s" <> "^" <> pom;
   pom2 = "x" <> pom // ToExpression;
AppendTo[neznameyps, pom2];
brzda--;
   j++;
  ];
  neznameyps;
  orez = Length[neznamex];
  nezvektor = neznamex;
  nezvektor = AppendTo[nezvektor, neznameyps];
  nezvektor = FlattenAt[nezvektor, orez + 1];
  polynomx = polynomx // ToExpression;
  polynomyps = polynomyps // ToExpression;
  polynomyps = faktor[polynomyps];
  rovnice = (promenna1 * polynomx) + (promenna2 * polynomyps) - 2 promenna3 // Expand;
  casti = CoefficientList[rovnice, s];
  delka = Length[casti];
  i = 1;
  resitel = {};
```

```
While[delka ≠ 0,
   AppendTo[resitel, casti[[i]] == 0];
   delka--;
   i++;
  ];
 vysledek = Solve[resitel, Take[nezvektor, orez]] // Flatten;
 i = 1;
 j = 1;
 If[promenna1 =!= promenna2, delka = Length[vysledek];
   While[delka > 0,
    AppendTo[result, ReplaceAll[nezvektor[[j]], vysledek[[i]]]];
    i++;
    j++;
    delka--];
   result;
   delka = Length[result];
   i = 1;
    k = 0;
    j = 1;
    hledanex = 0;
    While[i < delka + 1,
     hledanex = hledanex + result[[i]] * s ^k;
     i++;
     j++;
     k++];
   ,
   delka = Length[vysledek];
   While[delka > 0,
    AppendTo[result, ReplaceAll[nezvektor[[j]], vysledek[[i]]]];
    i++;
    j = j + 2;
    delka--];
   result;
delka = Length[result];
   i = 1;
   k = 0;
   j = 1;
   hledanex = 0;
   While[i < delka + 1,
    hledanex = hledanex + result[[i]] * s ^k;
    i++;
    j = j + 2;
    k = 2 + k];];
 Return[hledanex]
 ]
```

```
p = (s^2 - s - 2);
psdruz = faktor[p];
b = p * psdruz // Expand;
expo = Exponent[b, s];
koef = CoefficientList[b, s];
pomoc = (First[koef]^(1. / expo) + Last[koef]^(1. / expo) * s)^(expo / 2) // Expand;
bota = Rationalize[pomoc, 0.001];
pstart = pomoc;
pstartsdruz = faktor[pstart];
pstart = s^2 + 2.88 * s + 2 // Rationalize;
pstartsdruz = faktor[pstart];
b = p * psdruz;
i = 0;
While[i < 4,
 no = resenispr[pstart, pstartsdruz, b];
 Print["Reseni symericke polynomialni rovnice:", no];
 no = faktor[no];
 pom = (pstart + no) * 1 / 2 // Expand;
 Print["Faktor hledaného polynomu:", pom];
 pstart = pom;
 pstartsdruz = faktor[pstart];
 i++;
 If[no == 0, Break[]];
]
```

Řešení spektrální faktorizace polynomu které je využito při řešení komplexní úlohy v kapitole 6.2

Reseni symericke polynomialni rovnice: $2 - \frac{25\,s}{8} + s^2$

Faktor hledaného polynomu: $2 + \frac{1201\,s}{400} + s^2$

PŘÍLOHA P IV: UTILITA PRO ŘEŠENÍ 2DOF REGULATORU

Zdrojový kód programu Matlab:

```
% Synteza 2DOF
b1=1; b0=3; b=[b1 b0];
a2=1; a1=-1;a0=-2; a=[a2 a1 a0];
m=0.5;
% AFP+BQ=M
% poradi: p2 p1 q2 q1 q0
Matice1=[1 0 0 0 0; a1 1 b1 0 0;a0 a1 b0 b1 0; 0 a0 0 b0 b1; 0 0 0 0 b0];
D=[1 4 5.25 2.75 0.5]';
X1=Matice1\D; % Matice*X=D
p=[X1(1) X1(2) 0];
q=[X1(3) X1(4) X1(5)];
% FT+BR=M
% poradi: t3 t2 t1 t0 r0
Matice2=[1 0 0 0 0; 0 1 0 0 0;0 0 1 0 0; 0 0 0 1 b1; 0 0 0 0 b0];
X2=Matice2\D; % Matice*X=D

r=X2(5);

sim Poly_2DOF_ukazka
Poly_2DOF_ukazka
```