

Využití OOP v oblasti lineárních diskrétních dynamických systémů

Utilization of the OOP in the Linear Discrete Dynamical Systems Area

Jakub Hromada

Bakalářská práce
2013



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2012/2013

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Jakub HROMADA**
Osobní číslo: **A10031**
Studijní program: **B3902 Inženýrská informatika**
Studijní obor: **Informační a řídicí technologie**
Forma studia: **prezenční**

Téma práce: **Využití OOP v oblasti lineárních diskrétních dynamických systémů**

Zásady pro vypracování:

1. Zpracujte rešerši vybraných statí teorie automatického řízení s důrazem na problematiku lineárních diskrétních dynamických systémů (dále LDDS).
2. Zmapujte možnosti objektově orientovaného programování (dále OOP) a navrhněte jejich využití ve prospěch oblasti LDDS se zaměřením na jazyk C++.
3. Vyberte vhodné metody numerické matematiky za účelem implementace operací souvisejících s LDDS.
4. Realizujte aplikaci v jazyce C++ založenou na výhodách OOP užitých ve prospěch oblasti LDDS.
5. Ověřte správnost vytvořených řešení pomocí programového prostředí MATLAB.

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. BALÁTEĚ, Jaroslav. Automatické řízení. Praha: BEN, 2003. ISBN 80-7300-020-2.
2. VAŠEK, Vladimír. Teorie automatického řízení II. Brno: Vysoké učení technické v Brně, 1990. ISBN 80-214-0115-X.
3. DATTA, Biswa Nath. Numerical Methods for Linear Control Systems: Design and Analysis. Amsterdam: Elsevier Academic Press, 2004. ISBN 978-0-12-203590-6.
4. POKORNÝ, Pavel. Objektově orientované programování v C++. 2. vydání. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005. ISBN 80-7318-330-7.
5. LIBERTY, Jesse. Naučte se C++ za 21 dní. Praha: Computer Press, 2002. ISBN 80-7226-774-4.
6. PERŮTKA, Karel. MATLAB – Základy pro studenty automatizace a informačních technologií. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005. ISBN 80-7318-355-2.
7. KOVÁŘÍK, Martin. Matematické aplikace v Matlabu. Zlín: Univerzita Tomáše Bati ve Zlíně, 2008. ISBN 978-80-7318-753-8.
8. KARBAN, Pavel. Výpočty a simulace v programech Matlab a Simulink. Brno: Computer Press, 2006. ISBN 978-80-251-1448-3.

Vedoucí bakalářské práce:

Ing. Tomáš Barot
Ústav řízení procesů

Datum zadání bakalářské práce:

24. února 2013


Termín odevzdání bakalářské práce:

14. června 2013

Ve Zlíně dne 24. února 2013


prof. Ing. Vladimír Vašek, CSc.
děkan




prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Cílem bakalářské práce je skloubení objektově orientovaného programování s lineárními diskrétními dynamickými systémy. Je poukazováno na skutečnost, že OOP je vhodným nástrojem pro vytvoření jakékoliv logiky a kvalitně vrství jednotlivé podoblasti daného problému. Praktická část prezentuje aplikaci základních i specifických aspektů OOP v oblasti analýzy LDDS, jež jsou popsány v teoretických kapitolách. Styl práce byl koncipován z pohledu vlastního řešení numerických problémů. Práce obsahuje ověření dosažených výsledků v prostředí Matlab.

Klíčová slova:

LDDS, OOP, C++, numerické metody, diskrétní teorie systémů, Matlab

ABSTRACT

The aim of this thesis is the combination of object-oriented programming principles with linear discrete dynamical systems. It is pointed out that the OOP is an appropriate tool for creating any logic and well-layered areas of the problem. The practical part presents the application of basic and specific aspects of OOP in LDDS analysis, which are described in the theoretical chapters. Style of work was conceived from the perspective of own numerical solution of problems. The thesis includes the verification of the results in Matlab.

Keywords:

LDDS, OOP, C++, numerical methods, discrete systems theory, Matlab

Děkuji za podporu své rodině, která mě podporovala při mém studiu na Univerzitě Tomáše Bati. Děkuji svému vedoucímu bakalářské práce Ing. Barotovi za cenné rady při tvorbě práce.

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD	9
I TEORETICKÁ ČÁST	10
1 LDDS	11
1.1 ÚVOD	11
1.2 VNĚJŠÍ POPIS JEDNOROZMĚRNÉHO SYSTÉMU	11
1.2.1 Diferenční rovnice systému	11
1.2.2 Diskrétní přenos (Z - přenos)	12
1.3 VNĚJŠÍ POPIS VÍCEROZMĚRNÉHO SYSTÉMU	13
1.3.1 Soustava diferenčních rovnic	13
1.3.2 Popis mnohorozměrného systému v Z-transformaci	14
1.4 STAVOVÝ POPIS	15
1.4.1 Stavový popis jednorozměrný	15
1.4.2 Stavový popis mnohorozměrný	16
1.5 CHARAKTERISTIKY SYSTÉMU	17
1.5.1 Impulsní funkce	17
1.5.2 Přechodová funkce	17
1.6 STABILITA DISKRÉTNÍHO SYSTÉMU	19
1.6.1 Stabilita jednorozměrného vnějšího systému	19
1.6.2 Stabilita mnohorozměrného systému	19
1.6.3 Stabilita stavového popisu	20
1.7 PŘEVODY MEZI STAVOVÝM A VNĚJŠÍM POPISEM SYSTÉMŮ	22
1.7.1 Převod stavového popisu na vnější popis	22
1.7.2 Převod vnějšího popisu jednorozměrného systémů na stavový	23
1.7.3 Přepis vnějšího systému mnohorozměrného systému na stavový	25
2 NUMERICKÉ METODY	27
2.1 ÚVOD	27
2.2 GRAEFFOVA METODA	27
2.2.1 Příklad komplexních kořenů	28
2.3 GAUSSOVA ELIMINAČNÍ METODA	28
3 ZMAPOVÁNÍ MOŽNOSTÍ OOP	30
3.1 ZÁKLADNÍ ASPEKTY OOP	30
3.1.1 Třída	30
3.1.2 Zapouzdření	30
3.1.3 Dědičnost	30
3.1.4 Polymorfismus	30
3.2 SPECIFICKÉ ASPEKTY OOP	31
3.2.1 Přetěžování operátorů	31
3.2.2 Šablony	31
3.2.3 STL	31
3.2.4 Množina dalších specifických aspektů	31

4	MATLAB	32
4.1	ZÁKLADNÍ FUNKCE MATLABU PRO VERIFIKACI	32
4.1.1	Funkce tf.....	32
4.1.2	Funkce ss	33
4.1.3	Funkce impulse	34
4.1.4	Funkce step.....	34
II	PRAKTICKÁ ČÁST	35
5	ZMAPOVÁNÍ A PŘÍNOSY ASPEKTŮ OOP VE PROSPĚCH OBLASTI LDDS	36
5.1	VYUŽITÍ ZÁKLADNÍCH ASPEKTŮ	36
5.2	VYUŽITÍ SPECIFICKÝCH ASPEKTŮ.....	36
6	REALIZACE PROGRAMU V JAZYCE C++ S OVĚŘENÍM VÝSLEDKŮ POMOCÍ MATLABU.....	38
6.1	VYTVORENÍ SYSTÉMU	39
6.2	POLYMORFISMUS A DISKRÉTNÍ SYSTÉMY	42
6.3	PŘETĚŽOVÁNÍ OPERÁTORŮ PŘI ŘEŠENÍ STAVOVÝCH ROVNIC	46
6.4	VÍCENÁSOBNÁ DĚDIČNOST POUŽITA PRO PODOBNOST SISO A MIMO STAVOVÉHO POPISU	47
6.5	VYUŽITÍ KOPÍROVACÍHO KONSTRUKTORU PŘI PŘEVODU TYPŮ POPISU SYSTÉMU.	48
6.6	VYJÍMKY A JEJICH POUŽITÍ PŘI TESTOVÁNÍ SINGULARITY MATIC POPISU SYSTÉMU	50
6.7	KONTROLNÍ KÓDY FÁZE OVĚŘOVÁNÍ V PROSTŘEDÍ MATLAB	51
7	IMPLEMENTACE NUMERICKÝCH METOD.....	55
7.1	GRAEFFOVA METODA	55
7.2	GAUSSOVA ELIMINAČNÍ METODA	56
8	OVLÁDÁNÍ PROGRAMU	58
	ZÁVĚR	61
	ZÁVĚR V ANGLIČTINĚ.....	62
	SEZNAM POUŽITÉ LITERATURY.....	63
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	64
	SEZNAM OBRÁZKŮ	65
	SEZNAM TABULEK.....	66
	SEZNAM PŘÍLOH.....	67

ÚVOD

Náplní mé bakalářské práce bylo vytvoření rozhraní pro práci s LDDS a ukázat že při znalosti matematiky a objektového programování, lze udělat složitější rozhraní pro práci s danou problematikou.

Byly vybrány konkrétní statě z LDDS, základní charakteristiky, jako je přechodová charakteristika, impulsní charakteristika, konkrétní vstup - výstup systému, stabilita daného systému a převody mezi jednotlivými typy popisů systémů, to jak pro mnohorozměrný systém vnější, vnitřní, tak pro systém vnitřní, vnější s jedním vstupem a jedním výstupem.

Za jazyk programování byl vybrán C++, protože umí dobře pracovat se šablonami, hlavně s nativními typy. Šablony jsou v bakalářské práci jedním z pilířů programování. Jenž je hodně používán v praxi.

V praxi jsou používány zejména základní rysy a aspekty OOP. Práce se však zaměřuje na specifické často nepoužívané techniky. Ty jsou v práci mapovány a použity. Jedná se o více násobná dědičnost, polymorfismus, přetěžování operátorů, šablony, možnost práce s STL, spřátelené funkce, výjimky a kopírovací konstruktory.

Bylo též nutné implementovat některé algoritmy numerické matematiky, jako je výpočet kořenů polynomu, kvůli stabilitě, determinant polynomiální matice a celkové skloubení matic s polynomy.

Spojení výhod C++ se znalostí matematiky a LDDS, může přinést logické, dobře čitelné a škálovatelné kódy.

I. TEORETICKÁ ČÁST

1 LDDS

1.1 Úvod

Stejně jako je to u spojitéch dynamických systémů, tak i u diskretních dynamických systémů je několik možností způsobu vnějšího popisu, které vyjadřují závislost mezi vstupní a výstupní veličinou. [1]

Možnost vnitřního popisu chování diskretního systému obsahuje kromě výstupní veličiny $y(kT)$, a diskretní vstupní veličinou $u(kT)$ i diskretní stavové veličiny $x_i(kT)$. [1]

1.2 Vnější popis jednorozměrného systému

Jednorozměrný systém se vyznačuje jedním vstupem a jedním výstupem, je popsán jednou diferenční rovnicí, jedním diskretním Z - přenosem. [1]

1.2.1 Diferenční rovnice systému

Pro popis vlastností lineárně diskretního dynamického systému můžeme použít diferenční rovnici v rekurentním tvaru spolu s počátečními podmínkami: [1]

$$a_n y[(k+n)T] + \dots + a_1 y[(k+1)T] + a_0 y(kT) = b_m u[(k+m)T] + \dots + b_1 u[(k+1)T] + b_0 y(kT) \quad (1)$$

$$y(0), y(T), \dots, y[(n-1)T] \quad (2)$$

$$u(0), u(T), \dots, u[(m-1)T] \quad (3)$$

Aby diferenční rovnice popisovala reálný dynamický systém, musí se splnit podmínka fyzikální realizovatelnosti. [1]

$$m \leq n \quad (4)$$

Uvedená diferenční rovnice (1) je lineární, její koeficient a_0, a_1, \dots, a_n a b_0, b_1, \dots, b_n jsou časově invariantní - hovoříme o stacionárním LDDS. [1]

Též se někdy používá modifikace diferenční rovnice (1) :

$$\begin{aligned} a_n y(kT) + a_{n-1} y[(k-1)T] + \dots + a_1 y[(k-n+1)T] + a_0 y[(k-n)T] = \\ b_n y(kT) + b_{n-1} y[(k-1)T] + \dots + b_1 y[(k-m+1)T] + b_0 y[(k-m)T] \end{aligned} \quad (5)$$

Počáteční podmínky zůstávají stejné - viz (2), (3)

1.2.2 Diskrétní přenos (Z - přenos)

Po Z transformaci diferenční rovnice (1) spolu s počátečními podmínkami (2), (3), bude mít výsledný tvar algebraické rovnice.[1]

$$(a_n z^n + \dots + a_1 z^1 + a_0)Y(z) - L(z) = (b_m z^m + \dots + b_1 z^1 + b_0)U(z) - R(z) \quad (6)$$

kde je $Y(z)$ je obraz diskretního výstupního signálu, $U(z)$ je obraz diskretního vstupního signálu, $L(z)$ je mnohočlen nanejvýše n-tého stupně určený počátečními podmínkami levé strany diferenční rovnice (1), $R(z)$ je mnohočlen nanejvýše m-tého stupně určený počátečními podmínkami pravé strany diferenční rovnice (1). [1]

Z algebraické rovnice (6) vypočítáme obraz diskretního výstupního signálu:

$$Y(z) = G(z)U(z) + G_p(z) \quad (7)$$

kde racionální lomená funkce $G(z)$ je diskretní přenos v kladných mocninách z^n ve tvaru

$$G(z) = \frac{M(z)}{N(z)} = \frac{b_m z^m + \dots + b_1 z^1 + b_0}{a_n z^n + \dots + a_1 z^1 + a_0}, n \geq m \quad (8)$$

a

$$G_p(z) = \frac{L(z) - R(z)}{N(z)} \quad (9)$$

je racionální lomená funkce určená počátečními podmínkami. Z daného vyjádřeného je jasné, že diskretní přenos ve tvaru uvedeném:[1]

$$G(z) = \frac{Y(z)}{U(z)} = \frac{M(z)}{N(z)} \quad (10)$$

dostaneme z diferenčních rovnic (1) za dané podmínky nulových počátečních podmínek.

Jmenovatel přenosu $G(z)$ (jakož i funkce počátečních podmínek $G_p(z)$) je charakteristický polynom diskretního lineárního dynamického systému.[1]

$$N(z) = a_n z^n + \dots + a_1 z^1 + a_0 = a_n (z - z_1)(z - z_2) \dots (z - z_n) \quad (11)$$

kde z_0, z_1, \dots, z_n jsou kořeny charakteristického polynomu k počáteční diferenční rovnice (1)

$$N(z) = a_n z^n + \dots + a_1 z^1 + a_0 = 0 \quad (12)$$

a tedy z_i jsou póly přenosu $G(z)$. [1]

Z - přenos $G(z)$ rovnice (8) je vyjádřen v kladných mocninách z . Jeho úprava v záporných mocninách z^{-n} má tvar: [1]

$$G(z) = \frac{Y(z)}{U(z)} = \frac{(b_m + b_{m-1}z^{-1} \dots + b_1z^{-m+1} + b_0z^{-m})z^{-n+m}}{a_m + a_{m-1}z^{-1} \dots + a_1z^{-n+1} + a_0z^{-n}} \quad (13)$$

Obě možnosti zápisu diskrétního přenosu při dodržení nulových počátečních podmínek určují jednoznačné vlastnosti daného diskrétního lineárního dynamického systému v oblasti komplexních proměnných (prostor obrazů), stejně jeho diferenční rovnice (1), (5) v časové oblasti pro $k \geq 0$ (prostor originálu). [1]

1.3 Vnější popis vícerozměrného systému

Vícerozměrné systémy se vyznačují více jak jedním vstupem do systému a více jak jedním výstupem ze systému. [2]

1.3.1 Soustava diferenčních rovnic

Podle definice diferenční rovnice pro jednorozměrný systém v kapitole 1.2.1 se problém rozšiřuje na více vstupu a výstupu, nebude už jedna diferenční rovnice popisující systém, ale bude jich více jak jedna. [2]

Pro zjednodušení si vyjádříme počáteční y a budeme pracovat v minulých hodnotách:

Implementace diferenční rovnice pomocí sumy [2]

$$y_1(k) = \sum_{m=1}^l \left(\sum_{n=1}^{r_{1l}} -a_{1ln_{1l}} y_l(k - r_{1l}) + \sum_{n=1}^{s_{1l}} -b_{1ls_{1l}} u_l(k - s_{1l}) \right) \quad (14)$$

$$y_2(k) = \sum_{m=1}^l \left(\sum_{n=1}^{r_{2l}} -a_{2ln_{2l}} y_l(k - r_{2l}) + \sum_{n=1}^{s_{2l}} -b_{2ls_{2l}} u_l(k - s_{2l}) \right) \quad (15)$$

Universální vzorec:

$$y_v(k) = \sum_{m=1}^l \left(\sum_{n=1}^{r_{vl}} -a_{vln_{vl}} y_l(k - r_{vl}) + \sum_{n=1}^{s_{vl}} -b_{vls_{vl}} u_l(k - s_{vl}) \right) \quad (16)$$

$y_v(k)$ - je vyjádření počátečního výstupu konkrétní diferenční rovnice v soustavě diferenčních rovnic index v je konkrétní index diferenční rovnice.[2]

l - počet výstupů

r - maximální počet členů a v diferenční rovnici

s - maximální počet členů b v diferenční rovnici

1.3.2 Popis mnohorozměrného systému v Z-transformaci

Diskrétní z - přenos se vyjadřuje v mnohorozměrných systémech pomocí polynomiálních matic v z^{-1} . [2][3]

Základní polynomiální rovnice vyjádřená pomocí matic je:[2][3]

$$A(z^{-1}).Y(z^{-1}) = B(z^{-1}).U(z^{-1}) \quad (17)$$

$$Y(z^{-1}) = A^{-1}(z^{-1}).B(z^{-1}).U(z^{-1}) \quad (18)$$

kde

$$Y(z^{-1}) = [Y_1(z^{-1}), Y_2(z^{-1}) \dots Y_l(z^{-1})]^T \quad (19)$$

$$U(z^{-1}) = [U_1(z^{-1}), U_2(z^{-1}) \dots U_m(z^{-1})]^T \quad (20)$$

$A(z^{-1})$ - polynomiální matice v z^{-1} $l \times l$

$B(z^{-1})$ - polynomiální matice v z^{-1} $l \times m$

l - počet diferenčních rovnic pro popis MIMO systému

A zde je výsledná přenosová matice diskretních dílčích polynomu - racionálně lomených funkcí.[2][3]

$$G(z^{-1}) = \begin{bmatrix} G_{11}(z^{-1}) & G_{12}(z^{-1}) & \dots & G_{1m}(z^{-1}) \\ G_{21}(z^{-1}) & G_{22}(z^{-1}) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ G_{l1}(z^{-1}) & G_{l2}(z^{-1}) & \dots & G_{lm}(z^{-1}) \end{bmatrix} = A(z^{-1}).B(z^{-1}) \quad (21)$$

1.4 Stavový popis

Základní vlastností, kteréhokoliv dynamického systému je to, že jeho chování, v kterém kolik okamžiku času závisí nejen na silách, které na něj působí právě v tomto okamžiku, ale taky na silách, které na systém působily v minulosti. Můžeme konstatovat, že systém má „paměť“, v které je zapamatován podíl v minulosti účinkujících sil na chování systému v právě pozorovaném okamžiku. Stav systému, odvozený jako množina hodnot tzv. stavových veličin, představuje okamžitý stav buněk této paměti.[1][2]

1.4.1 Stavový popis jednorozměrný

Stavový popis je popsán stavovou rovnicí: [1][2]

$$\vec{x}(k) = F \cdot \vec{x}(k) + G \cdot u(k) \quad (22)$$

$$y(k) = H \cdot \vec{x}(k) + L \cdot u(k) \quad (23)$$

při počátečním stavovém vektoru $\vec{x}(0)$

kde:

Matice F - matice systému rozměrů $n \times n$

Matice G - matice buzení rozměrů $n \times 1$

Matice H - matice výstupní rozměrů $1 \times n$

Matice L - matice převodu rozměrů $1 \times 1 \dots$ pro striktně ryzí systém [0]

n - Počet stavových proměnných

Protože platí následující vztah:

$$\vec{x}(k) = F \cdot \vec{x}(0) + \left(\sum_{r=0}^{k-1} F^{k-r-1} \cdot G \cdot u(r) \right) \quad (24)$$

lze výstup vypočítat jako: [1][2]

$$y(k) = H \cdot F \cdot \vec{x}(0) + \left(\sum_{r=0}^{k-1} H \cdot F^{k-r-1} \cdot G \cdot u(r) \right) + L \cdot u(k) \quad (25)$$

1.4.2 Stavový popis mnohorozměrný

Rovnice a popisující matice daného systému jsou totožné jako jednorozměrného, pouze je více vstupu u a více výstupu y . [1][2][3]

Stavová rovnice:

$$\bar{x}(k+1) = F \cdot \bar{x}(k) + G \cdot \bar{u}(k) \quad (26)$$

$$\bar{y}(k) = H \cdot \bar{x}(k) + L \cdot \bar{u}(k) \quad (27)$$

Ze vstupu $\bar{u}(k)$ a výstupu $\bar{y}(k)$ se stal vektor

$$\bar{y}(k) = [y_1(k), y_2(k) \dots y_l(k)]^T \quad (28)$$

$$\bar{u}(k) = [u_1(k), u_2(k) \dots u_m(k)]^T \quad (29)$$

Obecné stavové rozměry u jednotlivých matic stavového MIMO systému: [1][2][3]

Matice F - matice systému rozměrů $n \times n$

Matice G - matice buzení rozměrů $n \times m$

Matice H - matice výstupní rozměrů $l \times n$

Matice L - matice převodu rozměrů $l \times m$

n - počet stavů

m - počet vstupů

l - počet výstupu

1.5 Charakteristiky systému

1.5.1 Impulsní funkce

Impulsní funkce je odezva na Diracův jednotkový impulz $\delta(kT)$. [1]

Pro $k = 0$

$$\delta(kT) = 1 \quad (30)$$

Pro $k \neq 0$

$$\delta(kT) = 0 \quad (31)$$

Diskrétní impulz $\delta(kT)$ je také někdy pojmenováván Kroneckerovým impulzem. [1]

Z - obraz diskrétního jednotkového impulzu $\delta(kT)$ je

$$U(z) = Z\{\delta(kT)\} = 1 \quad (32)$$

Diskrétní impulzní funkce bude mít Z - obraz: [1]

$$Y(z) = G(z)U(z) = G(z) \quad (33)$$

a originál: [1]

$$y(kT) = g(kT) = Z^{-1}\{G(z)\} \quad (34)$$

1.5.2 Přejchodová funkce

Přejchodová funkce je odezva systému na Heavisideovým jednotkovým skokem $\eta(kT)$ definovaným vztahem: [1]

pro $k = 0, 1, 2, \dots$

$$\eta(kT) = 1 \quad (35)$$

pro $k = -1, -2, -3, \dots$

$$\eta(kT) = 0 \quad (36)$$

Z-obraz diskrétního jednotkového skoku $\eta(kT)$ je

$$U(z) = Z\{\eta(kT)\} = \frac{z}{z-1} \quad (37)$$

Diskrétní přechodová funkce bude mít Z - obraz:[1]

$$Y(z) = H(z) = G(z)U(z) = G(z)\frac{z}{z-1} \quad (38)$$

a originál:[1]

$$y(kT) = h(kT) = Z^{-1}\left\{G(z)\frac{z}{z-1}\right\} \quad (39)$$

1.6 Stabilita diskrétního systému

Obecně platí, že systém je stabilní, pokud kořeny charakteristického polynomu leží uvnitř jednotkové kružnice. [1]

1.6.1 Stabilita jednorozměrného vnějšího systému

Přenos je zadán podle rovnice (8)[1]

$$G(z) = \frac{b_m z^m + \dots + b_1 z^1 + b_0}{a_n z^n + \dots + a_1 z^1 + a_0}$$

Abychom určili stabilitu systému, musíme určit kořeny jmenovatele polynomu čili:[1]

Položíme rovno nule [1]

$$a_n z^n + \dots + a_1 z^1 + a_0 = 0 \quad (40)$$

Jednotlivé kořeny P_1, \dots, P_n

A pro stabilitu platí: [1]

$$|P_i| < 1, i = 1, 2, \dots, n \quad (41)$$

1.6.2 Stabilita mnohorozměrného systému

Podle zadaného systému, podle rovnice (18)

$$Y(z^{-1}) = A^{-1}(z^{-1}) \cdot B(z^{-1}) \cdot U(z^{-1})$$

Vypočítáme determinant polynomiální matice A , musí být převedena do z -obrazu:[2][3]

$$A(z^{-1}) = \begin{bmatrix} 1 + a_{111}z^{-1} + a_{112}z^{-2} + \dots + a_{11n}z^{-n} & 1 + a_{1m1}z^{-1} + a_{1m1}z^{-2} + \dots + a_{1mn}z^{-n} \\ 1 + a_{211}z^{-1} + a_{212}z^{-2} + \dots + a_{21n}z^{-n} & \vdots \\ \vdots & \vdots \\ 1 + a_{l11}z^{-1} + a_{l22}z^{-2} + \dots + a_{l1n}z^{-n} \dots & 1 + a_{lm1}z^{-1} + a_{lm2}z^{-2} + \dots + a_{lmn}z^{-n} \end{bmatrix} \quad (42)$$

$$A(z) = \begin{bmatrix} z^n + a_{111}z^{n-1} + a_{112}z^{n-2} + \dots + a_{11n}z^0 & z^n + a_{1m1}z^{n-1} + a_{1m1}z^{n-2} + \dots + a_{1mn}z^0 \\ z^n + a_{211}z^{n-1} + a_{212}z^{n-2} + \dots + a_{21n}z^0 & \vdots \\ \vdots & \vdots \\ z^n + a_{l11}z^{n-1} + a_{l22}z^{n-2} + \dots + a_{l1n}z^0 \dots & z^n + a_{lm1}z^{n-1} + a_{lm2}z^{n-2} + \dots + a_{lmn}z^0 \end{bmatrix} \quad (43)$$

Poté se vypočítá determinant matice A:[1]

$$|A(z)| = a_n z^n + \dots + a_1 z^1 + a_0 \quad (44)$$

Výsledkem je polynom, položí se rovno 0.[1]

$$a_n z^n + \dots + a_1 z^1 + a_0 = 0 \quad (45)$$

Jednotlivé kořeny:[1]

$$P_1, \dots, P_n \quad (46)$$

A pro stabilitu platí: [1]

$$|P_i| < 1, i = 1, 2, \dots, n \quad (47)$$

1.6.3 Stabilita stavového popisu

Pro stabilitu stavového popisu platí a je jedno jestli je mnohorozměrný nebo jednorozměrný, že výsledná stabilita se počítá, podle matice F. [1]

Určí se vlastní čísla matice F[1]

Vychází se ze vztahu:[1]

$$(F - \lambda I) \quad (48)$$

kde:

Matice F - matice systému

λ - pomocná proměnná

I - jednotková matice

$$(F - \lambda I) = \begin{pmatrix} f_{11} & f_{12} & \dots & f_{1n} \\ f_{21} & f_{22} & \dots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{1n} & f_{2n} & \dots & f_{nn} \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} \quad (48)$$

$$(F - \lambda I) = \begin{pmatrix} f_{11} - \lambda & f_{12} & \dots & f_{1n} \\ f_{21} & f_{22} - \lambda & \dots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{1n} & f_{2n} & \dots & f_{nn} - \lambda \end{pmatrix} \quad (48)$$

Dále se vypočítá determinant $|(F - \lambda I)|$

Ze spočítaného determinantu vznikne polynom a opět se určí jeho kořeny.[1]

$$|(F - \lambda I)| = a_n z^n + \dots + a_1 z^1 + a_0 \quad (49)$$

Výsledkem je polynom, položí se rovno 0[1]

$$a_n z^n + \dots + a_1 z^1 + a_0 = 0 \quad (50)$$

Jednotlivé kořeny:[1]

$$P_1, \dots, P_n \quad (51)$$

A pro stabilitu platí: [1]

$$|P_i| < 1, i = 1, 2, \dots, n \quad (52)$$

1.7 Převody mezi stavovým a vnějším popisem systémů

Obecně platí, že systém může mít pouze jeden vnější popis, ale více vnitřních popisů.[1]

1.7.1 Převod stavového popisu na vnější popis

Převod stavového systému na vnější systém a je jedno, jestli je mnoho rozměrný nebo jednorozměrný. Vychází ze stavových rovnic (22),(23)[1]

Převede se na stavové rovnice v z - obrazech:[1]

$$z.X(z) = F.X(z) + G.U(z) \quad (53)$$

$$Y(z) = H.X(z) + L.U(z) \quad (54)$$

Kde: - $X(z)$ - vektor Z - obrazů stavových veličin $X(z) = [X_1(z) \ \dots \ X_n(z)]^T$

$$z.X(z) - F.X(z) = G.U(z) \quad (55)$$

$$(zI - F).X(z) = G.U(z) \quad (56)$$

$$X(z) = (zI - F)^{-1}G.U(z) \quad (57)$$

$$Y(z) = H.(zI - F)^{-1}G.U(z) + L.U(z) \quad (58)$$

$$Y(z) = (H.(zI - F)^{-1}G + L).U(z) \quad (59)$$

$$G(z) = H.(zI - F)^{-1}G + L \quad (60)$$

$$G(z) = \frac{1}{\det(zI - F)} H.adj(zI - F).G + L \quad (61)$$

Po dosazení vyjádření Z - obrazu vektoru stavových veličin (57) do Z -obrazu výstupní rovnice (54) získáme rovnici (58), ve které je po její úpravě na rovnici (59) patrné, jaký má tvar maticový přenos $G(z)$ (60), což je výsledný vztah pro převod diskretního stavového popisu na maticový přenos v Z - transformaci.[1]

1.7.2 Převod vnějšího popisu jednorozměrného systémů na stavový

Vycházíme z diferenčního tvaru zápisu (1) nejprve ji musíme přepsat do tvaru $(k-n)$ čili:[2]

$$\begin{aligned} a_0 y(kT) + a_1 y[(k-1)T] + \dots + a_{n-1} y[(k-n+1)T] + a_n y[(k-n)T] = \\ = a_0 u(kT) + a_1 u[(k-1)T] + \dots + b_m u[(k-m)T] \end{aligned} \quad (62)$$

Zavedme pro posunutí o i period vzorkování E^{-i} , tj.[2]

$$E^{-i} y(k) = y(k-i) \quad (63)$$

S označením (6.2) můžeme rovnici (62) přepsat na tvar[2]

$$\sum_{i=0}^n a_i E^{-i} y(k) = \sum_{i=0}^m b_i E^{-i} u(k) \quad (64)$$

$$\hat{A}y(k) = \hat{B}u(k) \quad (65)$$

kde zřejmě

$$\hat{A} = \sum_{i=0}^n a_i E^{-i} \text{ a } \hat{B} = \sum_{i=0}^m b_i E^{-i} \quad (66)$$

Rovnici (65) můžeme vyjádřit dvěma vztahy:[2]

$$\hat{A}x(k) = u(k) \quad (67)$$

$$\hat{B}x(k) = y(k) \quad (68)$$

Zavedeme - li nyní stavové veličiny[2]

$$x(k-n) = x_1(k), \quad (69)$$

$$x(k-n+1) = x_1(k+1) = x_2(k), \quad (70)$$

$$x(k-n+2) = x_1(k+1) = x_3(k), \quad (71)$$

⋮

$$x(k-1) = x_{n-1}(k+1) = x_n(k), \quad (72)$$

$$x(k) = x_n(k+1), \quad (73)$$

můžeme rovnici (67) přepsat na tvar[2]

$$x_n(k+1) = \frac{1}{a_0} [u(k) - a_1 x_n(k) - a_2 x_{n-1}(k) - \dots - a_n x_1(k)], a \neq 0 \quad (74)$$

Rovnice (67) je nyní se stavovými veličinami (69) a (73)[2]

$$y(k) = b_0 x_n(k+1) + b_1 x_n(k) + \dots + b_n x_1(k) \quad (75)$$

Dosadíme-li v (75) za $x_n(k+1)$ z (74) dostaneme[2]

$$y(k) = \frac{b_0}{a_0} [u(k) - a_1 x_n(k) - a_2 x_{n-1}(k) - \dots - a_n x_1(k)] + b_1 x_n(k) + \dots + b_n x_1(k) \quad (76)$$

$$y(k) = \frac{b_0}{a_0} u(k) + \left(b_1 - \frac{b_0 a_1}{a_0} \right) x_n(k) + \left(b_2 - \frac{b_0 a_2}{a_0} \right) x_{n-1}(k) + \dots + \left(b_n - \frac{b_0 a_n}{a_0} \right) x_1(k) \quad (77)$$

Rovnice (69) a (74) určují první stavovou rovnici[2]

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ \vdots \\ x_n(k+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ -\frac{a_n}{a_0} & -\frac{a_{n-1}}{a_0} & \dots & \dots & -\frac{a_1}{a_0} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_n(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \frac{1}{a_0} \end{bmatrix} u(k) \quad (78)$$

a rovnice (77) určuje druhou stavovou rovnici[2]

$$y(k) = \left[\left(b_n - \frac{b_0 a_n}{a_0} \right), \dots, \left(b_1 - \frac{b_0 a_1}{a_0} \right) \right] [x_1(k), \dots, x_n(k)]^T + \frac{b_0}{a_0} u(k) \quad (79)$$

1.7.3 Přepis vnějšího systému mnohorozměrného systému na stavový

Vycházíme z rovnice (17) musíme převést na z : [2][3]

$$A(z).y(k) = B(z).u(k) \quad (80)$$

Kde:

- $A(z)$ - polynomiální matice řádu N obsahující prvky $a_{ij}(z)$
- $B(z)$ - polynomiální matice řádu N obsahující prvky $b_{ij}(z)$
- $y(k)$ - vektor výstupních diskrétních veličin systému $y(k) = [y_1(k) \quad y_2(k) \quad \dots \quad y_N(k)]^T$
- $u(k)$ - vektor vstupních diskrétních veličin systému $u(k) = [u_1(k) \quad u_2(k) \quad \dots \quad u_N(k)]^T$

Dále provedeme úpravu v rovnici (80), ze které vzejde tvar maticové rovnice (81). [2][3]

$$\left(\sum_{p=0}^n A_p \cdot z^{n-p} \right) \cdot y(k) = \left(\sum_{p=0}^n B_p \cdot z^{n-p} \right) \cdot u(k) \quad (81)$$

Kde:

- n - maximální stupeň polynomů n_{ij} , z vyjádření A_p, B_p - čtvercové číselné matice (nikoliv již polynomiální), pro dané p

Pokud je matice A_p pro $p=0$ (tedy A_0) regulární, tzn. že je její determinant nenulový, provedeme definici nových matic dle vztahů (82) a (83). [2][3]

$$R_p = A_p \cdot A_0^{-1}, \quad \text{pro } p = 0, 1, \dots, n \quad (82)$$

$$S_p = B_p - A_p \cdot A_0^{-1} \cdot B_0, \quad \text{pro } p = 0, 1, \dots, n \quad (83)$$

Vnitřní stavový popis mnohorozměrného diskrétního systému má pak obecný tvar stavové rovnice (84) a výstupní rovnice (85). [2][3]

$$x(k+1) = \begin{bmatrix} -R_1 & I & 0 & \dots & 0 \\ -R_2 & 0 & I & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ -R_n & 0 & 0 & \dots & 0 \end{bmatrix} \cdot x(k) + \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{bmatrix} \cdot u(k) \quad (84)$$

$$y(k) = \begin{bmatrix} A_0^{-1} & 0 & 0 & \cdots & 0 \end{bmatrix} x(k) + \begin{bmatrix} A_0^{-1} B_0 \end{bmatrix} u(k) \quad (85)$$

- N - počet vstupů, výstupů systému soustavy pro soustavu za předpokladu, že $l = m = N$;

Matice F - matice systému rozměrů $n \times n$

Matice G - matice buzení rozměrů $n \times N$

Matice H - matice výstupní rozměrů $N \times n$

Matice L - matice převodu rozměrů $N \times N$ [9]

2 NUMERICKÉ METODY

2.1 Úvod

Numerické algoritmy jsou potřeba, kvůli vypočítání kořenů polynomiálních rovnic. Vybrána byla metoda Graeffova, jelikož u stability systému potřebují absolutní hodnoty kořenů, přesná hodnota mě nezajímá, na to je Graeffova metoda ideální. Další oblastí, kde jsem uplatnil numerické metody je počítání determinantu, zde jsem použil Gaussovu eliminační metodu. [4]-[7]

2.2 Graeffova metoda

Chceme najít aproximaci kořenů $\alpha_1, \alpha_2, \dots, \alpha_n$, polynomu n -tého stupně. [4]-[7]

$$P(z) = a_0 z^n + a_1 z^{n-1} + \dots + a_{n-1} z + a_n \quad (86)$$

přičemž předpokládáme, že kořeny jsou reálné a máme je očíslovány podle klesající absolutní hodnoty, tj. [4]-[7]

$$|\alpha_1| > |\alpha_2| > |\alpha_{n-1}| > |\alpha_n| \quad (87)$$

Definujeme posloupnost polynomů [4]-[7]

$$P^{(0)}(z) = P(z), P^{(1)}(z), P^{(2)}(z), \dots, P^{(k-1)}(z), P^{(k)}(z), \dots, \quad (88)$$

$$P^{(k)}(z) = a_0^{(k)} z^n + a_1^{(k)} z^{n-1} + \dots + a_{n-1}^{(k)} z + a_n^{(k)} \quad (89)$$

prostřednictvím vztahů pro koeficienty ($a_i^{(0)} = a_i, i = 0, 1, 2, \dots, n$): [4]-[7]

$$a_0^{(k)} = (a_0^{(k-1)})^2 \quad (90)$$

$$-a_1^{(k)} = (a_1^{(k-1)})^2 - 2a_0^{(k-1)} a_2^{(k-1)} \quad (91)$$

$$a_2^{(k)} = (a_2^{(k-1)})^2 - 2a_1^{(k-1)} a_3^{(k-1)} + 2a_0^{(k-1)} a_4^{(k-1)} \quad (92)$$

$$-a_3^{(k)} = (a_3^{(k-1)})^2 - 2a_2^{(k-1)} a_4^{(k-1)} + 2a_1^{(k-1)} a_5^{(k-1)} - 2a_0^{(k-1)} a_6^{(k-1)} \quad (93)$$

$$(-1)^{n-1} - a_{n-1}^{(k)} = (a_{n-1}^{(k-1)})^2 - 2a_{n-2}^{(k-1)} a_n^{(k-1)} \quad (94)$$

$$(-1)^n a_{n-1}^{(k)} = (a_{n-1}^{(k-1)})^2 \quad (95)$$

Se dá odvodit vztah[4]-[7]

$$|\alpha_j| \approx \sqrt[k]{\frac{a_j^k}{a_{j-1}^k}}, j = 1, 2, \dots, n \quad (96)$$

2.2.1 Příklad komplexních kořenů

Mají - li některé kořeny algebraické rovnice stejnou absolutní hodnotu, nastanou při užití Graeffovy metody jisté komplikace. [4]-[7]

$$\text{Platí - li : } |\alpha_1| > |\alpha_2| > \dots > |\alpha_{i-1}| > |\alpha_i| = |\alpha_{i+1}| > |\alpha_{i+2}| > \dots > |\alpha_n|$$

a že kořeny se stejnou absolutní hodnotou jsou komplexní, tj. [4]-[7]

$$\begin{aligned} \alpha_i &= r(\cos \alpha + i \sin \alpha) \\ \alpha_{i+1} &= r(\cos \alpha - i \sin \alpha), |\alpha_i| = |\alpha_{i+1}| = r \end{aligned} \quad (97)$$

Absolutní hodnotu r komplexních kořenů α_i, α_{i+1} určíme ze vztahu[4]-[7]

$$r^2 \approx \sqrt[k]{\frac{a_{i+1}^k}{a_{i-1}^k}}, j = 1, 2, \dots, n \quad (98)$$

2.3 Gaussova eliminační metoda

Princip Gaussovy metody spočívá v provedení takových operací, které nezmění výsledek determinantu, ale zjednoduší způsob výpočtu determinant. Výsledek prováděných úprav je trojúhelníková matice A (tedy pro $i > j$ je $a_j^i = 0$), neboť platí[4]-[7]

$$\det A = a_1^1 a_2^2 \dots a_n^n \quad (99)$$

tzn. determinant trojúhelníkové matice je roven součinu prvků hlavní diagonály matice.

Při úpravě řádku v matici postupujeme podle následujících pravidel:

Jesliže B vznikne z A výměnou dvou řádků nebo sloupců potom : [4]-[7]

$$\det B = -\det A \quad (100)$$

Jestliže B vznikne z A vynásobením řádku nebo sloupce skalárem c , potom

$$\det B = c \det A \quad (101)$$

Jestliže B vznikne z A přičtením násobku jednoho řádku k jinému, nebo přidáním násobku sloupce k jinému sloupci potom

$$\det B = \det A \quad (102)$$

Opakováním předešlých uvedených pravidel převedeme matici na matici trojúhelníkovou a poté snadno vypočteme determinant: [4]-[7]

3 ZMAPOVÁNÍ MOŽNOSTÍ OOP

3.1 Základní aspekty OOP

3.1.1 Třída

Hlavním pojmem OOP je objekt, čili instance třídy. Třída je speciálním datovým typem dané entity. Objekt může představovat subjekty reálného i imaginárního světa (komplexní číslo). Na rozdíl od datového typu struktura z jazyka C, lze do třídy specifikovat, kromě samotných dat spjatých s parametrickou stránkou objektu, též i metody neboli činnosti. S objektovým přístupem jsou svázány tři základní principy, a to zapouzdření, dědičnost a polymorfismus. Instance třídy lze vytvořit staticky i dynamicky. Zařazení objektů do pole nazýváme agregace.[8]-[10]

3.1.2 Zapouzdření

Na zapouzdření se můžeme dívat ze dvou pohledů. Prvním principem je uzavření entity objektů, při definici struktury do třídy. Další formou zapouzdření zmiňovaného odbornou literaturou, je aspekt viditelnosti parametrů třídy navenek, což též souvisí s přístupem k atributům ze strany práce s kódem. Rozlišujeme tři varianty viditelnosti atributů na venek třídy - public, private, protected.[8]-[10]

3.1.3 Dědičnost

Dědičnost můžeme definovat způsobem, jenž evokuje samotný název pojmu. Odvozené, neboli děděné třídy, od svých předků, získávají vlastnosti i metody původní třídy. Specifickým případem je více násobná dědičnost.[8]-[10]

3.1.4 Polymorfismus

Třetím základním aspektem objektově orientovaného programování je polymorfismus, neboli mnohotvarost. Dá se definovat, jakožto princip, kdy jedna entita, v tomto případě metoda, má shodný název, ale vykonává ve specifikovaných případech jinou náplň. Přínosy polymorfismu se projeví, při implementaci například v poli objektů, kde je nutné zahrnout též princip virtuálních metod, abstraktní třídy, tedy i dědičnosti.[8]-[10]

3.2 Specifické aspekty OOP

3.2.1 Přetěžování operátorů

Při zaměření na jazyk C++, o němž se bude týkat i další výklad níže, se dají definovat náplně operací při použití operátorů. Můžeme si stanovit, co budou operátory provádět za činnosti. Hlavní výhodou je definování vlastních operátorů do tříd což umožní, provádět operaci i se samotnými objekty například operace s maticemi, operace s polynomy, kdy matice nebo polynom je reprezentovaným objektem.[8]-[10]

3.2.2 Šablony

Šablony slouží k nadefinování obecného objektu, to znamená, že objekt může být jakýmkoliv typem i nativním typem, pokud jde o nativní typ nemusí se definovat jednotlivá kritéria tříd, ale pokud jde o typ jako objekt, daný objekt musí splňovat jednotlivá pravidla obecného objektu.[8]-[10]

3.2.3 STL

C++ má knihovnu STL, daná knihovna obsahuje šablonové datové struktury, které můžeme použít pro vlastní účely. STL obsahuje, třídu vektor, spojový seznam, multimapy, zásobník. Vektor může obsahovat dynamické vytvářené objekty. Má funkci dynamického pole, kdy lze měnit jeho velikost i za běhu, jeho použití je rychlejší oproti klasickému poli jazyka C++.[8]-[10]

3.2.4 Množina dalších specifických aspektů

Zde je na místě zmínit, detailní specifické vlastnosti, a to výjimky, kopírovací konstruktor, více násobná dědičnost a streamy a aspekt friend. Použití výjimek zpřehledňujeme kód a zachytávání chyb programu. Kopírovací konstruktor slouží k vytvoření kopie objektu a k předání nového objektu, kdy můžeme provést specifikaci činností při kopírovacím převodu, při vzniku objektu. Více násobná dědičnost slouží k po dědění dobrých vlastností z více tříd objektů. Streamy slouží k přesunu z paměťové entity do druhé kdy danou entitou může být samotný objekt. Bez aspektu friend by objekty různých tříd nemohly volat metody z private sekcí v rámci hierarchické struktury dědění.[8]-[10]

4 MATLAB

4.1 Základní funkce Matlabu pro verifikaci

4.1.1 Funkce tf

Pro zadání systému slouží funkce *tf* (transfer function), vstupem je polynom oddělený čárkami a perioda vzorkování. Polynom je zadáván v hranatých závorkách a jednotlivé koeficienty jsou odděleny mezerou př. [11]-[12]

```
System = tf([9 8 7 6 5],[7 7 6 6 4],1)
```

Výstupem je:

```
Transfer function:
 9 z^4 + 8 z^3 + 7 z^2 + 6 z + 5
-----
 7 z^4 + 7 z^3 + 6 z^2 + 6 z + 4
Sampling time (seconds): 1
```

Obr. 1 Výstup funkce tf

Takhle vytvoříme přenos pro jednorozměrný systém

Pro více rozměrný systém je potřeba použít složené závorky př. [11]-[12]

```
Gz = tf({[0 0 1 -2 1],[2 -1 2 0]};[0 2 -2 0 4],[1 4 2 0]},{[1 0 1 0 2],[1 0 1 0 2]};[1 0 1 0 2],[1 0 1 0 2]},1)
```

První složená závorka obsahuje polynomiální čitatele konkrétního přenosu a druhá složená závorka jsou polynomiální jmenovatelé daného přenosu, zde je výstup: [11]-[12]

```

Transfer function from input 1 to output...
      z^2 - 2 z + 1
#1:  -----
      z^4 + z^2 + 2

      2 z^3 - 2 z^2 + 4
#2:  -----
      z^4 + z^2 + 2

Transfer function from input 2 to output...
      2 z^3 - z^2 + 2 z
#1:  -----
      z^4 + z^2 + 2

      z^3 + 4 z^2 + 2 z
#2:  -----
      z^4 + z^2 + 2

Sampling time (seconds): 1

```

Obr. 2 Výstup funkce tf

4.1.2 Funkce ss

Funkce ss (state-space) převede jednotlivé zadané matice stavového popisu systému na jeden objekt, se kterým je možno dále manipulovat, např. vykreslit přechodovou, impulsní charakteristiku. Příklad použití: [11]-[12]

```

F = [0.98 0.24; -0.12 0.93];
G = [0.02;0.15];
H = [1 0];
L = [0];
T = 1;

```

```
diskrStavovySISO = ss(F,G,H,L,T)
```

Výstup je [11]-[12]

```

a =
      x1      x2
      x1  0.98  0.24
      x2 -0.12  0.93

b =
      u1
      x1  0.02
      x2  0.15

```

Obr. 3 Výstup funkce ss s maticemi F,G

```
c =  
      x1  x2  
y1    1   0  
  
d =  
      u1  
y1    0  
  
Sampling time (seconds): 1
```

Obr. 4 Výstup funkce ss s maticemi H, L

4.1.3 Funkce impulse

Pokud máme vytvořený systém pomocí *ss*, *tf*, můžeme použít funkci *impulse* která vykreslí impulsní charakteristiku. Za parametr bere daný systém. [11]-[12]

4.1.4 Funkce step

Jestliže je k dispozici objekt systému, je možné aplikovat funkci *step*, jenž zobrazí přechodovou charakteristiku. [11]-[12]

II. PRAKTICKÁ ČÁST

5 ZMAPOVÁNÍ A PŘÍNOSY ASPEKTŮ OOP VE PROSPĚCH OBLASTI LDDS

Obecné pojmy tykající se zmapovaných základních i specifických aspektů OOP vysvětlených v teoretické části práce budou nyní analyzovány z pohledu jejich uplatnění v oblasti LDDS. V této kapitole bude pojednáno o využitelnosti v obecném pohledu.

5.1 Využití základních aspektů

<i>Název aspektu</i>	<i>Forma využitelnost v LDDS</i>
Zapouzdření	Definice systému jako uzavřené entity
Dědičnost	Využitelná pro různé typy popisů systému (pro vnější, vnitřní popis, SISO, MIMO)
Polymorfismus	Jedna činnost má sice stejný název ale provádí různé operace pro různé typy popisů (Práce se signály, impulsní, přechodová charakteristika)

Tab. 1 Využitelnost základních aspektů

5.2 Využití specifických aspektů

<i>Název aspektu</i>	<i>Forma využitelnost v LDDS</i>
Více násobná dědičnost	Pro stavový popis, kdy lze SISO i MIMO případ definovat podobnými daty i operacemi, tedy lze odvodit určité podobnosti z více tříd
Přetěžování operátorů	Stavový popis používá maticových operací, kdy matice je prezentována objekty, pro operace s nimi je potřeba definovat nové operátory a specifikovat tyto činnosti
Šablony	Při použití datových struktur, které zahrnují entity numerické matematiky, jako jsou

	polynomiální matice
Kopírovací konstruktor	Slouží pro převod matematického vyjádření téhož systému na jiné
Streamy	Paměťový prostředek pro realizaci výstupních odezev systémů
Friend	Prefix kvůli přístupu k zapouzdřeným datům dané entity (nutné při zpracování výstupních odezev a při převodu na jiný typ popisu)
Vyjímky	Ošetření chybových stavů systému (nulový determinant při singulární matici)

Tab. 2 Využití specifických aspektů

6 REALIZACE PROGRAMU V JAZYCE C++ S OVĚŘENÍM VÝSLEDKŮ POMOCÍ MATLABU

Vytvořený program prezentuje využití nalezených aspektů jazyka C++ (respektive OOP přístupu) ve prospěch oblasti LDDS. Funkce programu je implementace kapitol analýzy LDDS systémů. Jedná se o konzolovou aplikaci spustitelnou na operačním systému Microsoft Windows a Mac OS. Pro windows bylo použito prostředí CodeBlocks verze 12.11 s kompilátorem MinGW, a pro Mac OS bylo použito prostředí Xcode 4.5.2. Obdobně by se daly přeložit zdrojové kódy na libovolné platformy. Dále budou prezentovány hlavní funkce programu s tím, že popis ovládání se nachází v závěrečné kapitole této práce. Budou uvedeny ukázky zdrojových kódů a ověření důležitých operací v prostředí Matlab. Při tvorbě programu byly použity metody numerické matematiky, které byly vybrány a popsány v teoretické části. Samotná aplikace je nemá vlastní menu, ale pracuje výhradně ve formě rozhraní příkazové řádky kde, požadavky programu jsou zadávány přes parametry příkazové řádky. Těmito parametry lze specifikovat, požadované volané funkce programu a výstupem řešení programu jsou ve formě textového souboru (možné u vstupu i výstupu).

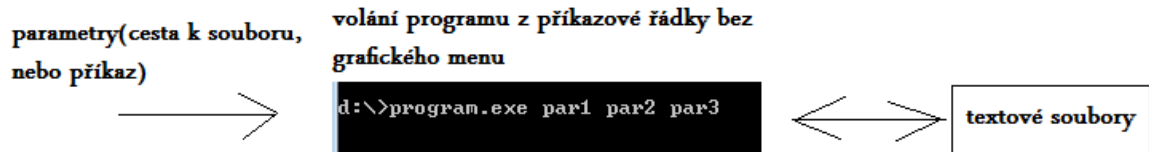
```
{
    fileIn.open(argv[5],std::ios_base::in);
    if(fileIn.is_open())
        cteniDatZeSouboruVstup(fileIn,u,SISO);
    else throw vyjm::FileError("Soubor : " + std::string((argv[5]== NULL) ? "" : argv[5]) + " se nepodarilo
    otevrit");
    fileIn.close();
} else throw vyjm::ArgumentError("Spatny pocet parametru ->" + std::string(IntToString(argc)) + " Chybi
vstup do vystupu systemu\n");

fileOut.open(argv[4],std::ios_base::out);
if(fileOut.is_open()) rozhrani->CSVexport(rozhrani->vystup_systemu(u),fileOut);
else throw vyjm::FileError("Soubor : " + std::string((argv[4]== NULL) ? "" : argv[4]) + " se nepodarilo
otevrit");
fileOut.close();
}
else if(!strcmp(argv[3],"PrevodNaStav" )
{
    if(!strcmp(argv[1],"VnejsiSISO"))
    {
        CLDDS_Vnejsi_SISO * vnejsiSISO = dynamic_cast<CLDDS_Vnejsi_SISO*>(rozhrani);
        CLDDS_Stav_SISO stavSISO(*vnejsiSISO);

        fileOut.open(argv[4],std::ios_base::out);

        if(fileOut.is_open())
            zapisDatDoSouboruStav(fileOut,stavSISO);
        else throw vyjm::FileError("Soubor : " + std::string((argv[4]== NULL) ? "" : argv[4]) + " se nepodarilo
otevrit");
        fileOut.close();
    }
    else if(!strcmp(argv[1],"VnejsiMIMO")
    {
        CLDDS_Vnejsi_MIMO * vnejsiMIMO = dynamic_cast<CLDDS_Vnejsi_MIMO*>(rozhrani);
        CLDDS_Stav_MIMO stavMIMO(*vnejsiMIMO);
```

Obr. 5 Náhled vývojového prostředí na OS Mac, v programu XCode



Obr. 6 Schéma činnosti programu

6.1 Vytvoření systému

Na začátku programu lze vytvořit, nový systém, a to SISO MIMO vnější, stavový a jejich kombinace.

Na této ukázce je definice třídy pro stavové SISO:

```
class CLDDS_Stav_SISO: public CLDDS,CLDDS_Stav
{
public:
    CLDDS_Stav_SISO();
    CLDDS_Stav_SISO(const Matrix<double> & F,const Matrix<double> & G,const Matrix<double> & H,
        const Matrix<double> & L,const Matrix<double> & x0,const int & n,
        const double & T):CLDDS_Stav(F,G,H,L,x0,n,T){}
    CLDDS_Stav_SISO(CLDDS_Vnejsi_SISO & vnejsiSISO);
    CLDDS_Stav_SISO operator=(CLDDS_Vnejsi_SISO & vnejsiSISO);
    const Matrix<double> & get_F(){return F;}
    const Matrix<double> & get_G(){return G;}
    const Matrix<double> & get_H(){return H;}
    const Matrix<double> & get_L(){return L;}
    const Matrix<double> & get_x0(){return x0;}
    virtual Matrix<double> vystup_systemu(Matrix<double> vstup)throw(std::logic_error);
    virtual Matrix<double> impulsni_funkce();
    virtual Matrix<double> prechodova_funkce();
    virtual bool stabilni(){return stavova_stabilita();}
    friend CLDDS_Vnejsi_SISO & operator>>(CLDDS_Stav_SISO & stav_SISO,CLDDS_Vnejsi_SISO & vnejsi_SISO);
    void CSVexport(Matrix<double> vystup,std::fstream & f)throw(std::logic_error);
};
```

Definice třídy objektu systému vnější SISO:

```
class CLDDS_Vnejsi_SISO:public CLDDS
{
private:
    Matrix<double> a,b;
    Polynom A,B;
    int r,s;
    double T;
    double b0;
    void prevodNaK();
    void prevodNaZ();
    void kontrola(const Polynom & A, const Polynom & B) throw(std::logic_error);
public:
    CLDDS_Vnejsi_SISO();
    CLDDS_Vnejsi_SISO(const Polynom & A, const Polynom & B);
    CLDDS_Vnejsi_SISO(const Zlomek<Polynom> & AB);
    CLDDS_Vnejsi_SISO(CLDDS_Stav_SISO & stav_SISO);
    CLDDS_Vnejsi_SISO operator=(CLDDS_Stav_SISO & stav_SISO);
    const Polynom & get_A(){return A;}
    const Polynom & get_B(){return B;}
    virtual Matrix<double> vystup_systemu(Matrix<double> vstup)throw(std::logic_error);
    virtual Matrix<double> impulsni_funkce();
    virtual bool stabilni();
    virtual Matrix<double> prechodova_funkce();
    virtual void CSVexport(Matrix<double> vystup,std::fstream & f)throw(std::logic_error);
    friend CLDDS_Stav_SISO & operator>>(CLDDS_Vnejsi_SISO & vnejsi_SISO,CLDDS_Stav_SISO & stav_SISO);
};
```

Definice třídy objektů stavové MIMO:

```
class CLDDS_Stav_MIMO: public CLDDS, CLDDS_Stav
{
private:
```

```

    int l; // pocet vystupu

    int m; // pocet vstupu

public:
    CLDDS_Stav_MIMO();

    CLDDS_Stav_MIMO(const int & l,const int & m,const Matrix<double> & F,
                    const Matrix<double> & G,const Matrix<double> & H,
                    const Matrix<double> & L,const Matrix<double> & x0,
                    const int & n,const double & T)throw(std::logic_error);

    CLDDS_Stav_MIMO(CLDDS_Vnejsi_MIMO & vnejsiMIMO);

    CLDDS_Stav_MIMO operator=(CLDDS_Vnejsi_MIMO & vnejsiMIMO);

    Matrix<double>get_F(){return F;}

    Matrix<double>get_G(){return G;}

    Matrix<double>get_H(){return H;}

    Matrix<double>get_L(){return L;}

    Matrix<double>get_x0(){return x0;}

    virtual Matrix<double> vystup_systemu(Matrix<double> vstup)throw(std::logic_error);

    virtual Matrix<double> impulsni_funkce();

    virtual Matrix<double> prechodova_funkce();

    virtual bool stabilni(){ return stavova_stabilita();}

    friend CLDDS_Vnejsi_MIMO & operator>>(CLDDS_Stav_MIMO & stav_MIMO,CLDDS_Vnejsi_MIMO &
    vnejsi_MIMO)throw(std::logic_error);

    void CSVexport(Matrix<double> vystup,stdout::fstream & f)throw (std::logic_error);

};

```

Definice třídy objektů vnější MIMO:

```

class CLDDS_Vnejsi_MIMO:public CLDDS
{
private:
    Matrix<Polynom> a;

    Matrix<Polynom> b;

    Matrix<Polynom> A;

    Matrix<Polynom> B;

```

```

Matrix<int> r,s;

int l,m;

double T;

int maxStupen_r;

int maxStupen_s;

int maxStupenA;

Matrix<double> b0;

void najdiNejvysiStupne();

void prevodNaZ();

void prevodNaK();

void kontrola(const Matrix<Polynom> & A,const Matrix<Polynom> & B)throw(std::logic_error);

void upravaA_B(Matrix<Polynom> & tmp_A,Matrix<Polynom> & tmp_B);

public:

CLDDS_Vnejsi_MIMO();

CLDDS_Vnejsi_MIMO(CLDDS_Stav_MIMO & stav_mimo);

CLDDS_Vnejsi_MIMO operator=(CLDDS_Stav_MIMO & stav_mimo);

CLDDS_Vnejsi_MIMO(const Matrix<Polynom> & A,const Matrix<Polynom> & B);

const Matrix<Polynom> & get_A(){return A;}

const Matrix<Polynom> & get_B(){return B;}

virtual Matrix<double> vystup_systemu(Matrix<double> vstup)throw(std::logic_error);

virtual Matrix<double> impulsni_funkce();

virtual bool stabilni();

virtual Matrix<double> prechodova_funkce();

friend CLDDS_Stav_MIMO & operator>>(CLDDS_Vnejsi_MIMO & vnejsiMIMO ,CLDDS_Stav_MIMO &
stavMIMO);

virtual void CSVexport(Matrix<double> vystup,std::fstream & f)throw(std::logic_error);

};

```

6.2 Polymorfismus a diskrétní systémy

Kód pro rozhraní jednotlivých systému

```

class CLDDS
{
    // cira metoda
public:
    virtual Matrix<double> vystup_systemu(Matrix<double> vstup) = 0;
    virtual Matrix<double> impulsni_funkce() = 0;
    virtual Matrix<double> prechodova_funkce() = 0;
    virtual bool stabilni() = 0;
    virtual void CSVexport(Matrix<double> vystup, std::ofstream & f) = 0;
};

```

Kód pro výpočet impulsní charakteristiky pro SISO vnější

```

Matrix<double> CLDDS_Vnejsi_SISO::impulsni_funkce()
{
    Matrix<double> u(50,1),i(50,1),p_y(r,1),p_u(s,1);
    double tmp_y = 0.0;
    double tmp_u = 0.0;
    u[0][0] = 1.0;
    for(int k = 0; k < 50;k++)
    {
        tmp_y = 0.0;
        tmp_u = 0.0;
        for(int o = 0; o < r;o++)
            tmp_y += a[o][0] * (p_y[o][0] * (-1));
        for(int o = 0; o < s;o++)
            tmp_u += b[o][0] * (p_u[o][0]);
        i[k][0] = tmp_y + tmp_u;
        if(k == 0) i[k][0] += b0;
        for(int q = r - 1; q > 0;q--)
            p_y[q][0] = p_y[q - 1][0];
        p_y[0][0] = i[k][0];
        for(int q = s - 1; q > 0;q--)
            p_u[q][0] = p_u[q - 1][0];
    }
}

```

```

    p_u[0][0] = u[k][0];
}
return i;
}

```

Kód pro výpočet impulsní charakteristiky pro MIMO vnější

```

Matrix<double> CLDDS_Vnejsi_MIMO::impulzni_funkce()
{
    Matrix<double> u(m,50),i(l,50),p_y(l,maxStupen_r),p_u(m,maxStupen_s);
    double tmp = 0.0;
    for(int o = 0; o < m; o++)
        u[o][0] = 1.0;
    for(int k = 0; k < 50; k++)
    {
        tmp = 0.0;
        for(int x = 0; x < l; x++)
        {
            for(int z = 0; z < m; z++)
            {
                for(int j = 0; j < r[x][z]; j++)
                    tmp += a[x][z][j] * p_y[z][j] * (-1);
                for(int j = 0; j < s[x][z]; j++)
                    tmp += b[x][z][j] * p_u[z][j];
            }
            i[x][k] = tmp;
            for(int j = 0; j < b0.get_columns() && k == 0; j++)
                i[x][k] += b0[x][j];
            tmp = 0.0;
        }
        for(int qq = 0; qq < l; qq++)
        {
            for(int q = maxStupen_r - 1; q > 0; q--)

```

```

        p_y[qq][q] = p_y[qq][q - 1];
        p_y[qq][0] = i[qq][k];
    }
    for(int qq = 0; qq < m; qq++)
    {
        for(int q = maxStupen_s - 1; q > 0; q--)
            p_u[qq][q] = p_u[qq][q - 1];
        p_u[qq][0] = u[qq][k];
    }
}
return i;
}

```

Kód pro výpočet impulsní charakteristiky pro MIMO stavový:

```

Matrix<double> CLDDS_Stav_MIMO::impulsni_funkce()
{
    Matrix<double> i(l,50),u(m,50),tmp(l,1),tmpS(l,1),tmpUr(m,1);
    for(int j = 0; j < m; j++)
        u[j][0] = 1.0;
    for(int k = 0; k < 50; k++)
    {
        tmp = H*(F^k)*x0;
        for(int r = 0; r <= (k - 1); r++)
        {
            tmpS = tmpS + (H * (F ^ (k-r-1)) * G * u.submatice(1,m,r + 1,r + 1)); // pozor u submatice cislujeme od 1
        }
        tmp = tmp + L * u.submatice(1,m,k + 1,k + 1) + tmpS; // pozor u submatice cislujeme od 1
        for(int o = 0; o < m; o++)
            i[o][k] = tmp[o][0];
        tmpS = Matrix<double>(l,1);
    }
    return i;
}

```

Kód pro výpočet impulsní charakteristiky pro SISO stavový:

```
Matrix<double> CLDDS_Stav_SISO::impulsni_funkce()
{
    Matrix<double> u(50,1),tmp(1,1),tmpS(1,1),i(50,1);
    u[0][0] = 1.0;
    for(int k = 0; k < (int)i.get_rows(); k++)
    {
        tmp = H * (F^k) * x0;
        for(int r = 0; r <= (k - 1) && u[r][0] != 0; r++)
            tmpS = tmpS + (H * (F^(k-r-1)) * G * u[r][0]);
        tmpS = tmp + tmpS + (L * u[k][0]);
        i[k][0] = tmpS[0][0];
        tmpS = Matrix<double>(1,1);
    }
    return i;
}
```

6.3 Přetěžování operátorů při řešení stavových rovnic

Použití přetěžování operátoru u operací se stavovým popisem MIMO i SISO systému:

```
template<typename T>
Matrix<T> Matrix<T>::operator-(const T & cislo)
{
    Matrix<T> tmp(this->_rows,this->_columns);
    for(int i = 0; i < _count; i++)
        tmp._matrix[i] = _matrix[i] - cislo;
    return tmp;
}

template<typename T>
Matrix<T> Matrix<T>::operator*(const T & cislo)
{
    Matrix<T> tmp(this->_rows,this->_columns);
```

```

    for(int i = 0; i < _count; i++)

        tmp._matrix[i] = _matrix[i] * cislo;

    return tmp;
}

std::ostream & operator<<(std::ostream & os, const Matrix<int> & matrix)
{
    for(int i = 0; i < matrix._count; i++)

    {
        os << std::fixed << std::setprecision(3) << matrix._matrix[i];

        ((i + 1) % matrix._columns == 0) ? os<< std::endl : os << " ";

    }

    return os;
}

std::ostream & operator<<(std::ostream & os, const Matrix< Zlomek<Polynom> > & matrix)
{
    for(int i = 0; i < matrix._count; i++)

    {
        os << std::fixed << std::setprecision(3) << matrix._matrix[i];

        ((i + 1) % matrix._columns == 0) ? os<< std::endl : os << " ";

    }

    return os;
}

```

6.4 Vícenásobná dědičnost použita pro podobnost SISO a MIMO stavového popisu

Použití vícenásobné dědičnosti při použití SISO i MIMO stavového popisu

```

class CLDDS_Stav_MIMO: public CLDDS, CLDDS_Stav
{
    ... Následuje další část kódu
};

```

Kód pro rodičovskou třídu od níž je odvozeno SISO i MIMO stavové:

```
class CLDDS_Stav
{
protected:
    Matrix<double> F,G,H,L,x0;
    int n;
    double T;
    void kontrola(const Matrix<double> & F,const Matrix<double> & G,const Matrix<double> & H,
        const Matrix<double> & L,const Matrix<double> & x0,const int & n,const double & T);
public:
    CLDDS_Stav();
    CLDDS_Stav(const Matrix<double> & F,const Matrix<double> & G,const Matrix<double> & H,
        const Matrix<double> & L,const Matrix<double> & x0,const int & n,const double & T);
    bool stavova_stabilita();
};
```

6.5 Využití kopírovacího konstruktora při převodu typů popisu systému.

Úkazka kódu pro použití kopírovacího konstruktora

```
CLDDS_Stav_MIMO::CLDDS_Stav_MIMO(CLDDS_Vnejsi_MIMO & vnejsiMIMO)
{
    vnejsiMIMO >> (*this);
}
CLDDS_Stav_MIMO & operator>>(CLDDS_Vnejsi_MIMO & vnejsiMIMO ,CLDDS_Stav_MIMO & stavMIMO)
{
    Matrix<Polynom> tmp_A;
    Matrix<Polynom> tmp_B;
    tmp_A = vnejsiMIMO.A;
    tmp_B = vnejsiMIMO.B;
    int maxDelka = vnejsiMIMO.maxStupenA + 1;
    Pole< Matrix<double> > poleMatrixA(maxDelka);
```

```

Pole< Matrix<double> > poleMatrixB(maxDelka);

for(int i = maxDelka - 1 ,ii = 0; i >= 0 ; i--,ii++)

{

    poleMatrixA[ii] = Matrix<double>(vnejsiMIMO.l,vnejsiMIMO.m);
    poleMatrixB[ii] = Matrix<double>(vnejsiMIMO.l,vnejsiMIMO.m);
    for(int j = 0; j < vnejsiMIMO.l; j++)
        for(int k = 0; k <vnejsiMIMO.m; k++)
            {
                poleMatrixA[ii][j][k] = ((tmp_A[j][k]).get_stupen() >= i) ? tmp_A[j][k][i] : 0;
                poleMatrixB[ii][j][k] = ((tmp_B[j][k]).get_stupen() >= i) ? tmp_B[j][k][i] : 0;
            }
        }
}

Pole< Matrix<double> > poleMatrixR(maxDelka);
Pole< Matrix<double> > poleMatrixS(maxDelka);
Matrix<double> inverzniMatice = poleMatrixA[0].inverze();
for(int i = 0; i < maxDelka; i++)
{
    Matrix<double> soucinA = poleMatrixA[i] * inverzniMatice;
    poleMatrixR[i] = (i == 0) ? soucinA : soucinA * (-1.0);
    poleMatrixS[i] = poleMatrixB[i] - soucinA * poleMatrixB[0];
}

if(vnejsiMIMO.l == vnejsiMIMO.m)
{
    Matrix<double> F((maxDelka - 1) * 2,(maxDelka - 1) * 2);
    Matrix<double> G((maxDelka - 1) * 2,vnejsiMIMO.l);
    Matrix<double> H(vnejsiMIMO.l,(maxDelka - 1) * 2);
    Matrix<double> L = inverzniMatice * poleMatrixB[0];
    for(int i = 0, k = 0; i < F.get_rows(); i+= vnejsiMIMO.l,k++)
    {
        F.vlozSubmatici(poleMatrixR[k + 1],i,0);
        G.vlozSubmatici(poleMatrixS[k + 1],i,0);
    }
}

```

```

}

for(int i = vnejsiMIMO.m, j = 0; j < F.get_rows() - vnejsiMIMO.m; i++, j++)

    F[j][i] = 1.0;

H.vlozSubmatici(inverzniMatice, 0, 0);

stavMIMO =
CLDDS_Stav_MIMO(vnejsiMIMO.m, vnejsiMIMO.l, F, G, H, L, Matrix<double>(F.get_rows(), 1), F.get_rows(), vnejsiMIMO.T);
}

return stavMIMO;
}

```

6.6 Vyjímky a jejich použití při testování singularity matic popisu systému

Kód pro testování singularity matice popisu systému:

```

template<typename T>
Matrix<T> Matrix<T>::inverze(void) throw(std::logic_error)
{
    T _det = det();
    if(_det != 0)
    {
        Matrix<T> tmp(this->_rows, this->_columns);
        for(int i = 0; i < tmp._count; i++)
        {
            detk = (submatice(i % _columns + 1, i / _rows + 1)).det();
            aa = tmp._matrix[i] = (((submatice(i % _columns + 1, i / _rows + 1)).det()) / _det) * (pow(-1.0, (i % _columns) + (i / _rows) + 2));
        }
        return tmp;
    }
    else throw vyjm::Null_det("Nelze udelat inversni matici nulovy determinant\n");
}

```

```

catch(vyjm::Bad_index & ex)
{
    std::cout << ex.what();
    std::cout << ex.chybny_index();
}
catch(vyjm::Null_det & nulldet)
{
    std::cout << nulldet.what();
}
catch(vyjm::Max_sizePolynom & maxSizePolynom)
{
    std::cout << maxSizePolynom.what();
    std::cout << "Dana velikost je " << maxSizePolynom.get_velikost() << std::endl;
}
catch(vyjm::ArgumentError & ArgumentError)
{
    std::cout << ArgumentError.what();
}
catch(vyjm::FileError & fileError)
{
    std::cout << fileError.what();
}

```

6.7 Kontrolní kódy fáze ověřování v prostředí Matlab

Byli zvoleny následující kódy, pro ověření důležitých partií při implementaci programu.

Kód pro ověřování práce se SISO vnější:

```

ds = tf([9 8 7 6 5],[7 7 6 6 4],1)
[i,t,k] = step(ds);
impulse(ds);
i;
roots([7 7 6 6 4]);
%[h,t,k] = step(ds)
%step(ds)é

```

Kód pro ověřování práce se SISO stavový:

```
%Třída CLDDS_Stavove_SISO
F = [0.98 0.24; -0.12 0.93];
G = [0.02;0.15];
H = [1 0];
L = [0];
T = 1;
diskrStavovySISO = ss(F,G,H,L,T,s);
ss2tf(diskrStavovySISO)
%impulse
%i,t,x] = impulse(diskrStavovySISO);
%***
[h,t,x] = step(diskrStavovySISO);
h;
```

Kód pro ověřování práce se MIMO vnější:

```
%Gz = tf([0.25 0 0.25 -1 0],[0 1 -0.5 1 0];[1 0 0 -1 0],[0 0.5 -2 1 0]],[[0.5 0 0.5 0 1],[0.5 0 0.5 0 1];[0.5 0 0.5 0 1],[0.5 0 0.5 0 1]],1)
%Gz = tf([30792.7937752510 -32045.3011623675 -73577.8649861530 -1177.3003753905],[-21473.2045977578 22448.1417581707 47884.3417923739 700.0480403572];[-30270.7979496634 31693.2236208567 79653.6212623956 1004.7673522333],[20002.8033444477 -20840.6441912592 -52278.2379936682 -582.2252484066]],[[695.4489818588 722.6410370535 248.8397685546 -25.1946492155 1],[695.4489818588 722.6410370535 248.8397685546 -25.1946492155 1];[695.4489818588 722.6410370535 248.8397685546 -25.1946492155 1],[695.4489818588 722.6410370535 248.8397685546 -25.1946492155 1]],1)%implicitni
%Gz = tf([1 -2 1 0 0],[2 -1 2 0];[4 -2 2 0 0],[2 -4 1 0]],[[2 0 1 0 1],[2 0 1 0 1];[2 0 1 0 1],[2 0 1 0 1]],1)
%uciteluv priklad
%Gz = tf([-44.3413 -60.1360 71.7842 -2.4341 0.0000],[30.9212 -79.0011 105.1199 103.2046 0.0000];[43.5897 -93.4589 103.3389 193.5435 -0.0000],[-28.8038 143.4776 -229.3313 -25.8012 0.0000]],[[1.0000 -25.1946 248.8398 -722.6410 695.4490],[1.0000 -25.1946 248.8398 -722.6410 695.4490];[1.0000 -25.1946 248.8398 -722.6410 695.4490],[1.0000 -25.1946 248.8398 -722.6410 695.4490]],1)
%Gz = tf([0 -2.4341 71.7842 -60.1360 -44.3413],[0 103.2046 105.1199 -79.0011 30.9212];[0 193.5435 103.3389 -93.4589 43.5897],[0 -25.8012 -229.3313 143.4776 -28.8038]],[[695.4490 -722.6410 248.8398 -25.1946 1],[695.4490 -722.6410 248.8398 -25.1946 1];[695.4490 -722.6410 248.8398 -25.1946 1],[695.4490 -722.6410 248.8398 -25.1946 1]],1)
```

```
%Gz = tf([-0.004 0.1039 -0.08664 -0.06395],[0.148 0.1517 -0.1136 0.04456],[0.278 0.149 -0.1349 0.06266],[-0.037 -
0.3299 0.2065 -0.04115]},{[1 -1.039 0.3572 -0.03597 0.001578],[1 -1.039 0.3572 -0.03597 0.001578];[1 -1.039
0.3572 -0.03597 0.001578],[1 -1.039 0.3572 -0.03597 0.001578]},1);
```

```
Gz = tf([0 0 1 -2 1],[2 -1 2 0];[0 2 -2 0 4],[1 4 2 0]},{[1 0 1 0 2],[1 0 1 0 2];[1 0 1 0 2],[1 0 1 0 2]},1)
```

```
step(Gz);
```

```
[i,t,x] = step(Gz);
```

```
i
```

```
%i
```

Kód pro ověřování práce se MIMO stavový:

```
%Trida CLDDS_Stavove_MIMO
```

```
%F = [1.28 0.28 -0.59 -0.15; -0.87 0.15 -0.84 -0.3; -1.76 -0.29 1.28 0.49; -2.87 -0.59 0.87 0.32];
```

```
%G = [0.66 0.11; -1.56 -0.19; -0.63 1.45; -1.33 -2.44];
```

```
%H = [1 0 0 0;0 0 1 0];
```

```
%L = [0 0;0 0];
```

```
%T = 1;
```

```
%
```

```
% F = [-32.279 -70.151 1.000 0.000; 29.086 57.474 0.000 1.000; 57.722 124.972 0.000 0.000; -61.617 -121.356 0.000
0.000];
```

```
% G = [-24.882 17.533;20.913 -13.700;44.341 -30.921;-43.590 28.804];
```

```
% H = [-57.722 -124.972 0.000 0.000;61.617 121.356 0.000 0.000];
```

```
% L = [-44.341 30.921;43.590 -28.804];
```

```
% T = 1;
```

```
% F = [0 -1 1 0; -1 0 0 1; -0.5 0 0 0; 0 -1 0 0];
```

```
% G = [-2 2;-1 1;-0.5 0;-2 0];
```

```
% H = [0.5 0 0 0;0 1 0 0];
```

```
% L = [0.5 0;2 0];
```

```
% T = 1
```

```
% F = [0.583 0.022 1.000 0.000; -0.017 0.456 0.000 1.000 ; -0.174 -0.180 0.000 0.000; 0.089 0.083 0.000 0.000];
```

```
% G = [-0.004 0.148;0.278 -0.037;0.096 0.220;0.311 -0.349];
```

```
% H = [1.000 -0.000 0.000 0.000;-0.000 1.000 0.000 0.000];
```

```
% L = [0 0;0 0];
```

```
% T = 1;
```

```
F = [-0.5 0 1 0 0 0;0 -0.143 0 1 0 0;0 -0.143 0 0 1 0;-0.5 0 0 0 0 0;-0.25 0 0 0 0 0;0 -0.143 0 0 0 0];
```

```
G = [1 3;0.857 1;2.857 2;0 7;0 0;-1.143 0];
```

```
H = [0.25 0 0 0 0 0;0 0.143 0 0 0 0];
```

```
L = [0 0;1.143 0];
```

```
T = I;
```

```
diskrStavovyMIMO = ss(F,G,H,L,T)
```

```
tf(ss(F,G(:,1),H,L(:,1)))
```

```
tf(ss(F,G(:,2),H,L(:,2)))
```

```
%impulse
```

```
[i,t,x] = step(diskrStavovyMIMO);
```

```
i
```

```
***
```

```
%[h,t,x] = step(diskrStavovySISO);
```

```
%impulse(diskrStavovySISO)
```

```
%[h,t,x] = step(diskrStavovyMIMO);
```

```
%abs(eig(F))
```

7 IMPLEMENTACE NUMERICKÝCH METOD

Byly implementovány dvě numerické metody. Metoda Graeffova a Gaussova eliminační metoda. Graeffova metoda slouží pro ověřování stability polynomu, což lze aplikovat pro analýzu napříč všemi variantami popisů systému. To to je základním těžištěm pro řešení stability ve vytvořeném programu.

7.1 Graeffova metoda

Graeffova metoda vypočítá přibližné hodnoty n-tého polynomu při položení rovno nule. Vypočítá absolutní hodnoty kořenu, to jak pro reálné, tak komplexní proměnné.

Ukázka kódu:

```
void Rovnice::GraeffSolution()
{
    try
    {
        Pole<double> *pole = new Pole<double>[K];
        for(int i = 0; i < K; i++)
            pole[i]._resize(delka_polynom);
        for(int i = 0; i < K; i++)
        {
            for(int j = stupen, k = 0; j >= 0; j--, k++)
            {
                if(!i)
                    pole[i][j] = pole_polynoms[j];
                else
                {
                    pole[i][j] = pow(pole[i - 1][j], 2);
                    if(j < stupen)
                    {
                        char znamenko = (k % 2) ? -1 : 1;
                        int k1 = stupen;
                        int k2 = (stupen - k * 2);
```

```

        while(k1 > j)
        {
            pole[i][j] += znamenko * 2 * pole[i - 1][k1--] * (k2 < 0 ? 0 : pole[i - 1][k2]);
            k2++;
            znamenko *= -1;
        }
        if(k % 2)
            pole[i][j] *= (-1);
    }
}
}
}

double exponent = 1/pow(2.0,(K - 1));
for(int i = 0; i < pole[K - 1].get_length() - 1; i++)
{
    if(changingSign(pole,i) == false)
        solution[i] = pow(fabs(pole[K - 1][i] / pole[K - 1][i + 1]),exponent);
    else
        solution[i] = sqrt(pow(fabs(pole[K - 1][komplexSloupec - 1] / pole[K - 1][komplexSloupec + 1]),exponent));
}

delete [] pole;
}

catch(std::bad_alloc & bd)
{
    std::cout << bd.what() << std::endl;
}
}
}

```

7.2 Gaussova eliminační metoda

Gaussova eliminační metoda slouží pro výpočet determinantů, kdy se daná matice převede na trojúhelníkový tvar a podle diagonálních prvků se vypočítá determinant.

Ukázka kódu:

```
template<typename T>
Matrix<T> Matrix<T>::Prevod_na_Trojuhelnikovy_tvar(Pole<T> & RozsirujiciCisla)
{
    Matrix<T> tmp(_rows, _columns, _matrix);

    int o = 0;

    for(int i = 0; i < _rows; i++)
    {
        // std::cout << tmp << std::endl;

        if(tmp[i][i] == 0)
            if(!uprava_radku(tmp,i)) return tmp;

        T vrchniHodnota = tmp[i][i];

        // std::cout << tmp << std::endl;

        for(int j = i + 1; j < _rows; j++)
        {
            T spodniHodnota = tmp[j][i];

            if(spodniHodnota != 0.0)
            {
                o++;

                RozsirujiciCisla._resize(o);

                RozsirujiciCisla[o - 1] = vrchniHodnota;

                for(int k = i; k < _columns; k++)
                {
                    tmp[j][k] = tmp[i][k] * spodniHodnota * (-1) + vrchniHodnota * tmp[j][k];
                }
            }
        }
    }

    return tmp;
}
```

8 OVLÁDÁNÍ PROGRAMU

Ovládání přes příkazovou řádku, funguje na principu předávání parametrů za název programu, které mohou plnit funkci identifikátoru názvu operace, nebo definovat cestku k souboru. Následná tabulka zahrnuje výčet důležitých kombinací práce s programem, ukázkou volání těchto operací, tykajících se oblasti LDDS.

<i>Typ popisu systému</i>	<i>Druh operace</i>	<i>Postup volání z příkazové řádky</i>
SISO Vnější	Impulsní charakteristika	PraktickaCast.exe VnejsiSISO vstup.txt Impulse vystupImpulse.txt
	Přechodová charakteristika	PraktickaCast.exe VnejsiSISO vstup.txt Prechod vystupPrechod.txt
	Odezva na vstupní signál	PraktickaCast.exe VnejsiSISO vstup.txt Vystup ObecnyVystup.txt vstupU.txt
	Převod na Vnitřní popis	PraktickaCast.exe VnejsiSISO vstup.txt PrevodNaStav StavovyPopis.txt
	Stabilita	PraktickaCast.exe VnejsiSISO vstup.txt Stabilita
MIMO Vnější	Impulsní charakteristika	PraktickaCast.exe VnejsiMIMO vstup.txt Impulse vystupImpulse.txt
	Přechodová charakteristika	PraktickaCast.exe VnejsiMIMO vstup.txt

		Prechod vystupPrechod.txt
	Odezva na vstupní signál	PraktickaCast.exe VnejsiMIMO vstup.txt Vystup ObecnyVystup.txt vstupU.txt
	Převod na Vnitřní popis	PraktickaCast.exe VnejsiMIMO vstup.txt PrevodNaStav StavovyPopis.txt
	Stabilita	PraktickaCast.exe VnejsiMIMO vstup.txt Stabilita
MIMO Stavové	Impulsní charakteristika	PraktickaCast.exe StavMIMO vstup.txt Impulse vystupImpulse.txt
	Přechodová charakteristika	PraktickaCast.exe StavMIMO vstup.txt Prechod vystupPrechod.txt
	Odezva na vstupní signál	PraktickaCast.exe StavMIMO vstup.txt Vystup ObecnyVystup.txt vstupU.txt
	Převod na Vnější popis	PraktickaCast.exe StavMIMO vstup.txt PrevodNaVnejsi VnejsiPopis.txt
	Stabilita	PraktickaCast.exe StavMIMO vstup.txt Stabilita

SISO Stavové	Impulsní charakteristika	PraktickaCast.exe StavSISO vstup.txt Impulse vystupImpulse.txt
	Přechodová charakteristika	PraktickaCast.exe StavSISO vstup.txt Prechod vystupPrechod.txt
	Odezva na vstupní signál	PraktickaCast.exe StavSISO vstup.txt Vystup ObecnVystup.txt vstupU.txt
	Převod na Vnější popis	PraktickaCast.exe StavMIMO vstup.txt PrevodNaVnejsi VnejsiPopis.txt
	Stabilita	PraktickaCast.exe StavSISO vstup.txt Stabilita

Tab. 3 Příkazy pokročilého ovládání

Na přiloženém CD v podadresářích se nacházejí připravené příklady volání programu spustitelné přes přítomné *.bat soubory, kdy textové soubory zahrnují vlastní definice systémů. Výstupy programů jsou směrovány též do textových souborů. V adresáři na CD spustitelný soubor pro MAC - OS.

ZÁVĚR

Těžištěm práce bylo zmapování známých i specifických vlastností a aspektů OOP přístupu se zaměřením na C++, a to ve prospěch zejména analýzy LDDS. Byly vybrány kapitoly diskrétního řízení. C++ je chápán jako prostředek pro realizaci cílů praktické oblasti diskrétního řízení. C++ není zkoumán samostatně, ale jeho použití jako nástroje pro realizaci konkrétní oblasti praxe. Důkaz přínosu zmapovaných specifik OOP v C++ je vytvořením aplikace. Byla vytvořena konzolová aplikace, popsána z pohledu jejího ovládání, která zvládá řešit, právě vybrané kapitoly z diskrétního řízení. Dokáže analyzovat systémy různých matematických popisů a pracovat s nimi z pohledu signálů. Byly implementovány, též metody (matice, polynomy, operace numerické i symbolické, operace se zlomky). Byla ověřena správnost v řešení programu Matlab. Bylo možné naprogramovat i složité záležitosti z teorie systémů vlastními silami s použitím numerických metod a výhod aspektů C++. I obtížnější záležitosti jako práce s MIMO systémy. Nebylo cílem pokrýt všechny komplexní funkce teorie systémů, ale prezentace takových operací, které dokazují výhody použití nalezených aspektů. Vlastní přístup je opakem nasazení komerčních prostředků Matlab nebo volně dostupného prostředí Scilab.

ZÁVĚR V ANGLIČTINĚ

The aim of this thesis was to find the known and the specific features and aspects of OOP approach, focusing on C++, in particular for analysis of LDDS. They were selected chapters of discrete control. C++ is seen as a means for achieving the objectives of the practical field of discrete control. C++ is studied separately, but its use as a tool for the implementation of specific areas of practice. Evidence of mapped specifics OOP in C++ for developing applications. It was created a console application, described in terms of its control, which manages to deal with just the selected chapters of discrete control. It could analyze systems of different mathematical descriptions and works with signals. Also were implemented the methods (matrix, polynomials, numerical and symbolic operations, operations with fractions). It was verified the solution in Matlab. It can't be programmed all complex issues of systems theory using numerical methods and aspects of C ++ .For example were implemented the issues about MIMO systems. This own solution is the complement opposite of commercial tools Matlab and free available Scilab.

SEZNAM POUŽITÉ LITERATURY

- [1] BALÁTĚ, Jaroslav. *Automatické řízení*. Praha: BEN, 2003. ISBN 80-7300-020-2.
- [2] STREJČ, Vladimír. *Stavová teorie lineárního diskrétního řízení*. Praha: Nakladatelství Československé akademie věd, 1978.
- [3] KUČERA, Vladimír. *Analysis and Design of Discrete Linear Control Systems*. Praha: Nakladatelství Československé akademie věd, 1991. ISBN 80-200-0252-9.
- [4] VITÁSEK, Emil. *Numerické metody*. Praha: Státní nakladatelství technické literatury, 1987.
- [5] KUČERA, Luděk a Jaroslav NEŠETŘIL. *Algebraické metody diskrétní matematiky*. Praha: SNTL, 1989. ISBN 80-03-00107-2.
- [6] DEMIDOVIČ, Boris Pavlovič a Isaak Abramovič MARON. *Základy numerické matematiky*. Praha: Státní nakladatelství technické literatury, 1966.
- [7] VLACH, Milan. *Základní numerické metody*. Praha: Státní nakladatelství technické literatury, 1971.
- [8] LIBERTY, Jesse. *Naučte se C++ za 21 dní*. Praha: Computer Press, 2002. ISBN 80-7226-774-4.
- [9] POKORNÝ, Pavel. *Objektově orientované programování v C++*. 2. vydání. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005. ISBN 80-7318-330-7.
- [10] SUTTER, Herb a Andrei ALEXANDRESCU. *C++: 101 programovacích technik*. Brno: Zoner Press, 2005. Encyklopedie Zoner Press. ISBN 80-86815-28-5.
- [11] PERŮTKA, Karel. *MATLAB - Základy pro studenty automatizace a informačních technologií*. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005. ISBN 80-7318-355-2.
- [12] KARBAN, Pavel. *Výpočty a simulace v programech Matlab a Simulink*. Brno: Computer Press, 2006. ISBN 978-80-251-1448-3.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

OOP Objektově orientované programování.

LDDS Lineární diskrétní dynamické systémy.

STL Standart template library.

SISO Single input single output

MIMO Multi input multi output

SEZNAM OBRÁZKŮ

Obr. 1 Výstup funkce tf.....	32
Obr. 2 Výstup funkce tf.....	33
Obr. 3 Výstup funkce ss s maticemi F,G	33
Obr. 4 Výstup funkce ss s maticemi H, L	34
Obr. 5 Náhled vývojového prostředí na OS Mac, v programu XCode	38
Obr. 6 Schéma činnosti programu	39

SEZNAM TABULEK

Tab. 1 Využitelnost základních aspektů	36
Tab. 2 Využití specifických aspektů	37
Tab. 3 Příkazy pokročilého ovládání	60

SEZNAM PŘÍLOH

P1: nosič CD-ROM